BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Program magang dijalankan di PT. Global Innovation Technology. Penulis disini memiliki posisi sebagai *Fullstack Web Developer* dan menjalankan program magang ini bersama dengan empat *intern* lainnya. Program magang ini di rancang untuk mengembangkan website CRM atau *Customer Relationship Management*. Terdapat *person in charge* sebagai koordinator dan pemberi arahan selama program magang berlangsung. Dalam pengembangan website ini, koordinatornya adalah Bapak Marcel dan Bapak Roni. Sedangan CEO dari perusahan ini adalah Bapak Aditya Pratama.

3.2 Tugas yang Dilakukan

Penulis diberikan tugas untuk mengembangkan *website* CRM atau *Customer Relationship Management* sebagai *fullstack website developer* bersama - sama dengan anggota lainnya. Penulis disini memiliki tugas utama mengembangkan fitur *dashboard* yang terdisi dari 4 macam, yaitu *Sales Performance, My Performance, Sales Analytics,* dan *Sales Activities*. Tugas lainnya adalah mengubah tampilan *frontend* pada *website* CRM.

Dalam proses pengembangan *website*, penulis sebelumnya telah diberikan kesempatan untuk *Self learning* mengenai penggunaan bahasa pemograman Spring boot untuk bagian back end dan bahasa pemograman Angular untuk bagian *frontend*. Kemudian diikuti dengan pembuatan *website* CRM. Berikut merupakan pekerjaan yang dilakukan selama program magang berlangsung.

- Melakukan pembuatan kode untuk mengembangkan *website* yang ada
- Melakukan pembuatan kode untuk membuat fitur - fitur baru dalam *website*
- Melakukan optimisasi kode yang sudah ada
- Memperbaiki *bugs* dan kesalahan yang ada selama proses pengembangan
- Melakukan komunikasi dengan tim untuk *weekly update*

3.3 Uraian Pelaksanaan Magang

Pelaksanaan atau penerapan program magang ini dimulai dengan *Self learning* menggunakan *project demo*. *Framework* yang diaplikasikan oleh perusahaan adalah Spring boot untuk bagian *backend* dan Angular untuk bagian *frontend*. Pembelajaran mandiri dilakukan dengan menerapkan *project demo* yang sudah disediakan oleh Koordinator dari magang tersebut. Setelah pembelajaran mandiri, dilakukanlah analisis dari *website* CRM dari segi *backend* dan juga *frontend*. Kemudian koordinator membagikan *source file* dari *website* CRM yang akan dikembangkan.

Dalam pengerjaannya, tim magang telah dibagi berdasarkan tugas dan tanggung jawabnya masing-masing. Setelah itu setiap pribadi akan mengerjakan bagiannya pada *local* masing - masing, yang kemudian akan diserahkan kepada koordinator untuk dilakukan *merger* dengan *source file* utama.

Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1.

