BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Kegiatan kerja magang dilakukan yang dilakukan di PT. Teknologi Kreasi Bangsa di bawah pengawasan Bapak Irfan Budipradipta sebagai supervisi dan *project manager* dalam proyek yang dikembangkan selama kerja magang. Proses membimbing dan mengawasi kerja magang dilakukan oleh supervisi. Request untuk penambahan fitur hanya diberikan oleh project manager.

3.2 Tugas yang Dilakukan

Tugas yang dilakukan selama kerja magang adalah mengembangkan aplikasi *mobile* berbasis android yang bernama Mitra BOS. Tugas-tugas yang dikerjakan adalah mengimplementasikan *design* yang sudah ada ke dalam aplikasi, mengembangkan logika pada aplikasi, mengambil dan memperbarui data dalam basis data melalui API, menambahkan fitur *location tracking* menggunakan Google Maps API, dan melakukan testing aplikasi. Kegiatan-kegiatan tersebut dikembangkan menggunakan bahasa pemrograman Java untuk kode utama dan XML untuk tampilan pada android.

Perangkat lunak yang digunakan selama kerja magang adalah Android Studio dan Postman. Android Studio digunakan untuk mengembangkan aplikasi android, sedangkan Postman digunakan untuk melakukan testing API yang dibutuhkan sebelum diimplementasikan ke dalam aplikasi. *Source code* aplikasi diberikan melalui *flash drive* dan tidak menggunakan aplikasi pengontrol versi apa pun (seperti Git) selama kerja magang berlangsung. Dengan demikian, *source code*

aplikasi yang dikembangkan disimpan secara lokal selama kerja magang berlangsung.

Pemberian dan pembagian tugas dilakukan oleh Bapak Irfan yang dibantu dengan aplikasi Trello. Selain itu, terdapat beberapa rapat yang dilakukan untuk melaporkan progres pengembangan aplikasi selama pengembangan berlangsung. Rapat dilakukan sebanyak seminggu sekali.

3.3 Uraian Pelaksaan Kerja Magang

Proses pelaksanaan kerja magang dibagi menjadi tiga bagian, yaitu proses pelaksanaan, kendala yang ditemukan, dan solusi atas kendala yang ditemukan.

3.3.1 Proses Pelaksanaan

Proses pengembangan aplikasi Mitra BOS yang berbasis android memerlukan perangkat lunak dan perangkat keras. Perangkat lunak yang digunakan dalam mengembangkan aplikasi tersebut adalah sebagai berikut.

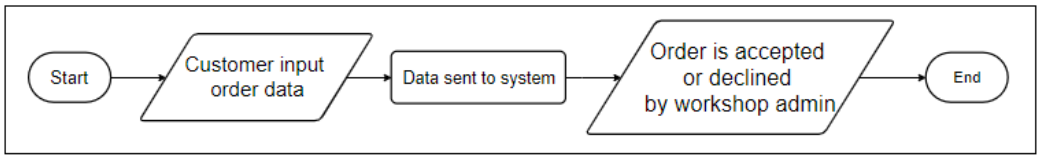
1. Android Studio versi 3.4.1
2. Postman versi 7.0
3. Sistem operasi Windows 10 Pro 1903 64 bit
4. Android emulator API 24

Perangkat keras yang digunakan selama pengembangan aplikasi menggunakan laptop Asus X550JX dengan spesifikasi sebagai berikut.

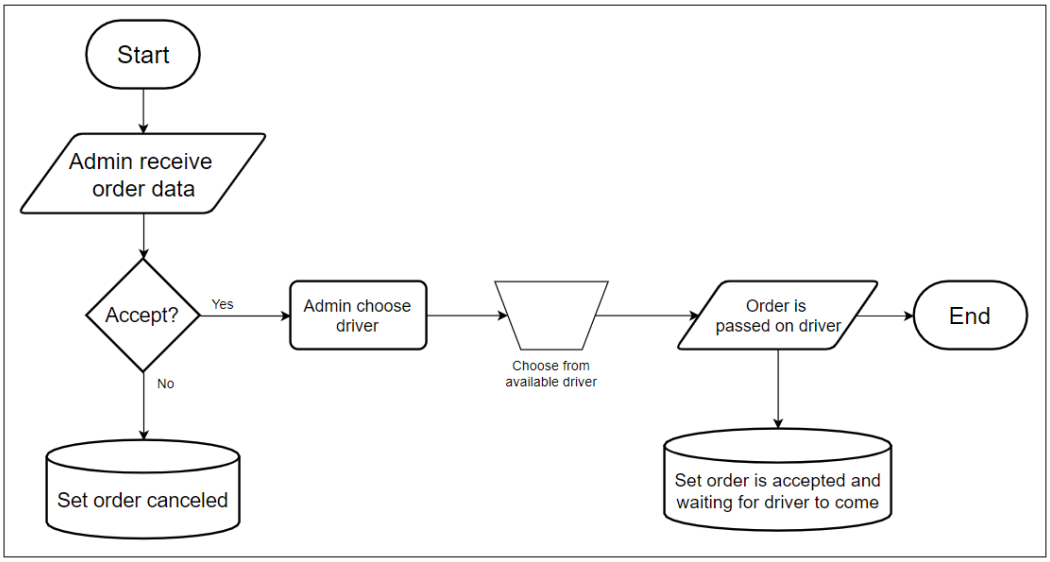
1. Prosesor Intel i7 4720HQ 2,6GHz
2. RAM 12GB 1600Mhz DDR3
3. NVIDIA Geforce GTX 950M

Perangkat keras lainnya adalah sebuah *smartphone* Xiaomi Redmi Note 4 Pro yang digunakan untuk melakukan testing pada perangkat keras asli.

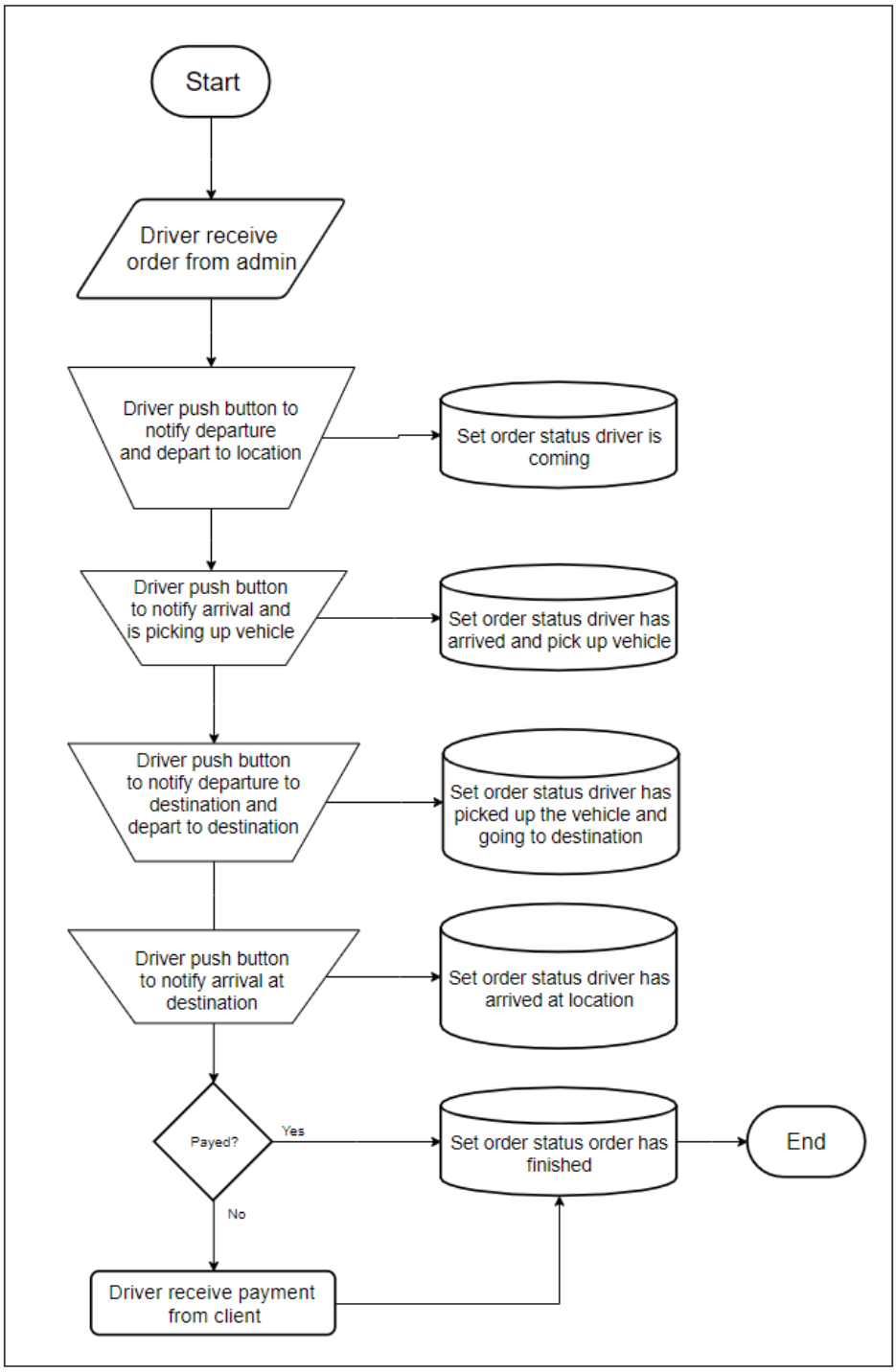
Proses pengerjaan selama pengembangan dibagi menjadi tiga bagian, yaitu mengimplementasikan susunan tampilan (*layout*) suatu halaman, memasukkan logika, dan melakukan testing. Berikut ini adalah alur diagram dan *data flow diagram* (DFD) aplikasi.



