BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Proyek yang dikerjakan dalam dalam kegiatan magang ini adalah design dan implementasi *front-end* dan *back-end* sebuah aplikasi *android*. Posisi pekerjaan yang dilaksakan dalam kegiatan magang ini adalah sebagai *intern* yang mengembangkan ide dari sebuah masalah yang ada dalam perusahaan dengan supervisi dari Bapak Eko Purwito. Sebagai bagian dari proyek yang dikerjakan, dilaksanakannya kegiatan *brainstorming* mengenai bagian-bagian yang diperlukan oleh aplikasi dan struktur yang akan digunakan dalam pembuatan *prototype* aplikasi.

Kegiatan magang ini menggunakan interaksi pada *Renomeet* dan *Whatsapp*. *Renomeet* merupakan *website* buatan perusahaan yang digunakan untuk melakukan video call terhadap topik yang berkaitan dengan data internal PT AGCI. Komunikasi dilakukan melalui aplikasi Whatsapp sebagai alat tanya jawab dan absensi. Pesan Whatsapp yang dikirimkan harus memberikan ringkasan singkat terhadap latihan atau kegiatan yang dikerjakan pada hari kerja tersebut.

3.2 Tugas yang Dilakukan

Berdasarkan lampiran KM04, tugas yang dilakukan dalam kegiatan magang bisa dibagi menjadi 3 bagian utama. Bagian latihan, pembuatan desain dan pembahasan struktur *datababse*, dan pembuatan *front-end* dan *back-end* aplikasi *android*. Program yang digunakan adalah yaitu

- Figma
- AstahUML
- StarUML
- Android Studio
- Eclipse Kepler 4.1.3
- PostGre SQL

- Postman

Dalam durasi kegiatan magang, proyek yang dilakukan adalah pembuatan desain dan membuat struktur dasar untuk digitalisasi kegiatan penerimaan dan pengiriman makanan dalam sebuah restoran. Setelah desain awal, struktur *database* dan *flowchart* dari aplikasi telah ditentukan, kelanjutannya adalah pembuatan dari *front-end prototype* dari aplikasi tersebut. Setelah bulan pertama berlalu, dilaksanakannya latihan untuk mempelajari dan mengimplementasikan sistem *back-endv* untuk aplikasi *android* tersebut yang mencangkup *database* lokal melalui akses yang melalui internet dan pengiriman SQL.

Pada selaku kegiatan magang, ada kesempatan untuk menjadwalkan *meeting* untuk melakukan pembahasan mengenai hasil yang sudah dikerjakan dan pembahasan mengenai kelanjuttan dari kegiatan magang. Pada akhir dari kegiatan magang, diadakannya rekap dalam bentuk presentasi untuk menunjukkan keseluruhan dari kegiatan magang yang telah terjadi.

3.2.1 Uraian Pelaksanaan Kerja Magang

Persis dengan lampiran KM04, berikut merupakan tabel kegiatan magang yang terjadi pada tiap minggu:

Minggu	Rangkuman dari pekerjaan yang dilakukan tiap minggu
1	Perkenalan terhadap supervisor dan HRD dan latihan soal terhadap penulisan laporan dan kemampuan pemogramman
2	Latihan lanjutan mengenai Java dan SQL serta pembahasan dari soal
3	Pembahasan masalah proyek, pembuatan mockup layar aplikasi, dan design diagram ERD, SD, dan Use-Case
4	Pembuatan XML dan Activity *front-end* aplikasi Android Studio
5	Optimisasi *front-end* dan latihan awal back-end aplikasi
6	Pembelajaran, pembahasan, dan implementasi dari webservice aplikasi
7	Implementasi lanjutan *back-end* aplikasi
8	Revisi, Implementasi dan perbaikan bug dalam aplikasi
9	Pembahasan akhir dan presentasi hasil dari prototype aplikasi

Tabel 3.1. Rangkuman kegiatan magang setiap minggu

Pada awal dan sebelum dari kegiatan magang, diadakannya kegiatan tes dan latihan awal mengenai materi yang akan dikerjakan. Materi utama yang dibahas adalah Java dan SQL.

Pada selang antara latihan pemahaman dan pengetesan kemampuan, terjadi pembahasan mengenai beberapa pilihan proyek yang akan dikembangkan pada selang waktu kegiatan magang. Dari masalah yang ada, proyek yang dikembangkan dalam kegiatan magang adalah pembuatan design *front-end* dan *back-end* aplikasi *android* terhadap digitalisasi pada kegiatan penerimaan dan pendataan pesanan dari pembeli dengan dapur dalam sebuah restoran.

Minggu ke 3, telah dibuat design untuk diagram dan *mockup* layar yang akan menjadi pedoman dari minggu selanjutnya. Ada juga kesempatan untuk melakukan pembahasan dan perbaikan terhadap tes dan latihan yang telah dikerjakan. Diagram yang dikerjakan mencangkup *Entity Relationship, Sequence Diagram* dan *Use-case* Diagram. Mockup halaman menunjukkan tiap activity yang perlu dibuat sesuai dengan *Sequence* Diagram yang telah dibuat menggunakan Figma.

Pada minggu ke 4, telah dibuat desain XML untuk tampilan *front-end* dari proyek di *Android Studiov*. *Activity* yang dibuat sebagai *front-end* pada *Android Studio* mencangkup halaman login, halaman list meja dalam sebuah restoran, halaman isi pesanan terhadap meja, dan halaman pengisian pesanan dari meja tersebut. Minggu selanjutnya digunakan untuk optimisasi XML dan pembelajaran awal materi *back-end* serta aplikasi lainnya yang dibutuhkan dalam pengerjaan desain *back-end* pada data yang bersifat lokal.

Minggu ke 6 dan 7 digunakan untuk pemahaman dan implementasi dari *back-end* aplikasi. *Back-end* tersebut mencangkup *webservice*, *local database*, dan hasil *front-end* aplikasi. Minggu ke 8 adalah bagian terakhir dari implementasi terhadap halaman-halaman *activity* berserta SQL yang diperlukan dalam aplikasi. Pada minggu terakhir, diberikannya kesempatan untuk memperbaiki beberapa bug pada SQL dan logika *back-end* dan mengimplementasikan update pada saat berpindah halaman.

Minggu terakhir kegiatan magang digunakan untuk rekap ulang kegiatan magang dalam bentuk presentasi. Diadakannya kesempatan untuk pembuatan presentasi dan revisi untuk mengakhiri kegiatan magang.

3.3 Proses Pelaksanaan

Proses dari kegiatan magang bisa dibagi menjadi 4 bagian, yaitu *brainstorming* dan pembahasan mengenai kebutuhan dalam aplikasi yang perlu untuk dibuat, pembuatan struktur diagram dan *mockup* dasar dari aplikasi, pembuatan *front-end* dari aplikasi, dan pembuatan *back-end* dari aplikasi.

