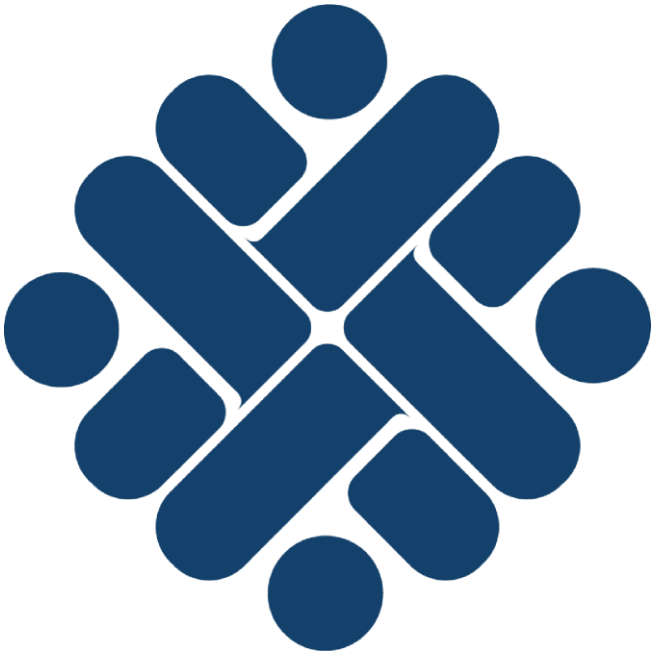
LAPORAN TUGAS AKHIR
APLIKASI MONITORING KEBUN HIDROPONIK BERBASIS MOBILE



Disusun oleh :

Fernanda Daymara Hasna

MOBILE APPLICATION AND TECHNOLOGY

KEJURUAN TEKNOLOGI INFORMASI DAN KOMUNIKASI

BALAI BESAR PENGEMBANGAN LATIHAN KERJA

BBPLK BEKASI

2021

KATA PENGANTAR

Assalamualaikum Wr. Wb.

Puji syukur kehadirat Tuhan Yang Maha Esa atas segala rahmat-Nya, penulis dapat menyelesaikan laporan tugas akhir dengan judul **Laporan Tugas Akhir Aplikasi Monitoring Kebun Hidroponik Berbasis *Mobile***. Pengembangan ini disusun dalam rangka memenuhi salah satu persyaratan untuk menyelesaikan Pelatihan Berbasis Kompetensi (PBK) Tahap 1 "*Mobile Application and Technology*" yang diselenggarakan oleh Balai Besar Pengembangan Latihan Kerja (BBPLK) Bekasi sejak Februari 2021 s.d. Oktober 2021.

Penulis menyadari bahwa dalam penyusunan proposal ini jauh dari kata sempurna, untuk itu penulis memohon kritik dan saran yang membangun sehingga dapat menjadi lebih baik lagi. Harapannya penelitian ini dapat berguna sebagai acuan pengembangan selanjutnya dan bermanfaat bagi kita semua. Aamiin.

Wassalamualaikum Wr. Wb.

Bekasi, 05 Oktober 2021

Penulis

Fernanda Daymara Hasna

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE

BAB I
PENDAHULUAN

A. LATAR BELAKANG

Saat ini, bercocok tanam tidak hanya dilakukan dengan media tanam berupa tanah. Banyak media tanam lain yang dapat dijadikan alternatif lain jika tanah tidak memungkinkan untuk digunakan, contohnya adalah hidroponik. Hidroponik merupakan teknik budidaya tanaman yang memanfaatkan air (tanpa tanah) sebagai media tanamnya, dimana akar tanaman terendam dan tumbuh pada air dangkal yang bernutrisi tersirkulasi. Pada hidroponik yang terbuka, lebih rentan terkena hama dan penyakit seperti belalang (membuat daun berlubang), lalat buah (membuat daun memiliki garis meliuk tak beraturan), hama kutu (membuat daun keriting menggulung dan batang melemah), dll. sehingga dibutuhkan mekanisme tambahan penghalau hama penyakit. Sedangkan pada hidroponik tertutup seperti *greenhouse* di daerah perkotaan (biasanya menggunakan paranet), membuat suhu udara di dalamnya menjadi lebih panas dan kadar kelembaban menurun, sehingga membutuhkan mekanisme tambahan untuk menstabilkan keadaan tersebut.