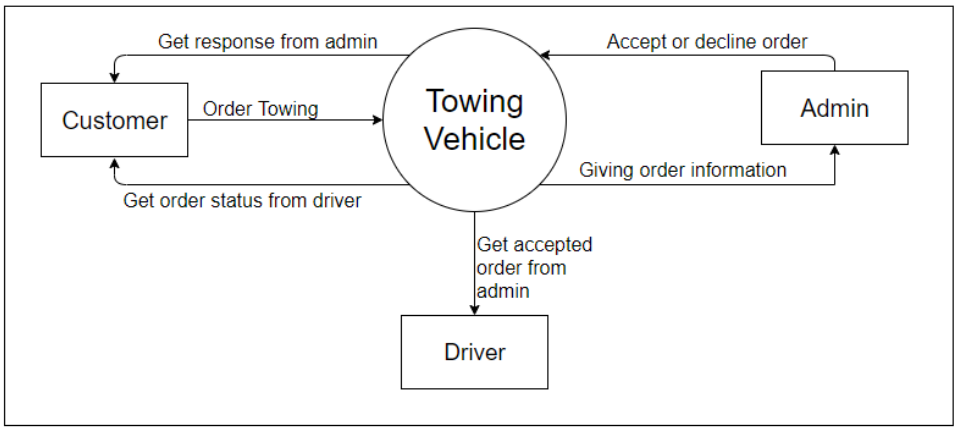
Gambar 3.1 Diagram alur untuk *customer*



Gambar 3.2 Diagram alur untuk sisi admin



Gambar 3.3 Diagram alur untuk sisi *driver*

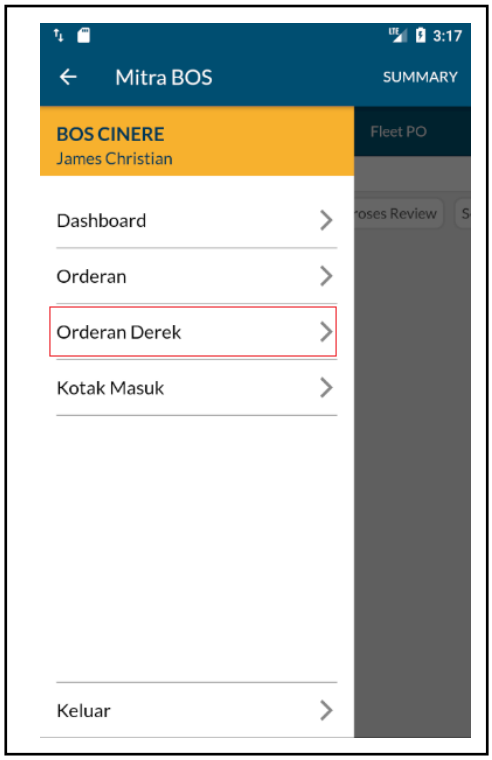


Gambar 3.4 Diagram alur data fitur Orderan Derek

A. Implementasi Layout

A.1 Sisi Admin

Menu Orderan Derek pada awalnya tidak ada sama sekali, sehingga harus ditambahkan.



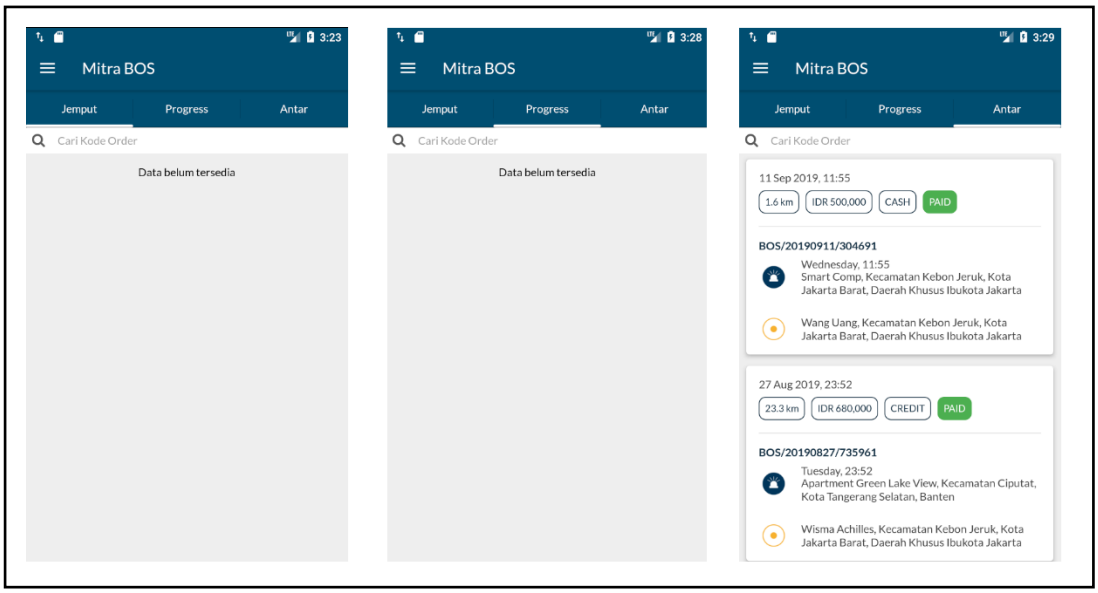
Gambar 3.5 Menu Orderan Derek

Penambahan menu Orderan Derek dilakukan *programmatically* melalui NavigationDrawerFragment.java yang meng-handle logika dari fragment ini. Berikut adalah potongan *code* yang digunakan untuk menambahkan menu Orderan Derek.

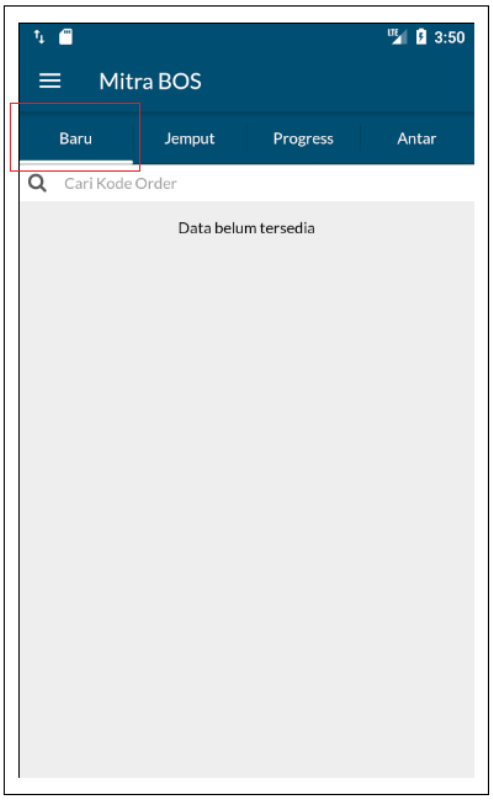
```
public static List<NavigationMenuItem> getWorkshopNavigationMenuList(Context mContext) {
    List<NavigationMenuItem> navigationMenuItemList = new ArrayList<>();
    navigationMenuItemList.add(
            new NavigationMenuItem(NavigationMenuItem.ID_DASHBOARD,
            mContext.getString(R.string.navigation_menu_dashboard_title)));
    navigationMenuItemList.add(
            new NavigationMenuItem(NavigationMenuItem.ID_UPCOMING_ORDER,
                mContext.getString(R.string.navigation_menu_order_title)));
    navigationMenuItemList.add(
            new NavigationMenuItem(NavigationMenuItem.ID_ORDER_DEREK,
            mContext.getString(R.string.navigation_menu_derek_title)));
    navigationMenuItemList.add(
            new NavigationMenuItem(NavigationMenuItem.ID_INBOX,
            mContext.getString(R.string.navigation_menu_inbox_title)));

    return navigationMenuItemList;
}
```

Gambar 3.6 *Code* untuk menambahkan menu Orderan Derek

Halaman selanjutnya adalah halaman Orderan Derek itu sendiri. Berikut ini adalah tampilan yang diajukan oleh tim design.