3.3.1 Analisis Kebutuhan Proyek

Prototype yang dikerjakan adalah untuk mendigitalkan kegiatan interaksi antara pembeli dan pegawai dari restoran. Program tersebut mencangkup pengambilan pesanan, pendataan pesanan, pengataran pesanan dan pembayaran pada kasir. Pada aplikasi ini, *prototype* yang akan dibuat diperlukan untuk menyimpan data terhadap interaksi antara pembeli dan pegawai sesuai interaksi yang barusan disebutkan kecuali pengiriman pesanan dari dapur kepada pembeli.

Berdasarkan interaksi yang ada dalam restoran, dapat ditemukan 3 jenis pengguna aplikasi yaitu pembeli, pramusaji dan kasir. Pada kegiatan magang ini, *prototype* diminta untuk mengerjakan bagian interaksi terhadap pramusaji saja.

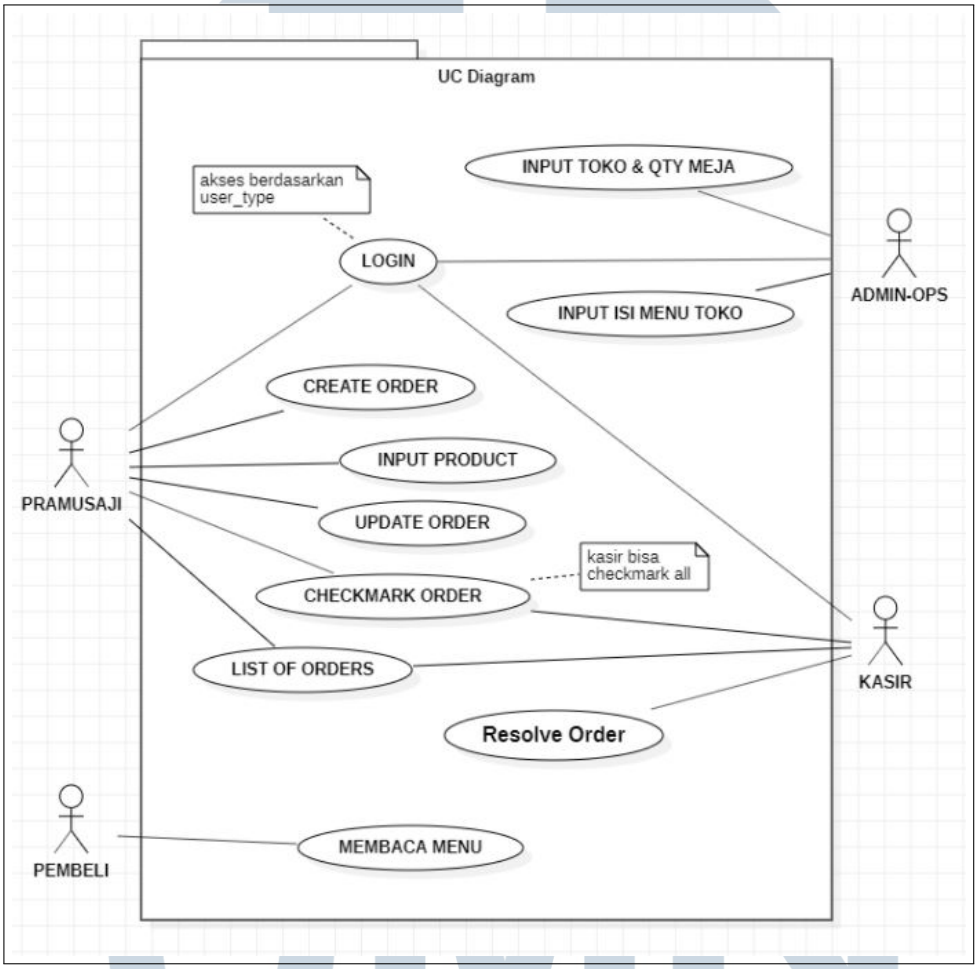
Tugas utama pramusaji adalah untuk menerima pesanan dari pembeli dan mengirimkan pesanan dari pembeli. Dari inti masalah tersebut dibuatkanlah aplikasi yang mendatakan setiap meja dalam restoran yang bisa di ketuk untuk membuka keterangan isi meja seperti pesanan, nama pembeli dan keterangan dari pesanan tersebut.

3.3.2 Struktur Diagram

Berdasarkan penjelasan sebelumnya, bagian dari pembuatan design struktur aplikasi mencangkup diagram dari struktur aplikasi, mockup *front-end* aplikasi. Dalam proyek tersebut bisa ditemukan 3 jenis diagram, yaitu *Entity Relationship Diagram, Use-case Diagram* dan *Sequence Diagram.*

A. Use-Case Diagram

Berikut merupakan diagram use-case yanng digunakan sebagai struktur dasar interaksi dalam aplikasi.



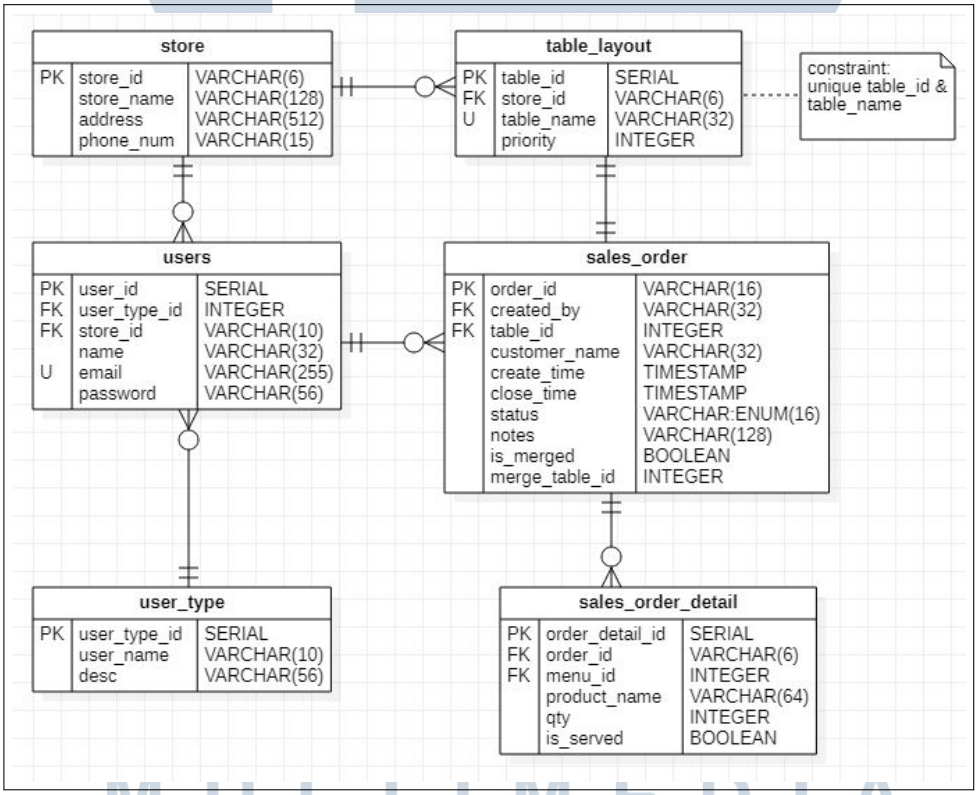
Gambar 3.1. Diagram Use-Case mengenai interaksi antara manusia dengan aplikasi

Sesuai dengan gambar 3.1, kegiatan interaksi antar manusia dalam sebuah restoran dapat dibagi menjadi 4 bagian, yaitu pramusaji, kasir dan pembeli. Admin operasional yang ada pada diagram diatas merupakan pengurus dalam yang mengatur isi data dari toko. Sebelum kegiatan transaksi dalam restoran berjalan, admin operasional disini harus mengisi data dari layout meja dalam sebuah restoran. Selanjutnya pada kegiatan transaksi dalam restoran, pembeli akan bertemu dengan pramusaji yang akan mencatat pesanan dari pembeli. Setelah pembeli selesai makan, pembeli akan menuju kasir untuk membayar makanan yang sudah dimakan.