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

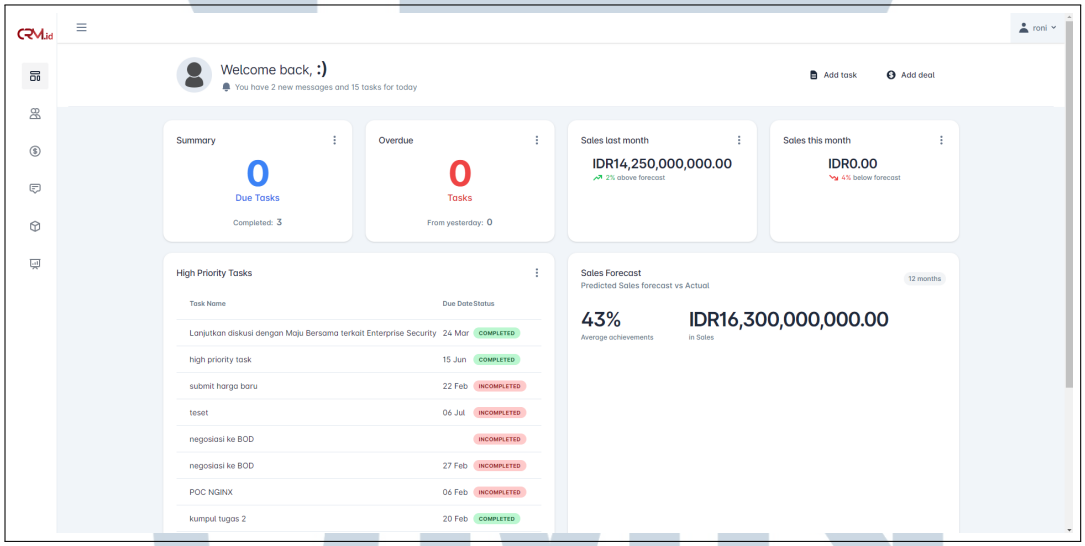
Minggu Ke -	**Pekerjaan yang dilakukan**
1	*Self learning framework* Springboot untuk pembuatan *website*
2	*Self learning framework* Angular untuk pembuatan *website*
3	Pembuatan *project demo* dan menganalisa *website* CRM
4	*Self learning* dan mencoba penggunaan API highcharts dalam *project demo*
5	Membuat *Pie Charts* menggunakan API highcharts dan mengimplementasikannya pada *website* CRM
6	Pengerjaan *dashboard Sales Analytics* dan menambah *funnel charts* kedalam dashboard
7	Membuat *multi charts* yang kemudian akan diimplemetasikan kedalam *dashboard*
8	Mengkoneksikan *Dashboard* dengan fitur - fitur yang telah dibuat
9	Melakukan koneksi *charts* yang ada dengan data asli
10	Melakukan analisis terhadap produk serupa dari perusahaan lain
11	Membuat tampilan *frontend dashboard* baru dengan menambahkan beberapa fitur baru didalamnya
12	Membuat tampilan fitur baru yaitu Tabel *High Priority Deals*
13	Membuat *service backend* pada fitur baru yaitu Tabel *High Priority Deals*
14	Melanjutkan membuat *service backend* pada fitur baru dan mengkoneksikan dengan *database*
15	Melakukan koneksi antara *frontend* dan *backend* pada fitur baru Tabel *High Priority Deals*
16	Membuat tampilan fitur baru yaitu *Sales Performance by Month*
17	Membuat *service backend* pada fitur baru yaitu *Sales Performance by Month*
18	Melakukan koneksi antara *frontend* dan *backend* pada fitur baru *Sales Performance by Month*
19	Membuat tampilan fitur baru yaitu *Target By Month*
20	Membuat *service backend* pada fitur baru yaitu *Target By Mont*h
21	Melakukan koneksi antara *frontend* dan *backend* pada fitur baru *Target By Month*
22	Membuat tampilan fitur baru yaitu *Tracking Sales*

3.4 Perancangan Fitur

Penulis mendapatkan tugas dan tanggung jawab berupa pengerjaan fitur tambahan, yaitu halaman *dashboard* utama yang berisi 4 halaman yaitu *Sales Activities, My Performance, Sales Performance,* dan *Sales Analytics*. Dari dashboard tersebut kemudian ditambahkan dengan total 3 fitur baru, yaitu Tabel *High Performance Deals, My Performance Every Month,* dan *Target by Month.*

3.4.1 Pembuatan fitur Tabel High Performance Deals pada dashboard Sales Activities

Berikut adalah tampilan dari halaman *dashboard Sales Activities* sebelum dilakukan perubahan yang terlihat pada gambar 3.1.



Gambar 3.1. Tampilan halaman *dashboard Sales Activities*

Berikut adalah fitur Tabel *High Performance Deals* yang telah penulis kerjakan pada gambar 3.2. Dalam fitur ini, tabel akan menampilkan data yang berisi nama *deals*, jumlah nominal yang didapatkan, tanggal tenggat waktu, dan *status* dari *deals* tersebut.

