# BAB 3
# PELAKSANAAN KERJA MAGANG

## 3.1 Kedudukan dan Koordinasi

Proyek ini dilakukan di bawah supervisi Bpk. Aang Wiandi yang mengarahkan dan memberikan bimbingan untuk tugas-tugas yang dijalankan. Bimbingan yang diberi mencakup menjawab pertanyaan berkaitan dengan pekerjaan magang, memeriksa hasil kerja yang dilakukan, dan membantu kendala-kendala teknis yang dihadapi saat mengerjakan tugas dari segi perusahaan. Posisi yang diduduki merupakan sebagai seorang *Web Developer* yang berperan atas pengembangan *web email-blast* karyawan. Tugas-tugas yang dijalankan melibatkan proses desain, aplikasi, dan revisi atau pengubahan aplikasi yang dilakukan selama durasi magang.

## 3.2 Tugas yang Dilakukan

Tugas yang dilakukan meliputi pengembangan *web email-blast* yang akan menjadi hasil dari proses magang. Oleh sebab itu, penyelesaian fitur serta perubahan penampilan *web* yang rapi menjadi tujuan pekerjaan ini yang dilaksanakan sebagai seorang *Web Developer*. Berikut adalah tugas-tugas yang dilakukan dalam proses pengembangan *web* ini:

1. Merancang fitur halaman *dashboard* baru dan mengembangkan fitur yang telah tersedia
2. Membuat penampilan yang efisien dan rapi
3. Berkoordinasi dan mengadakan rapat dengan supervisi dalam pembahasan progres pengerjaan *web*
4. Memastikan semua fitur berjalan dengan lancar

## 3.3 Uraian Pelaksanaan Magang

Pelaksanaan kegiatan magang diuraikan seperti pada Tabel 3.1.

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

| Minggu Ke- | Pekerjaan yang Dilakukan |
|---|---|
| 1 | Membuat *framework* penampilan *Queue Management System* |
| 2 | Membuat *mockup* penampilan *Queue Management System* |
| 3 | Melakukan revisi penampilan berdasarkan supervisi |
| 4 | Mendiskusikan *framework* dan fitur-fitur yang akan diimplementasikan |
| 5 | Merancang fitur grafik *bar* |
| 6 | Merancang fitur grafik *bar* |
| 7 | Merancang fitur *calendar* |
| 8 | Melanjutkan implementasi fitur *calendar* |
| 9 | Merancang fitur *to-do list* |
| 10 | Merancang fitur *to-do list* |
| 11 | Melakukan *debugging* pada fitur |
| 12 | Merancang fitur grafik *line* |
| 13 | Merancang fitur grafik *line* |
| 14 | Penambahan fungsi pada fitur *calendar* |
| 15 | Penambahan fungsi pada fitur *to-do list* |
| 16 | Melakukan pengujian akhir dan evaluasi |

Proyek magang ini dilaksanakan menggunakan perangkat keras *laptop* pribadi. *Laptop* pribadi yang digunakan adalah HP OMEN 16 dengan spesifikasi sebagai berikut:

1. CPU: AMD Ryzen 7 7840HS w/ Radeon 780M Graphics
2. RAM: 32GB
3. GPU: NVIDIA GeForce 4070

### 3.3.1 Pengembangan Web

Kegiatan ini mencakup proses serta hasil pengembangan *web* yang dikembangkan. Perubahan dalam tampilan serta penambahan fitur pada halaman *web dashboard* merupakan tugas-tugas yang tercakup dalam kegiatan ini. Tampilan visual dirancang dengan pemikiran untuk memudahkan pengguna dalam

menggunakan *web*. Fitur-fitur yang ditambahkan juga berfungsi untuk melengkapi kebutuhan pengguna dan memberikan informasi yang mendukung tujuan utama *web*, yaitu pengiriman *email-blast*. Alur penggunaan *web* mengacu pada cara pengguna berinteraksi dengan setiap halaman dan fitur *web* sesuai kebutuhan.

Perancangan *web* dilakukan secara kolaboratif dengan seorang rekan kerja sesuai dengan arahan *supervisor*. Perincian desain dan revisi dilakukan setiap minggu antar *web developer*. Pelaporan progres perancangan *web* dilakukan secara *online* melalui *Trello* kepada *supervisor* magang. Pembahasan secara *offline* juga dilakukan beberapa kali di kantor perusahaan terkait tugas baru. Diskusi yang diselenggarakan berisi arahan terhadap fitur yang sedang dibuat serta revisi untuk fitur yang telah selesai. Perubahan desain yang direncanakan juga diterapkan agar *web developer* memiliki gambaran yang jelas mengenai visi yang diinginkan.

Perbedaan *web* sebelum dan sesudah penambahan fitur dapat dilihat pada Tabel 3.2.

Tabel 3.2. Pengembangan *web* yang dilakukan selama pelaksanaan magang

| ***Web* Sebelumnya** | ***Web* Sekarang** |
|---|---|
| Belum memiliki fitur *Bar Chart* | Telah memiliki fitur *Bar Chart* |
| Belum memiliki fitur *Line Chart* | Telah memiliki fitur *Line Chart* |
| Belum memiliki fitur *Calendar* | Telah memiliki fitur *Calendar* |
| Belum memiliki fitur *To-do List* | Telah memiliki fitur *To-do List* |
| *Layout dashboard* masih kosong | *Layout dashboard* terisi beragam fitur dengan rapi dan tidak bertabrakan |

### 3.3.2 Perancangan *Dashboard*

Kegiatan perancangan *dashboard* meliputi pengembangan sebuah desain *dashboard* yang sederhana di mana hanya jumlah *email* yang berhasil dan gagal yang ditampilkan. Kegiatan ini memaparkan perancangan setiap fitur dari desain awal hingga revisi terakhir. Proses perancangan fitur melewati beberapa tahap revisi. Pengerjaan *dashboard* dilakukan pada beberapa aspek yang ditambahkan selama menjalani kegiatan ini.

Bahasa pemrograman yang digunakan berupa *PHP* dengan *Laravel Framework* yang terintegrasi dengan *Blade*, *JavaScript*, *CSS*, *SQL*, *JSON*,

dan *YAML*. Perusahaan telah memberikan *project file* sebelumnya yang sudah terintegrasi menjadi *Laravel project* dengan *file* `.env` yang disediakan.

Pengembangan *user interface* pada sistem *web* mengacu pada prinsip *Eight Golden Rules of Interface Design* yang dikemukakan oleh Ben Shneiderman[6]. Prinsip-prinsip ini digunakan sebagai landasan dalam perangkaian tata letak serta fungsi fitur-fitur yang akan ditambahkan pada halaman *dashboard* dalam harapan untuk hasil antarmuka yang intuitif, efisien, dan nyaman bagi pengguna. Adapun prinsip-prinsip tersebut dijelaskan sebagai berikut:

1. **Konsistensi dalam Desain** (*Strive for Consistency*)
   Elemen-elemen antarmuka seperti tombol, ikon, warna, dan menu harus didesain secara konsisten di seluruh bagian sistem. Konsistensi ini membantu pengguna mengenali pola interaksi dan mengurangi kebutuhan untuk mempelajari ulang cara penggunaan setiap bagian.

2. **Dukungan untuk Pengguna Berpengalaman** (*Enable Frequent Users to Use Shortcuts*)
   Sistem sebaiknya menyediakan pintasan atau cara cepat bagi pengguna yang sudah terbiasa, seperti penggunaan *shortcut keyboard* atau *menu kontekstual*, sehingga mereka dapat menyelesaikan tugas dengan lebih efisien.

3. **Umpan Balik yang Informatif** (*Offer Informative Feedback*)
   Setiap aksi yang dilakukan oleh pengguna harus diberikan respons yang jelas oleh sistem, misalnya dengan menampilkan notifikasi sukses setelah formulir dikirim, atau indikator pemrosesan saat sistem sedang bekerja.

4. **Dialog yang Memberikan Kejelasan** (*Design Dialogs to Yield Closure*)
   Setiap proses atau rangkaian aksi pengguna harus memiliki struktur yang jelas: awal, proses, dan akhir. Hal ini memberikan rasa selesai pada setiap tugas yang dilakukan pengguna, contohnya dalam proses pengiriman *email* yang terdiri dari tahap pembuatan, pratinjau, dan konfirmasi pengiriman.

5. **Penanganan Kesalahan yang Sederhana** (*Offer Simple Error Handling*)
   Sistem harus mampu mencegah terjadinya kesalahan, namun jika kesalahan terjadi, sistem harus menampilkan pesan yang mudah dipahami dan memberikan solusi yang dapat dilakukan pengguna untuk memperbaikinya.

6. **Kemudahan untuk Membatalkan Aksi** (*Permit Easy Reversal of Actions*)
   Pengguna harus diberi kemampuan untuk membatalkan atau mengulang suatu

tindakan. Fitur ini penting untuk mengurangi kecemasan pengguna terhadap kesalahan dan mendorong eksplorasi fitur sistem.

7. **Pengendalian oleh Pengguna** (*Support Internal Locus of Control*)
   Sistem sebaiknya memberikan kendali penuh kepada pengguna. Antarmuka yang baik tidak membuat pengguna merasa dikendalikan oleh sistem, melainkan sebaliknya, pengguna merasa bebas dan nyaman dalam berinteraksi.