Di beberapa kota di Indonesia juga banyak berkembang perkebunan hidroponik bahkan sudah memiliki cabang kebun di wilayah yang berbeda. Hal ini menimbulkan permasalahan tentang mekanisme pelaporan monitoring keadaan hidroponik, khususnya bagi petani pada skala industri, dikarenakan kebun yang harus dipantau semakin banyak dan beragam. Sistem pemantauan secara konvensional harus menunggu rekap dari petani, dimana petani harus menyelesaikan *jobdesk* terlebih dahulu yang menyebabkan kondisi yang dilaporkan tidak *real time.* Terutama pada situasi COVID-19 yang menyulitkan proses mobilisasi karyawan yang terlibat. Hal ini dapat mempengaruhi tingkat akurasi data keadaan kebun dan tingkat efisiensi kinerja karyawan.

B. PERMASALAHAN

Belum adanya mekanisme pelaporan keadaan kebun hidroponik dengan memanfaatkan teknologi digital menyebabkan kurang efektifnya pelaporan secara konvensional untuk tetap diterapkan pada pertanian hidroponik skala industri. Dengan

sistem ini, diharapkan dapat mempermudah *user* dalam memberikan laporan keadaan dan memantau kondisi lingkungan kebun hidroponik yang menunjang kualitas tanaman dan hasil panennya.

C. BATASAN MASALAH

Untuk memfokuskan permasalahan yang diangkat, maka batasan masalah tersebut diantaranya adalah :

a) Aplikasi *mobile* yang dikembangkan untuk Android.
b) Setiap akun hanya dapat memiliki 1 jenis *role* yang tidak dapat diubah, yaitu *Owner* (pemilik)
c) Setiap akun hanya terintegrasi dengan 1 kebun. *Owner* hanya dapat memiliki 1 kebun.
d) Setiap kebun dapat memiliki beberapa kit yang dapat dibuat oleh setiap anggota kebun.
e) Data monitoring kebun dapat diinput manual melalui aplikasi yang dirancang.
f) Pengelolaan tanaman yang ditanam hanya meliputi penambahan tanaman.
g) Pengembangan fitur lainnya dapat dilakukan selama tidak melebihi waktu pengembangan yang disediakan.

D. TUJUAN DAN MANFAAT

Tujuan dari aplikasi yang dikembangkan ini adalah untuk meningkatkan kinerja petani hidroponik dengan memudahkan pelaporan hasil monitoring berbasis aplikasi sehingga data dapat diakses secara *real time* oleh petani maupun *stakeholder* kebun di berbagai lokasi serta dapat menunjang kualitas tanaman, Diharapkan aplikasi ini dapat memberikan manfaat berupa :

a) Membantu petani hidroponik dalam memudahkan system database dan pelaporan hasil monitoring.
b) Menjadi ilmu dan media informasi pembelajaran mengenai pengembangan aplikasi *mobile.*
c) Memberikan referensi mengenai pengembangan fitur yang lebih baik untuk aplikasi sejenis.

BAB II
LANDASAN TEORI

A. ANDROID

Android adalah sebuah *operating system* untuk *mobile device* berbasis Linux. Android merupakan salah satu OS yang bersifat *open source*, sehingga banyak pihak yang bereksplorasi dengan OS ini. Sebagai contohnya yaitu *mobile devices* yang menggunakan Android sebagai OS nya, seperti *smartphones* dan *tablet*, dapat memiliki *graphical user interface* (GUI) yang berbeda meskipun menerapkan OS yang sama karena kostumisasi yang dilakukan setiap developernya. Tidak hanya GUI-nya, beberapa *mobile devices* juga memiliki *built-in application* dan juga mendukung *third-party programs* lainnya [1]. Hal ini dimanfaatkan sebagai ciri khas dari brand *mobile devices* tersebut.