High Priority Deals

Deals Name	Amount	Due Date	Status
BCA - NGINX	24000000000	30 Mar	LOST
Mandiri - NGINX	2000000000	05 Feb	POC
LKPP	1000000000		NEW
Telkomsel	1000000000		WON
SIEM - Kemhan	9000000000		LOST
Site Scope	2200000000	29 Sep	NEW
Coca Cola Training Sales	1000000000	30 Mar	WON
FDS - Bank DKI	19844000000		LOST
Setup Asset Prediction System for Pensiun	14250000000	13 Apr	WON
Assesment	1025000000	29 Apr	WON

Gambar 3.2. Tampilan fitur Tabel *High Performance Deals*

Alur dari pengerjaan fitur tersebut adalah sebagai berikut, *backend* akan dikerjakan terlebih dahulu dengan menggunakan *framework* Springboot, yang kemudian hasilnya akan di cek melalui Postman. Setelah *service* berhasil mengambil *data*, baru kemudian akan dikoneksikan dengan *frontend* dengan menggunakan *framework* Angular.

1. **Pembuatan Service Backend** Untuk membuat *service backend*, diperlukan *controller* yang akan menampung *service* dan *query* yang ada. Pada fitur ini, *controller* mengambil *function* getHighPriorityDeals() Berikut adalah *codingan controller* yang menampung *service* yang ada pada gambar 3.3.

```
@RolesAllowed({ "admin", "user" })
@GetMapping("/deals/highpriories")
public ResponseEntity<?> getHighPrioriesDeals()
{
    try {
        return ResponseHandler.setResponse("OK", HttpStatus.OK, dashboardService.getHighPriorityDeals());
    } catch (Exception e) {
        String stacktrace = ExceptionUtils.getStackTrace(e);
        return ResponseHandler.setResponse(e.getMessage(), HttpStatus.BAD_REQUEST, null);
    }
}
```

Gambar 3.3. Tampilan *codingan service* fitur *Tabel High Performance Deals*

Pada gambar 3.4, terlihat bahwa *controller* menampung *service* yang berisi *data-data* yang diperlukan, sehingga *data* tersebut dapat digunakan untuk bagian *frontend*.

```
@Override
public List<DashboardHighPriorityDealsResponseDto> getHighPriorityDeals() {
    List<DashboardHighPriorityDealsResponseDto> dashboardHighPriorityDealsResponse = new ArrayList<DashboardHighPriorityDealsResponseDto>();

    BigDecimal cardDealValue = BigDecimal.valueOf(1000000000);

    List<DealCards> dealList = dealCardRepo.getHighPriorityDeals(cardDealValue);
    List<Deals> dealList1 = dealRepo.getHighPriorityDeals2(cardDealValue);
    for (DealCards deal: dealList) {
        DashboardHighPriorityDealsResponseDto dashboardHighPriorityDeals = new DashboardHighPriorityDealsResponseDto();
        dashboardHighPriorityDeals.setDealsName(deal.getCardTitle());
        dashboardHighPriorityDeals.setAmount(deal.getCardDealValue());
        dashboardHighPriorityDeals.setDueDate(deal.getCardDueDate());
        for (Deals deal1: dealList1) {
            List <DealCards> dealCardList = deal1.getDealCards();
            for(DealCards deal2: dealCardList) {
                if (deal.getDealCardId()== deal2.getDealCardId()) {
                    dashboardHighPriorityDeals.setStatus(deal1.getDealDisplayName());
                }
            }
        }
        dashboardHighPriorityDealsResponse.add(dashboardHighPriorityDeals);
    }
    return dashboardHighPriorityDealsResponse;
}
```