8. **Mengurangi Beban Memori Jangka Pendek** (*Reduce Short-Term Memory Load*)
   Informasi penting harus ditampilkan secara langsung di layar agar pengguna tidak perlu mengingat terlalu banyak hal. Misalnya, menampilkan label atau petunjuk penggunaan langsung di dekat elemen yang relevan, bukan mengharuskan pengguna mengingat instruksi dari halaman sebelumnya.

Jenis font yang digunakan dalam pengembangan ini adalah *Aceh SoftMedium* sebagai jenis huruf utama. Font ini memiliki tampilan yang sederhana dan mudah dibaca, sehingga memudahkan pengguna dalam memahami isi laporan secara keseluruhan. *Aceh SoftMedium* memberikan kesan visual yang ringan namun tetap profesional. Penggunaan font ini dimasukkan ke dalam semua fitur untuk membangun prinsip *consistency* dan *familiarity*.

Warna pada pengembangan *web* berupa warna dasar yakni hitam dan putih dengan beberapa warna *highlight* untuk menarik fokus pengguna pada tombol atau elemen. Warna *highlight* berupa warna hijau dengan kode `#16c66b` serta warna merah dengan kode `#ff112b`. Warna mayoritas pada *dashboard* merupakan warna abu dan putih untuk mendukung penampilan yang jelas dan memberikan *whitespace* yang cukup bagi pengguna. Pemilihan warna *highlight* dilakukan untuk memberikan warna yang cukup terang untuk memberikan efek penekanan pada data tapi tidak terlalu cerah agar tetap nyaman dilihat. Kedua warna juga memberikan informasi akan hasil data yang diinginkan dan dihindari. Beberapa tombol juga menggunakan warna biru untuk memberikan kesan persiapan dalam penggunaan.

Alur perancangan tiap fitur diawali dengan analisis kebutuhan. Bagian ini menjabarkan latar belakang perancangan serta tujuan fitur. Aspek-aspek yang dibutuhkan dari fitur yang dirancang juga berada di bagian ini.
Alur berlanjut pada tahap *Perancangan Logika* yang memperlihatkan proses perakitan cara kerja fitur dan interaksi dengan sisi *backend*. Logika yang dihasilkan

dari tahap ini menjadi landasan dari kinerja fitur.

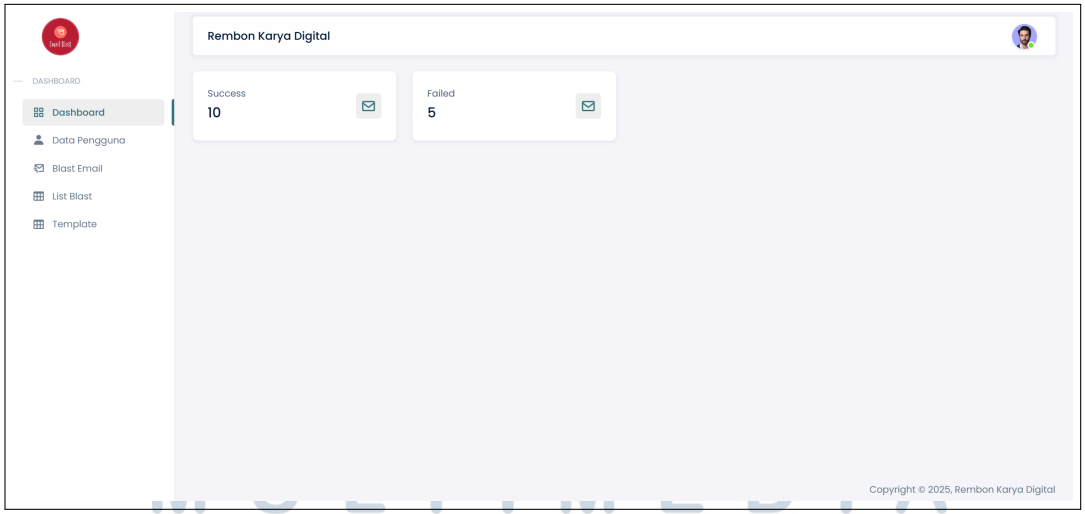
Tahapan selanjutnya berupa implementasi dari logika yang telah dibuat beserta pendekatan dalam merakit penampilan fitur. Tahap ini memaparkan proses realisasi fitur yang mengintegrasikan sisi *frontend* dan *backend*. Perubahan pada gambaran atau arah pengerjaan fitur juga disebutkan pada tahap ini.

Alur perancangan berakhir pada bagian pengujian di mana fitur yang telah berhasil dibuat diuji agar berjalan sesuai dengan fungsi yang diharapkan.

## A Bar Chart

### A.1 Analisis Kebutuhan

Permasalahan utama dari *dashboard* sebelumnya mengarah pada kekosongan halaman *dashboard* tersebut dengan penampilan informasi yang minim dan hanya berupa jumlah *email* yang gagal dan berhasil terkirim. Pengguna *web* tidak menerima informasi lain dan tidak dapat membandingkan data dengan rinci. Perlu adanya suatu gambaran mengenai hasil kerja *email-blast* yang telah dilakukan sebagai bentuk konfirmasi pengiriman yang lengkap dan terstruktur. Tampilan *dashboard* sebelum adanya pengembangan *web* dapat dilihat pada Gambar 3.1.




Gambar 3.1. Tampilan *Dashboard* sebelum adanya pengembangan *web*

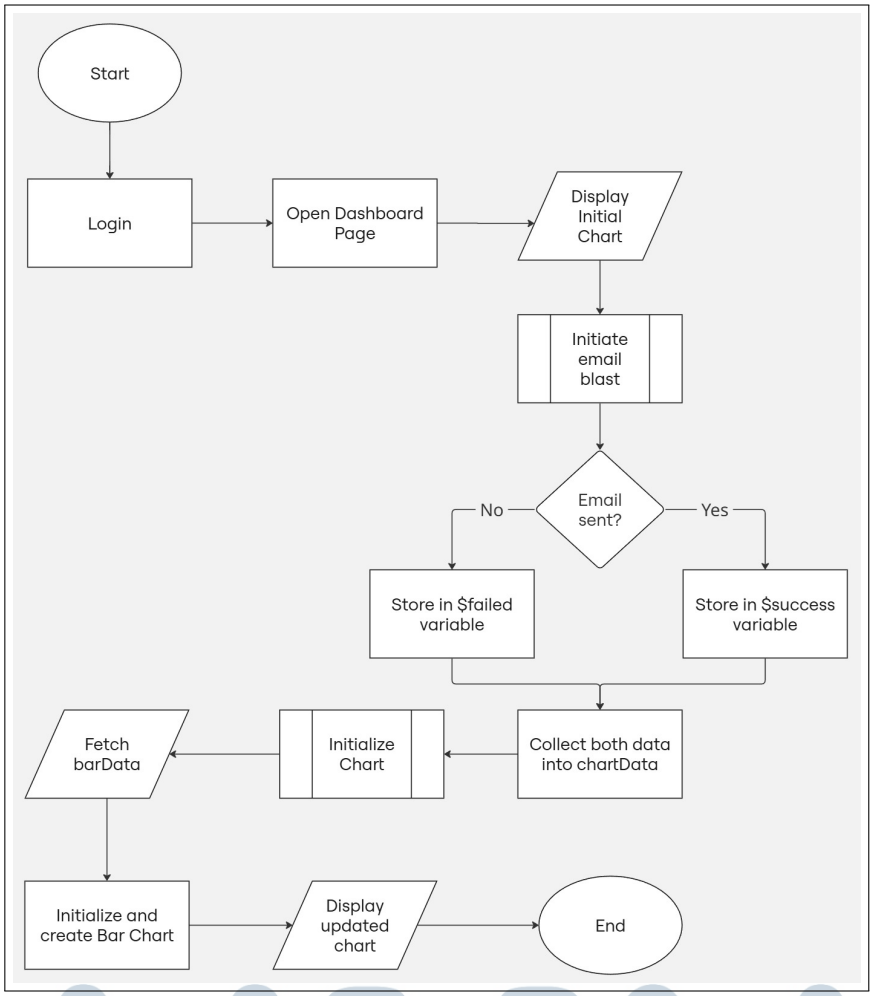
Usulan pembuatan *chart* diberikan oleh *supervisor* sebagai bentuk informasi visual agar pengguna dapat mengetahui hasil *email-blast* secara nyata. Tipe *chart* yang ditentukan dari hasil diskusi dengan *supervisor* merupakan *bar chart*. Pemilihan tipe grafik ini dikarenakan kesederhanaan data yaitu jumlah *email* yang

gagal dan berhasil terkirim. *Bar chart* memberikan pemaparan data yang linear dan menjadi tipe representasi visual yang cocok dalam konteks ini. *Styling* grafik berencana untuk memiliki beberapa modifikasi dimana warna data dibedakan untuk membedakan hasil yang diinginkan dan dihindari dari kedua variabel data tersebut.

*Bar Chart* merupakan fitur baru pertama yang dibuat pada *dashboard* dengan tujuan memberikan informasi kepada pengguna *web*. Jika *email* yang dikirim gagal, maka data tersebut akan terlihat di *bar chart* dan pengguna dapat mengirim ulang *email*. Data pada grafik akan berubah secara dinamis sesuai dengan perubahan variabel *email* yang berhasil dan gagal terkirim. Dengan ini, pengguna mempunyai referensi visual data hasil pengiriman *email-blast*.