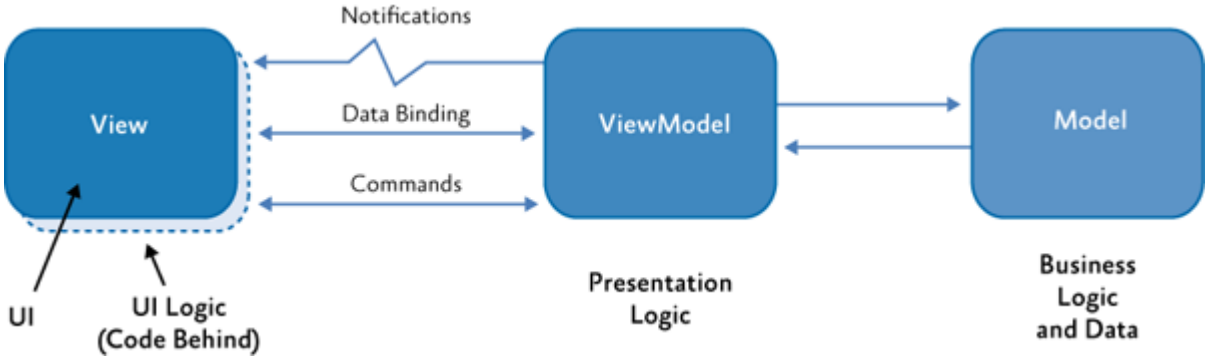
Dikenal sebagai produk yang dikembangkan oleh Google (GOOGL), namun pengembangan awalnya diinisasi oleh Android Inc. pada 2003 yaitu sebuah perusahaan software yang berbasis di Silicon Valley sebelum diakuisisi oleh Google pada 2005 [2]. Android dirilis pertama kali ke publik pada Oktober 2008 dengan versi Android 1.0, dimana *codename* yang umum dikenal sekarang belum diterapkan pada versi 1.0 dan 1.1. Semenjak versi 1.5 hingga saat ini, setiap versi memiliki *codename* yang unik, untuk versi 1.5 hingga 9 memiliki *codename* dengan tema makanan sedangkan versi sejak 10 hingga saat ini, versi terbaru Android 12 yang dirilis pada 4 Oktober 2021, menerapkan urutan angka sebagai *codename*-nya [3].

Developer dapat membuat aplikasi untuk Android menggunakan Android *support developer kit* (SDK) yang berbasis Java, sehingga mendukung bahasa pemograman turunan dari Java seperti Kotlin [2]. Biasanya SDK sudah terintegrasi dengan *integrated development environment* (IDE) yang mendukung SDK tersebut agar memudahkan developer dalam mengembangkan aplikasi tersebut. IDE untuk Android yang dikembangkan oleh Google adalah Android Studio yang juga mendukung *build system* yang lebih fleksibel, sehingga memungkinkan untuk membuat banyak variasi aplikasi untuk *mobile devices* yang berbeda dalam satu proyek. Dengan kata lain, Android SDK adalah "otak" untuk membuat aplikasi dan Android Studio adalah *tools* yang digunakan

untuk membuat aplikasi tersebut, sehingga sangat memungkinkan untuk membuat aplikasi dengan *tools* lainnya [4].

Dalam membangun aplikasi Android, menerapkan *architectural pattern* Android akan memudahkan dalam proses pengembangan karena *block codes* yang dibangun menjadi lebih terstruktur dan terorganisir. Beberapa arsitektur yang banyak digunakan adalah *Model-View-Controller* (MVC), *Model-View-Presenter* (MVP), dan *Model-View-ViewModel* (MVVM).

