BAB 3
PELAKSANAAN KERJA MAGANG

Selama lima bulan kerja magang yang dilakukan, terdapat tugas yang harus dikerjakan agar tujuan dari kerja magang ini tercapai. Pada bab ini, akan dijelaskan kedudukan, tugas-tugas, dan kendala serta solusi yang dilakukan selama kerja magang ini.

3.1 Kedudukan dan Organisasi

Kedudukan kerja magang ini adalah IT Developer di divisi MAI (*Management Automation Information*). Pembuatan *website* CMS (*Content Management System*) ini dilakukan oleh seorang diri dengan beberapa bantuan dari rekan kerja sekitar. Mayoritas ketentuan tugas harian diberikan oleh supervisor dan terkadang dari rekan kerja sekitar agar *website* CMS sesuai dengan yang *user* inginkan.

3.2 Tugas yang Dilakukan

Tugas kerja magang yang dilakukan sesuai dengan tujuan magang, yaitu membuat *website* CMS untuk aplikasi internal OT. Konten-konten yang dapat diatur melalui *website* CMS ini tidak hanya konten yang berada di OTLink, tetapi juga untuk beberapa aplikasi perusahaan OT lainnya seperti ABBA (Arta Boga Bersama Anda). Selain dari pembuatan *website* CMS, tugas penulisan laporan ini juga dibuat sesuai peraturan kampus dan bimbingan dosen pembimbing.

Dalam melaksanakan kerja magang ini, terdapat tugas harian yang dilakukan. Agar tulisan lebih ringkas, maka tugas harian tersebut dirangkum menjadi per mingguan. Berikut pelaksanaan kerja magang yang dilakukan selama lima bulan diuraikan seperti pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	Pekerjaan yang dilakukan
1	• Mempelajari struktur proyek dan Laravel. • Memperbaiki masalah di proyek yang sudah ada dan membuat struktur dan fungsi dasar *website* (*login* dan pemasukan data *banner*).
2	• Menambah fitur pemasukan data *news*, *event*, dan swab (jadwal swab dan swab pantry). • Menambah fitur untuk menampilkan data konten (*banner*, *news*, *event*) yang sudah ada ke suatu tabel. (Bagian A.) • Membuat struktur dasar untuk menambah/mengurangkan menu samping *user*.
3	• Membuat tampilan dan penambah/pengurangan menu samping lebih adaptif terhadap perubahan data di *database*. (Bagian B.) • Menambah fitur *reset* PIN. • Penambahan konten *banner event* di aplikasi OTLink juga dapat masuk untuk aplikasi ABBA. • Menambahkan fitur untuk edit dan hapus pada konten-konten OTLink yang sudah ada. • Membuat dokumentasi *website* mengenai *software* dan *library* apa yang dipakai.
4	• Membuat fitur di CMS OTlink untuk menambahkan konten (buletin dan inbox) ke aplikasi ABBA. • Mengimplementasikan fitur *rich text input* dengan TinyMCE. (Bagian C.3) • Melakukan testing dan pembetulan masalah yang ada.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang (lanjutan)

5	• Membuat menu untuk menambahkan menu (*parent* dan *child*) di aplikasi OTLink. (Bagian C.4) • Menambahkan fitur untuk menambahkan menu di CMS OTLink. • Menambahkan menu ABBA *Event* untuk menambahkan *event* baru khusus untuk ABBA. • Menambahkan menu *Manage User* untuk memberikan akses ke CMS OTLink. • Membetulkan masalah yang ditemukan.
6	• Mendukung *input text emoji* agar dapat masuk ke *database*. • Membuat petunjuk dan perubahan tampilan agar lebih *user friendly*. • Menambah fitur edit untuk konten-konten ABBA. • Menambah menu ABBA *Upload* Bintang dan ABBA Hadiah Bintang untuk mengakomodasi fitur baru di aplikasi ABBA. • Membetulkan masalah *website* dan menerapkan permintaan *user*.
7	• Membetulkan masalah *website* dan menerapkan permintaan *user*. • Membuat menu ABBA Bintang di CMS OTLink. • Membuat *webview* ABBA di CI (CodeIgniter) untuk menukarkan bintang dengan barang tertentu. (Bagian 3.3.2)
8	• Belajar Flutter untuk membuat menu baru di aplikasi OTLink. • Membuat tampilan isi menu GSP (*Good Storage Practices*). (Bagian 3.3.3)
9	• Melanjutkan membuat tampilan isi menu GSP dan luarannya. (Bagian 3.3.3) • Membuat API untuk *fetch* dan *upload* data ke *database* untuk menu GSP.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang (lanjutan)