## A.2 Perancangan Logika

Bagian ini menjelaskan proses cara kerja *bar chart* yang diawali dengan pengumpulan data dari fungsi *web* utama. Data yang diterima kemudian akan diolah menjadi data yang digunakan dalam *bar chart* secara spesifik dengan *label* dan *value* tersendiri. Data kemudian dibaca oleh *chart* dan diperlihatkan secara langsung, menyajikan grafik visual untuk hasil data fungsi *email-blast*. Visualisasi alur perancangan *bar chart* dapat dilihat pada Gambar 3.2.




Gambar 3.2. *Flowchart Bar Chart*

## A.3 Implementasi

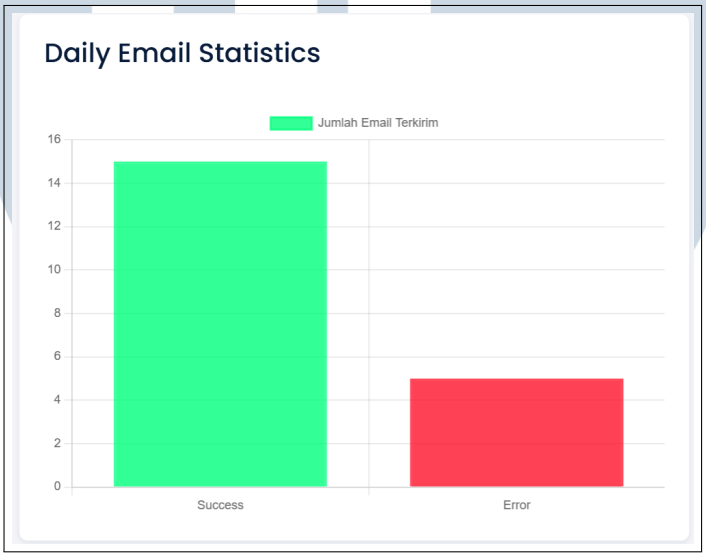
Pembuatan grafik *bar chart* menggunakan *ChartJS*, suatu *library JavaScript open-source* yang digunakan untuk menampilkan grafik interaktif dan responsif di halaman *web* menggunakan elemen dari *HTML5*. Grafik diintegrasikan dengan *backend* melalui `batch::sum` data *email* yang gagal dan berhasil terkirim. Data berikut dilabel menjadi *y value*. *Label email* yang gagal dan berhasil terkirim dinyatakan sebagai *x value*. *Bar Chart* diinisialisasi dengan membuat *canvas* untuk *chart* pada *container*. *Canvas id chart* akan ditangkap oleh *function* `initChart` untuk mengetahui letak *chart* berada sembari membuat kerangka *chart*. Variabel `chartData` akan dipanggil untuk dimasukkan ke dalam *chart* kemudian ditampilkan ketika fungsi *initialize bar chart* dipanggil. Kode implementasi *bar chart* dapat dilihat pada lampiran 3.1.

```
// inisialisasi canvas chart
<div class="chart-container">
    <canvas id="barChart" height="400" style="height: 400px;"></canvas>
</div>

//function initChart universal
function initChart(canvasId, type, data, options) {
                const canvas = document.getElementById(canvasId);
                if (!canvas) return null;

                const existingChart = Chart.getChart(canvas);
                if (existingChart) {
                    existingChart.destroy();
                }

                return new Chart(canvas, {
                    type: type,
                    data: data,
                    options: options
                });
            }

//function memanggil chartData
const barData = {
                    labels: chartData.labels,
                    datasets: [{
                        label: 'Jumlah Email Terkirim',
                        data: chartData.data,
                        backgroundColor: [
                            'rgba(0, 255, 125, 0.8)',
                            'rgba(255, 17, 43, 0.8)',
                        ],
                        borderColor: [
                            'rgba(0, 255, 125, 0.8)',
                            'rgba(255, 17, 43, 0.8)',
                        ],
                        borderWidth: 2
                    }]
                };
```

Kode 3.1: Kode pembuatan *bar chart*

*Error handling* diterapkan pada *bar chart* untuk mencegah *error* ketika

inisialisasi *bar chart*. *Function* `initChart` akan memeriksa apakah *chart* yang akan dirancang telah dibuat atau belum. Jika terdapat *chart* dengan kriteria yang sama, fungsi akan menghapus *chart* tersebut agar tidak bertabrakan dengan *chart* yang akan dirancang. *Styling bar chart* diterapkan dengan menggunakan warna pada setiap *bar*. Warna hijau digunakan untuk *label* ”Berhasil” untuk mengindikasikan *outcome* yang diinginkan, sementara warna merah digunakan untuk *label* ”Gagal” untuk mengindikasikan *outcome* yang tidak diinginkan. Hasil implementasi dapat dilihat pada Gambar 3.3.




Gambar 3.3. *Bar Chart Dashboard*

## B Line Chart

### B.1 Analisis Kebutuhan

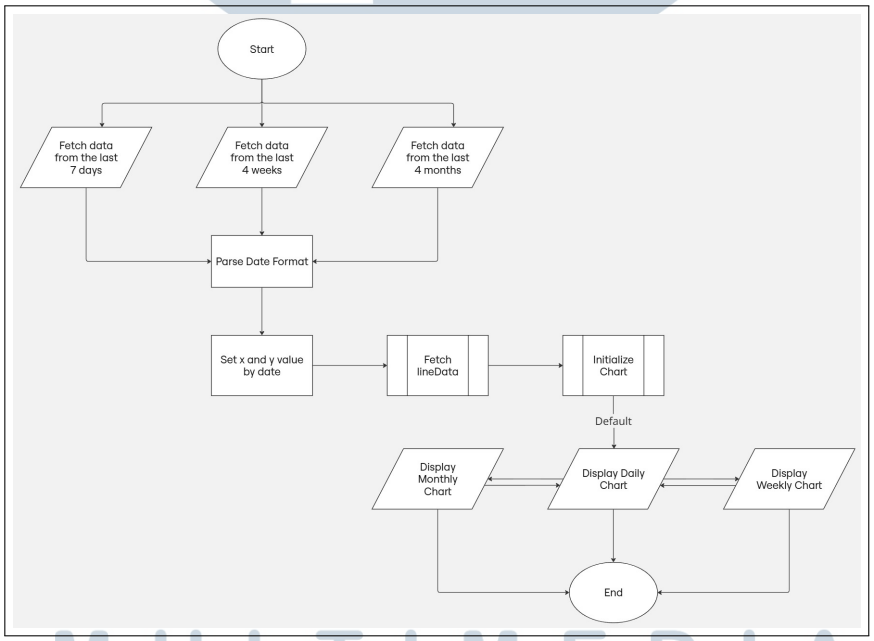
Dengan tampilan *bar chart* pada *dashboard*, tertera gambaran grafik untuk memaparkan data hasil pengiriman *email* secara terus-terang. Pemaparan informasi yang diberikan bersifat sederhana dan mudah untuk dimengerti. Namun, jenis pemaparan *bar chart* kurang sepadan untuk perbandingan serta analisa data dalam periode waktu yang lama. *Dashboard web* tidak memiliki gambaran langsung untuk melakukan evaluasi atau analisis tren. Diperlukan penyajian data yang lebih informatif, terstruktur, dan dinamis sebagai bahan pertimbangan pengguna dalam mengevaluasi performa pengiriman *email-blast*.

Oleh karena permasalahan yang ada, penambahan visualisasi dalam bentuk *line chart* yang memuat data dalam rentang waktu tertentu ditentukan untuk

mengatasi isu yang ada. Berbeda dari *bar chart* yang bersifat statis, *line chart* menampilkan kecenderungan *(tren)* data dari waktu ke waktu, yang sangat berguna dalam menilai performa dan konsistensi pengiriman. Pengguna dapat memilih periode tampilan data berdasarkan harian, mingguan, maupun bulanan melalui opsi *filter* yang disediakan.

## B.2 Perancangan Logika

Bagian ini menjelaskan alur pengolahan data ke dalam *line chart*. Data mentah diambil dari referensi perancangan `chartData` dari *bar chart*, yang kemudian dimodifikasi menjadi *dataset* terstruktur dengan pasangan *label* waktu dan jumlah pengiriman. Pengelompokan data diformat sesuai dengan periode yang dipilih oleh pengguna (harian, mingguan, bulanan). Setelah itu, data ini digunakan untuk menghasilkan visualisasi berupa *line chart* yang menggambarkan fluktuasi jumlah pengiriman *email* seiring waktu. Skema alur logika perancangan dapat dilihat pada Gambar 3.4.




Gambar 3.4. *Flowchart Line Chart*

## B.3 Implementasi

Grafik garis (*line chart*) dikembangkan menggunakan *ChartJS*, *library* yang telah digunakan pada fitur *bar chart*. Grafik ini dihubungkan dengan sistem *backend* melalui pemrosesan data dinamis berdasarkan waktu. Data yang diperoleh

akan disesuaikan dengan periode yang dipilih pengguna (harian, mingguan, atau bulanan) dan dikelompokkan dalam format *label* waktu sebagai *x value* dan jumlah *email* sebagai *y value*.