Gambar 3.4. Tampilan *codingan service* fitur *Tabel High Performance Deals*

2. **Koneksi dengan frontend** Setelah *data* berhasil diambil, selanjutnya akan dilakukan koneksi dengan bagian *frontend* sehingga *data* bisa ditampilkan ke *website* CRM. Berikut adalah *codingan* dari bagian *frontend* pada gambar 3.5. Pada *codingan* tersebut telah digunakan API yang telah dibuat pada *backend* dan dimasukan kedalam sebuah *function* yang akan digunakan kedepannya.

```
getHighPriorityDeals(): Observable<highPriorityDeals[]> {
  return this._httpClient
    .get<highPriorityDeals[]>(
      `${environment.apiBaseUrl}/dashboard/deals/highpriories`
    )
    .pipe(
      tap((body: any) => {
        console.log(body.data);
        this._highPriorityDeals.next(body.data);
      }),
      catchError(
        (error: HttpErrorResponse): Observable<any> => {
          return of(false);
        }
      )
    );
}
```

Gambar 3.5. Tampilan codingan frontend fitur Tabel *High Performance Deals*

Selanjutnya API tersebut akan diambil *data* yang ada didalamnya dan digunakan untuk fitur *Tabel High Priority Deals* dengan menggunakan *codingan* yang ada pada gambar 3.6.

```
this._dashboardService.highPriorityDeals$
  .pipe(takeUntil(this._unsubscribeAll))
  .subscribe((result: highPriorityDeals[]) => {
    console.log("highPriorityDeals", result);
    this.highPriorityDeals = result;
    this.dataSourceDeals = result;
    this._changeDetectorRef.markForCheck();
  });
```

Gambar 3.6. Tampilan *codingan frontend* fitur Tabel *High Performance Deals*

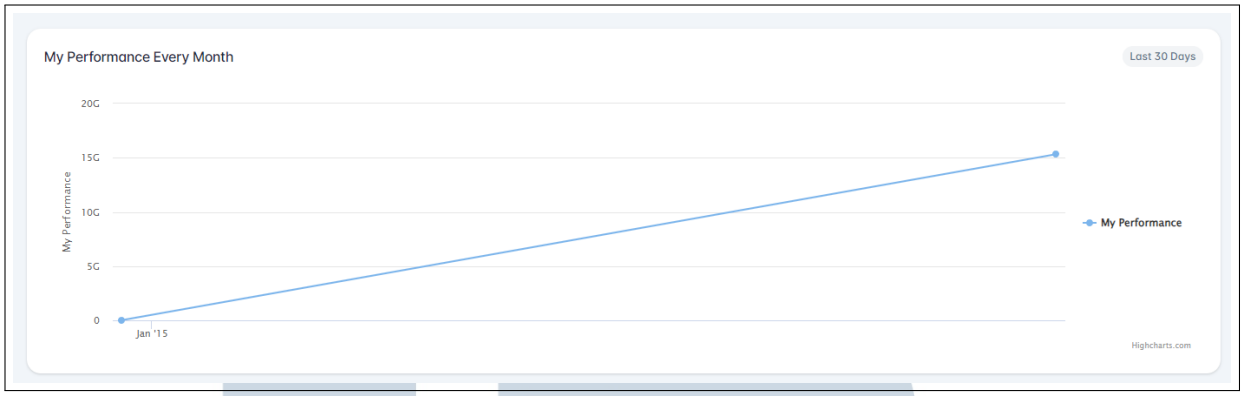
Kemudian hasil dari *data* yang telah didapatkan akan digunakan pada pembuatan tabel yang terdapat pada html yang ada. *Codingan* tersebut tertera pada gambar 3.7

```
<table
  mat-table
  [dataSource]="dataSourceDeals"
  class="min-w-0 overflow-y-visible"
>
  <ng-container matColumnDef="dealsName">
    <th mat-header-cell *matHeaderCellDef>Deals Name</th>
    <td mat-cell *matCellDef="let element">
      {{element.dealsName}}
    </td>
  </ng-container>

  <ng-container matColumnDef="amount">
    <th mat-header-cell *matHeaderCellDef>Amount</th>
    <td mat-cell *matCellDef="let element">
      {{element.amount}}
    </td>
  </ng-container>

  <ng-container matColumnDef="dueDate">
    <th mat-header-cell *matHeaderCellDef>Due Date</th>
    <td mat-cell *matCellDef="let element">
      {{element.dueDate | date: 'dd MMM'}}
    </td>
  </ng-container>
```