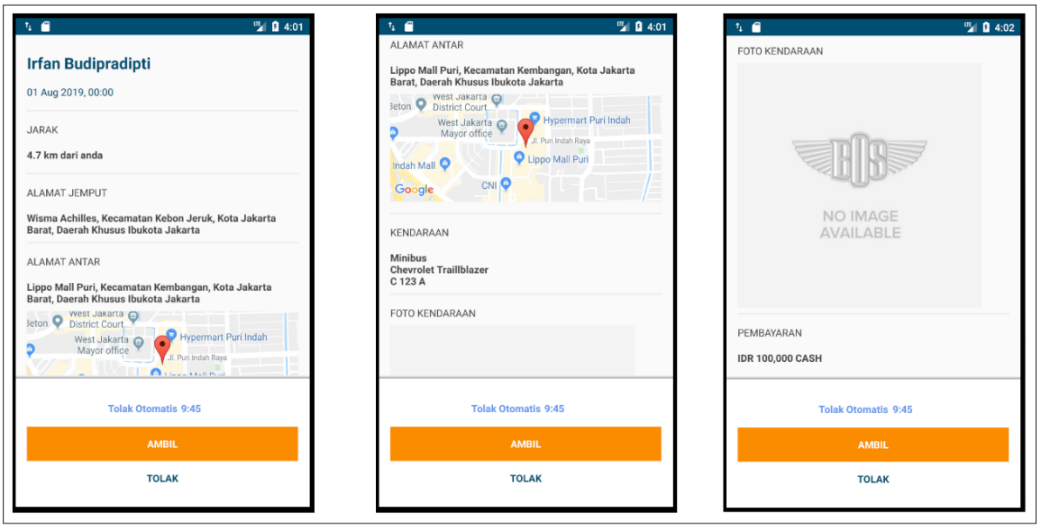


Gambar 3.7 Design halaman Orderan Derek sisi driver



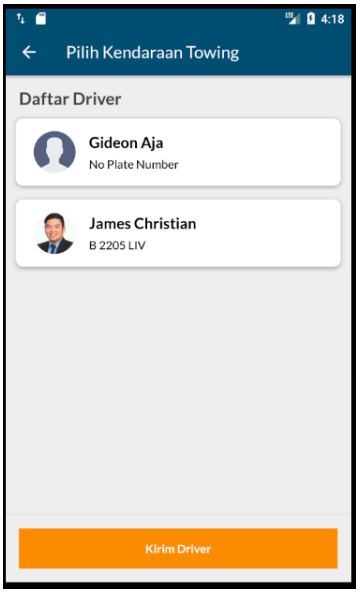
Gambar 3.8 Design halaman Orderan Derek sisi admin

Kedua tampilan tidak jauh berbeda, sehingga *code* digabungkan menjadi satu dan diberi logika juga. Kedua tampilan ini juga dibuat *programatically* dan dijelaskan cara pembuatannya pada subbab selanjutnya. Halaman selanjutnya adalah halaman detail saat admin menerima pesanan dari pelanggan (TowingNewDetailActivity.java). Halaman ini hanya menampilkan detail pesanan dari pelanggan. Driver tidak memiliki halaman ini. Berikut ini adalah tampilan dari halaman detail dari sisi admin.



Gambar 3.9 Halaman Detail Order dari sisi admin

Tampilan pada halaman detail ini diatur susunannya menggunakan XML (tidak ditunjukkan pada laporan ini). Halaman selanjutnya adalah halaman admin untuk menugaskan driver pada order tersebut. Halaman tersebut di-handle oleh file TowingNewChooseDriverActivity.java dan susunan pada halaman tersebut dibuat menggunakan XML (tidak ditunjukkan). Berikut ini adalah tampilan saat admin memilih driver.

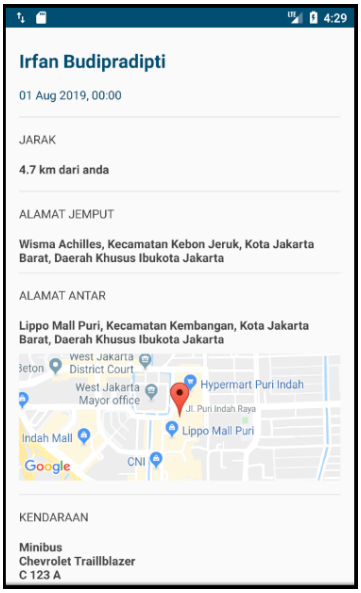


Gambar 3.10 Tampilan admin memilih driver

Halaman selanjutnya hanya berupa halaman konfirmasi yang hanya memberitahukan bahwa order telah disampaikan kepada driver. Tampilan pada halaman ini disusun menggunakan XML (tidak ditunjukkan). Berikut ini adalah tampilan halaman tersebut.

Gambar 3.11 Tampilan order telah diteruskan kepada driver

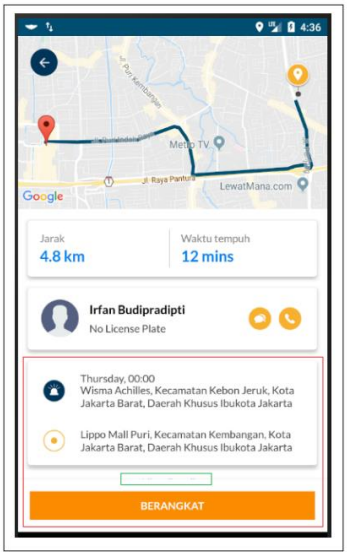
Tampilan terakhir dari sisi admin adalah tampilan detail, namun tidak ada timer yang berjalan, hanya detail pesanan yang sedang berjalan. Berikut ini adalah tampilan halaman tersebut.



Gambar 3.12 Tampilan detail tidak ada timer

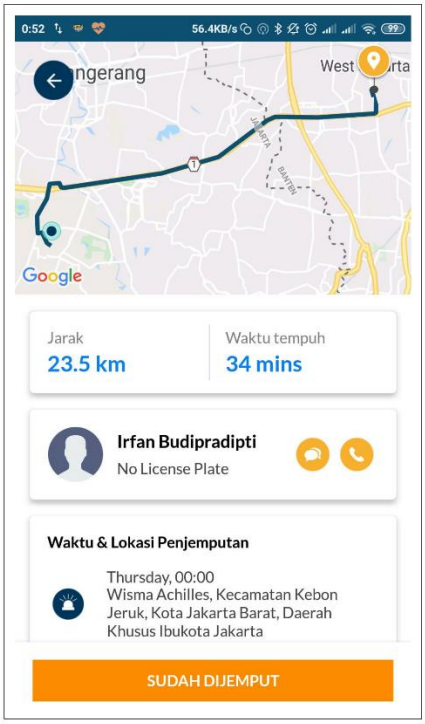
A.2 Sisi Driver

Order baru yang diterima driver langsung ada pada tab menu jemput. Pada halaman ini, *map* pada halaman tersebut menampilkan jarak dan waktu yang harus ditempuh driver untuk menderek mobil dari tempat awal hingga tujuan. Berikut ini adalah tampilan order baru dari sisi driver.