Implementasi diinisiasi dengan pembuatan *canvas* dan identifikasi elemen grafik yang diinginkan. Fungsi `initChart` digunakan untuk menginisialisasi grafik, termasuk validasi apakah grafik sebelumnya sudah ada atau perlu dihapus. *Dataset* diproses dari *controller* ke dalam variabel `lineData`, kemudian ditampilkan melalui fungsi inisialisasi ketika pengguna memilih periode yang diinginkan. Tampilan *line chart* akan selalu diperlihatkan pada periode harian sebagai *default*. Kode perakitan *line chart* dapat dilihat pada kode 3.2.

```
// Inisialisasi canvas chart
<div class="chart-container">
    <canvas id="lineChart" height="400" style="height: 400px;"></canvas>
</div>

// Fungsi universal initChart
function initChart(canvasId, type, data, options) {
    const canvas = document.getElementById(canvasId);
    if (!canvas) return null;

    const existingChart = Chart.getChart(canvas);
    if (existingChart) {
        existingChart.destroy();
    }

    return new Chart(canvas, {
        type: type,
        data: data,
        options: options
    });
}

// Dataset line chart berdasarkan periode
                const initialSuccessData = realDayData.success.map(item => ({
                    x: Date.parse(item.x + ' 00:00:00 GMT+0700'),
                    y: item.y
                }));

                const initialErrorData = realDayData.error.map(
```

```
    item => ({
                    x: Date.parse(item.x + ' 00:00:00 GMT+0700'),
                    y: item.y
                }));

                const lineData = {
                    datasets: [{
                        label: 'Success Count',
                        data: initialSuccessData,
                        backgroundColor: 'rgba(40, 167, 69, 0.1)',
                        borderColor: 'rgba(40, 167, 69, 1)',
                        borderWidth: 3,
                        fill: false,
                        tension: 0.4,
                        pointBackgroundColor: 'rgba(40, 167, 69,
1)',
                        pointBorderColor: '#fff',
                        pointBorderWidth: 2,
                        pointRadius: 6
                    }, {
                        label: 'Error Count',
                        data: initialErrorData,
                        backgroundColor: 'rgba(220, 53, 69, 0.1)',
                        borderColor: 'rgba(220, 53, 69, 1)',
                        borderWidth: 3,
                        fill: false,
                        tension: 0.4,
                        pointBackgroundColor: 'rgba(220, 53, 69,
1)',
                        pointBorderColor: '#fff',
                        pointBorderWidth: 2,
                        pointRadius: 6
                    }]
                };
```

Kode 3.2: Kode pembuatan *line chart*

*Error handling* pada *bar chart* turut diterapkan pada fitur ini untuk menghindari konflik ketika pengguna mengganti periode tampilan. *Function* memeriksa apakah ada grafik aktif yang sedang ditampilkan, lalu menghapusnya sebelum menampilkan data baru. Pewarnaan grafik juga mengarah pada kenyamanan visual dengan warna merah dan hijau agar membedakan kedua variabel. Hasil implementasi dapat dilihat pada Gambar 3.5.


(a) *Line Chart Day* (b) *Line Chart Week*

(c) *Line Chart Month*

Gambar 3.5. *Line chart* dengan fitur periode

## C Calendar

### C.1 Analisis Kebutuhan

Penyusunan waktu dan jadwal pengiriman menjadi unsur penting dalam menjaga sistem pengiriman email yang efisien. Halaman *dashboard email-blast* saat ini belum mempunyai sistem manajemen waktu terintegrasi, sehingga menyulitkan pengguna dalam melakukan perencanaan dan eksekusi pengiriman *email* dengan sistematis. Informasi yang ditampilkan *Dashboard* hanya berupa jumlah *email* yang berhasil dan gagal terkirim. Kondisi halaman ini tidak memberikan wadah untuk menjadwalkan pengiriman dengan referensi waktu atau pengingat tanggal. Berdasarkan kebutuhan ini, diusulkan pengembangan fitur kalender interaktif yang terintegrasi dengan sistem *email-blast* dan to-do list. Fitur bertujuan dalam tiga fungsi utama:

1. Penjadwalan strategis pengiriman *email* berdasarkan penjadwalan waktu optimal

2. Manajemen *deadline* dan pengingat tugas pendukung yang terintegrasi dengan fitur *to-do list*

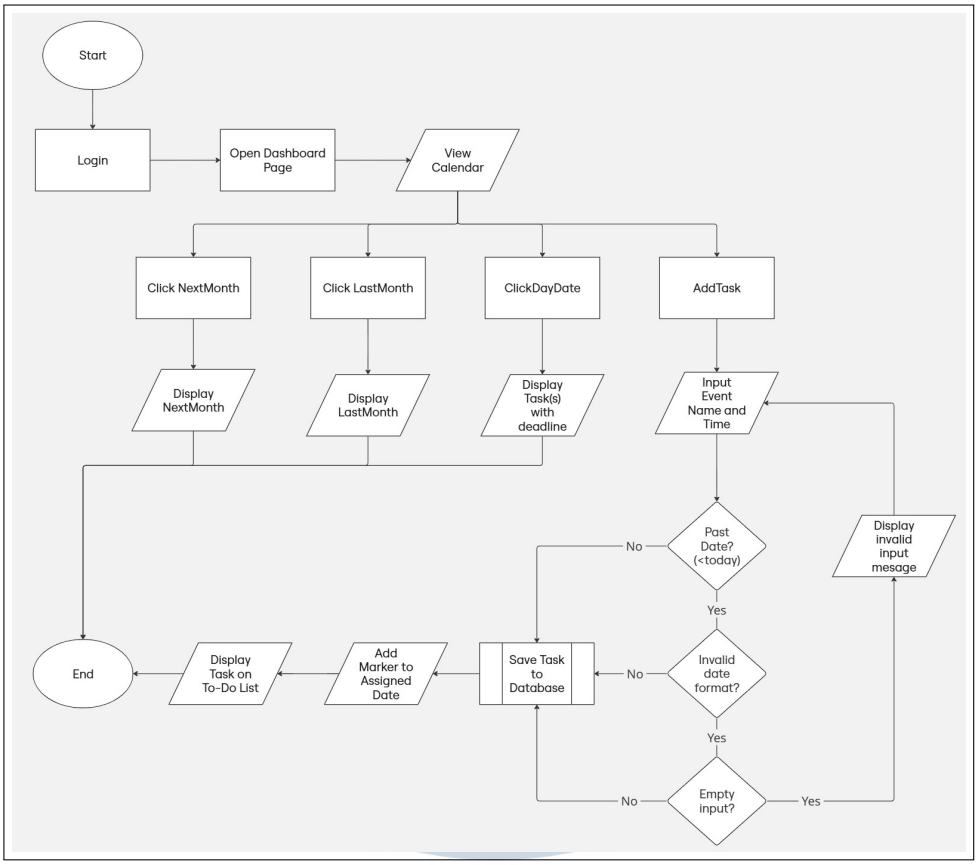
3. Penjabaran task to-do list berdasarkan tanggal (deadline yang telah ditenntukan

Fitur kalender mencakup beberapa unsur fungsional dimana salah satunya mencakup tombol perpindahan bulan. Tombol ini dapat menjangkau hingga pergantian tahun dan dapat menelusuri ke tanggal masa lalu dan masa depan. Tombol penambahan task juga diimplementasikan agar penambahan tugas dapat dimasukan dari kalendar. Tugas yang dimasukkan akan terintegrasi dengan fitur to-do list dan akan tertampil dengan daftar *task* lainnya. Fitur terakhir berupa pemantauan tugas pada kalender sesuai dengan tanggal yang telah dimasukkan saat penambahan task tersebut.

## C.2 Perancangan *Logic*

Perancangan fitur kalender dimulai dengan pencarian referensi akan implementasi fitur ini yang kompatibel dengan tampilan *dashboard*. Untuk penyesuaian layout dari halaman *dashboard* beserta fitur lainnya, perancangan kalender manual *calendar widget* kustom yang didasari dengan kode *html, javascript dan AJAX*. Jenis implementasi ini mendukung fleksibilitas dalam penambahan aspek sesuai dengan kebutuhan pengguna. styling fitur menggunakan Bootstrap sebagai dasar dengan penambahan pengaturan CSS sebagai pelengkap.

Alur fitur kalender yang diharapkan tertera lebih jelas pada gambar 3.6.




Gambar 3.6. *Flowchart* Fitur Kalender

Fitur akan sudah ditampilkan pada *dashboard* ketika pengguna mengakses *web* setelah melakukan *login*. Pengguna memiliki beberapa opsi interaksi: mengklik tombol "*NextMonth*" atau "*LastMonth*" untuk perpindahan bulan, menampilkan *task* yang tertera pada tanggal kalender dengan rincian, atau menambahkan *task* baru dengan menekan tombol "*Add Task*".