Gambar 3.7. Tampilan *codingan frontend* fitur *Tabel High Performance Deals*

3.4.2 Pembuatan fitur My Performance Every Month pada dashboard My Performance

Berikut adalah *fitur My Performance Every Month* yang telah penulis kerjakan pada gambar 3.8. Dalam fitur ini, terdapat *line charts* yang akan menampilkan akumulasi penjualan *sales* perbulannya.



Gambar 3.8. Tampilan fitur *My Performance by Month*

Alur dari pengerjaan fitur tersebut adalah sebagai berikut, *backend* akan dikerjakan terlebih dahulu dengan menggunakan *framework* Springboot, yang kemudian hasilnya akan di cek melalui Postman. Setelah *service* berhasil mengambil *data*, baru kemudian akan dikoneksikan dengan *frontend* dengan menggunakan *framework* Angular.

1. **Pembuatan Service Backend** Untuk membuat *service* backend, diperlukan *controller* yang akan menampung *service* dan *query* yang ada. Pada gambar 3.9 merupakan gambar dari *codingan controller* yang menampung *service* yang diperlukan.

```
@RolesAllowed({ "admin", "user" })
@GetMapping("/sales/myperformance")
public ResponseEntity<?> getMyPerformance()
{
    try {
        return ResponseHandler.setResponse("OK", HttpStatus.OK, dashboardService.getDashboardMyPerformance());
    } catch (Exception e) {
        String stacktrace = ExceptionUtils.getStackTrace(e);
        return ResponseHandler.setResponse(e.getMessage(), HttpStatus.BAD_REQUEST, null);
    }
}
```

Gambar 3.9. Tampilan *codingan service* fitur *My Performance Every Month*

Kemudian *controller* itu akan menampung *service* yang ada pada gambar 3.10. Dari *codingan* tersebut, maka *data* yang diperlukan bisa diambil sehingga dapat digunakan untuk bagian *frontend*.

```java
@Override
public DashboardMyPerformanceResponseDto getDashboardMyPerformance() {
    DashboardMyPerformanceResponseDto dashboardMyPerformanceResponse = new DashboardMyPerformanceResponseDto();

    String usrId = UserUtils.getUserId();

    List<BigDecimal> totalSalesEveryMonth = dealCardProductRepo.getTotalSalesEveryMonth(usrId);

    System.out.println("test"+dashboardMyPerformanceResponse);
    dashboardMyPerformanceResponse.setTotalSalesEveryMonth(totalSalesEveryMonth);
    return dashboardMyPerformanceResponse;
}
```

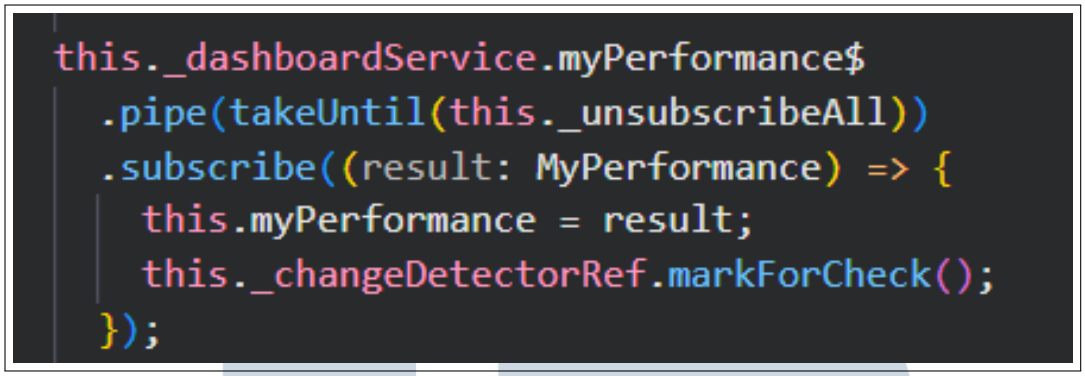
Gambar 3.10. Tampilan *codingan service* fitur *My Performance Every Month*