10	• Menyimpan data hasil *form* ke penyimpanan lokal perangkat. (Bagian 3.3.3) • Menambahkan *listener* agar data hasil *form* dapat diunggah ke *server* jika internet *user* aktif. (Bagian 3.3.3) • Membuat tampilan luar dan isi menu GSP responsif. (Bagian 3.3.3) • Membetulkan *warning* yang ada dan masalah yang ditemukan di menu GSP.
11	• Testing menu GSP yang sudah dibuat menggunakan perangkat Android dan iOS. • Membuat menu luaran dari menu GSP yang telah dibuat. (Bagian 3.3.3) • Mengeksport aplikasi OTLink agar *user* dapat tes hasil menu GSP. • Membuat API, *store function*, dan laporan untuk membantu proyek SFA (*Sales Force Automation*) dalam mengecek jika data foto di dalam *database* benar tersimpan di *server*.
12	• Membuat menu ABBA *Monitoring* di CMS OTLink untuk mengunggah *file excel* hasil bintang *user* ke *database*. • Menambahkan ABBA *webview Target and Rewards* bintang yang menggunakan data *monitoring* hasil input di CMS OTLink. • Menambahkan notifikasi di menu GSP jika terdapat data yang belum terunggah ke *server*.
13	• Menambahkan fitur di CMS OTLink menu ABBA Inbox agar *user* dapat menjadwalkan pesan untuk aktif pada tanggal tertentu. • Menambahkan menu dan tambahan untuk GSP di dalam proyek CMS OT (ASP.NET). • Membuat laporan survei kompetitor untuk SFA menggunakan aplikasi FineReport.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang (lanjutan)

14	• Belajar untuk membuat program .bat untuk menjalankan FTP (*File Transfer Protocol*) dan diautomasi menggunakan Windows task scheduler. • Menyelesaikan tampilan dan fungsi *webview* ABBA Hadiah dan Histori Bintang. (Bagian 3.3.2)
15	• Membetulkan masalah pada website CMS OTLink dan *webview* ABBA berdasarkan yang dilaporkan *user*. • Membuat tampilan dasar dan fungsi grafik data menggunakan Google Charts untuk laman proyek CDP (*Customer Data Platform*).
16	• Membetulkan masalah pada website CMS OTLink dan *webview* ABBA berdasarkan yang dilaporkan *user*.

3.3 Uraian Pelaksanaan Magang

Dalam pelaksanaan kerja magang ini, dibuatnya *website* CMS OTLink dan beberapa proyek yang lebih kecil lainnya. Pada bagian ini, detail pekerjaan proyek tersebut dan tugas yang substansial dijelaskan.

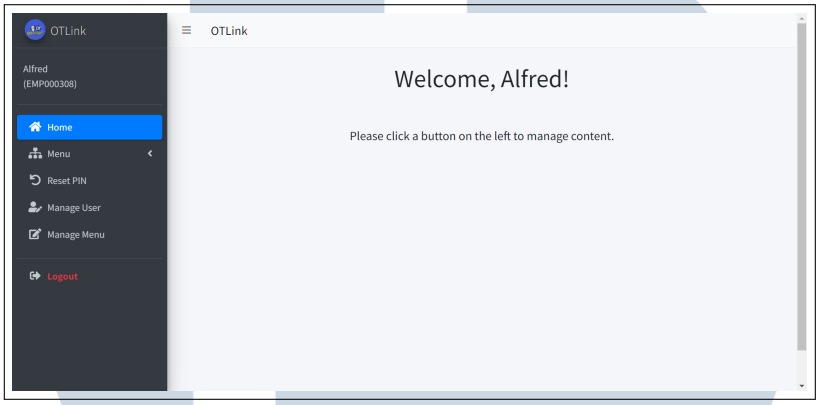
3.3.1 CMS OTLink

CMS OTLink merupakan proyek pembuatan *website* untuk mengelola konten dan beberapa data di aplikasi OTLink maupun ABBA. Proyek ini dibuat menggunakan *framework* Laravel dan *database* yang digunakan adalah SQL Server. *Website* ini memiliki *sidebar* yang dapat berubah secara dinamis berdasarkan *database* dan penggunaan beberapa *library* lainnya yang dijelaskan di bagian berikut ini.

A. Tampilan *Website*

Terdapat tiga komponen UI (*User Interface*) dasar yang ada di *website* CMS OTLink. Pertama, *navbar* putih yang berfungsi untuk men-*toggle sidebar* dan tombol judul yang jika diklik akan membawa *user* ke laman *home*. Kedua,

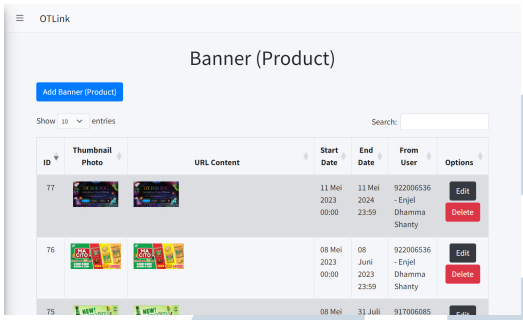
sidebar di kiri laman yang berfungsi sebagai metode untuk *user* dapat mengunjungi menu-menu CMS yang telah diberikan akses. Ketiga, isi konten menu yang berada di bawah *navbar*. Tampilan dasar *website* CMS OTLink yang dibahas seperti di Gambar 3.1.



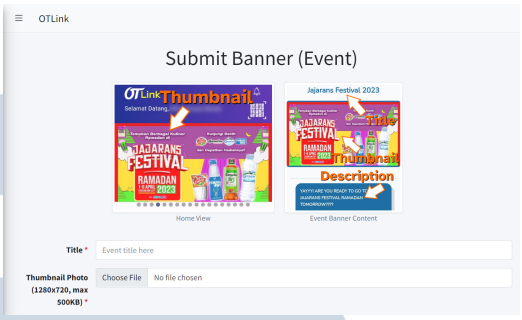
Gambar 3.1. Tampilan Dasar CMS OTLink

Struktur dasar UI, Animasi gerak *sidebar*, dan *styling* UI menggunakan *template* AdminLTE dan Bootstrap 5. Selain itu, CMS OTLink menggunakan *library* yang terpisah untuk UI tertentu seperti tabel data (Bagian C.1), *icon* (Bagian C.2), *rich text area* (Bagian C.3), dan *sortable list* (Bagian C.4).