Penambahan task baru akan terdiri dari input nama *task* beserta *deadline*. Input yang diterima akan tersimpan pada database tabel *todos* dengan id task tersendiri dan tanggal task dibuat. Setelah data *task* berhasil tersimpan pada database, kalender akan menampilkan *task* baru tersebut pada to-do list. Fitur mendukung pemaparan task berjumlah banyak pada setiap tanggal. Ketika menekan suatu tanggal yang tertandai, kalender akan mengeluarkan modal yang menampilkan daftar task yang tercantum pada tanggal tersebut

## C.3 Implementasi

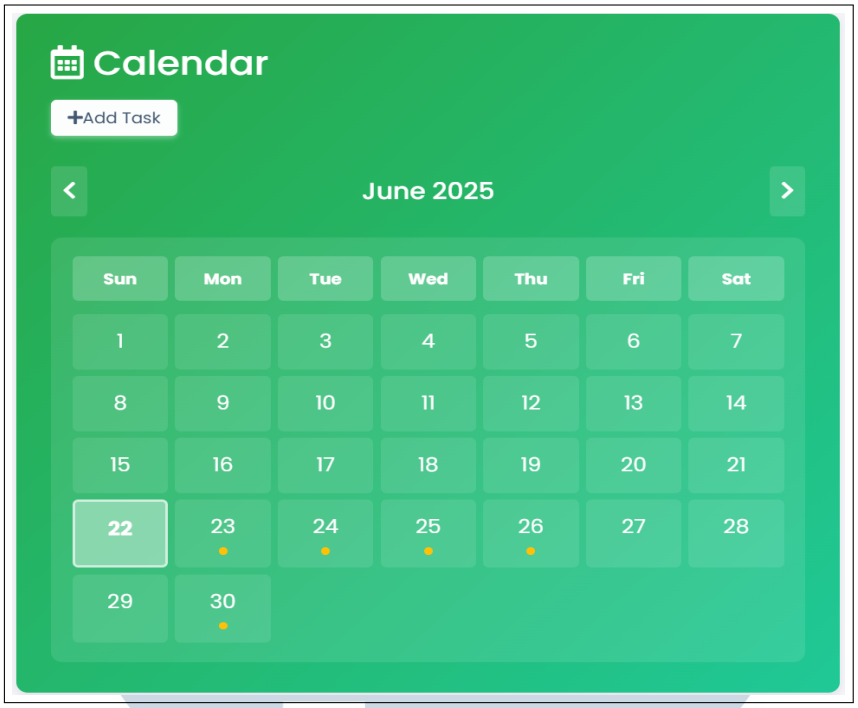
Implementasi fitur ini akan menggunakan *javascript*. Percobaan yang diaplikasikan mencakup pembuatan kalender yang mengandalkan pembuatan

struktur kalender tanpa ketergantungan pada *library* eksternal. Struktur data seperti bulan dan tanggal dimasukkan secara manual dan disambungkan ke dalam sistem. Jenis perancangan ini menyebabkan pengelompokkan function sesuai dengan aspek yang dikembangkan. Pengembangan dasar dari fitur ini berawal dari pembuatan *grid* kalender dan penempatan hari pada setiap bulan. daftar hari ditampilkan dengan header diatas grid hari. Pengisian tanggal menggunakan metode pencarian tanggal terakhir agar mengetahui jumlah hari pada setiap bulan. Pencarian hari pertama dari tanggal awal bulan juga diaplikasikan untuk mengidentifikasi kolom kosong pada *grid*. Dengan ditemukannya tanggal awal dan akhir bulan, pemasukan setiap tanggal pada grid kalender dijalani, menyesuaikan layout tanggal pada setiap bulan.

Perpindahan bulan pada kalender dapat diakses menggunakan kedua tombol navigasi di sisi kiri dan kanan kalender. tombol navigasi menggunakan [eventListener] untuk membaca bulan yang sekarang, dan mengurangi atau menambahkan variabel bulan untuk berganti bulan sesuai dengan urutan masing-masing bulan. Ketika mencapai bulan Desember atau Januari, tombol akan berpindah ke tahun sebelum atau ke depannya berdasarkan tombol navigasi yang ditekan. Hal ini dirancang agar kalender tidak terbatas dengan tahun kini saja namun juga untuk tahun lalu atau bahkan tahun depan.

Pemasukan *task* pada fitur kalender terdapat pada tombol Add Task di sebelah kiri atas. Tombol ini menggunakan AJAX dengan penambahan task baru ke dalam *database* . Modal yang tertampil ketika mengklik tombol *Add Task* akan meminta input nama *task* dan tanggal tenggat waktu *task* tersebut. Modal menerapkan *error handling* dimana input nama dan tanggal harus terisi dan sesuai dengan format tanggal yang tertera (dd-mm-YYYY). Penyimpangan dari input tersebut akan menimbulkan alert yang memberitahukan input yang invalid.

Dalam penyelesaian penambahan *task* baru, fitur *to-do list* akan menampilkan *database* yang telah diperbarui lengkap dengan sisa waktu sebelum *deadline*. *task* yang ditambahkan juga dapat dilihat pada tanggal tenggat waktu *task* tersebut. Suatu modal akan tertampak ketika mengklik suatu tanggal yang akan menampilkan daftar *task*. Jika tidak ada *task* yang terdaftar di tanggal tersebut. Modal akan mengirimkan pesan bahwa tidak ada tugas yang terkait pada tanggal ini. Modal akan mengkaji informasi berupa nama *task* dengan tenggat waktu yang dimasukkan ketika menggunakan tombol *Add Task*. Modal ini juga mendukung penjabaran *task* dalam jumlah banyak dalam satu hari. Hasil implementasi kalender dapat dilihat pada gambar 3.7.




Gambar 3.7. Fitur *Calendar*

## D *To-do List*

### D.1 Analisis Kebutuhan

Pada tampilan *email-blast* sebelum pengembangan, belum terdapat fitur pengelolaan tugas yang mendukung proses perencanaan kegiatan pengiriman secara sistematis. Hal ini menghambat kinerja pengguna dalam pengaturan pengerjaan tugas berkaitan dengan pengiriman email. Keseluruhan tahapan tersebut memerlukan sistem pencatatan dan pengingat agar tidak terjadi keterlambatan ataupun kelalaian dalam eksekusi. Fitur *to-do list* bertujuan untuk menanggulangi isu tersebut sebagai wadah pengingat akan tugas yang akan dikerjakan. Fitur ini ditujukan agar membantu pengguna mengelola dan memantau penyelesaian tugas-tugas yang mendukung kelancaran pengiriman *email*. Adapun kebutuhan utama dari fitur ini yang dijabarkan sebagai berikut:

1. **Pencatatan Tugas Harian dan Berkala**
   Pengguna dapat memasukkan tugas dalam daftar *task* yang harus diselesaikan, dengan periode Tenggat waktu tertentu.

2. **Penentuan Status dan Tenggat Waktu**
   Setiap tugas diberi label status (to-do, on progress, done) dan batas waktu

penyelesaian untuk pemantauan dan pengaturan pengerjaan tugas sesuai dengan efisiensi pengguna.

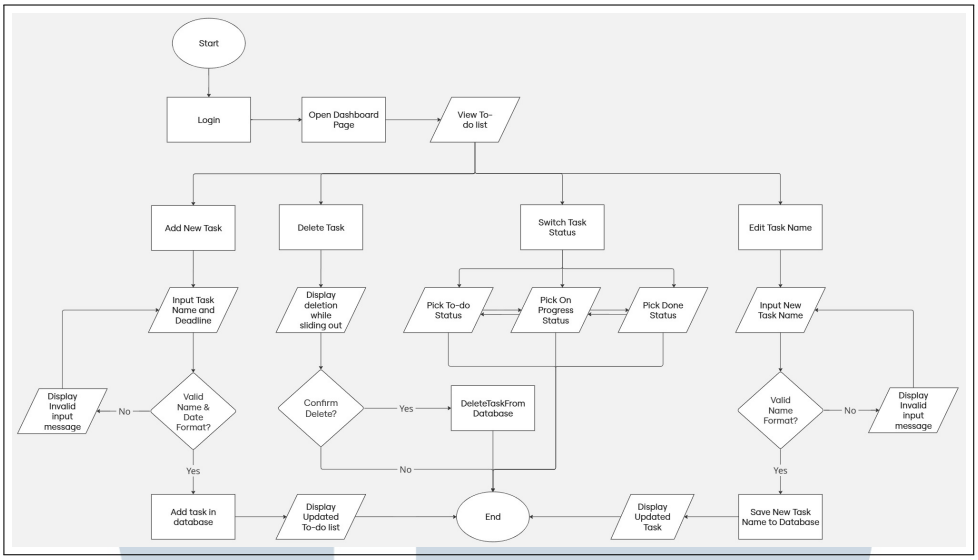
3. **Pengubahan Nama dan Penghapusan Tugas**
   Pengguna dapat mengubah nama *task* yang telah ada dengan *input* beserta dengan penghapusan tugas dari daftar untuk menjaga kerapihan dan ukuran *to-do list*.

Fitur *to-do list* diharapkan sebagai alat bantu produktivitas yang meningkatkan efisiensi kerja. Dengan adanya fitur ini, proses kerja didorong agar menjadi lebih tertata, tercatat.

## D.2 Perancangan Logika

Logika fitur *to-do list* dimulai dengan pemaparan daftar tugas beserta elemen-elemen pelengkap pada *task card*. Daftar tugas akan tertera berdasarkan urutan pembuatan tugas, dimana setiap tugas baru yang dibuat akan berada di baris paling bawah di daftar tugas. Setiap tugas mempunyai *id* masing-masing untuk referensi fungsi lain sejak inisialisasi tugas pada *database*. Elemen-elemen yang ada pada setiap *task card* berfungsi sebagai bentuk *CRUD* bagi setiap tugas yang ditampilkan.