Pada aplikasi yang telah dikerjakan, belum ada perbedaan akses antara kasir, admin operasional atau pramusaji. Hal tersebut disebabkan oleh perbedaan dari lokasi akses yang akan digunakan oleh user tersebut. Berdasarkan pembahasan singkat dalam kegiatan magang, akses aplikasi tersebut akan dibuatkan terpisah sebagai aplikasi *embedded* dalam sebuah gadget tablet yang akan tersedia pada restoran untuk bagian kasir. Dan topik-topik yang perlu diisi untuk melakukan tes pada bagian admin operasional perlu untuk membahas mengenai rahasia perusahaan, oleh karena itu kegiatan magang tidak mencangkup bagian perbedaan akses user seperti admin operasional ataupun kasir.

B. Entity Relationship Diagram

Berikut merupakan ERD dari *prototype* aplikasi yang dibuat:



Gambar 3.2. Entity Relationship Diagram dari back-end aplikasi

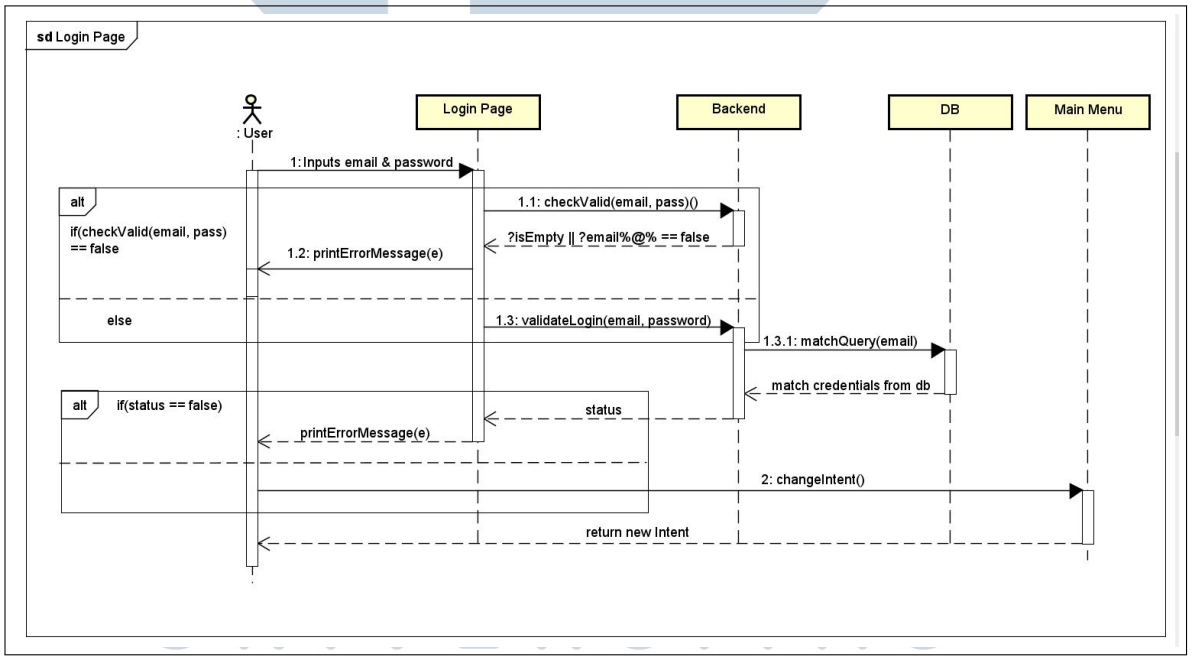
Berdasarkan diagram seperti gambar 3.2, bisa ditemukan 6 tabel dalam *database* yaitu *store*, *users*, *user_type*, *table_layout*, *sales_order*, dan *sales_order_detail*. Relasi tabel diatas adalah *many to many* kecuali *users* dengan *user_type* yang memiliki relasi one to many. Tabel dalam ERD diimplementasikan

dengan *local database* yang diisi dengan *dummy data* pada komputer local dimana prototype tersebut dibuat untuk menunjukkan simulasi dari salah satu toko dalam perusahaan.

C. Sequence Diagram

Pembuatan diagram untuk kegiatan proyek diharuskan untuk menggunakan Astah UML dalam pembuatan diagram *sequence*. Diagram *sequence* ini akan digunakan sebagai pedoman dari struktur *back-end* yang akan dibuat terhadap beberapa halaman yang memerlukan akses kepada *database*. Dari semua diagram yang ada, kegiatan magang ini difokuskan untuk menyelesaikan *front-end* dan *back-end* dari bagian pramusaji.