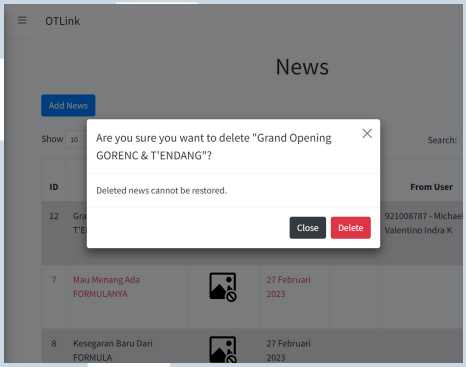
Pada umumnya, isi tampilan depan menu CMS adalah tabel data yang berisi data-data bersangkutan. Di atas tabel tersebut, terdapat tombol untuk menambahkan data baru. Di beberapa menu, tabel data memiliki tombol tambahan untuk edit atau hapus salah satu baris datanya seperti di Gambar 3.2a. Jika di dalam menu tersebut dapat menambahkan atau mengedit konten baru, maka tampilan laman tersebut adalah *form* sesuai data konten yang ingin ditambah atau diubah seperti di Gambar 3.2b. Bila *user* mencoba untuk menghapus baris tabel data, maka akan ada tampilan *pop up* seperti di Gambar 3.2c.



(a) Tampilan Tabel Data CMS OTLink



(b) Tampilan *Form* CMS OTLink

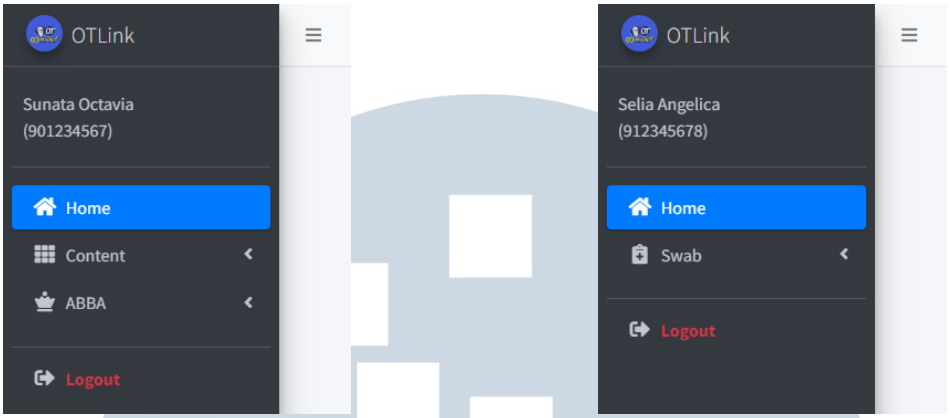


(c) Tampilan modal CMS OTLink

Gambar 3.2. Tampilan Dasar Isi Menu CMS OTLink

B. *Sidebar* Dinamis

Dalam *website* CMS OTLink yang dibuat, terdapat tampilan *sidebar* di kiri layar. *Sidebar* tersebut terdapat menu-menu CMS yang digunakan untuk mengatur konten atau *user* CMS itu sendiri. Setiap *user* memiliki hak akses masing-masing, sehingga satu *user* dapat mengakses menu Konten atau ABBA, sedangkan *user* lain hanya dapat mengakses menu Swab seperti di Gambar 3.3. Daftar menu tersebut dan hak akses setiap *user* terdapat di *database*, sehingga *sidebar* tersebut harus dikonstruksi secara dinamis agar hanya menu yang *user* dapat akses tertampil.



(a) (b)

Gambar 3.3. Komparasi *Sidebar* Berdasarkan Hak Akses *User*

Menu yang dikonstruksi dibuat sebagai *tree* dengan kedalaman satu. Setiap menu terdapat data *parent*-nya. Jika menu tersebut berupa *parent*, maka ID *parent*-nya dipasang nol. Sedangkan jika menu *child*, ID *parent* akan di set dengan ID menu *parent*-nya. Tabel menu di *database* dapat dilihat di Gambar 3.4. Dalam mengkonstruksi *sidebar*, digunakan API yang keluarannya berupa *string* HTML rekonstruksi *sidebar*. Dari *string* HTML ini, dapat menggunakan JavaScript untuk ditampilkan di laman HTML seperti di Kode 3.1.

	MenuID	MenuName	MenuOrder	ParentID	MenuIcon	AllowChild	LockOrder
1	1	Home	1	0	fas fa-home	0	0
2	2	Content	2	0	fas fa-th	1	0
3	3	Banner	3	2	far fa-images	0	0
4	4	News	4	2	far fa-newspaper	0	0
5	5	Event	5	2	far fa-calendar	0	0
6	6	Swab	6	0	fas fa-notes-medical	1	0
7	7	Upload Jadwal Swab	7	6	fas fa-calendar-alt	0	0
8	8	Swab Pantry	8	6	fas fa-calendar-alt	0	0
9	9	Manage Menu	100	0	fas fa-edit	0	1
10	10	Reset PIN	50	0	fas fa-undo	0	1
11	11	ABBA	9	0	fas fa-chess-queen	1	0