2. **Koneksi dengan frontend** Setelah *data* berhasil diambil, selanjutnya akan dilakukan koneksi dengan bagian *frontend* sehingga *data* bisa ditampilkan ke *website* CRM. Berikut adalah *codingan* dari bagian *frontend* pada gambar 3.11. Pada *codingan* tersebut telah digunakan API yang telah dibuat pada *backend* dan dimasukan kedalam sebuah *function* yang akan digunakan kedepannya.

```typescript
getMyPerformance(): Observable<MyPerformance> {
  return this._httpClient
    .get<MyPerformance>(
      `${environment.apiBaseUrl}/dashboard/sales/myperformance`
    )
    .pipe(
      tap((body: any) => {
        this._myPerformance.next(body.data);
      }),
      catchError(
        (error: HttpErrorResponse): Observable<any> => {
          return of(false);
        }
      )
    );
}
```

Gambar 3.11. Tampilan *codingan frontend* fitur *My Performance Every Month*

Selanjutnya API tersebut akan diambil *data* yang ada didalamnya dan digunakan untuk fitur *My Performance Every Month* dengan menggunakan *codingan* yang ada pada gambar 3.18.



```
this._dashboardService.myPerformance$
  .pipe(takeUntil(this._unsubscribeAll))
  .subscribe((result: MyPerformance) => {
    this.myPerformance = result;
    this._changeDetectorRef.markForCheck();
  });
```

Gambar 3.12. Tampilan *codingan frontend* fitur *My Performance Every Month*

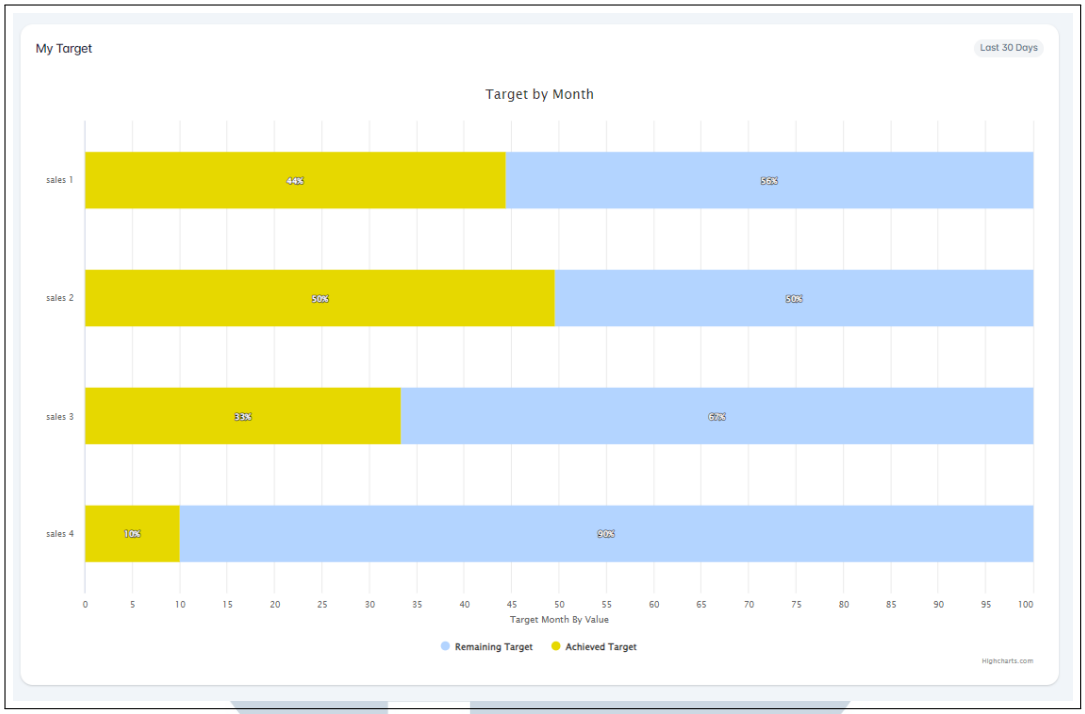
Kemudian hasil dari *data* yang telah didapatkan akan digunakan pada pembuatan tabel yang terdapat pada html yang ada. *Codingan* tersebut tertera pada gambar 3.19

```
myPerformanceOpt: Highcharts.Options = {
  title: {
    text: " ",
    align: "left",
  },

  subtitle: {
    align: "left",
  },

  yAxis: {
    title: {
      text: "My Performance",
    },
  },

  xAxis: {
    accessibility: {
      rangeDescription: "Range: January to February",
    },
    type: "datetime",
  },

  legend: {
    layout: "vertical",
    align: "right",
    verticalAlign: "middle",
  },
```