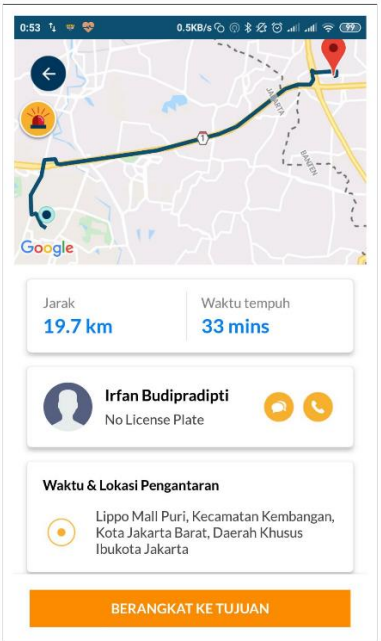
Gambar 3.13 Order masuk untuk driver

Halaman-halaman selanjutnya hanya mengganti isi dari kotak merah dan *map* yang ada di atas halaman sesuai dengan status pesanan yang sedang berjalan. Kotak hijau yang terdapat di dalam kotak merah adalah teks “Lihat Detail >” yang dapat digunakan *driver* untuk melihat detail pesanan. Tampilan halaman tersebut dapat dilihat pada Gambar 3.12. Logika pergantian isi dari kotak merah akan dibahas di subbab selanjutnya. Halaman selanjutnya, ketika *driver* berangkat ke tempat pelanggan, akan mengganti isi kotak merah menjadi seperti berikut.



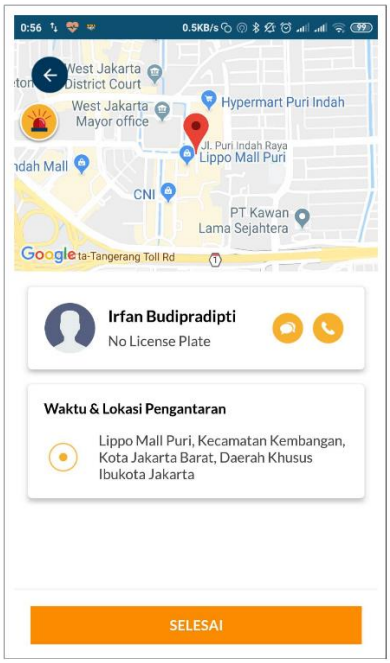
Gambar 3.14 Halaman *driver* sudah menjemput pelanggan

Setelah *driver* menjemput pelanggan dan memroses kendaraan yang diangkut, *driver* akan diarahkan ke halaman selanjutnya setelah menekan tombol pada “SUDAH DIJEMPUT”. Berikut ini adalah tampilan halaman selanjutnya.



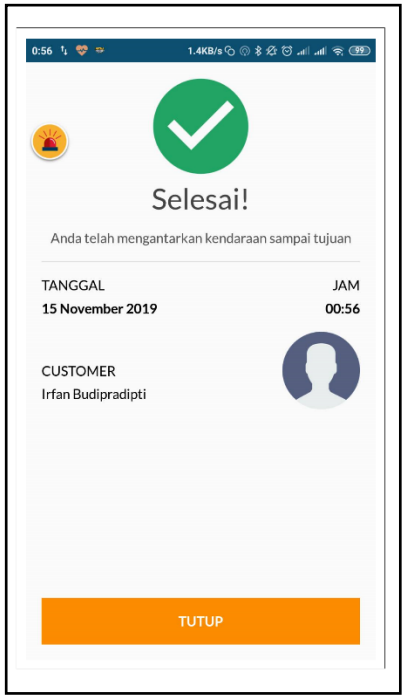
Gambar 3.15 Halaman *driver* akan berangkat ke tujuan

Tombol tersebut akan memberitahu pelanggan bahwa kendaraan yang diangkut akan dan sedang diantar ke tujuan. Ketika tombol tersebut ditekan, halaman akan berganti menjadi seperti berikut.



Gambar 3.16 Halaman *driver* selesai mengantarkan kendaraan

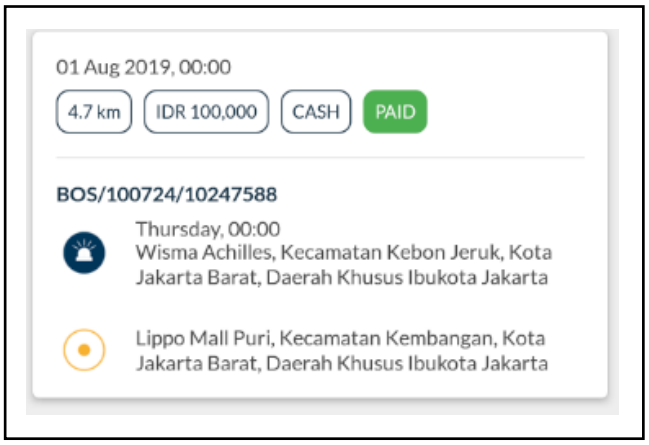
Driver mengakhiri order dengan menekan tombol “SELESAI” dan diakhiri dengan halaman berikut ini.



Gambar 3.17 Halaman *driver* menyelesaikan order

A.3 List Order

Setiap tab menu memiliki list order masing-masing. List order merupakan daftar order yang dibuat berupa card. Contoh card order dapat dilihat pada gambar berikut ini.



Gambar 3.18 Card order

B. Implementasi Logika

B.1 Status Order dan Pembayaran

Setiap order yang diterima dan yang sedang dijalankan memiliki statusnya masing-masing. Berikut adalah daftar status tersebut.

a. NEW: Order baru dari pelanggan
b. WAITING_ASSIGN_DRIVER: Order baru diterima oleh admin dan sedang menunggu penugasan ke *driver*
c. DRIVER_ASSIGNED: Order telah disampaikan ke *driver*
d. DRIVER_PICK_UP: *Driver* berangkat ke lokasi pelanggan
e. DRIVER_ARRIVED_AT_ORIGIN: *Driver* telah sampai ke lokasi pelanggan dan memulai proses penderekan mobil
f. DRIVER_GO_TO_DESTINATION: *Driver* memulai perjalanan untuk mengantar kendaraan yang diderek ke tujuan.
g. DRIVER_ARRIVED_AT_DESTINATION: *Driver* telah sampai ke tujuan
h. FINISHED: Order telah selesai dikerjakan *driver* dengan pembayaran telah terkonfirmasi
i. CANCELLED_BY_MITRA: Order dibatalkan mitra (pemilik bengkel) atau dibatalkan otomatis
j. CANCELLED_BY_CUSTOMER: Order dibatalkan oleh pelanggan

Selain status order, terdapat juga status pembayaran, yaitu:

a. PAID: Pelanggan telah melakukan pembayaran
b. UNPAID: Pelanggan belum melakukan pembayaran

B.2 Halaman Utama Orderan Derek

Halaman utama Orderan Derek dapat dilihat pada Gambar 3.7 dan 3.8. Pembuatan halaman ini dilihat pada potongan *code* di bawah ini.

```
try {
    List<FragmentPagerItem> pagerItemList = new ArrayList<>();
    if(mMitraUser.level == null || mMitraUser.level.equals(MMitraUser.MITRA_USER_LEVEL_ADMIN))
    {
        pagerItemList.add(new FragmentPagerItem(
                getString(R.string.tab_towing_title_new),
                TowingNewFragment.getFragment(MOrderTowing.ORDER_TOWING_STATUS_NEW))
        );
    }
    pagerItemList.add(new FragmentPagerItem(
            getString(R.string.tab_towing_title_pickup),
            TowingNewFragment.getFragment(MOrderTowing.ORDER_TOWING_STATUS_PICK_UP))
    );
    pagerItemList.add(new FragmentPagerItem(
            getString(R.string.tab_towing_title_progress),
            TowingNewFragment.getFragment(MOrderTowing.ORDER_TOWING_STATUS_PROGRESS))
    );
    pagerItemList.add(new FragmentPagerItem(
            getString(R.string.tab_towing_title_antar),
            TowingNewFragment.getFragment(MOrderTowing.ORDER_TOWING_STATUS_ARRIVED))
    );

    CustomFragmentStatePagerAdapter pagerAdapter = new CustomFragmentStatePagerAdapter(
            mContext, fragmentManager, pagerItemList
    );
    if(mMitraUser.level == null || mMitraUser.level.equals(MMitraUser.MITRA_USER_LEVEL_ADMIN))
    {
        viewPager.setOffscreenPageLimit(4);
    }
    else{
        viewPager.setOffscreenPageLimit(3);
    }

    viewPager.setAdapter(pagerAdapter);
    tabLayout.setViewPager(viewPager);
} catch(Exception e) { e.printStackTrace(); }
```

Gambar 3.19 *Code* yang digunakan untuk membuat halaman Orderan Derek

Bagian *code* yang diberi kotak berwarna merah merupakan logika yang dibutuhkan untuk membedakan antara halaman admin dan *driver*, sedangkan bagian *code* yang diberi kotak berwarna biru ada pada kedua sisi akun (*driver* maupun admin). *Code* yang ada pada kotak merah hanya akan berjalan ketika pelanggan yang sedang login adalah seorang admin. Kotak merah pertama (atas) digunakan untuk membuat tab menu Baru, seperti yang ditunjukkan pada Gambar 3.8, sedangkan kotak merah kedua (bawah) digunakan untuk membatasi jumlah tab pada menu, empat pada admin dan tiga pada *driver*.