C.1 Login SD

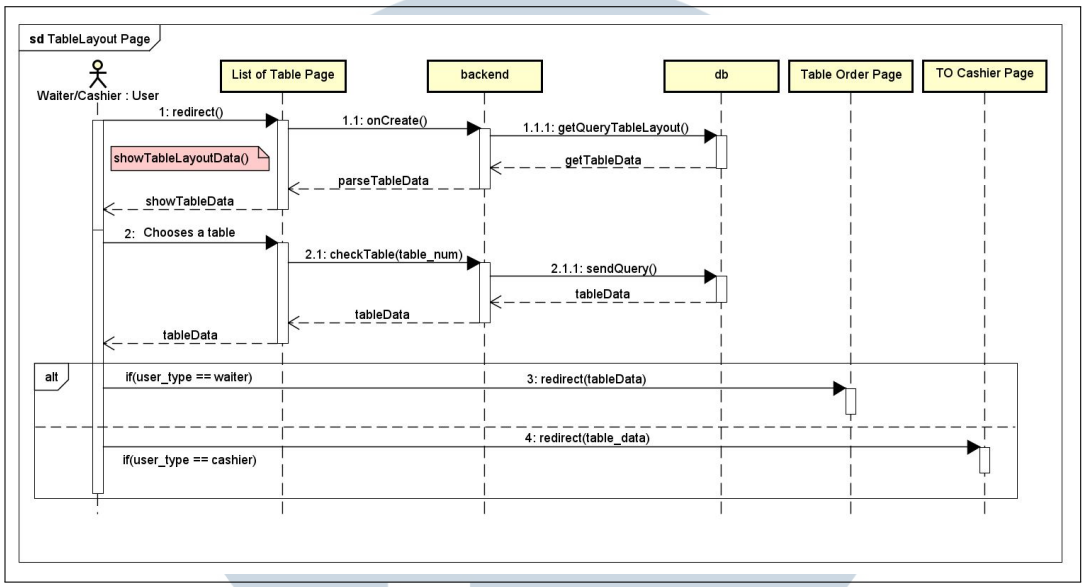


Gambar 3.3. Sequence Diagram untuk halaman login

Gambar 3.3 menunjukkan alur utama dari kegiatan login aplikasi *prototype*. Pada halaman login ini pengguna akan mendapatkan data yang akurat terhadap kesalahan *email/username* dan *password* yang diisi. Halaman login ini akan memanggil tabel dalam *database* dengan judul *users* untuk mengecek kebenaran dari *username* dan *password* yang diisi.

Diagram pada gambar 3.3 juga menunjukkan interaksi antara halaman *login* dengan *database*. Tabel ERD yang didigunakan adalah tabel *users* yang digunakan sebagai pedoman utama pengecekkan email dan password pengguna.

C.2 TableLayout SD



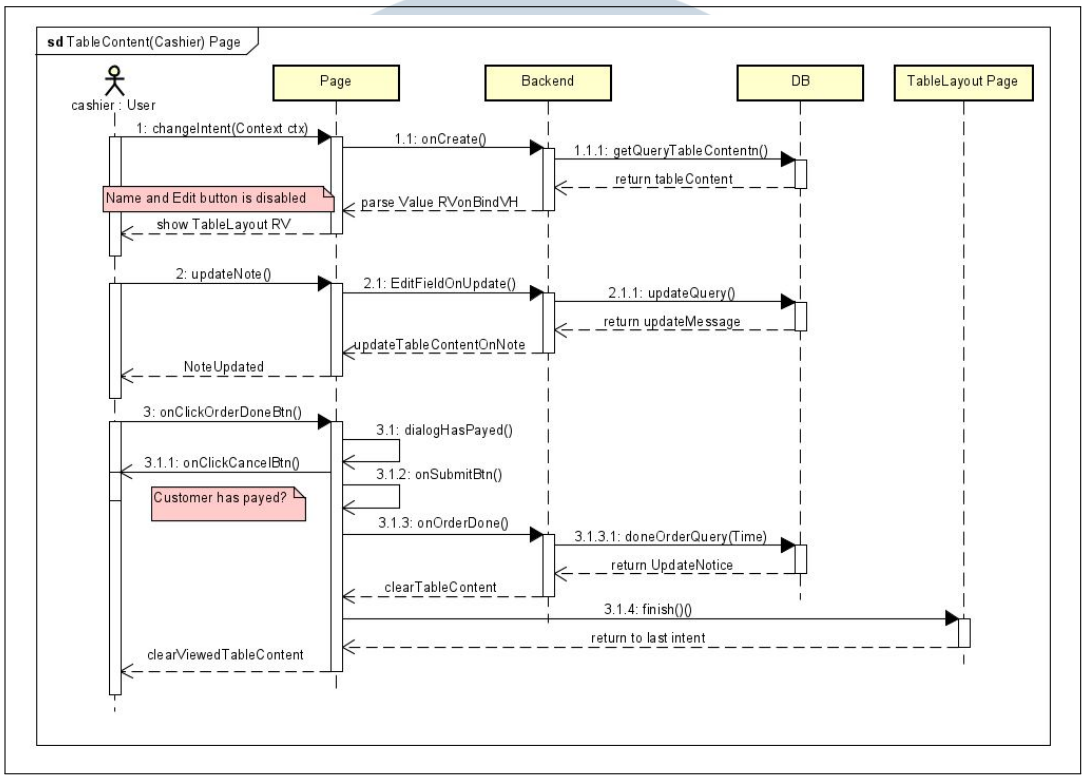
Gambar 3.4. Sequence Diagram untuk halaman *TableLayout* milik pramusaji

Berdasarkan gambar 3.4, bisa ditemukan secara mendetil interaksi yang terjadi dalam aplikasi untuk menunjukkan data yang perlu ditampilkan ke layar pada halaman utama aplikasi. Pada saat halaman tersebut dipanggil, aplikasi diharapkan untuk menampilkan data dari semua meja dalam toko sesuai dengan toko tempat karyawan tersebut bekerja. Dari data yang akan ditampilkan dalam halaman tersebut, tiap data dari meja diharapkan untuk dapat mengambil dan menampilkan data mengenai meja yang dipilih kepada pengguna.

Halaman *TableLayout* milik pramusaji merupakan halaman utama setelah *login*. Hhalaman ini mengakses *database* dengan nama *table_layout* yang akan menunjukkan semua meja dalam sebuah toko.

Halaman *TableLayout* milik pramusaji mengakses *database* bagian *sales_order* juga. Dari tabel tersebut, aplikasi dapat menemukan rincian dari meja yang ada dalam toko tersebut. Rincian ini dapat di ubah oleh pramusaji sesuai dengan kebutuhan.