Mekanisme fitur textitCRUD mencakup penambahan, penampilan nama tugas dengan durasi tenggat waktu, pengubahan urutan, pengubahan nama tugas dan penghapusan tugas. Setiap fungsi mempunyai elemen masing-masing pada *task card* dalam daftar tugas. Penambahan tugas menggunakan kolom isi nama dan tenggat waktu. Penambahan tugas menyelesaikan proses dengan tombol *Add task* yang menyimpan *input* tugas ke dalam tabel *todos*. Pengubahan nama tugas mempunyai similaritas pada penambahan tugas dimana sebuah *input* berupa teks akan diminta oleh kolom isi, dan akan diperbarui dalam *backend*. Untuk penghapusan, sistem akan mengirim modal konfirmasi penghapusan yang mendukung prinsip *Permit Easy Reversal of Actions* pada 8 Golden Rules. Setelah memverifikasi penghapusan, maka tugas yang dipilih akan memberikan deskripsi bahwa tugas sedang dihapus sebelum bergeser keatas. Sistem menyertakan fungsi `deleteTaskFromDatabase()` yang mengirim permintaan *AJAX* dengan *token CSRF* untuk memastikan keamanan. *Logic* fitur ini dapat dilihat lebih jelas pada *flowchart* di gambar 3.8.




Gambar 3.8. *Flowchart To-Do List*

## D.3 Implementasi

Implementasi fitur *To-Do List* ini dibangun dengan pengaplikasian interface berbasis *HTML* dengan fungsionalitas dinamis menggunakan *JavaScript*. Fitur ini terdiri dari *body* berisi daftar tugas dalam bentuk *unordered list*, dan *footer* yang berisi *form input* untuk menambahkan tugas baru. setiap *task card* akan berunsur akan nama tugas, tombol *handle* untuk fungsi *drag-and-drop*, *badge* waktu dengan warna berbeda sesuai sisa durasi tenggat waktu, tombol *edit* dan *hapus*.

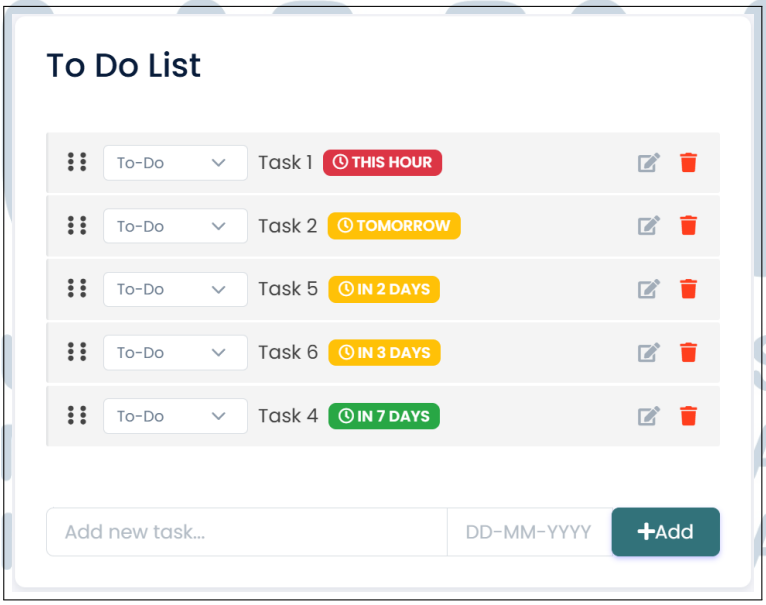
Fungsi penambahan *task* akan menyimpan dua jenis variabel berbentuk teks dan tanggal dalam format dd-mm-YYYY. Kedua variabel ini adalah nama dan tenggat waktu dari tugas tersebut. Data akan dikirim dan disimpan dalam tabel *mySQL todos* yang menyimpan kedua variabel tersebut dan menambahkan *id*, dan *status task* sebagai *to-do* secara *default*. Fitur akan memperbarui diri dengan memanggil kembali daftar *task* pada tabel *todos* dengan *task* baru. Sisa durasi sebelum tenggat waktu ditampilkan disamping kanan nama *task* yang bersifat dinamis berdasarkan dekatnya tanggal *deadline*. *deadline* dalam waktu lebih dari 3 hari akan menampilkan *badge* waktu berwarna hijau. *deadline* dalam kurun waktu 1 sampai 3 hari akan menampilkan *badge* waktu berwarna kuning untuk menunjukkan kepentingan *task* yang disarankan untuk diselesaikan. *deadline* dalam kurun waktu kurang dari 1 hari akan menampilkan *badge* waktu berwarna merah yang mengindikasikan urgensi *task* yang harus diselesaikan dalam waktu dekat.

Pengubahan nama dan penghapusan *task* dilakukan dengan mengklik kedua elemen *edit* dan *trash*. Menjelang pengubahan nama, *input* akan diterima dengan

kolom isi akan nama baru dari *task* yang terpilih. Setelah mengkonfirmasi pengubahan maka nama *task* baru akan tertera secara langsung dalam *to-do list* melalui AJAX. nama *task* pada *database* juga akan diubah menjadi nama *task* baru. Ketika mengklik tombol *trash*, modal konfirmasi penghapusan akan keluar agar mencegah kesalahan penghapusan *task* yang tidak diinginkan. Aspek ini juga mendukung prinsip *8 Golden Rules* yang mendukung kemudahan untuk membatalkan aksi pengguna. Setelah mengkonfirmasi, *task* yang terpilih akan memberikan deskripsi dalam proses penghapusan sebelum bergeser ke atas dan terhapus dari *to-do list* dan *database* menggunakan method *DELETE*.

Fitur *drag handle* menggunakan fungsi *placeholder* dalam penempatan setiap *task card*. Tombol *handle* dapat diklik dan ditahan untuk menarik *task* yang dipegang ke atas atau ke bawah sesuai dengan urutan yang diinginkan. Urutan daftar tugas akan tersimpan dalam *database* pada kolom *position*. Dengan ini, posisi urutan *task* akan tersimpan jika halaman diperbarui.

Untuk integrasi dengan *backend Laravel*, implementasi ini menyediakan fungsi khusus yang mengirim permintaan *AJAX* ke *endpoint RESTful* dengan *method PUT* untuk *update* dan *DELETE* untuk penghapusan, selalu menyertakan *CSRF token* sebagai bagian dari keamanan. Pendekatan *event delegation* digunakan untuk memastikan semua fungsi interaksi tetap bekerja pada elemen yang ditambahkan secara dinamis. Hasil penerapan fitur dapat dilihat pada gambar 3.9.



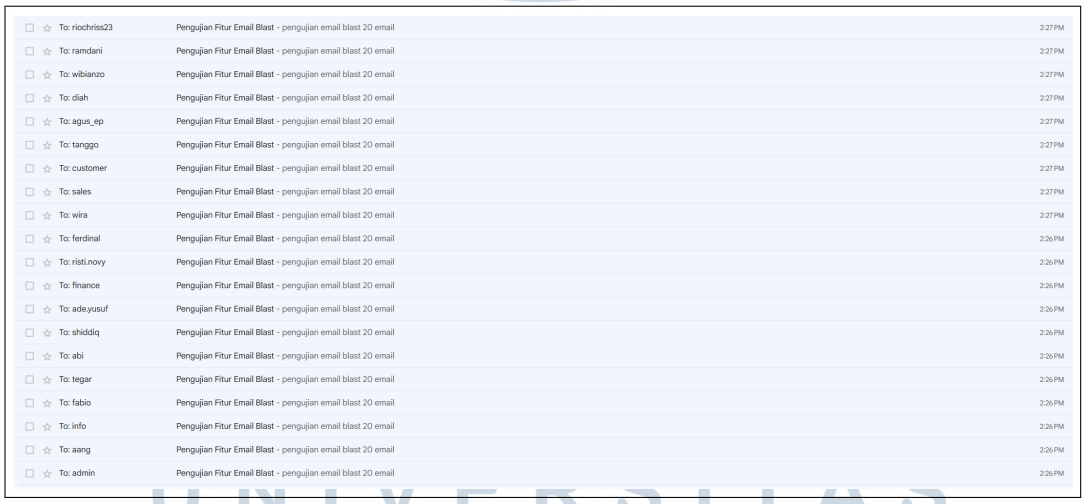

Gambar 3.9. Fitur *To-Do List*

### 3.3.3 Pengujian dan Evaluasi

#### A Pengujian fitur *email blast*

Pengujian fungsi utama *web* dijalankan dengan pengiriman *email blast* berjumlah 20 *email*. 19 *email* yang dikirim merupakan *email* karyawan perusahaan dan 1 lainnya merupakan akun pribadi untuk mengkonfirmasi pengujian. Daftar *email* pengujian dijabarkan dalam satu *file excel* dan dikirim pada halaman *blast email web*. Hasil pengujian fitur *email blast* dapat dilihat pada gambar 3.10 dan 3.11.


Gambar 3.10. Pengiriman *email* dalam *web*




Gambar 3.11. Pengiriman *email* dalam *gmail*

Pengujian fitur *email blast* berakhir dengan keberhasilan pengiriman 20 jumlah *email* yang direncanakan. Pengujian telah membuktikan pengiriman berhasil dikirim dan tertuju pada masing-masing *email* yang *valid* dan aktif. Dengan ini, fitur-fitur berupa grafik dapat menampilkan informasi yang sesuai dan akurat.

## B Pengujian fitur *Dashboard*

Pengujian fitur *dashboard* dilakukan dengan *black box testing*. Evaluasi fitur dapat dinilai dalam kategori *PASS* dimana fitur berhasil diimplementasikan atau *FAILED* yang memberitahukan bahwa fitur yang ditujukan gagal dalam implementasi. *Testing* fitur berupa tabel yang berisi segala fungsi pada fitur yang telah ditambahkan pada halaman *dashboard* beserta metode yang digunakan dan hasi yang diharapkan dari fitur tersebut. Hasil pengujian fitur *bar chart* tertampak pada Tabel 3.3.