Gambar 3.13. Tampilan *codingan frontend* fitur *My Performance Every Month*

3.4.3 Pembuatan fitur Target by Month pada dashboard My Performance

Berikut adalah fitur *Target by Month* yang telah penulis kerjakan pada gambar 3.14. Dalam fitur ini, terdapat *bar charts* yang akan menampilkan *achieved targe*t dan juga *remaining target* yang telah dikerjakan *sales*.



Gambar 3.14. Tampilan fitur *Target by Month*

Alur dari pengerjaan fitur tersebut adalah sebagai berikut, *backend* akan dikerjakan terlebih dahulu dengan menggunakan *framework* Springboot, yang kemudian hasilnya akan di cek melalui Postman. Setelah *service* berhasil mengambil *data*, baru kemudian akan dikoneksikan dengan *frontend* dengan menggunakan *framework* Angular.

1. **Pembuatan Service Backend** Untuk membuat *service* backend, diperlukan *controller* yang akan menampung *service* dan *query* yang ada. Pada gambar 3.15 merupakan gambar dari *codingan controller* yang menampung *service* yang diperlukan.

```
@RolesAllowed({ "admin", "user" })
@GetMapping("/sales/mytarget")
public ResponseEntity<?> getMyTarget()
{
    try {
        return ResponseHandler.setResponse("OK", HttpStatus.OK, dashboardService.getDashboardMyTarget());
    } catch (Exception e) {
        String stacktrace = ExceptionUtils.getStackTrace(e);
        return ResponseHandler.setResponse(e.getMessage(), HttpStatus.BAD_REQUEST, null);
    }
}
```