Gambar 3.4. Tabel Menu Di *Database*

```
$.ajax({
    url:"{{ route('get_sidebar') }}",
    type: 'GET',
    dataType  : "json",
    data: {
        name: 'output'
    },
    success: function(data){
        // Make the sidebar
        let html = $.parseHTML(data.output);
        $("#sidebar_menu").prepend(html);
    },
    error: function (xhr, b, c) {
        console.log("xhr=" + xhr + " b=" + b + " c=" + c);
    }
});
```

Kode 3.1: Pengambilan Data Dari API *Sidebar* dan Menampilkannya di *Element Sidebar*

C. Penggunaan *library* eksternal

Dalam membuat *website* CMS ini, terdapat beberapa masalah rumit yang harus diselesaikan. Agar tidak menghabiskan waktu dan tenaga dengan membuat semuanya dari awal, maka dari itu digunakan beberapa *library* agar pembuatan *website* ini dapat berjalan dengan cepat.

C.1 Tabel Data dengan jQuery DataTables

Mayoritas tampilan depan menu CMS OTLink adalah tabel data yang berisikan data yang bersangkutan. Agar pembuatan tabel data lebih mudah dan cepat, maka *library* jQuery DataTables dipakai. *Library* ini menyediakan tampilan UI tabel, penyortiran data, pencarian data, dan juga AJAX data agar pemrosesan data dapat dilakukan secara asinkron. Dalam pemasukkan data ke tabel, digunakan kode Laravel dengan cara mengiterasikan hasil *query* ke dalam tabel data seperti di Kode 3.2.

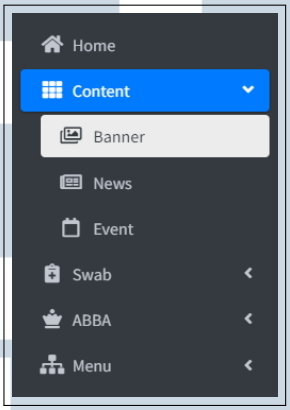
```
<div class="px-5 table-responsive">
  <table class="table table-bordered table-striped table-hover" id
    ="table">
    <thead>
      <tr>
        <th class="text-center">ID</th>
        <th class="text-center">Title</th>
        <th class="text-center">Message</th>
      </tr>
    </thead>
    <tbody>
      @foreach ($tableData as $data)
        <tr>
          <td>{{ $data->id }}</td>
          <td>{{ $data->judul }}</td>
          <td>{{ $data->isi }}</td>
        </tr>
      @endforeach
    </tbody>
  </table>
</div>

<script>
  $(document).ready(function() {
    $('#table').DataTable({
      "order": [[ 0, "desc" ]],
      pagingType: "numbers",
      columns: [
        { data: 'id', name: 'id', class: 'text-center' },
        { data: 'judul', name: 'judul' },
        { data: 'isi', name: 'isi' },
      ]
    });
  });
</script>
```

Kode 3.2: Contoh Implementasi jQuery DataTables

C.2 Font Awesome dan *Icon Preview*

Di setiap menu di CMS OTLink, terdapat gambar *icon* yang tampak di *sidebar website* seperti di Gambar 3.5. Gambar *icon* ini tidak dibuat dalam bentuk format gambar, melainkan menggunakan *library* Font Awesome 5. *Library* ini

menyediakan berbagai *icon* untuk dipakai dan implementasi *icon* tersebut seperti di Kode 3.3. Dipakainya *library* ini dikarenakan untuk mengurangi pemakaian gambar dan juga memudahkan saat menambah *icon* menu di *website* CMS. Dalam *database*, hanya diperlukan teks nama *icon* tersebut seperti di Gambar 3.4.



Gambar 3.5. Tampilan *Sidebar Website* CMS

```
<!-- Shows home icon -->
<i class="nav-icon fas fa-home"></i>
```

Kode 3.3: Contoh Implementasi Font Awesome 5

Saat pembuatan menu baru di CMS, *user* diberikan input teks untuk *icon* yang diinginkan. Di bawah input tersebut, terdapat *preview icon* yang berdasarkan teks input *user*. Saat *user* mengganti input teks, maka gambar *preview* juga akan berubah seperti di Gambar 3.6. Agar *icon preview* dapat berubah saat input diganti, maka dibuatlah seperti Kode 3.4.

(a) (b)

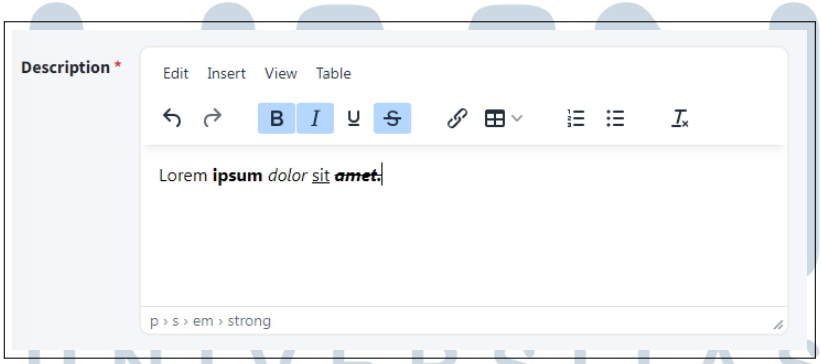
Gambar 3.6. Tampilan *Icon Preview*

```
<p class="lead">
  <i class="fa-3x picker-target far fa-circle"></i>
</p>

<script>
  function update_icon(obj) {
    if (obj.value === '') $('.lead .picker-target').get(0).
    className = 'picker-target fa-3x far fa-circle';
    else $('.lead .picker-target').get(0).className = 'picker-
    target fa-3x ' + obj.value;
  }
</script>
```