B.3 Halaman Detail Order Admin

Halaman detail order admin, seperti yang ditunjukkan pada Gambar 3.9 hingga 3.12, memiliki logikanya masing-maisng.

B.3.1 Detail Order

Halaman detail order dibuat dalam file TowingNewDetailActivity.java. Logika pada halaman ini terdapat pada pengambilan detail order, timer, tombol "AMBIL", dan tombol "Tolak". Proses pengambilan data dapat dilihat pada potongan *code* di bawah ini.

```
public void getData(){
    try {
        H.getService(mContext).getOrderTowingDetail(towingId).enqueue(
                new CustomCallback<MOrderTowing>(
                        mContext,
                        new DefaultCallback<MitraBosResponse<MOrderTowing>>() {
            @Override
            public void onResponse(
                    Call<MitraBosResponse<MOrderTowing>> call,
                    Response<MitraBosResponse<MOrderTowing>> response,
                    MitraBosResponse<MOrderTowing> mOrderTowingMitraBosResponse,
                    DefaultResponse<MitraBosResponse<MOrderTowing>> defaultResponse
            ) {
                if(defaultResponse.isOk()){
                    mOrderTowing = mOrderTowingMitraBosResponse.result;
                    setupView();

                    try {
                        LatLng ny = new LatLng(
                                mOrderTowing.destinationLatitude,
                                mOrderTowing.destinationLongitude);
                        gmap.moveCamera(CameraUpdateFactory.newLatLng(ny));
                        gmap.addMarker(new MarkerOptions().position(ny));
                        gmap.animateCamera(CameraUpdateFactory.newLatLngZoom(ny, v: 15.0f));
                        gmap.getUiSettings().setAllGesturesEnabled(true);
                    } catch (Exception e) { e.printStackTrace(); }
                }
            }

            @Override
            public void onFailure(
                    Call<MitraBosResponse<MOrderTowing>> call,
                    DefaultError<MitraBosResponse<MOrderTowing>> defaultError) {}
        }, showDialog: true));
    } catch(Exception e) { e.printStackTrace(); }
}
```

Gambar 3.20 Fungsi getData

Pengambilan data dibantu oleh fungsi getOrderTowingDetail() yang ada pada *file* H.java di *service*. Data yang terambil langsung dimasukkan ke dalam

objek mOrderTowing. Selanjutnya data ditampilkan pada setupView() dan koordinat tujuan order dimasukkan ke dalam Google Map untuk ditunjukkan. Proses perhitungan *timer* terdapat dalam fungsi setupView(). Berikut ini adalah potongan *code* untuk *timer*.

```
Date currentDate = Calendar.getInstance().getTime();
long difference = mOrderTowing.expiredDatetime.getTime() - currentDate.getTime();

if(difference > 0){
    towingDetailTimer.setText(mOrderTowing.expiredDatetime.toString());
    if(mOrderTowing.status.equals(MOrderTowing.ORDER_TOWING_STATUS_NEW)){
        timerCountDown = true;
        setTimer(difference);
    }
}
else{
    linearAction.setVisibility(View.GONE);
    towingDetailTimer.setText(R.string.towing_new_automated_canceled);
    adminDeclineOrder(towingId);
}
```

Gambar 3.21 Logika timer

Variabel *difference* menghitung selisih tanggal kadaluarsa order dengan waktu saat order dibuka. Jika *difference* lebih besar dari 0 (nol), maka order dinyatakan masih valid. Jika *difference* tidak lebih besar dari 0 (nol), maka *timer* dihentikan dan order langsung dibatalkan menggunakan fungsi adminDeclineOrder(). Berikut ini adalah *code* pada fungsi adminDeclineOrder().

```
public void adminDeclineOrder(int towingId){
    H.getService(mContext).adminDeclineOrder(towingId).enqueue(new CustomCallback<MOrderTowing>(
        mContext, new DefaultCallback<MitraBosResponse<MOrderTowing>>() {
            @Override
            public void onResponse(
                    Call<MitraBosResponse<MOrderTowing>> call,
                    Response<MitraBosResponse<MOrderTowing>> response,
                    MitraBosResponse<MOrderTowing> mOrderTowingMitraBosResponse,
                    DefaultResponse<MitraBosResponse<MOrderTowing>> defaultResponse
            ) {}

            @Override
            public void onFailure(
                    Call<MitraBosResponse<MOrderTowing>> call,
                    DefaultError<MitraBosResponse<MOrderTowing>> defaultError
            ) {}
        })
    );
}
```

Gambar 3.22 Fungsi adminDeclineOrder

Fungsi, yang juga digunakan oleh tombol “Tolak”, digunakan untuk membatalkan order dengan mengirimkan ID order dan mengubah status pesanan dari NEW menjadi CANCELLED_BY_MITRA. Pengiriman ID order dan penggantian status dibantu dengan menggunakan API. Logika pada tombol “AMBIL” diatasi oleh fungsi adminAcceptOrder(). Berikut ini adalah potongan *code* fungsi adminAcceptOrder().

```
public void adminAcceptOrder(int towingId){
    H.getService(mContext).adminAcceptOrder(towingId).enqueue(
        new CustomCallback<MOrderTowing>(mContext,
        new DefaultCallback<MitraBosResponse<MOrderTowing>>() {
            @Override
            public void onResponse(
                    Call<MitraBosResponse<MOrderTowing>> call,
                    Response<MitraBosResponse<MOrderTowing>> response,
                    MitraBosResponse<MOrderTowing> mOrderTowingMitraBosResponse,
                    DefaultResponse<MitraBosResponse<MOrderTowing>> defaultResponse
            ) {}

            @Override
            public void onFailure(
                    Call<MitraBosResponse<MOrderTowing>> call,
                    DefaultError<MitraBosResponse<MOrderTowing>> defaultError
            ) {}
        })
    );
}
```

Gambar 3.23 Fungsi adminAcceptOrder

Fungsi ini digunakan untuk mengambil atau menerima pesanan dengan mengirimkan ID order dan mengubah status pesanan dari NEW menjadi WATING_ASSIGN_DRIVER. Admin yang telah menerima pesanan akan diarahkan ke halaman pemilihan *driver* (Gambar 3.10).

B.3.2 Halaman Assign Driver

Halaman ini merupakan halaman lanjutan dari subbab sebelumnya. Halaman ini digunakan admin untuk memilih *driver* untuk ditugaskan. Proses yang terjadi di halaman ini adalah pengambilan daftar *driver* yang tersedia dan penugasan *driver* yang dipilih. Proses ini dilakukan di

TowingNewChooseDriverActivity.java. Berikut ini adalah potongan *code* proses pengambilan data.

```
public void getData(){
    H.getService(mContext).getDriverList().enqueue(
            new CustomCallback<List<MMitraUser>>(
                    mContext,
                    new DefaultCallback<MitraBosResponse<List<MMitraUser>>>() {
        @Override
        public void onResponse(
                Call<MitraBosResponse<List<MMitraUser>>> call,
                Response<MitraBosResponse<List<MMitraUser>>> response,
                MitraBosResponse<List<MMitraUser>> listMitraBosResponse,
                DefaultResponse<MitraBosResponse<List<MMitraUser>>> defaultResponse) {
            if(defaultResponse.isOk()){
                driverList.addAll(listMitraBosResponse.result);
                recyclerDriverList.getAdapter().notifyDataSetChanged();

                swipeRefreshLayout.setRefreshing(false);
            }
        }

        @Override
        public void onFailure(
                Call<MitraBosResponse<List<MMitraUser>>> call,
                DefaultError<MitraBosResponse<List<MMitraUser>>> defaultError
        ) {}
    }));
}
```