Tabel 3.3. Hasil Pengujian Fitur *Web*

| No | Test Scenario | Metode Pengujian | Expected Result | Result |
|---|---|---|---|---|
| 1 | Membuka *website* | *Website dibuka* | Berhasil masuk ke *website* | PASS |
| 2 | *Login* | Masuk ke akun pribadi | Berhasil masuk ke akun pribadi | PASS |
| 3 | Membuka halaman *dashboard* | Halaman terbuka | Halaman *dashboard* berhasil terbuka | PASS |
| 4 | Fitur *Bar Chart* | Tampilan fitur | Fitur tampil dengan benar | PASS |
| 5 | Fitur *Line Chart* | Tampilan fitur | Berhasil menampilkan isi detail artikel | PASS |
| 6 | Fitur *Calendar* | Tampilan fitur | Menampilkan komentar dan tersimpan di *database* | PASS |
| 7 | Fitur *To-do list* | Tampilan fitur | Menampilkan komentar sebelumnya | PASS |

Lanjut ke halaman berikutnya

Tabel 3.3 – lanjutan

| No | Test Scenario | Metode Pengujian | Expected Result | Result |
|---|---|---|---|---|
| 8 | Tampilan *Bar Chart* dinamis | Data berubah sesuai input | Menampilkan data dinamis akurat | PASS |
| 9 | Pemilihan periode *Line Chart* | Klik tombol day/week/month | Menampilkan data sesuai periode | PASS |
| 10 | Letak tanggal dan bulan *Calendar* | Navigasi bulan | Tanggal tampil real-time | PASS |
| 11 | Perpindahan bulan *Calendar* | Tombol kiri/kanan | Kalender bulan tampil sesuai | PASS |
| 12 | Modal tambah *task* baru | Klik tombol *add task* | Modal muncul dengan benar | PASS |
| 13 | Task baru ke *To-do List* | Klik tombol tambah task | Task tampil di *To-do List* | PASS |
| 14 | Marker tanggal *Calendar* | Tambah task via kalender | Marker muncul di tanggal sesuai | PASS |
| 15 | Daftar task tanggal *Calendar* | Klik tanggal kalender | Modal *ViewTaskModal* tampil | PASS |
| 16 | Daftar task *To-do List* | Menampilkan dari *database* | Semua elemen task tampil | PASS |
| 17 | Tambah task baru | Isi nama dan deadline lalu klik tambah | Task baru muncul | PASS |
| 18 | Edit task | Klik ikon *edit* | Nama task diperbarui di *database* | PASS |
| 19 | Hapus task | Klik ikon *trash* | Muncul modal konfirmasi | PASS |
| 20 | *Drag and drop* task | Geser task card | Posisi berubah dan tersimpan | FAILED |

Lanjut ke halaman berikutnya

| Tabel 3.3 – lanjutan | | | | |
|---|---|---|---|---|
| **No** | **Test Scenario** | **Metode Pengujian** | **Expected Result** | **Result** |
| 21 | Elemen durasi deadline | Hitung durasi otomatis | Durasi berubah dinamis | PASS |
| 22 | Modal penghapusan task | Konfirmasi hapus | Task terhapus dengan animasi | PASS |
| 23 | Integrasi *database* | Manipulasi data via MySQL | Fungsi CRUD berjalan baik | PASS |

Berdasarkan hasil pengujian fitur-fitur yang telah diimplementasikan pada halaman *dashboard email-blast*, sebagian besar fitur secara umum dapat berjalan dengan baik dan sesuai dengan harapan dari hasil yang diinginkan. Fitur-fitur utama yakni *line chart* dan *bar chart* sebagai visualisasi pengiriman email, kalender interaktif, serta fitur *to-do list* telah terintegrasi secara fungsional dan responsif.

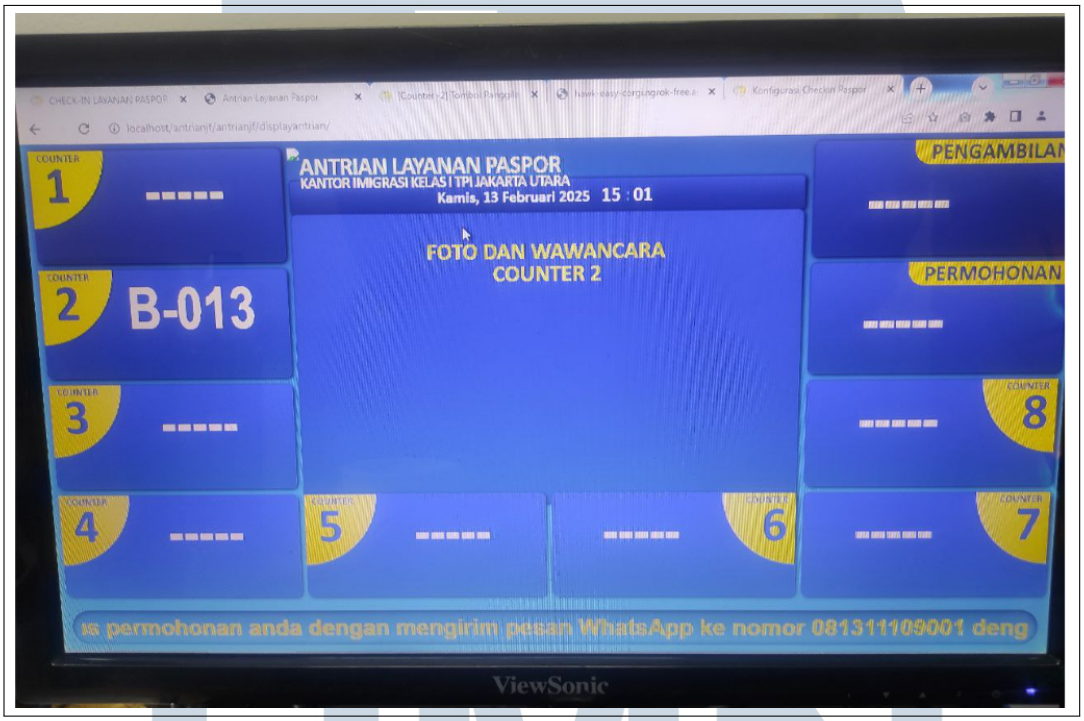
Secara tampilan antarmuka, *layout* dari *dashboard* tersusun dengan rapi dan tidak menimbulkan konflik antar elemen. Perancangan antarmuka juga didasarkan teori *8 Golden Rules* sebagai landasan penerapan dan penempatan elemen pada setiap fitur. Penempatan fitur disesuaikan dengan kebutuhan pengguna agar mempermudah navigasi penggunaan bersifat intuitif. Penggunaan kerangka desain dinamis juga memastikan bahwa informasi yang ditampilkan akan selalu berdasarkan *real-time* sesuai denga penggunaan *web*.

Menjelang proses *testing*, Fitur yang belum berhasil diimplementasikan merupakan fungsi pengubahan urutan tugas dengan *drag and drop handle*. Tugas dapat diubah secara tata letak dengan lancar namun akan kembali kepada posisi semula menjelang pembaruan halaman *web*. Selain itu, semua fitur tertampil dan responsif beserta elemen-elemen yang terkait dengan fitur tersebut. Beberapa *bug* minor sempat ditemukan, terutama pada tahap awal integrasi fitur kalender, yang kemudian berhasil diperbaiki melalui revisi metode implementasi dan pengecekan *syntax* kode. Hasil textittesting dapat dilihat pada lampiran *testing*.

Secara keseluruhan, *dashboard* yang telah dikembangkan berhasil meningkatkan kapabilitas sistem dalam memberikan informasi dan kontrol terhadap kampanye *email-blast*. Fitur-fitur yang tersedia mendukung pengguna dalam mengambil keputusan yang lebih baik berbasis data dan jadwal, serta memberikan pengalaman penggunaan yang lebih terarah dan efisien.

### 3.3.4 Perancangan *Interface Queue Management System*

Bagian ini menjelaskan tugas awal yang diberikan oleh *supervisor* sebelum tugas utama dalam pengembangan *web*. Tugas yang diberikan mencakup merancang *mockup* tampilan sebuah *Queue Management System* (QMS) layanan paspor. Tampilan sebelumnya diberikan oleh *supervisor* sebagai referensi untuk faktor yang ingin diubah dan dipertahankan sesuai dengan keinginan klien.




Gambar 3.12. Tampilan *Queue Management System* layanan paspor sebelumnya

Arahan *supervisor* meliputi penambahan jumlah konter pada satu tampilan dengan rapi dan tertata. Warna, dan tata letak *interface* juga disarankan untuk tidak jauh berbeda dari referensi. Terdapat informasi yang juga dihimbau untuk ditampilkan beserta dengan daftar konter, yaitu jenis pelayanan pada tiap konter. Tugas yang diberikan juga memaparkan untuk membuat 3 desain yang berbeda dengan kelebihan dan kekurangan masing-masing. *Layout* setiap desain perlu terasa unik dan dapat diterapkan secara nyata untuk kemudahan pelanggan.