1. ARSITEKTUR MVVM

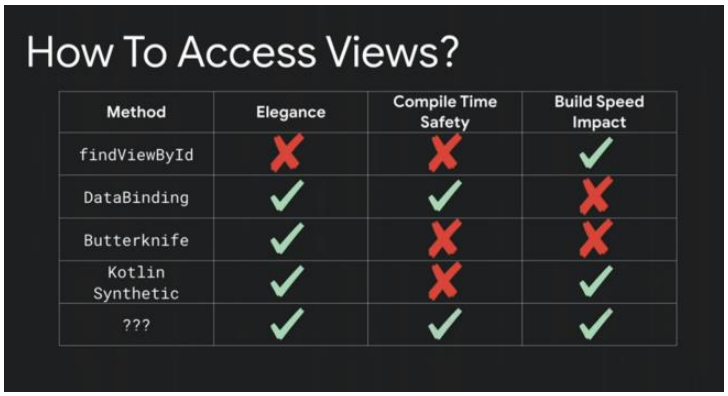


Gambar 1. Ilustrasi arsitektur MVVM

Sumber : https://docs.microsoft.com/

Arsitektur MVVM (*Model-View-ViewModel*) merupakan arsitektur yang direkomendasikan oleh Android, dimana memisahkan antara *business logic* dengan GUI. Arsitektur MVVM dibagi menjadi 3 layer, yaitu *Model Layer* berupa representasi dari data yang digunakan seperti *Plain Old Java Object* (POJO), *Data Class,* dan lainnya, *View Layer* berupa representasi dari *user interface* pada aplikasi seperti *Activity* dan *Fragment,* dan *ViewModel Layer* yang berinteraksi dengan Model dan menyalurkan data ke *View Layer* [5]. Dengan menerapkan arsitektur ini, data akan tetap konsisten meskipun terjadi konfigurasi pada GUI seperti perubahan orientasi *mobile device* saat terjadi *rotate* dari *portrait* ke *landscape* maupun sebaliknya.

Untuk berinteraksi dengan komponen yang ada pada Layout XML yang berada pada *View Layer*, banyak cara yang dapat dilakukan seperti findViewById, DataBinding, Butterknife, dan Kotlin Syntethic. Namun berdasarkan Gambar 2, masing-masing metode yang telah disebutkan di atas memiliki kekurangan yang dijawab melalui metode terakhir. Metode terakhir dan terbaru yang dikembangkan oleh Android ini bernama *View Binding* yang dirilis pada Februari 2020 [6].



Gambar 2. Perbandingan cara mengakses komponen pada *View Layer*

Sumber : https://www.youtube.com/

2. VIEW BINDING

Gambar 3. Cara penggunaan *View Binding*

Sumber : https://medium.com/

View Binding adalah metode untuk mengakses komponen pada Layout XML dengan cara memanggil id dari komponen yang berkaitan dari *binding layout* yang digunakan seperti pada Gambar 3. Hal ini bisa dilakukan karena setiap kali ada perubahan pada Layout XML, *View Binding* akan otomatis menyimpan komponen tersebut dalam sebuah variable sesuai dengan id yang didefinisikan pada Layout XML dan langsung dapat digunakan pada layout yang bersesuaian. Untuk menggunakan *View Binding,* pastikan *minimum requirement* berupa Android Studio 3.6 dan Android Gradle Plugin 3.6 terpenuhi. Kemudian dapat mengonfigurasi pada build.gradle dengan menambahkan pada `build.gradle [7]`.

```
android{
    ...
    buildFeatures{
        viewBinding true
    }
}
```

Kode 1. Mengaktifkan *View Binding* pada build.gradle

B. FIREBASE

Firebase adalah salah satu layanan dari Google yang menyediakan beragam fitur untuk dapat digunakan pada produk digital, seperti aplikasi atau *website,* dan dapat menerapkan banyak fitur dalam satu proyek sekaligus. Hal ini mempercepat pekerjaan developer dikarenakan mengurangi beban kerja terhadap bagian *back end* sehingga bisa focus pada pengembangan produk digital itu sendiri. Layanan pertama Firebase adalah Firebase Real Time Database pada 2011 (oleh Andrew Lee dan James Tamplin) sebelum diakusisi oleh Google pada 2014 dan berkembang sebagai layanan yang menyediakan beragam fitur. Firebase pun dapat digunakan secara gratis (*billing plan* Spark) maupun berbayar sesuai dengan pemakaiannya (*billing plan* Blaze) [8].

Karena Firebase dan Android merupakan produk yang sama-sama dikembangkan oleh Google, maka disediakan pula konfigurasi yang mudah untuk menghubungkan antar kedua layanan tersebut. Dari Android Studio, pengguna dapat langsung menggunakan Firebase melalui menu *Tools* dan memilih layanan Firebase yang akan digunakan. Dari sisi Firebase, pengguna perlu mendaftarkan aplikasi Android pada *Project Settings*. Kedua konfigurasi ini dapat dilakukan sekaligus memilih layanan Firebase, dikarenakan pengguna diharuskan untuk menghubungkan Firebase dan memasukkan *dependency library* yang bersangkutan hanya dengan menekan tombol.