Gambar 3.15. Tampilan *codingan service* fitur *Target by Month*

Kemudian *controller* itu akan menampung *service* yang ada pada gambar

3.16. Dari *codingan* tersebut, maka *data* yang diperlukan bisa diambil sehingga dapat digunakan untuk bagian *frontend*.

```
@Override
public List<DashboardMyTargetResponseDto> getDashboardMyTarget(){
    List<DashboardMyTargetResponseDto> dashboardMyTargetResponse = new ArrayList<DashboardMyTargetResponseDto>();

    String usrId = UserUtils.getUserId();

    List<KpiTargets> mytargetList = kpiTargetRepo.getMyTarget(usrId);
    List<KpiTargetDetails> mytargetList2 = kpiTargetDetailRepo.getMyTarget2(usrId);
    BigDecimal mytargetList4 = dealCardRepo.getMyTarget3(usrId);

    for (KpiTargetDetails mytarget: mytargetList2) {
        DashboardMyTargetResponseDto dashboardMyTarget = new DashboardMyTargetResponseDto();
/       dashboardMyTarget.setSalesName(mytarget.getKpiDescription());
        dashboardMyTarget.setTarget(mytarget.getTargetPoint());
/       dashboardMyTarget.setAchieved(mytarget.getTargetPoint());

        for(KpiTargets target1: mytargetList) {
            List<KpiTargetDetails> mytargetList3 = target1.getKpiTargetDetail();

            for(KpiTargetDetails target2: mytargetList3) {

                if(mytarget.getKpiDetailId() == target2.getKpiDetailId()) {
                    dashboardMyTarget.setSalesName(target1.getKpiName());
                }
            }
        }
        dashboardMyTarget.setAchieved(dealCardRepo.getMyTarget3(usrId));
        dashboardMyTargetResponse.add(dashboardMyTarget);
    }
    return dashboardMyTargetResponse;
}
```

Gambar 3.16. Tampilan *codingan service* fitur *Target by Month*

2. **Koneksi dengan frontend** Setelah *data* berhasil diambil, selanjutnya akan dilakukan koneksi dengan bagian *frontend* sehingga *data* bisa ditampilkan ke *website* CRM. Berikut adalah *codingan* dari bagian *frontend* pada gambar 3.17. Pada *codingan* tersebut telah digunakan API yang telah dibuat pada *backend* dan dimasukan kedalam sebuah *function* yang akan digunakan kedepannya.

```
getMyTarget(): Observable<MyTarget[]> {
  return this._httpClient
    .get<MyTarget[]>(`${environment.apiBaseUrl}/dashboard/sales/mytarget`)
    .pipe(
      tap((body: any) => {
        this._myTarget.next(body.data);
      }),
      catchError(
        (error: HttpErrorResponse): Observable<any> => {
          return of(false);
        }
      )
    );
}
```

Gambar 3.17. Tampilan *codingan frontend* fitur *Target by Month*

Selanjutnya API tersebut akan diambil *data* yang ada didalamnya dan digunakan untuk *Target by Month* dengan menggunakan *codingan* yang ada pada gambar 3.18.

```
this._dashboardService.myTarget$
  .pipe(takeUntil(this._unsubscribeAll))
  .subscribe((result: MyTarget[]) => {
    this.myTarget = result;
    this.dataSource = result;
    console.log("myTarget", result);
    this._changeDetectorRef.markForCheck();
  });
this._changeDetectorRef.markForCheck();
```

Gambar 3.18. Tampilan *codingan frontend* fitur *Target by Month*

Kemudian hasil dari *data* yang telah didapatkan akan digunakan pada pembuatan tabel yang terdapat pada html yang ada. *Codingan* tersebut tertera pada gambar 3.19

```
this.targetByMonthOpt.series = [
  {
    type: undefined,
    name: "Remaining Target",
    // data: [100000, 600000],
    data: this.myTarget.map((x) => x.target - x.achieved),
  },
  {
    type: undefined,
    name: "Achieved Target",
    // data: [80000, 590000],
    data: this.myTarget.map((x) => x.achieved),
  },
];
```

Gambar 3.19. Tampilan *codingan frontend* fitur *Target by Month*

3.5 Kendala dan Solusi yang Ditemukan

Kendala yang ditemukan adalah tidak diajarkannya *framework* Springboot untuk pembuatan *backend* pada Universitas Multimedia Nusantara. Hal ini membuat pengerjaan tugas sangat sulit dilakukan karena sangat minimnya pengetahuan mengenai *framework* tersebut dan kurangnya pengalaman. Selain itu hal ini terbilang sulit karena tidak mirip dengan *framework* lainnya yang diajarkan didalam perkuliahan. Solusi yang ditemukan adalah dengan meluangkan lebih banyak waktu untuk belajar secara mandiri, maupun bertanya ke koordinator ataupun ke teman-teman *intern* lainnya mengenai hal-hal yang belum dimengerti.