C.3 TableContent Kasir SD

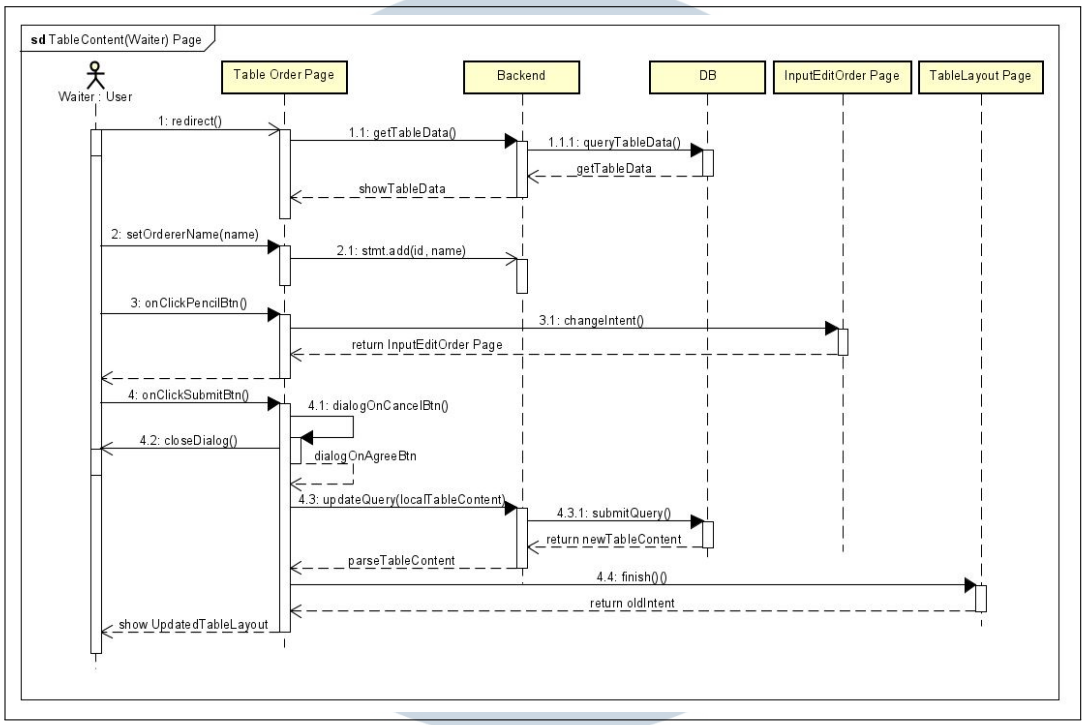


Gambar 3.5. Sequence Diagram untuk halaman *TableContent* milik kasir

Diagram pada gambar 3.5 merupakan struktur diagram yang menjadi garis besar mengenai bagian kasir proyek yang akan dikerjakan perusahaan. Bagian dari diagram ini tidak dikerjakan tidak termasuk bagian dari kegiatan magang melainkan desain struktur mengenai akses dari kasir terhadap aplikasi.

Diagram *sequence* pada gambar 3.5 menunjukkan interaksi antara bagian kasir dalam restoran dimana aplikasi diminta untuk menunjukkan data dari pesanan pembeli. Dan dari interaksi tersebut diharapkan juga aplikasi dapat mendatakan tiap order yang dipesan sebagai terbayar supaya tidak ada kelalaian antara kasir ataupun pembeli jika mereka lupa membayar.

C.4 TableContent Pramusaji SD

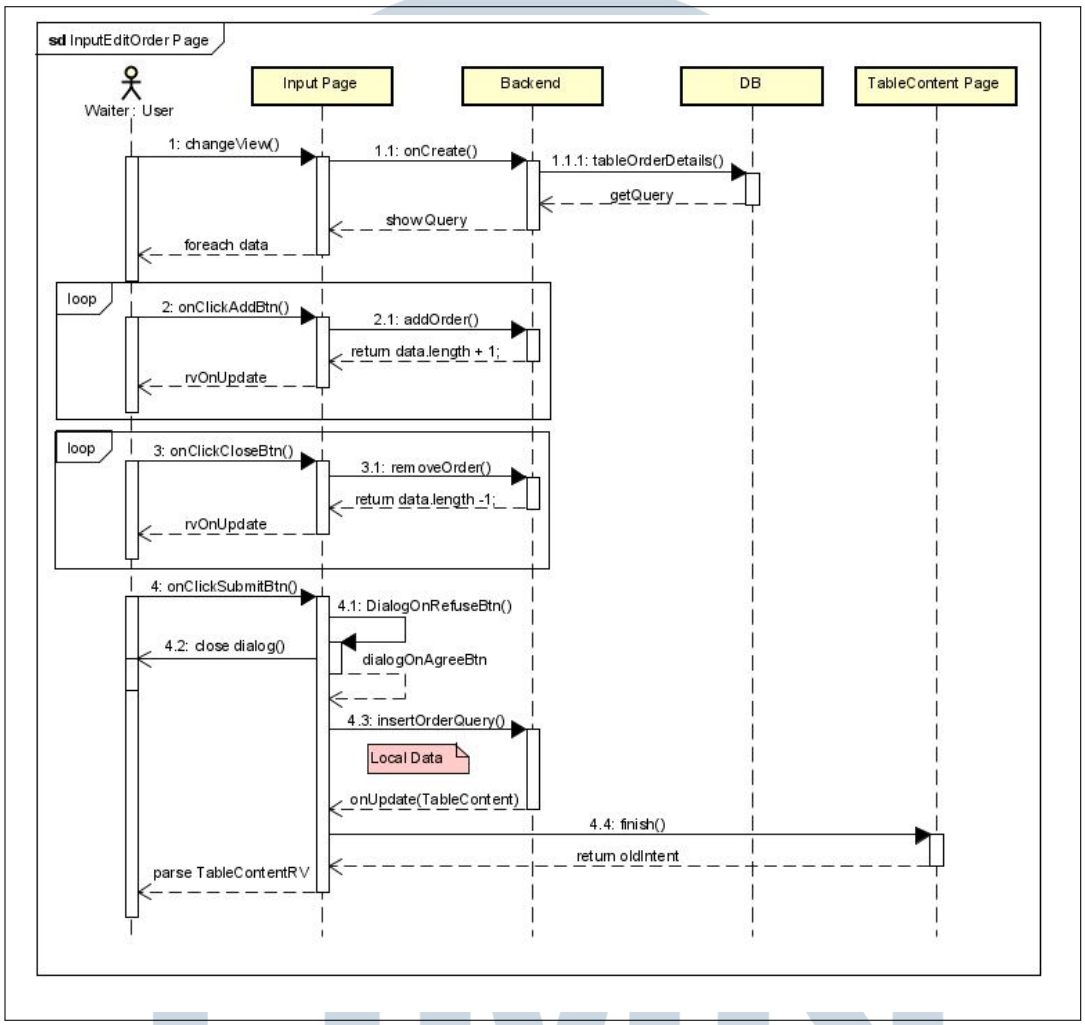


Gambar 3.6. Sequence Diagram untuk halaman *TableContent* milik pramusaji

Diagram *sequence* sesuai dengan gambar 3.6 merupakan alur mengenai halaman *TableContent* milik pramusaji. Pada halaman ini pramusaji dapat mengisi nama dan membaca pesanan yang sudah diisi oleh pramusaji pada halaman selanjutnya. Hasil yang ada dalam halaman ini akan sesuai dengan data yang telah atau akan dimasukkan oleh pramusaji lainnya.

Pada halaman ini aplikasi mengakses *database* bagian *table_layout*, *sales_order* dan *sales_order_detail* untuk menunjukkan data dari meja yang telah dipilih oleh pengguna. Data tersebut digunakan untuk menunjukkan nama dan isi pesanan yang pernah diisi dan belum menyelesaikan kegiatan transaksi dalam restoran.

C.5 InputEditOrder SD



Gambar 3.7. Sequence Diagram untuk halaman *InputEditOrder* milik Pramusaji

Pada diagram *sequence* yang ada pada gambar 3.7, alur yang terjadi adalah pada saat pramusaji akan mengubah isi pesanan dalam sebuah meja. Aplikasi dapat menunjukkan pesanan lainnya yang sudah diisi pada saat halaman ini dibuka kembali. Pada halaman *InputEditOrder*, pramusaji bisa menambahkan pesanan kosong untuk semua pesanan yang ada. Dalam pesanan kosong tersebut, pengguna aplikasi perlu mengisi nama pesanan dan jumlah dari pesanan tersebut. Setelah pengisisan pesanan selesai, diperlukan untuk pramusaji nenekan tombol submit supaya informasi tersebut tersimpan dalam *database*.