1. FIREBASE AUTHENTICATION

Firebase Authentication adalah layanan Firebase untuk proses autentikasi ke produk digital yang dibuat. Autentikasi, yaitu proses validasi identitas saat memasuki suatu sistem, yang didukung oleh Firebase Authentication di antaranya yaitu dengan email, nomor telepon, secara anonymous, maupun provider lain seperti Google, Facebook, GitHub, Apple, Microsoft, Twitter, Yahoo, dan sebagainya. Firebase Authentication juga terintegrasi dengan layanan Firebase lainnya [8]. Berdasarkan dokumentasi Firebase Authentication untuk Android [9], terdapat beberapa *syntax* yang membangun fungsi-fungsi dasar Firebase Authentication seperti mengecek *auth* terkini (apakah ada yang aktif atau

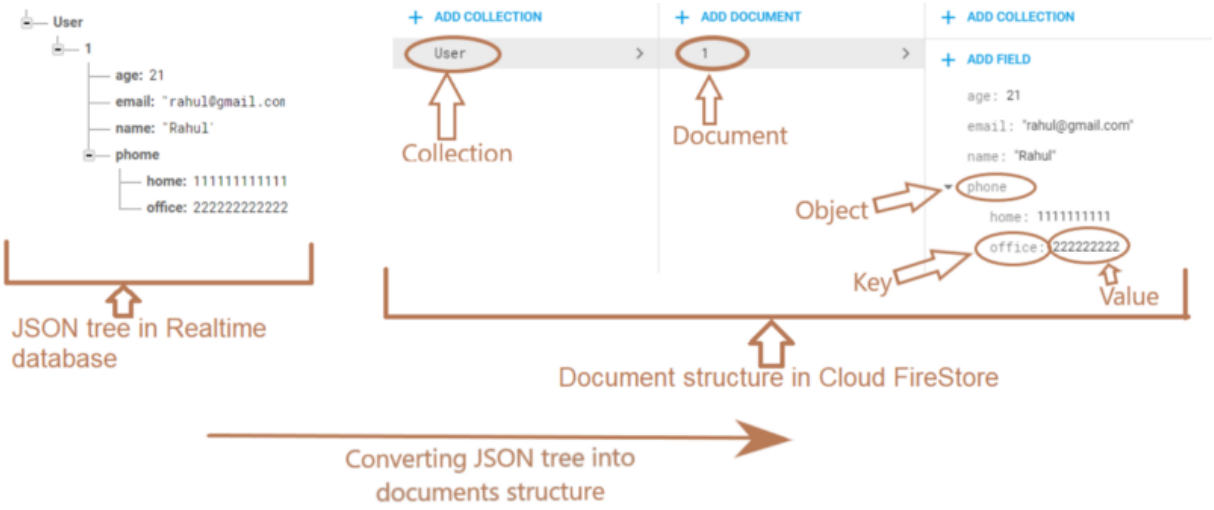
```
fun firebaseAuth(email:String, password:String){
    val auth : FirebaseAuth = Firebase.auth

    /** Check current user auth **/
    when(auth.currentUser){
        null -> "No currentUser data"
        else -> "currentUser is found"
    }

    /** Sign Up new user**/
    auth.createUserWithEmailAndPassword(email, password)

    /** Sign In with email password **/
    auth.signInWithEmailAndPassword(email, password)

    /** Sign Out current user **/
    auth.signOut()
}
```

Kode 2. *Syntax* dasar pada Firebase Authentication

tidak), melakukan *sign in* dan *sign out*, maupun sebagainya yang umum digunakan pada aplikasi *mobile* yang ditunjukkan melalui Kode 2.

2. FIREBASE CLOUD FIRESTORE

Firebase menyediakan 2 jenis layanan database, yaitu Firebase Real Time Database dan Firebase Cloud Firestore. Keduanya dapat langsung tersinkronisasi secara *real time* ke setiap pengguna. Hal ini sangat memudahkan developer karena saat membuat produk digital lintas platform, saat terjadi perubahan data mana semua pengguna akan mendapatkan *update* secara serentak dan otomatis [8].