Kode 3.4: Implementasi *Icon Preview*

C.3 *Rich Text Area* dengan TinyMCE

Di beberapa kasus dalam memasukkan konten di *website* CMS, diperlukan teks input HTML yang dapat diberikan dekorasi teks seperti *bold*, *italic*, *hyperlink*, hingga *list*. Menggunakan *text area* HTML pada umumnya tidak memberikan kemudahan dalam pembuatan dekorasi teks tersebut. Maka dari itu, agar membantu *user* dalam pembuatan konten teks, maka digunakan *library* TinyMCE agar *user* dapat dengan mudah menambahkan dekorasi teks seperti di Gambar 3.7.



Gambar 3.7. Tampilan *Text Area* TinyMCE

Dalam CMS ini, TinyMCE di-*host* sendiri dari *server* karena semua fitur yang dibutuhkan sudah tersedia. Cara kerjanya, yaitu dengan membuat kode JavaScript untuk menginisialisasi TinyMCE dengan *pointer* ke salah satu *text area* yang ada. Setup TinyMCE di dalam HTML dapat dilihat di Kode 3.5.

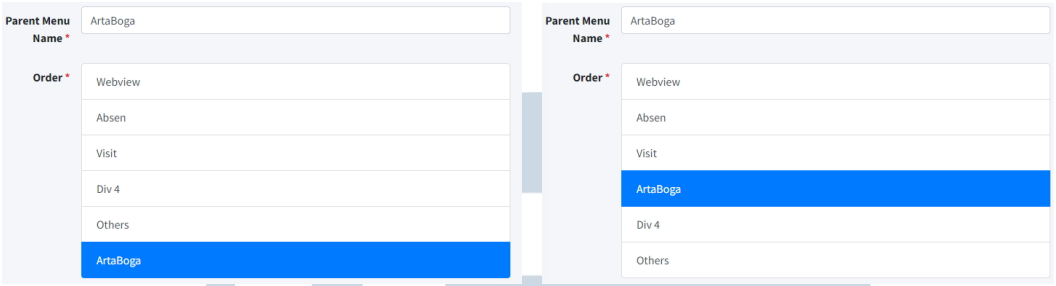
```
<textarea name="desc" id="desc" rows="6" required></textarea>

<script>
  tinymce.init({
    selector: 'textarea#desc',
    menubar: 'edit insert view table',
    plugins: 'lists table link',
    toolbar: 'undo redo | bold italic underline strikethrough |
   link table | numlist bullist | removeformat',
    setup: function (editor) {
        editor.on('change', function () {
            tinymce.triggerSave();
        });
    },
    promotion: false,
    branding: false,
  });
</script>
```

Kode 3.5: Contoh Implementasi TinyMCE

C.4 *Sortable list* dengan jQuery UI

Salah satu menu di CMS, yaitu pembuatan menu *parent* baru di aplikasi OTLink. Setiap *parent* menu memiliki urutan letak, maka dari itu diperlukan penginputan urutan letak tampilan. Semisalnya ada *parent* menu baru yang diinput, menu tersebut dapat diletakkan di paling bawah, paling atas, atau di antara menu-menu lainnya. Penginputan tradisional seperti *dropdown* atau *text box* kurang intuitif untuk dipakai karena tidak ada contoh tampilan letak menu-menu setelah menu baru itu dimasukkan. Maka dari itu, CMS ini memakai *sortable list* yang disediakan oleh jQuery UI.

Tampilan *sortable list* yang dipakai di CMS OTLink terlihat seperti di Gambar 3.8a. Secara umum, menu baru akan berada di bawah dan diberi warna agar mengindikasikan bahwa menu tersebut adalah menu baru. Kemudian *list* menu-menu lainnya diambil dari *database*. Setiap isi *list* dapat di-*drag and drop* sesuai urutan yang diinginkan *user* seperti di Gambar 3.8b. Agar *sortable list* ini dapat dijadikan sebagai input, maka setiap *list* berupa input yang dapat dilihat di Kode 3.6.



(a) (b)

Gambar 3.8. Tampilan Input Berupa *Sortable List*

```
<div class="list-group" id="sortable">
  @foreach ($query as $item)
  <input type="text" class="list-group-item list-group-item-action
    " id="menu{{ $item->id }}" name="menu_check[]" value="{{ $item
    ->categories_grup }}" readonly>
  @endforeach
</div>

<script>
  $( function() {
    $( "#sortable" ).sortable({ cancel: null });
    $( "#sortable" ).disableSelection();
  } );
</script>
```

Kode 3.6: Contoh Implementasi *Sortable List*

C.5 Pemrosesan CSV dan Excel dengan Laravel Excel

Di menu CMS ini diperlukan *input* berupa CSV atau Excel seperti di Konten OTLink agar hanya *user* tertentu yang dapat melihat konten dan menu *upload* jadwal swab dan hasil swab Pantry. Dalam Laravel, terdapat *library* luar untuk mengurus data CSV dan Excel yang bernama Laravel Excel. Dalam pemakaiannya, *library* ini digunakan untuk memasukkan data CSV ke *database* model secara langsung seperti di Kode 3.7 atau mengubah data tersebut ke dalam *array* terlebih dahulu seperti di Kode 3.8.

```
use Maatwebsite\Excel\Facades\Excel;
use Illuminate\Http\Request;

public function add(Request $request){
    $csv = $request->csv_input;
    Excel::import(new UploadJadwalSwabImport(), $csv);
}
```