3.3.3 Tampilan Mockup Aplikasi

Dari diagram yang telah dibuat sesuai dengan gambar 3.2 sampai 3.7, digunakannya *website* dengan nama Figma untuk membuat tampilan awal aplikasi.

A. Halaman Login

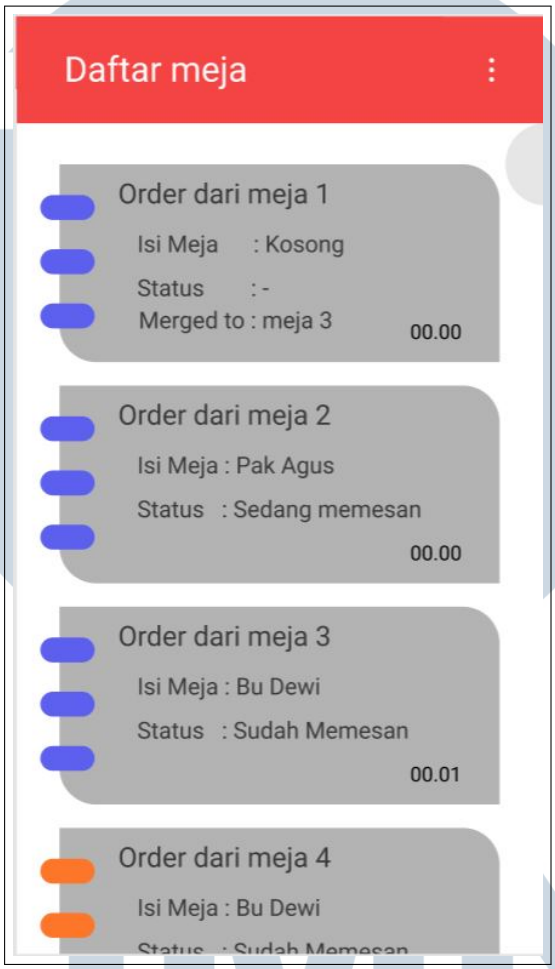
Berikut merupakan mockup dari halaman login.

Gambar 3.8. Mockup untuk halaman login

Pada gambar 3.8 pengguna aplikasi dapat berinteraksi dengan *textBox email* dan *password*, tombol dengan tulisan login dan tulisan kecil register. Textbox akan memberikan tampilan error jika textbox tersebut belum diisi dengan kriteria email dan password yang ada. Tombol registrasi yang ada pada halaman ini hanya digunakan sebagai *hyperlink* untuk menjadi jalan tikus dalam aplikasi prototype ini sesuai dengan persetujuan dan diskusi.

B. Halaman TableLayout

Gambar 3.9 merupakan mockup dari halaman *TableLayout*.

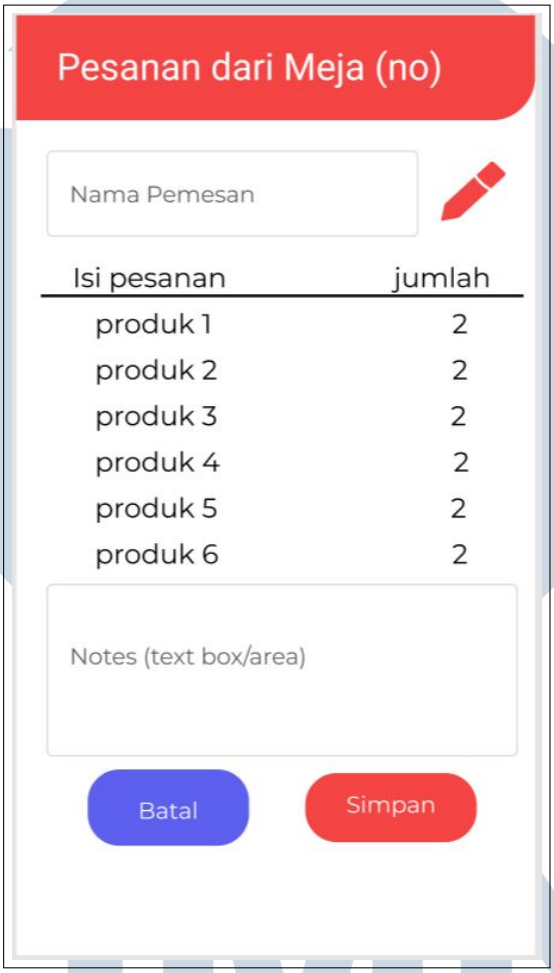


Gambar 3.9. Mockup untuk halaman TableLayout

Pada halaman ini, pramusaji dapat melakukan *scrolling* pada halaman untuk melihat meja lainnya dalam toko. Halaman ini memiliki interaksi pada tombol titik tiga diatas sebagai tombol setting dan berbagai macam meja yang akan ditunjukkan berdasarkan *dummy data* untuk menunjukkan keadaan sebuah meja. Berdasarkan *dummy data* yang terlihat, data tersebut akan menyesuaikan text yang ditampilkan berdasarkan isi dari *database*. Setiap kotak yang ada dalam halaman ini dapat membuka halaman selanjutnya dimana halaman *TableContent* bisa dilihat sesuai dengan keterangan dari meja yang telah dipilih.

C. Halaman TableContent

Gambar 3.10 merupakan mockup dari halaman *TableContent*.

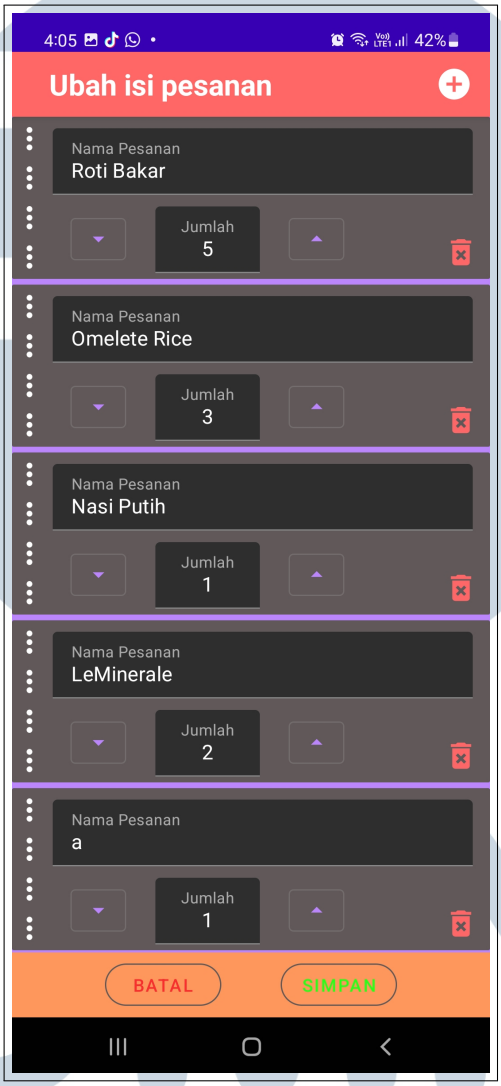


Gambar 3.10. Mockup untuk halaman TableContent

Pada halaman ini pengguna dapat berinteraksi dengan *textBox* nama yang akan menjadi indikasi terhadap salah satu pembeli pada meja tersebut. Tombol pensil di sebelah nama merupakan akses menuju halaman *InputEditOrder*.