Gambar 4. Perbandingan struktur data Real Time Database dan Cloud Firestore

Sumber : https://stackoverflow.com/

Namun yang membedakan keduanya adalah struktur datanya seperti pada Gambar 4. Firebase Real Time menyimpan data dalam bentuk JSON sedangkan Firebase Firestore disimpan dalam bentuk *collection* berupa kumpulan *document* yang berisi *key-value*. Firebase Cloud Firestore hadir untuk memudahkan pengembangan *mobile application*. Sebagai contoh, kedua database dapat melakukan sort atau filter. Namun, pada Firebase Cloud Firestore juga

```
fun firebaseRealTimeDatabase(id:Int, data:HashMap<String,Any>){
    val db : FirebaseDatabase = Firebase.database

    /** Add Data **/
    db.getReference( path: "users").child(id.toString()).setValue(data)

    /** **/
    db.getReference( path: "users").child(id.toString()).removeValue()

    /** Order Datas **/
    db.getReference( path: "users").orderByChild( path: "age")

}

fun firebaseCloudFirestore(id:Int, data:HashMap<String,Any>){
    val db : FirebaseFirestore = Firebase.firestore

    /** Add Data **/
    db.collection( collectionPath: "users").document(id.toString()).set(data)

    /** Delete Data **/
    db.collection( collectionPath: "users").document(id.toString()).delete()

    /** Order Datas **/
    db.collection( collectionPath: "users").orderBy( field: "age")

    /** Order with Querry **/
    db.collection( collectionPath: "users")
        .whereEqualTo( field: "name",  value: "%fer%")
        .orderBy( field: "age")
}
```

Kode 3. Perbandingan *syntax* Real Time Database dan Cloud Firestore

dapat menambahkan *querry* [10]. Perbandingan *syntax* antara kedua layana Firebase Database dijabarkan melalui Kode 3.

3. FIREBASE STORAGE

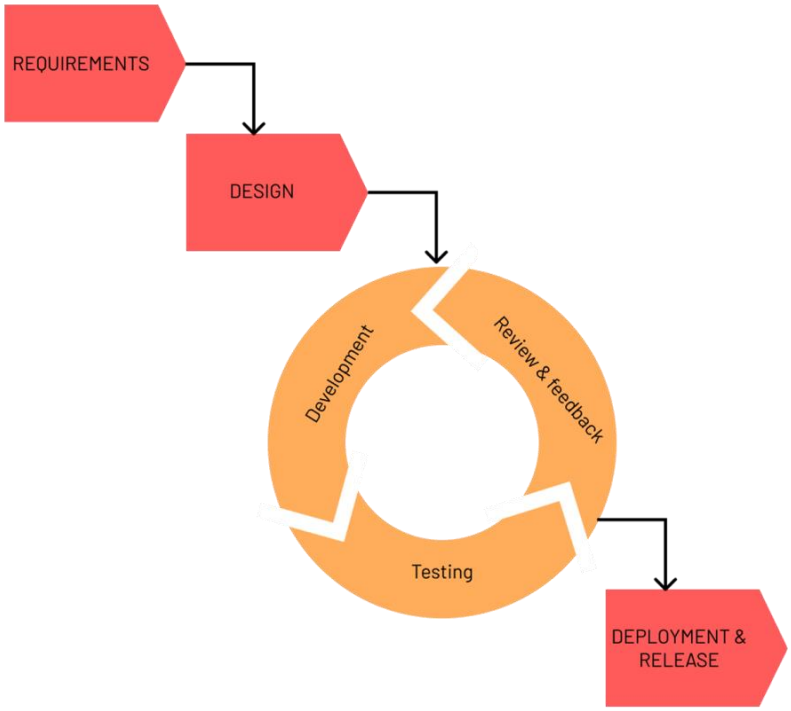
Firebase Storage adalah layanan untuk menyediakan penyimpanan file seperti gambar, dokumen, dan video. Layanan ini terintegrasi dengan Firebase Authentication, sehingga dapat digunakan untuk media penyimpanan berkas *client-based.* Setelah mengunggah berkas, maka akan mendapatkan url berkas sehingga dilihat dan dimodifikasi oleh *client* (dibutuhkan mekanisme validasi antara penggunggah dan *client* yang mengakses berkas tersebut). Berdasarkan dokumentasi Firebase Storage untuk Android [11], terdapat beberapa *syntax* dasar yang membangun fungsi-fungsi dasar Firebase Storage seperti untuk *upload* dan *download* file. Sebagai contoh pada Kode 4 menunjukkan cara untuk menggunggah foto untuk *user profile.*

```
fun firebaseStorage(
    reference : String = "profile_images",
    fileUri:Uri){
        val storage : FirebaseStorage = Firebase.storage

        /** Define storage reference (location of file) **/
        val storageReference : StorageReference =
            storage.getReference(reference)

        /** Upload file **/
        storageReference.putFile(fileUri)
            .continueWithTask {  it: Task<UploadTask.TaskSnapshot!>
                storageReference.downloadUrl
            }
}
```