Gambar 3.24 Fungsi getData

Data *driver* yang diterima, dimasukkan ke dalam objek driverList, lalu ditampilkan ke halaman. Halaman ini menyediakan terlebih dahulu tampilannya yang dilanjutkan dengan mengambil data. Oleh karena itu, penampilan data dilakukan setelah menerima data dengan memberitahu tampilan bahwa ada data yang berubah menggunakan notifyDataSetChanged(). Tombol “KIRIM DRIVER” digunakan untuk menugaskan *driver* yang dipilih. Berikut ini adalah *code* yang digunakan untuk tombol tersebut.

```java
public void sendDriver(){
    H.getService(mContext).assignDriver(towingId, DRIVER_ID).enqueue(
        new CustomCallback<MOrderTowing>(
            mContext,
            new DefaultCallback<MitraBosResponse<MOrderTowing>>() {
                @Override
                public void onResponse(
                        Call<MitraBosResponse<MOrderTowing>> call,
                        Response<MitraBosResponse<MOrderTowing>> response,
                        MitraBosResponse<MOrderTowing> mOrderTowingMitraBosResponse,
                        DefaultResponse<MitraBosResponse<MOrderTowing>> defaultResponse) {
                    if(defaultResponse.isOk()){
                        Toast.makeText(mContext, text: "Driver Sent", Toast.LENGTH_SHORT).show();
                        Intent intent = new Intent(
                                packageContext: TowingNewChooseDriverActivity.this,
                                TowingDriverSentActivity.class
                        );
                        intent.putExtra( name: "DRIVER_NAME", DRIVER_NAME);
                        intent.putExtra( name: "DRIVER_IMAGE_PATH", DRIVER_IMAGE_PATH);
                        startActivityForResult(intent, REQUEST_DRIVER_SENT);
                        finish();
                    }
                }

                @Override
                public void onFailure(
                        Call<MitraBosResponse<MOrderTowing>> call,
                        DefaultError<MitraBosResponse<MOrderTowing>> defaultError
                ) {}
            }
        )
    );
}
```

Gambar 3.25 Fungsi sendDriver

Fungsi ini mengirimkan ID order dan *driver* ke server melalui API. Tombol ini juga mengubah status pesanan dari WAITING_ASSIGN_DRIVER menjadi DRIVER_ASSINGED. Setelah tombol ini ditekan, halaman akan berpindah ke Gambar 3.11.

B.4 Halaman Detail Order Driver

Code pada Halaman ini dibuat pada TowingDriverDetailActivity.java. Halaman ini dibedakan berdasarkan status order yang sedang berjalan. Detail order diambil dari server saat dibuka, lalu data yang didapat ditampilkan ke dalam komponen pada halaman. Komponen pada halaman ini dibagi menjadi dua, yaitu komponen primer dan sekunder.

Komponen primer pada halaman terdiri dari maps (peta), card jarak dan waktu tempuh, dan card kontak pelanggan. Komponen ini disebut sebagai komponen primer karena selalu ada di setiap status pesanan, kecuali ketika pesanan selesai. Disisi lain, komponen sekunder berubah sesuai dengan status pesanan. Komponen sekunder pada halaman terdiri dari card detail perjalanan dan tombol untuk mengganti status.

B.4.1 Inisialisasi Data dan Komponen Maps

Pada membuka halaman, persiapan pertama yang dilakukan adalah mempersiapkan Google Maps. Jika dilihat pada gambar di bawah ini, tidak ada fungsi getData() di dalam fungsi onCreate().

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_towing_driver_detail);
    ButterKnife.bind( target: this);

    askForSystemOverlayPermission();

    isDistanceCalculated = false;

    Bundle bundle = getIntent().getExtras();
    towingId = bundle.getInt(BUNDLE_KEY_TOWING_NEW_OBBJECT);

    fabBack.setOnClickListener((v) -> { finish(); });

    imgBack.bringToFront();
    imgBack.setOnClickListener((v) -> {
            fragmentManager.popBackStack();
            finish();
    });

    Bundle mapViewBundle = null;
    mapView.onCreate(mapViewBundle);
    mapView.getMapAsync( onMapReadyCallback: this);
}
```

Gambar 3.26 Fungsi onCreate

Pada membuka halaman, persiapan pertama yang dilakukan adalah mempersiapkan Google Maps. Jika dilihat pada gambar di bawah ini, tidak ada fungsi getData() di dalam fungsi onCreate().

Code yang ada di dalam kotak berwarna hijau digunakan untuk menginisialisasi halaman. Code yang ada di dalam kotak berwarna merah digunakan untuk menginisialisasi Google Maps.

```
@Override
public void onMapReady(GoogleMap googleMap) {
    gmap = googleMap;
    gmap.setMinZoomPreference(10);
    try {
        double lat = -5.1817177;
        double lng = 102.7670478;
        LatLng ny = new LatLng(lat, lng);
        gmap.moveCamera(CameraUpdateFactory.newLatLng(ny));
    } catch (Exception e) { e.printStackTrace(); }
    getData();
}
```

Gambar 3.27 Fungsi onMapReady untuk Google Maps

Berdasarkan gambar di atas, fungsi getData() dipanggil setelah Google Map selesai diinisialisasi.

```java
public void getData() {
  try {
    H.getService(mContext).getOrderTowingDetail(towingId).enqueue(
      new CustomCallback<MOrderTowing>(mContext,
        new DefaultCallback<MitraBosResponse<MOrderTowing>>() {
        @Override
        public void onResponse(Call<MitraBosResponse<MOrderTowing>> call,
          Response<MitraBosResponse<MOrderTowing>> response,
          MitraBosResponse<MOrderTowing> mOrderTowingMitraBosResponse,
          DefaultResponse<MitraBosResponse<MOrderTowing>> defaultResponse) {
          if (defaultResponse.isOk()) {
            mOrderTowing = mOrderTowingMitraBosResponse.result;

            originLat = mOrderTowing.originLatitude;
            originLng = mOrderTowing.originLongitude;
            destLat = mOrderTowing.destinationLatitude;
            destLng = mOrderTowing.destinationLongitude;

            if(mOrderTowing.photoPath != null){
              String url = mOrderTowing.photoPath;
              Glide.with(mContext).load(url).asBitmap().centerCrop().into(
                new BitmapImageViewTarget(customerProfPic) {
                  @Override
                  protected void setResource(Bitmap resource) {
                    RoundedBitmapDrawable circularBitmapDrawable =
                      RoundedBitmapDrawableFactory.create(mContext.getResources(), resource);
                    circularBitmapDrawable.setCircular(true);
                    customerProfPic.setImageDrawable(circularBitmapDrawable);
                  }
              });
            } else{ customerProfPic.setImageResource(R.drawable.user); }
            if(mOrderTowing.status.equals(MOrderTowing.ORDER_TOWING_STATUS_DRIVER_ASSIGNED)){
              getDistance(originLat, originLng, destLat, destLng);
              try {
                LatLng origin = new LatLng(originLat, originLng);
                LatLng dest = new LatLng(destLat, destLng);

                String url = getDirectionsUrl(origin, dest);
                DownloadTask downloadTask = new DownloadTask();
                downloadTask.execute(url);

                LatLngBounds.Builder builder = LatLngBounds.builder();
                builder.include(origin);
                builder.include(dest);
                LatLngBounds bounds = builder.build();

                CameraUpdate cu = CameraUpdateFactory.newLatLngBounds(bounds, i: 100);

                gmap.moveCamera(cu);
                gmap.animateCamera(cu);
                gmap.addMarker(new MarkerOptions()
                        .icon(BitmapDescriptorFactory.fromResource(R.drawable.point))
                        .position(origin)
                );
                gmap.addMarker(new MarkerOptions().position(dest));
              } catch (Exception e) { e.printStackTrace(); }
            } else { getCurrentLocation(); }
            setupView();
          }
        }
        @Override
        public void onFailure(Call<MitraBosResponse<MOrderTowing>> call,
          DefaultError<MitraBosResponse<MOrderTowing>> defaultError) {}
      }, showDialog: true));
  } catch (Exception e) { e.printStackTrace(); }
}
```

Gambar 3.28 Fungsi getData

Data yang diterima dimasukkan ke dalam objek mOrderTowing. Potongan *code* yang ada di dalam kotak berwarna hijau digunakan untuk menetapkan nilai *longitude* dan *latitude* dari *origin* (lokasi *driver* saat itu atau lokasi awal pelanggan) dan *destination* (lokasi awal pelanggan atau lokasi tujuan pelanggan). Perhitungan jarak dan waktu tempuh juga dilakukan di dalamnya dibantu dengan fungsi getDistance(). Potongan *code* yang ada di dalam kota berwarna biru digunakan untuk mengambil gambar dari server dan mengatur gambar agar terlihat bulat. Fungsi setupView() yang di dalam kotak berwarna merah mengatur logika tampilan halaman tersebut.

B.4.2 Komponen Jarak dan Waktu Tempuh dan Komponen Kontak

Kedua komponen ini diatur di dalam fungsi getDistance(). Berikut ini adalah potongan code untuk jarak dan waktu tempuh.