D. Halaman InputEditOrder

Gambar 3.11 merupakan mockup dari halaman *TableContent*.

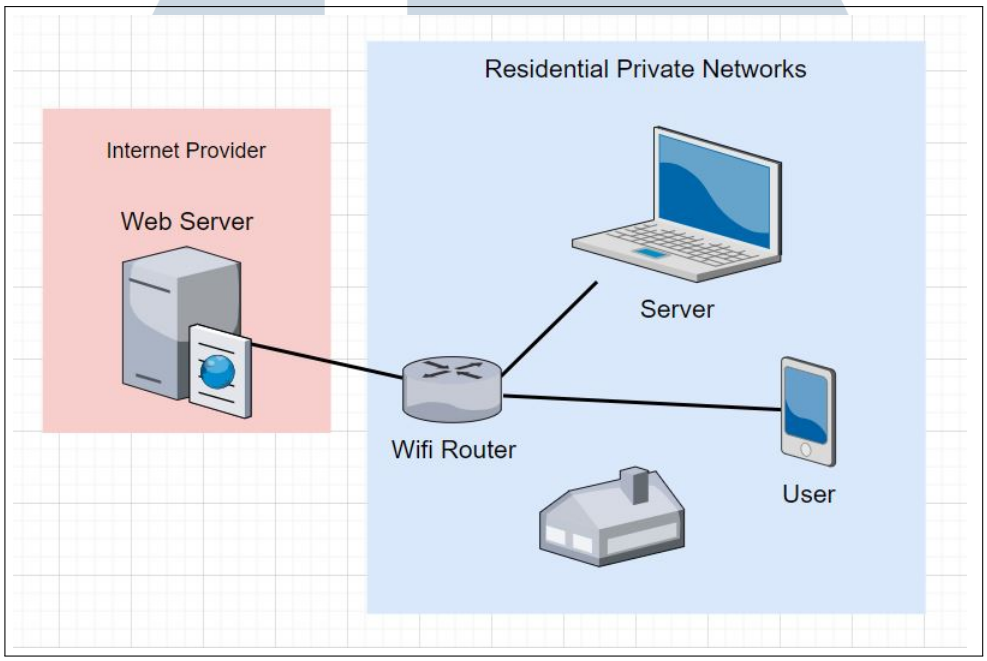


Gambar 3.11. Mockup untuk halaman InputEditOrder

Dari gambar diatas, pengguna dapat mengubah nama dan jumlah dari pesanan yang akan dipesan dalam sebuah meja. pesanan tersebut bisa dihapus sesuai dengan tombol merah pada tiap pesanan. Pramusaji juga bisa untuk menambahkan jumlah pesanan dengan menekan tombol + pada kanan atas.

3.3.4 Implementasi Back-end

Setelah selesai membuat ulang XML sesuai dengan mockup diatas, keterlanjutan dari prototype memerlukan untuk adanya interaksi *webservice* dari *Android Studio*, penulisan bahasa pemograman SQL, dan pembuatan koneksi dari komputer lokal kepada *webservice* sebagai berikut.



Gambar 3.12. Arsitektur Back-End Aplikasi

Dengan menggunakan *Eclipse Kepler* dan *PostGreSQL*, *database* yang ada dalam komputer lokal dapat digunakan untuk membuat dan menampung *dummy data* demi kepentingan magang.

Dalam interaksi antara aplikasi dengan internet, bagian dari aplikasi *android* perlu untuk memanggil perintah sesuai kebutuhan kepada *database* aplikasi. Dan pada *login* tersebut, diperlukan interaksi dari *webservice* untuk membuat *script* SQL sesuai kebutuhan dari aplikasi untuk mengirimkan data yang diperlukan. Postman merupakan aplikasi yang digunakan di antara interaksi *webservice* dan aplikasi pada *Android* untuk menunjukkan bahwa perintah yang dipanggil dari *webservice* sudah sesuai dengan keperluan aplikasi.

Pada implementasi dari *prototype* aplikasi, desain yang digunakan pada *Android Studio* untuk mengerjakan bagian *back-end* ini adalah design MVVM. MVVM atau *Model-View-ViewModel* merupakan struktur penulisan file dalam

aplikasi *Android Studio* dimana sebuah activity dibagi menjadi 3 bagian yaitu *Model, View* dan *ViewModel*.

3.3.5 Contoh Script Aplikasi

Untuk pemograman utama dalam kegiatan *Prototyping* pada kegiatan magang adalah menggunakan *Android Studio* dan *Eclipse Kepler*. Interaksi utama terhadap *webservice* adalah dengan menggunakan *API Helper Interface* dan *Error handling* yang terjadi dalam *Android Studio*

Struktur dari aplikasi *Android Studio* yang digunakan untuk melakukan interaksi *back-end* kepada *webservice* adlaah design MVVM. Dalam sebuah MVVM, sebuah *script* dibagi menjadi 3 bagian. Bagian model digunakan untuk interaksi dasar atau lapisan paling atas mengenai *script* pemograman. Pada kegiatan magang ini, *ViewModel* digunakan sebagai perantara untuk kalkulasi kompleks atau jalur tengah untuk memanggil data yang dibutuhkan dari *database*. File *repository* digunakan sebagai bagian error handling untuk memanggil data dari *database* dan memanggil *APIHelperInterface*.

A. Script Repository

Berikut merupakan *script repository* yang digunakan sebagai interaksi terhadap akses menuju *database*.

Kode 3.1: Kode inisialisasi repository di Android Studio

```
public MutableLiveData<Result<List<TableLayout>>>
getTableLayout() {
        MutableLiveData<Result<List<TableLayout>>>
        mLiveData = new MutableLiveData<>();

        Timber.tag(TAG).d("Start of
TablelayoutRepos");

        Gson gson = new
        GsonBuilder().setDateFormat("yyyy-MM-dd'T'H
H:mm:ss.ZZZ").create();
```

```
Map<String, String> stringMap = new
HashMap<>();
stringMap.put("store_id", "A001");
// TODO: Change value above to match
user.getStoreId on launch
//  by putExtra User.getStoreId() from
loginActivity

SendData sendData = new
SendData(appState.getDeviceId(),
        "getTablelayoutByStoreId",
        stringMap);
String data = gson.toJson(sendData);

ApiHelperInterface apiHelperInterface =
ApiHelper.getClient()
        .create(ApiHelperInterface.class);

Call<String> call =
apiHelperInterface.getTableLayoutByStoreId(
data);

call.enqueue(new Callback<String>() {
```

Script diatas merupakan inisialisasi awal dari *repository* sebelum aplikasi memanggil *webservice* untuk mendapatkan atau mengirimkan data yang diperlukan oleh aplikasi.