Kode 3.7: Contoh Implementasi Laravel Excel Untuk Memasukkan Data CSV Ke Dalam *Database* Melalui Model

```
use Maatwebsite\Excel\Facades\Excel;
use Illuminate\Http\Request;

public function add(Request $request){
    $file = $request->excel_input;
    $sheets = Excel::toArray([], $file);
}
```

Kode 3.8: Contoh Implementasi Laravel Excel Untuk Mengubah *File* Excel Menjadi *Array*

3.3.2 ABBA Bintang *Webview*

Pada aplikasi ABBA, terdapat fitur baru, yaitu ABBA Bintang. ABBA Bintang merupakan sistem poin untuk *user* aplikasi ABBA di mana dari pembelian dan bonus, mereka akan mendapatkan bintang. Bintang tersebut dapat digunakan untuk ditukarkan dengan hadiah-hadiah yang ditentukan admin. Tugas yang diberikan untuk ABBA bintang, yaitu

1. Membuat *webview* ABBA untuk menukar bintang yang dimiliki *user*.
2. Membuat *webview* ABBA untuk melihat histori transaksi bintang.
3. Membuat *webview* ABBA untuk *target and rewards* khusus untuk bintang.
4. Membuat menu CMS OTLink baru agar admin dapat mengunggah data bintang *user* dalam bentuk *excel*.
5. Membuat menu CMS OTLink agar admin dapat menambah daftar hadiah yang dapat ditukar.

Untuk penambahan data bintang *user* dan hadiah, dibuatlah CMS di dalam proyek CMS OTLink yang telah dibuat sebelumnya dengan menambahkan

menu baru di bawah menu *parent* ABBA. *Webview* ABBA dibuat menggunakan *framework* CodeIgniter karena dibuat di dalam proyek *webview* ABBA yang dasar *framework*-nya adalah CodeIgniter 3. Desain tampilan penukaran dan histori bintang telah dibuat sebelumnya dalam bentuk gambar dan diberikan dari supervisi untuk dibuatkan ke dalam laman *webview*.

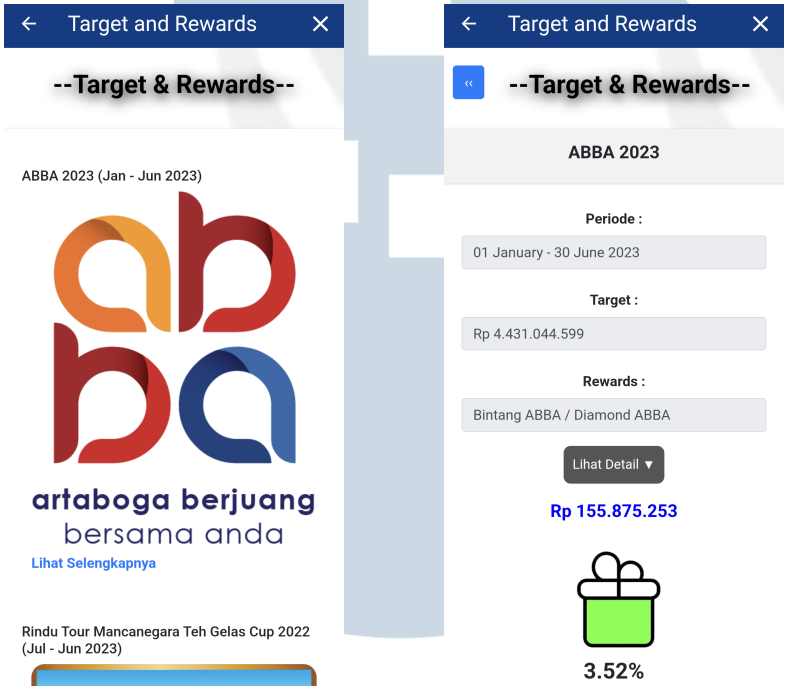
Dalam mengakses tampilan *webview* yang dibuat, *user* dapat menggunakan aplikasi ABBA dan menu bintang terdapat di halaman awal seperti di Gambar 3.9a. Di halaman awal ini juga terdapat teks jumlah bintang yang dimiliki *user*. Jika *user* menekan tombol Hadiah Bintang ABBA, maka *user* akan dipindahkan ke *webview* penukaran bintang seperti di Gambar 3.9b. Di laman ini, terdapat pilihan barang-barang yang disiapkan admin untuk *user* tukar dengan bintang yang ia miliki. Jika barang tersebut ditekan, maka terdapat tampilan yang muncul untuk mengonfirmasi pilihan *user* sebelum barang tersebut ditukar. Histori penukaran bintang *user* dan pendapatan bintang dapat dilihat di laman *webview* histori seperti di Gambar 3.9c.