```
public void getDistance(double lat1, double lng1, double lat2, double lng2){
  H.getService(mContext).getDistance(lat1, lng1, lat2, lng2).enqueue(
    new CustomCallback<>(mContext, new DefaultCallback<MitraBosResponse<MMapsDistance>>() {
      @Override
      public void onResponse(Call<MitraBosResponse<MMapsDistance>> call,
        Response<MitraBosResponse<MMapsDistance>> response,
        MitraBosResponse<MMapsDistance> mOrderTowingMitraBosResponse,
        DefaultResponse<MitraBosResponse<MMapsDistance>> defaultResponse) {
        if(defaultResponse.isOk()){
          if(mOrderTowingMitraBosResponse.result != null){
            mMapsDistance = mOrderTowingMitraBosResponse.result;

            driverDistance.setVisibility(View.VISIBLE);
            driverDurationTime.setVisibility(View.VISIBLE);
            progressJarak.setVisibility(View.GONE);
            progressWaktu.setVisibility(View.GONE);

            driverDistance.setText(mMapsDistance.distance);
            driverDurationTime.setText(mMapsDistance.duration);
          }
          else{
            driverDistance.setText("N/A");
            driverDurationTime.setText("N/A");
          }
        }
      }

      @Override
      public void onFailure(Call<MitraBosResponse<MMapsDistance>> call,
        DefaultError<MitraBosResponse<MMapsDistance>> defaultError) {}
    })
  );
}
```

Gambar 3.29 Fungsi getDistance

Jarak dan waktu tempuh yang didapat langsung ditampilkan di halaman detail. Komponen kontak diatur di fungsi setupView(). Berikut ini adalah potongan code yang mengatur komponen kontak.

```
customerName.setText(mOrderTowing.customer.name);
customerPlateNumber.setText(
        mOrderTowing.customerGarage.licensePlate == null ?
                "No License Plate" :
                mOrderTowing.customerGarage.licensePlate
);
```

Gambar 3.30 Komponen kontak nama dan nomor pelat kendaraan

```
callBtn.setOnClickListener((v) → {
    if( ContextCompat.checkSelfPermission( context: TowingDriverDetailActivity.this,
            Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED ) {
        ActivityCompat.requestPermissions(
                activity: TowingDriverDetailActivity.this,
                new String[]{Manifest.permission.CALL_PHONE},  requestCode: 333
        );
    }
    else{
        Intent callIntent = new Intent(Intent.ACTION_CALL);
        callIntent.setData(Uri.parse("tel:" + mOrderTowing.customer.phone));
        startActivity(callIntent);
    }
});

messageBtn.setOnClickListener((v) → {
        Intent n = new Intent(Intent.ACTION_VIEW);
        n.setType("vnd.android-dir/mms-sms");
        n.putExtra( name: "address", mOrderTowing.customer.phone);
        startActivity(n);
});
```

Gambar 3.31 Komponen kontak telepon dan SMS

Ketika tombol telepon yang ada di dalam komponen kontak ditekan, aplikasi akan mengarahkan *driver* ke aplikasi telepon bawaan dan langsung menghubungi pelanggan. Ketika tombol SMS ditekan, *driver* akan dialihkan ke aplikasi SMS bawaan dan membuat SMS baru dengan nomor telepon pelanggan sudah tercantum.

B.4.3 Komponen Sekunder

Komponen ini diatur oleh fungsi setupFragments(). Fungsi yang terdapat di dalam setupView() ini diatur paling terakhir setelah semua komponen primer

selesai dibuat. Fungsi setupFragments() merupakan fungsi yang bertanggung jawab untuk mengganti tampilan order detail *driver* yang ada di kotak merah pada Gambar 3.13. Berdasarkan Gambar 3.7, *driver* hanya memiliki 3 tab menu dan ketiga tampilan pada tiap tab menu diatur oleh fungsi ini. Berikut ini merupakan isi dari fungsi setupFragments().

```
public void setupFragments(){
    //Fragment Jemput
    setupFragmentJemput();

    //Fragment Progress
    setupFragmentProgress();

    //Fragment Antar
    setupFragmentAntar();
}
```

Gambar 3.32 Isi fungsi setupFragments

Fungsi setupFragmentJemput() digunakan untuk mengatasi tampilan *fragment* yang status ordernya berada pada tab menu Jemput. Tab menu jemput mengurus status order DRIVER_ASSIGN dan DRIVER_PICK_UP. Saat status order berupa DRIVER_ASSIGN, tampilan *fragment* disiapkan oleh fungsi setupFragmentDriverAssigned(). Tampilannya dapat dilihat pada Gambar 3.13. Fungsi driverPickup() di dalamnya berguna untuk mengganti status pesanan dari DRIVER_ASSIGNED menjadi DRIVER_PICK_UP. Saat status berupa DRIVER_PICK_UP, tampilan *fragment* disiapkan oleh fungsi setupFragmentDriverPickUp(). Tampilannya dapat dilihat pada Gambar 3.14. Fungsi driverPickup() di dalamnya berguna untuk mengganti status order dari DRIVER_PICK_UP menjadi DRIVER_ARRIVED_AT_ORIGIN.

```
public void setupFragmentJemput(){
    if(mOrderTowing.status.equals(MOrderTowing.ORDER_TOWING_STATUS_DRIVER_ASSIGNED)){
        setupFragmentDriverAssigned();
    }
    else if(mOrderTowing.status.equals(MOrderTowing.ORDER_TOWING_STATUS_DRIVER_PICK_UP)){
        setupFragmentDriverPickUp();
    }
}

public void setupFragmentDriverAssigned(){
    changeStatusBtn.setText(MOrderTowing.ORDER_TOWING_BUTTON_DEPART);
    changeStatusBtn.setOnClickListener((v) -> {
            isDistanceCalculated = false;
            driverDistance.setVisibility(View.GONE);
            driverDurationTime.setVisibility(View.GONE);
            progressJarak.setVisibility(View.VISIBLE);
            progressWaktu.setVisibility(View.VISIBLE);

            driverPickup();
            setupFragmentDriverPickUp();
    });
    fragmentManager.beginTransaction().replace(
            R.id.towing_detail_status,
            TowingDriverAssignedFragment.getFragment(towingId)
    ).commit();
}

public void setupFragmentDriverPickUp(){
    changeStatusBtn.setText(MOrderTowing.ORDER_TOWING_BUTTON_PICKEDUP);
    changeStatusBtn.setOnClickListener((v) -> {
            isDistanceCalculated = false;
            driverDistance.setVisibility(View.GONE);
            driverDurationTime.setVisibility(View.GONE);
            progressJarak.setVisibility(View.VISIBLE);
            progressWaktu.setVisibility(View.VISIBLE);

            driverArrivedAtOrigin();
            setupFragmentDriverArrivedAtOrigin();
    });
    fragmentManager.beginTransaction().replace(
            R.id.towing_detail_status,
            TowingDriverPickupFragment.getFragment(towingId)
    ).commit();
}
```

Gambar 3.33 Isi fungsi setupFragmentJemput dan fungsi pendukungnya

Fungsi setupFragmentProgress(), pada Gambar 3.32, digunakan untuk mengatasi tampilan *fragment* yang status ordernya berada di tab menu Progress. Tab menu ini mengurus status order DRIVER_ARRIVED_AT_ORIGIN. Saat status order berupa DRIVER_ARRIVED_AT_ORIGIN, tampilan *fragment* disiapkan oleh fungsi setupFragmentGoToDestination(). Tampilannya dapat dilihat pada Gambar 3.15. Fungsi driverGoToDestination() di dalamnya berguna untuk mengganti status pesanan dari DRIVER_ARRIVED_AT_ORIGIN menjadi DRIVER_GO_TO_DESTINATION.