B. Script Call Enqueue dalam Repository

Berikut merupakan lanjutan dari *script* dalam *repository*.

Kode 3.2: Kode Call Enqueue Dalam Repository di Android Studio

```
if (response.body() != null &&
response.isSuccessful()){
    try {
```

```
        JSONObject jsonObj = new JSONObject(
        response.body( ) );

        String status = jsonObj.getString("status");
        String msg = jsonObj.getString("message");

        if (status.equalsIgnoreCase("ok")
                &&
                msg.equalsIgnoreCase("success")){
            String data =
            jsonObj.getString("data");
            String strTableLayoutData = null;

            JSONObject jsonObjectData = new
            JSONObject(data);

            if
            (jsonObjectData.has("tableLayoutList")
            {
                strTableLayoutData =
            jsonObjectData.getString(
            "tableLayoutList");
            }

            if (strTableLayoutData == null) {
                mLiveData.setValue(new Result.Error
                        (new Exception( "Empty
TableLayout in store") )
                        );
                return;
            }

            Type typeTableLayout = new TypeToken
            <List<TableLayout>>(){}
            .getType();
            List<TableLayout> tableLayout =
```

```
                gson.fromJson(strTableLayoutData,
                typeTableLayout);

                if (tableLayout != null){
                    mLiveData.setValue(new
                    Result.Success<>(tableLayout));

                } else {
                    mLiveData.setValue(new Result.Error(
                            new Exception("Store or
tableLayout not found")
                    ));
                } // end of if tableLayout != null
            }

        }catch (JSONException e){
            Timber.tag(TAG).e(e);
            mLiveData.setValue(new Result.Error(new
            IOException("JSON Error", e)));
        }
}
```

Script diatas merupakan bagian dari *error handling* aplikasi untuk memastikan bahwa panggilan yang dilakukan pada *webservice* tidak menimbulkan *fatal error*

C. Script API Helper Interface

Berikut merupakan *script* dari *APIHelperInterface*

Kode 3.3: Kode APIHelperInterface di Android Studio

```
/** Get List Customer */
@Headers("Content-Type: application/json")
@POST("customer/getCustomers")
Call<String> getCustomers(
        @Body String data
);

@Headers({"Accept: application/json",
"Content-Type: application/json; charset=utf-8"})
@GET("salesorder/getLastSalesOrder")
Call<String> getLastSalesOrder();
```

Pada kode diatas, bisa ditemukan beberapa bagian utama dalam sebuah fungsi, yaitu: *Header, @post* atau *@get*, dan nama fungsi tersebut. *Headers* ini gunakan sebagai *url* yang akan dipanggil kepada *webservice* untuk menerima isi dari *database* sesuai dengan SQL yang telah dibuat. Fungsi dari simbol @ disini menunjukkan tipe panggilan kepada *database* yang dibutuhkan. *POST* memiliki arti dimana aplikasi ingin mengirimkan data dan juga menerima data kepada *database* sedangkan *GET* merupakan fungsi yang dipanggil jika aplikasi hanya ingin mengambil data dari *database*.

D. Script SQL

Berikut merupakan salah satu *script* SQL yang digunakan pada *Eclipse Kepler* untuk mengirimkan data dari *database* kepada aplikasi.

Kode 3.4: Kode SQL di Eclipse Kepler

```
@POST
@Path(value = "/getTableLayoutByStoreId")
public String getTableLayoutByStoreId(String json){
        LOGGER.info("Start fetching tables for store");
        Response response = new Response();
        Gson gson = new GsonBuilder().create();

        try{
                response.setStatus("OK");
                response.setMessage("Success");
                Map<String, Object> map = new
                HashMap<String, Object>();
                JSONObject jsonObjectAll = new
                JSONObject(json);

                LOGGER.info("Device ID
:"+jsonObjectAll.getString("deviceId"));
                LOGGER.info("Data
:"+jsonObjectAll.getJSONObject("data"));

                JSONObject data =
                jsonObjectAll.getJSONObject("data");
                String id = data.getString("store_id");

                map.put("tableLayoutList",
                tableLayoutbl.
                getTableLayoutByStoreId(id) );
                response.setData(map);

                return gson.toJson(response);
```

```
        }catch (Exception e) {
                LOGGER.error(e.getMessage(), e );
                response.setStatus("Error");
                response.setMessage(e.getMessage( ) );

                return gson.toJson(response);
        }
}

public List<TableLayout>
getTableLayoutByStoreId(String id) throws
SQLException{

        StringBuffer sql = new StringBuffer("SELECT
t.table_id , t.store_id , t.priority ,
t.table_name , ");
        sql.append("(SELECT notes FROM
rsps_app.sales_order WHERE
sales_order.table_id = t.table_id ");
        sql.append("AND close_time IS null ORDER BY
order_id DESC LIMIT 1), ");
        sql.append("(SELECT order_id FROM
rsps_app.sales_order WHERE
sales_order.table_id = t.table_id ");
        sql.append("AND close_time IS null ORDER BY
order_id DESC LIMIT 1), ");
        sql.append("(SELECT customer_name FROM
rsps_app.sales_order ");
        sql.append("WHERE sales_order.table_id =
t.table_id ");
        sql.append("AND close_time IS null ORDER BY
order_id DESC LIMIT 1 ");
        sql.append("), ");
        sql.append("(SELECT status FROM
rsps_app.sales_order WHERE
sales_order.table_id = t.table_id ");
```

```
            sql.append("AND close_time IS null ORDER BY
order_id DESC LIMIT 1 ");
            sql.append(") ");
            sql.append("FROM rsps_app.table_layout t ");
            sql.append("WHERE t.store_id = ? ");
            sql.append("ORDER BY t.priority ");

            return jdbcOperations.query(sql.toString()
            , new Object[]{id}, MAPPER);
}
```

Script java diatas mencangkup *script* SQL yang digunakan untuk memanggil data dari *database PostGreSQL*.

3.4 Kendala dan Solusi yang Ditemukan

Beberapa kendala dalam kegiatan magang yang ditemui dalam kegiatan magang adalah:

1. Kesulitan terhadap logika SQL yang cukup panjang.
2. Pemahaman terhadap interaksi dari aplikasi dengan model MVVM.
3. Kelengkapan dari brainstorming terhadap aplikasi prototype.
4. *System Testing* dan alur selanjutnya dalam pembuatan aplikasi.

Berdasarkan kendala yang ditemukan diatas, solusi yang bisa diterapkan untuk meneyelesaikan kendala tersebut adalah:

1. Bertanya kepada supervisor.
2. Mengikut seminar terhadap materi MVVM yang diadakan oleh kantor.
3. Diharapkan presentasi yang telah berikan pada akhir dari kegiatan magang bisa menjadi bantuan untuk kelengkapan dari hasil aplikasi sesuai *brainstorming* yang telah dilakukan.
4. Keterlanjutan dari proyek berdasarkan hasil dari magang