Kode 4. *Syntax* dasar untuk upload foto profil pada Firebase Storage

BAB III
METODE PENGEMBANGAN

A. METODE PENGEMBANGAN



Gambar 5. Ilustrasi metode pengembangan *Hybrid*

Sumber : https://cleancommit.io/

Dalam menjalankan kegiatan, pengembangan aplikasi ini menerapkan *Project Management Method* berupa *Hybrid* atau kombinasi metodologi *Waterfall* dan *Agile* dikarenakan proyek yang dilaksanakan sudah terstruktur namun membutuhkan fleksibilitas dalam pelaksanannya. Fitur atau layanan produk yang ingin dihasilkan dapat mengalami perubahan, dimana terbuka dengan eksperimen dan penambahan selama sesuai dengan tujuan yang dibutuhkan. Setiap tahapan memiliki fokus cakupan pekerjaan yang berbeda-beda sesuai dengan tujuan yang dicapai dari setiap tahapan.

1. INITIATION (REQUIREMENTS)

Tahapan ini menjadi fundamental pada pengembangan aplikasi yang mencakup :

a) Menentukan latar belakang sehingga menjadi topik yang akan dikerjakan dan menyediakan solusi dari permasalahan yang diangkat.

b) Membatasi kompleksitas proses pengembangan aplikasi sehingga dapat menentukan anggota kelompok dan pembagian jobdesk yang diinginkan.

2. PLANNING (DESIGN)

a) Menentukan rancangan fitur, layanan, dan kebutuhan utama dari solusi yang ingin disediakan.

b) Menentukan kebutuhan dan melakukan perancangan atau desain dari keseluruhan aplikasi mencakup :

(1) Tampilan *front end* dan data yang ditampilkan maupun diolah untuk *user.*

(2) Mekanisme *front end* agar aplikasi mudah dan nyaman untuk digunakan, seperti *loading progress* dan aktivasi *button*.

(3) Struktur database, apakah berbentuk JSON atau berupa *collection-document.*

(4) Jenis layanan Firebase yang akan digunakan, apakah Database Realtime, Firestore, dsb.

(5) Hal-hal lain yang berkaitan dengan input/output data dari *user* ke aplikasi maupun sebaliknya.

3. EXECUTING (DEVELOPMENT, TESTING, REVIEW & FEEDBACK)

Tahapan ini membutuhkan fleksibilitas karena memungkinkan adanya penyesuaian fitur yang ingin dibangun selama tetap searah dengan tujuan yang ingin dicapai. Pada bagian ini pula terjadi integrasi antara *front end* dan *back end* dapat membangun system yang dihendaki dan melakukan testing terhadap system tersebut. Untuk *front* end, tahapan ini berfokus pada apakah *user interface* dapat menampilkan dan menerima *input* sesuai dengan algoritma yang dibangun. Sedangkan *back end* berfokus pada konsistensi data, dimana data yang ditampilkan untuk *user* akan tetap sama sesuai dengan data yang diinputkan sebelumnya. Sehingga aplikasi dapat memberikan respon dari *user* sesuai dengan algoritma yang dibangun.

4. CLOSSURE (DEPLOYMENT & RELEASE)

Tahapan ini menjadi akhir dari proses pengembangan yang dilaksanakan setelah aplikasi selesai dibangun dan berjalan sebagaimana mestinya. Finalisasi dan evaluasi perlu dilakukan agar memastikan layanan yang disediakan sesuai dengan perancangan awal. Kegiatan ini berakhir dengan dokumentasi akhir berupa pembuatan laporan dan proses presentasi.