(a) Tampilan Depan Aplikasi ABBA (b) Tampilan Laman Tukar Bintang (c) Tampilan Laman Histori Bintang

Gambar 3.9. Tampilan Hadiah Bintang ABBA

Dalam laman awal ABBA, terdapat menu *Target and Rewards*. Isi dalam menu ini adalah kumpulan target dan hadiah yang *user* dapat capai berdasarkan ketentuan masing-masing target seperti di Gambar 3.10a. Target yang baru khusus

untuk bintang adalah ABBA 2023. Bentuk tampilan menu ini dibuat berdasarkan tampilan laman target sebelumnya. Dalam target ini terdapat informasi mengenai durasi target, hadiah target, target yang harus dicapai, dan detail mengenai *progress user*. Tampilan *target and rewards* yang dibuat terdapat di Gambar 3.10b.



(a) Tampilan Depan *Target and Rewards*

(b) Tampilan *Target and Rewards* ABBA 2023

Gambar 3.10. Tampilan Menu ABBA *Target and Rewards*

3.3.3 Menu GSP OTLink

Dalam aplikasi OTLink, terdapat menu GSP (Good Storage Practices) yang digunakan untuk mendata dan memonitor setiap kondisi gudang yang dipegang *user*. Sebelumnya, isi menu ini merupakan *webview*. Hanya saja, ada *user* yang komplain bahwa data yang telah di-*submit* tidak terunggah ke *database* karena koneksi *user* yang tidak stabil. Maka dari itu, tugas ini adalah membuat menu GSP di dalam aplikasi agar hasil jawaban *user* dapat tersimpan dan diunggah saat internet tersedia.

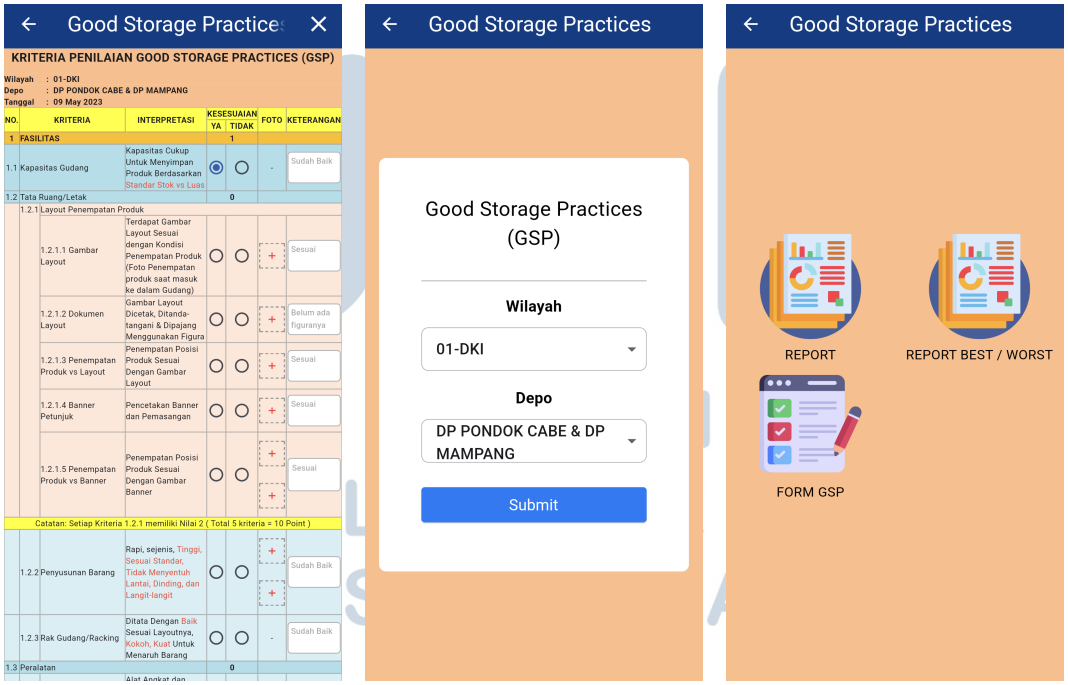
Aplikasi OTLink menggunakan *framework* Flutter, maka tugas ini dilakukan menggunakan Flutter. Dalam *fetching* dan *upload* data, diperlukan API yang dibuat di dalam proyek CMS OTLink. Maka pembuatan API untuk menu ini menggunakan Laravel. Tampilan menu GSP dibuat sesuai tampilan *webview* GSP sebelumnya dan

juga arahan dari pegawai lain.

Menu GSP memiliki *form* yang isinya berupa tabel dengan setiap barisnya adalah judul bagian, penjelasan bagian, *radio button* untuk mengatakan setuju atau tidak setuju, teks input, dan di beberapa baris terdapat input gambar seperti dalam Gambar 3.11a. Saat *form* di-*submit*, setiap baris akan disimpan di SQLite dan saat internet *user* tersedia, maka data di SQLite tersebut akan diunggah ke *database* melalui API.

Tampilan depan dari *form* GSP sebelumnya, terdapat pilihan yang terdiri dari dua *dropdown* untuk wilayah dan depo tujuan seperti di Gambar 3.11b. Saat *user* masuk ke dalam tampilan ini, akan dilakukan *fetch* data wilayah apa saja yang *user* dapat pilih. Saat *user* memilih salah satu wilayah, maka akan di-*fetch* depo apa saja yang terdapat di wilayah tersebut untuk *user* isi.