```
public void setupFragmentProgress(){
    if(mOrderTowing.status.equals(MOrderTowing.ORDER_TOWING_STATUS_DRIVER_ARRIVED_AT_ORIGIN)){
        changeStatusBtn.setText(MOrderTowing.ORDER_TOWING_BUTTON_DEPART_TO_DESTINATION);
        changeStatusBtn.setOnClickListener((v) -> {
                isDistanceCalculated = false;
                driverGoToDestinatioin();
                setupFragmentDriverGoToDestination();
        });
        fragmentManager.beginTransaction().replace(
                R.id.towing_detail_status,
                TowingDriverArrivedAtOriginFragment.getFragment(towingId)
        ).commit();
    }
}

public void setupFragmentDriverArrivedAtOrigin(){
    changeStatusBtn.setText(MOrderTowing.ORDER_TOWING_BUTTON_DEPART_TO_DESTINATION);
    changeStatusBtn.setOnClickListener((v) -> {
            isDistanceCalculated = false;
            driverGoToDestinatioin();
            setupFragmentDriverGoToDestination();
    });
    fragmentManager.beginTransaction().replace(
            R.id.towing_detail_status,
            TowingDriverArrivedAtOriginFragment.getFragment(towingId)
    ).commit();
}

public void setupFragmentDriverGoToDestination(){
    changeStatusBtn.setText(MOrderTowing.ORDER_TOWING_BUTTON_ARRIVED);
    changeStatusBtn.setOnClickListener((v) -> {
            isDistanceCalculated = false;
            driverArrivedAtDestination();
            setupFragmentDriverArrivedAtDestination();
    });
    fragmentManager.beginTransaction().replace(
            R.id.towing_detail_status,
            TowingDriverArrivedAtOriginFragment.getFragment(towingId)
    ).commit();
}
```

Gambar 3.34 Isi fungsi setupFragmentProgress dan fungsi pendukungnya

Fungsi setupFragmentAntar(), pada gambar 3.35, digunakan untuk mengatasi tampilan yang status ordernya berada pada tab menu Antar. Tab menu Antar mengurus status order DRIVER_GO_TO_DESTINATION, DRIVER_ARRIVED_AT_DESTINATION dan FINISHED. Saat status berupa DRIVER_GO_TO_DESTINATION, tampilan *fragment* disiapkan oleh fungsi setupFragmentDriverGoToDestination(). Tampilannya dapat dilihat pada Gambar 3.15. Fungsi driverArrivedAtDestination() di dalamnya berguna untuk mengganti status order dari DRIVER_GO_TO_DESTINATION menjadi DRIVER_ARRIVED_AT_DESTINATION. Saat status berupa DRIVER_ARRIVED_AT_DESTINATION, tampilan *fragment* disiapkan oleh fungsi setupFragmentDriverArrivedAtDestination(). Tampilannya dapat dilihat

pada Gambar 3.16. Sebelum menyelesaikan order, sistem mengecek apakah order sudah dibayar atau belum. Jika sudah dibayar, fungsi driverFinished() dijalankan dan mengganti status order dari DRIVER_ARRIVED_AT_DESTINATION menjadi FINISHED.

```
public void setupFragmentAntar(){
    if(mOrderTowing.status.equals(MOrderTowing.ORDER_TOWING_STATUS_DRIVER_GO_TO_DESTINATION)){
        setupFragmentDriverGoToDestination();
    }
    else if(mOrderTowing.status.equals(MOrderTowing.ORDER_TOWING_STATUS_DRIVER_ARRIVED_AT_DESTINATION)){
        setupFragmentDriverArrivedAtDestination();
    }
    else if(mOrderTowing.status.equals(MOrderTowing.ORDER_TOWING_STATUS_FINISHED)){
        setupFragmentDriverFinished();
    }
}

public void setupFragmentDriverGoToDestination(){
    changeStatusBtn.setText(MOrderTowing.ORDER_TOWING_BUTTON_ARRIVED);
    changeStatusBtn.setOnClickListener((v) -> {
            isDistanceCalculated = false;
            driverArrivedAtDestination();
            setupFragmentDriverArrivedAtDestination();
    });
    fragmentManager.beginTransaction().replace(
            R.id.towing_detail_status,
            TowingDriverArrivedAtOriginFragment.getFragment(towingId)
    ).commit();
}
```

Gambar 3.35 Isi fungsi setupFragmentAntar dan fungsi pendukungnya

```java
public void setupFragmentDriverArrivedAtDestination(){
    travelCard.setVisibility(View.GONE);
    changeStatusBtn.setText(MOrderTowing.ORDER_TOWIN^_BUTTON_FINISHED);
    changeStatusBtn.setOnClickListener((v) -> {
            //cek payment
            if (mOrderTowing.paymentStatus.equals(MOrderTowing.ORDER_TOWING_PAYMENT_STATUS_PAID)){
                driverFinished();
                Intent intent = new Intent(
                        packageContext: TowingDriverDetailActivity.this,
                        TowingDriverFinished.class
                );
                intent.putExtra( name: "CUSTOMERNAME", mOrderTowing.customer.name);
                intent.putExtra( name: "CUSTOMERIMAGEPATH", mOrderTowing.customer.imagePath);
                startActivity(intent);
                finish();
            }
            else{
                Intent intent = new Intent(
                        packageContext: TowingDriverDetailActivity.this,
                        TowingPayment.class
                );
                intent.putExtra(BUNDLE_KEY_TOWING_NEW_OBBJECT, towingId);
                startActivity(intent);
                finish();
            }
    });
    fragmentManager.beginTransaction().replace(
            R.id.towing_detail_status,
            TowingDriverArrivedAtOriginFragment.getFragment(towingId)
    ).commit();
}

public void setupFragmentDriverFinished(){
    fragmentManager.beginTransaction().replace(
            R.id.towing_detail_status,
            TowingDriverAssignedFragment.getFragment(towingId)
    ).commit();
}
```

Gambar 3.36 Isi fungsi pendukung setupFragmentAntar

```java
public void setupFragmentDriverArrivedAtDestination(){
    travelCard.setVisibility(View.GONE);
    changeStatusBtn.setText(MOrderTowing.ORDER_TOWIN^_BUTTON_FINISHED);
    changeStatusBtn.setOnClickListener((v) -> {
            //cek payment
            if (mOrderTowing.paymentStatus.equals(MOrderTowing.ORDER_TOWING_PAYMENT_STATUS_PAID)){
                driverFinished();
                Intent intent = new Intent(
                        packageContext: TowingDriverDetailActivity.this,
                        TowingDriverFinished.class
                );
                intent.putExtra( name: "CUSTOMERNAME", mOrderTowing.customer.name);
                intent.putExtra( name: "CUSTOMERIMAGEPATH", mOrderTowing.customer.imagePath);
                startActivity(intent);
                finish();
            }
            else{
                Intent intent = new Intent(
                        packageContext: TowingDriverDetailActivity.this,
                        TowingPayment.class
                );
                intent.putExtra(BUNDLE_KEY_TOWING_NEW_OBBJECT, towingId);
                startActivity(intent);
                finish();
            }
    });
    fragmentManager.beginTransaction().replace(
            R.id.towing_detail_status,
            TowingDriverArrivedAtOriginFragment.getFragment(towingId)
    ).commit();
}

public void setupFragmentDriverFinished(){
    fragmentManager.beginTransaction().replace(
            R.id.towing_detail_status,
            TowingDriverAssignedFragment.getFragment(towingId)
    ).commit();
}
```

Gambar 3.37 Fungsi pendukung setupFragmentAntar

3.3.2 Kendala yang Ditemukan

Dalam proses pelaksanaan kerja magang, terdapat beberapa kendala yang dihadapi. Kendala-kendala yang ditemukan adalah implementasi Google Maps pada aplikasi android karena belum pernah dipelajari sebelumnya.

3.3.3 Solusi atas Kendala yang Ditemukan

Berdasarkan kendala-kendala yang ditemukan selama proses kerja magang, beberapa solusi ditemukan untuk menyelesaikan kendala tersebut agar tidak menghambat pengerjaan tugas yang dilakukan selama kerja magang. Solusi dari kendala-kendala yang ditemukan adalah dengan mempelajari cara menggunakan Google Maps pada aplikasi android dengan menonton video tutorial di YouTube, membaca dokumentasi dari StackOverflow, dan bertanya langsung kepada salah satu karyawan yang pernah mempelajarinya.