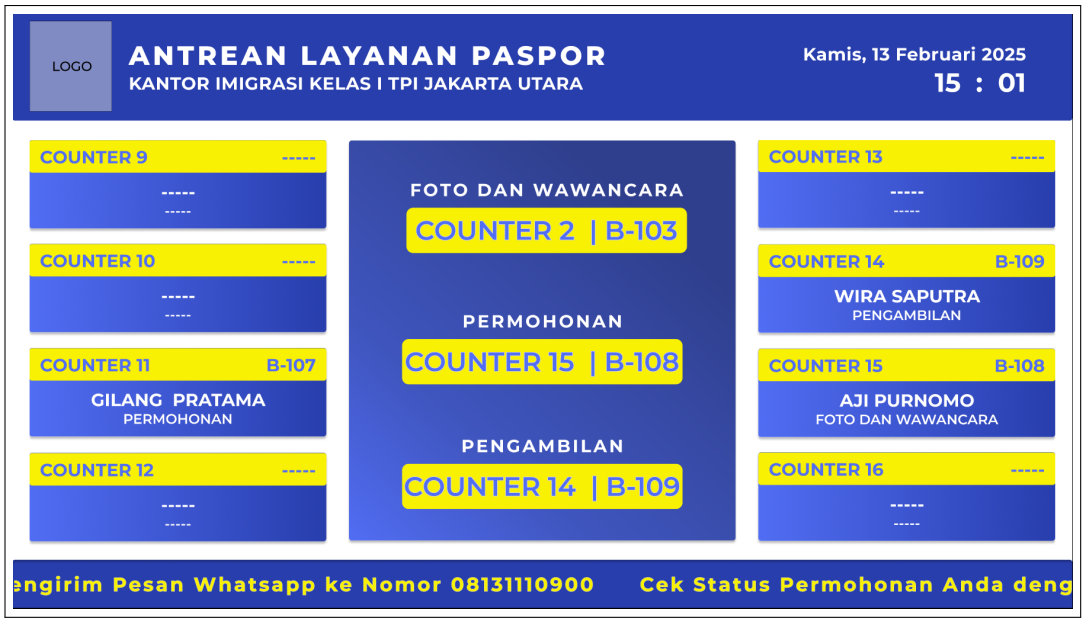
Pengerjaan tugas tampilan QMS dilakukan dengan *Figma* pada PC. Alur perancangan tampilan mencakup diskusi berkaitan dengan gambaran *mockup* sesuai dengan aspek-aspek yang diinginkan pada tampilan yang dirancang dalam bentuk sketsa. Gambaran kasar tersebut lalu dirombak menjadi *wireframe* setiap desain yang memfinalisasikan *layout* setiap informasi. Tahap terakhir merupakan

perancangan *mockup* setiap desain dengan fitur masing-masing beserta revisi dari desain awal. Pada proses perancangan, terdapat beberapa faktor yang berfungsi sebagai patokan setiap desain. Faktor yang dipaparkan dapat dilihat sebagai berikut:

1. Desain yang mudah dibaca dan jelas saat dipandang sekilas
2. Desain yang dinamis dan fleksibel sesuai dengan keadaan layanan
3. Desain yang informatif dan mencakup semua data dan kebutuhan pengguna dalam layanan paspor

Faktor yang telah tertera di atas menjadi wadah pembuatan setiap desain. Segi faktor tersebut juga didiskusikan dengan *supervisor* magang yang juga menyetujui akan tujuan desain. Tujuan utama dari desain mengarah pada efisiensi setiap tampilan dan kemudahan pelanggan untuk membaca informasi yang ditampilkan.

## A Desain 1




Gambar 3.13. *Mockup* Desain 1

Desain ini melingkupi semua informasi yang ingin disampaikan kepada pelanggan. Daftar konter terbagi di sebelah kanan dan kiri dengan jenis konter ditampilkan di tengah. Kolom informasi di tengah diatur menjadi lebih besar dari

daftar konter agar pengguna dapat melihat nomor antrian serta nomor konter yang sedang dilayani saat mengantri. Nomor antrian dan konter akan mengedip setiap kali layanan telah selesai dan pelanggan baru dipanggil untuk mengarah ke konter. Tercakup 8 dari 16 konter di layar yang akan mengedip setiap beberapa detik dan berotasi sesuai dengan urutan konter. Setiap daftar konter juga mempunyai nama serta jenis pelayanan untuk mengidentifikasi pelanggan secara rinci.

Desain 1 unggul pada penyampaian informasi yang praktis dan lengkap. Kolom informasi yang dijabarkan di tengah beserta daftar konter di sebelah bertujuan untuk memberikan pelanggan kenyamanan dalam memberikan informasi dengan detail, tanpa membuat teks terlalu kecil hingga tidak bisa dibaca. Namun, kekurangan desain ini beralih pada ukuran daftar konter serta pemberitaan informasi yang tertunda. Ukuran daftar konter akan bersifat tidak efisien dengan nama pelanggan yang panjang, mengakibatkan teks lebih sulit dibaca dari jauh. Tampilan daftar konter yang berubah setiap beberapa detik juga dapat mengecoh informasi terkini jika pelanggan tidak memperhatikan layar.

## B Desain 2


Gambar 3.14. *Mockup* Desain 2

Desain ini menampilkan informasi layanan dengan tata letak yang pragmatis dan sederhana. Seluruh tampilan dibagi ke dalam *grid* 2 baris x 4 kolom, menampung 8 dari total 16 konter aktif secara bersamaan. Masing-masing

*panel* menyajikan nomor antrian, jenis layanan, dan nama pemohon secara utuh. Informasi waktu, tanggal, serta lokasi layanan ditampilkan di bagian atas sebagai elemen navigasi waktu yang jelas. Seiring berjalannya sistem, tampilan konter akan berpindah secara otomatis ke kiri dan kanan setiap 5 detik, memastikan seluruh konter dapat ditinjau secara berkala oleh para pengunjung.

Desain 2 memiliki kekuatan pada keseimbangan visual dan keterbacaan. Semua elemen disusun dalam ukuran yang konsisten, sehingga tidak ada informasi yang terasa menonjol secara berlebihan maupun tersembunyi. Tampilan ini memberi kesan modern dan tertata tanpa mengorbankan kelengkapan data. Meskipun begitu, tidak adanya satu titik fokus utama dapat menyebabkan kebingungan kecil bagi pengguna yang mencari antrian mereka dengan cepat. Faktor ini juga dapat menyebabkan mata pelanggan untuk tidak dapat menangkap dan membaca nomor antrian yang tertera. Selain itu, pergeseran otomatis masih rentan untuk terlewatnya informasi apabila pengunjung tidak memperhatikan layar pada waktu yang tepat.

## C Desain 3


Gambar 3.15. *Mockup* Desain 3

Desain ini menyajikan format dua kolom yang terpisah fungsinya secara jelas: Kolom kiri berperan sebagai titik fokus informasi utama, sedangkan kolom kanan menampilkan daftar konter yang sedang aktif. Bagian kiri menampilkan

secara langsung layanan yang tengah berlangsung lengkap dengan nomor konter dan antrian dalam format mencolok. Di sisi lain, empat konter ditampilkan secara simultan di bagian kanan, dengan keterangan lengkap: nomor antrian, jenis layanan, nama pemohon, serta posisi konternya. Setiap beberapa detik, daftar ini akan berpindah ke empat konter berikutnya. Setelah mencapai akhir daftar, tampilan akan kembali ke awal, membentuk rotasi berkelanjutan. Indikator pergantian ini dipertegas dengan efek kedipan pada kolom informasi ketika ada pelanggan baru yang dilayani.

Desain 3 menonjolkan fokus informasi secara visual dan fungsional. Dengan memisahkan area informasi utama dan daftar konter, pengguna dapat langsung mengenali apakah giliran mereka sudah tiba tanpa harus membaca semua data sekaligus. Spasi teks yang luas pada daftar konter juga menampung nama pelanggan yang terkesan panjang. Efek berkedip saat antrian berubah memberi petunjuk dinamis yang menangkap perhatian dan membantu pengunjung tetap waspada. Meskipun begitu, pembagian kolom ini menyisakan ruang terbatas bagi daftar konter, sehingga teks dapat terlihat padat jika nama pelanggan atau deskripsi layanan terlalu panjang. Selain itu, karena rotasi informasi bersifat bertahap per empat konter, pelanggan akan menunggu giliran tampil dengan durasi yang lama agar bisa memantau status mereka.

## D Kendala dan Solusi Pelaksanaan Magang

Berikut merupakan kendala yang dialami selama pelaksanaan magang dijalankan:

1. Keterbatasan kolaborasi antar pengembang *web* dikarenakan pengerjaan tugas magang yang mayoritas dilaksanakan secara *online*
2. Keterbatasan komunikasi dengan *supervisor* magang dan hanya dapat dilakukan via *Trello* dan rapat berkaitan dengan pemberian dan revisi pekerjaan magang.
3. Timbulnya beragam *bug* pada fitur *dashboard* yang berdampak pada fitur lain dan menghambat pengembangan *web* dalam jangka panjang.

Berikut tertera solusi yang diterapkan untuk menanggulangi kendala dan permasalahan yang terjadi menjelang pelaksanaan magang:

1. Mengadakan rapat rutin secara mingguan dengan rekan kerja untuk membangun koordinasi dan memberi serta menerima *feedback* dari hasil pekerjaan magang.
2. Memberikan laporan progres yang sering dilakukan agar *supervisor* dapat memberi revisi dan arahan selagi pembangunan fitur untuk menghindari terjadinya miskomunikasi.
3. Penerapan *callback* terhadap fitur yang bermasalah dan mencari penyebab permasalahan tersebut yang disertakan dengan pencarian solusi.