Saat *user* masuk ke dalam menu GSP dari laman *home*, terdapat tampilan menu *report*, *report best/worst*, dan *form* GSP seperti di Gambar 3.11c. Di awal, peran *user* dicek melalui API. Hasil peran ini menentukan jika *user* dapat melihat menu *report best/worst* atau tidak. Menu *report* dan *report best/worst* tetap menggunakan *webview* karena tidak ada masalah, sehingga tidak perlu dibuat ulang. Sedangkan *form* GSP akan mengoper *user* ke dalam tampilan luar *form* GSP yang telah disebutkan sebelumnya.



(a) *Form* GSP (b) Awal *Form* GSP (c) Awal menu GSP

Gambar 3.11. Tampilan Isi Menu GSP

Dalam *form* GSP, input yang paling kompleks adalah input foto karena foto *user* perlu disimpan dan diproses terlebih dahulu sebelum dikirim. Berikut detail mengenai pengunggahan input foto dari awal input *form* dijelaskan.

A. Simpan Input Gambar

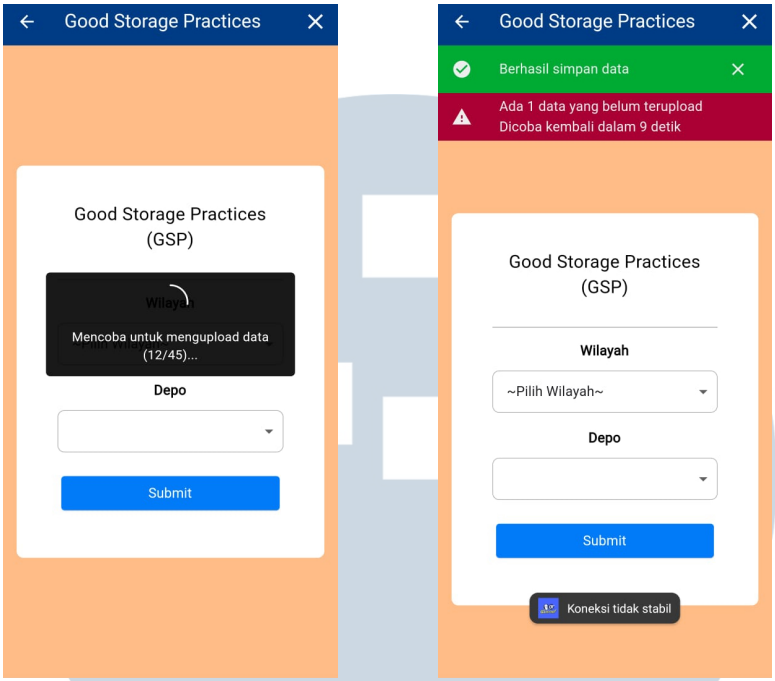
Tabel *form* GSP memiliki beberapa kolom input. Salah satu input itu adalah foto. Pada kolom tertentu, terdapat tombol dengan ikon tambah yang jika *user* menekannya akan diberikan *prompt* untuk mengambil foto dari galeri atau mengambil foto dari kamera. Setelah *user* mengambil gambar, gambar tersebut akan ditampilkan di *form* tersebut dan tersimpan di penyimpanan sementara perangkat.

Setelah *user* selesai mengisi dan men-*submit form*, maka data foto akan disimpan di folder *download* agar tersimpan walaupun *user* telah keluar dari aplikasi. Data *path* letak foto tersebut disimpan di SQLite bersamaan dengan data baris lainnya untuk digunakan saat data tersebut ingin diunggah.

B. Unggah Input Gambar

Koneksi *user* dicek saat tombol *submit* ditekan, tetapi jika akses internet *user* tidak stabil maka data SQLite akan ditahan seperti di Gambar 3.12. Pada laman awal *form* GSP, internet *user* akan selalu dicek jika tersedia setiap sepuluh detik. Jika tersedia dan data SQLite tidak kosong, maka data tersebut siap untuk diunggah.

Saat data diunggah, data foto diakses menggunakan *path* yang telah disimpan sebelumnya dan kemudian di-*encode* ke dalam *base64* agar data foto dapat diunggah dengan konsisten. Di dalam API yang digunakan untuk menerima data tersebut, data foto di-*decode* dan memasukkan hasil foto tersebut ke dalam penyimpanan *server* dan *database*. Hasil keluaran dari API berupa 0 jika gagal dan 1 jika sukses. Jika sukses, data baris SQLite yang diunggah akan dihapus dan melanjutkan ke baris berikutnya hingga *database* SQLite kosong.



(a) Tampilan Saat Data *Form* Diunggah

(b) Tampilan Saat Data *Form* Gagal Diunggah

Gambar 3.12. Tampilan Pengunggahan Data *Form* GSP

3.4 Kendala yang Ditemukan

Dalam pembuatan *website* CMS, ditemui beberapa kendala yang dihadapi. Kendala yang dimaksud sebagai berikut.

1. Keterbatasan pengetahuan dalam *framework* Laravel karena belum pernah mempelajari tentang *framework* tersebut.
2. Jaringan internet di perangkat komputer kantor terputus setiap jam lima sore secara otomatis karena merupakan aturan dari perusahaan.

3.5 Solusi Atas Kendala yang Ditemukan

Berdasarkan kendala yang telah disebutkan sebelumnya, ditemukan solusi atas setiap kendala tersebut. Solusi tersebut sebagai berikut.

1. Mempelajari *framework* Laravel dengan cara melihat sumber dari internet dan bantuan dari rekan kerja kantor.

2. Menyalakan jaringan internet di perangkat kantor yang digunakan agar dapat meneruskan pengerjaan proyek.