B. RENCANA JADWAL PENGEMBANGAN

Tabel 1. Rencana jadwal pengembangan

No	Rincian Kegiatan	Hari ke-														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1.	Pencarian dan Pengembangan Topik															
2.	Menganalisa kebutuhan, spesifikasi, dan fitur															
3.	Mendesain struktur database															
4.	Mendesain *front end* (*wireframe*)															
5.	Menentukan blok diagram data dan *front end* yang terlibat															
6.	Melakukan pengkodean dan penyesuaian															
7.	Finalisasi dan Evaluasi Akhir															
8.	Pelaporan dan dokumentasi akhir															

C. SUMBER DAYA

1. PERLENGKAPAN PERANGKAT KERAS

a) Komputer Lab 28

Sebagai *device* untuk membuat aplikasi saat dikerjakan di BBPLK Bekasi dengan spesifikasi sebagai berikut :

- *Processor* : Intel Core i7-7700
- *Graphic Card* : AMD Radeon R7 450
- *Storage Unit* : 1TB HDD
- RAM : 16 GB
- *Operating System* : Windows 10 Enterprise 64-bit

b) Asus ROG Strix GL-553VD

Sebagai *device* untuk membuat aplikasi saat dikerjakan di luar jam aktif BBPLK Bekasi dengan spesifikasi sebagai berikut :

- *Processor* : Intel Core i7-7700HQ
- *Graphic Card* : NVIDIA GeForce GTX 1050
- *Storage Unit* : 1TB HDD + 128 GB SSD
- RAM : 16 GB
- *Operating System* : Windows 10 Education 64-bit

c) Xiaomi Redmi Note 8 Pro

Sebagai *device* untuk menjalankan aplikasi pada perangkat *mobile* eksternal dengan spesifikasi sebagai berikut :

- *CPU* : Octa-core
- *GPU* : Mali-G76 MC4
- *Android Version* : Android 11 (API 30)
- *Storage Unit* : 128 GB (Internal)
- RAM : 6 GB

2. PERLENGKAPAN PERANGKAT LUNAK

a) Android Studio Arctic Fox

Sebagai IDE dalam pembuatan aplikasi Android dengan spesifikasi sebagai berikut :

- *Version* : 2020.3.1.Patch 2
- *VM* : OpenJDK 64-bit Server VM

b) Firebase Console

Sebagai sistem penyedia layanan untuk database dalam aplikasi dengan spesifikasi sebagai berikut :

- *Billing Plan* : Spark (free $0/month)
- *Firebase BoM Version* : 28.4.1 (September 13, 2021)

BAB IV
HASIL DAN PEMBAHASAN

A. INITIATION (REQUIREMENTS)

Pada tahap ini dilakukan pemilihan topik dan metode pengembangan melalui metode studi literatur, yaitu metode pengumpulan data berdasarkan referensi yang relevan dengan topik berkaitan. Literatur utama yang digunakan bersumber dari penulis pribadi berupa tugas mata kuliah dan portfolio saat uji kompetensi *IT Project Management* yang kemudian disesuaikan dengan materi yang didapatkan selama Pelatihan Berbasis Kompetensi (PBK) *Mobile Application*. Topik monitoring kebun hidroponik dipilih dengan pertimbangan bahwa latar belakang dan permasalahan yang dijabarkan sudah di alami oleh petani hidroponik di Surabaya. Sedangkan metode pengembangan *Hybrid* dipilih karena sudah ada kebutuhan awal untuk pengembangan aplikasi dan dapat mempercepat proses di tahap ini, namun dibutuhkan fleksibilitas agar dapat dilakukan penyesuaian dalam proses selanjutnya.

B. PLANNING (DESIGN)

Pada pengembangan kali ini, perancangan dilakukan berdasarkan bagian *front end* yang berkaitan dengan penyajian dan alur navigasi antar tampilan yang bersesuaian dan *back end* untuk mekanisme pengolahan data pada aplikasi. Agar kedua bagian ini dapat berfungsi sebagaimana mestinya, dimana aplikasi dapat merespon dan memberikan *output* sesuai *input* yang diberikan *user*, maka dibutuhkan perancangan *storyboard* penggunaan sebagai referensi algoritma keseluruhan.

1. DESAIN STORYBOARD

Alur penggunaan aplikasi ini dijelaskan sebagai berikut :

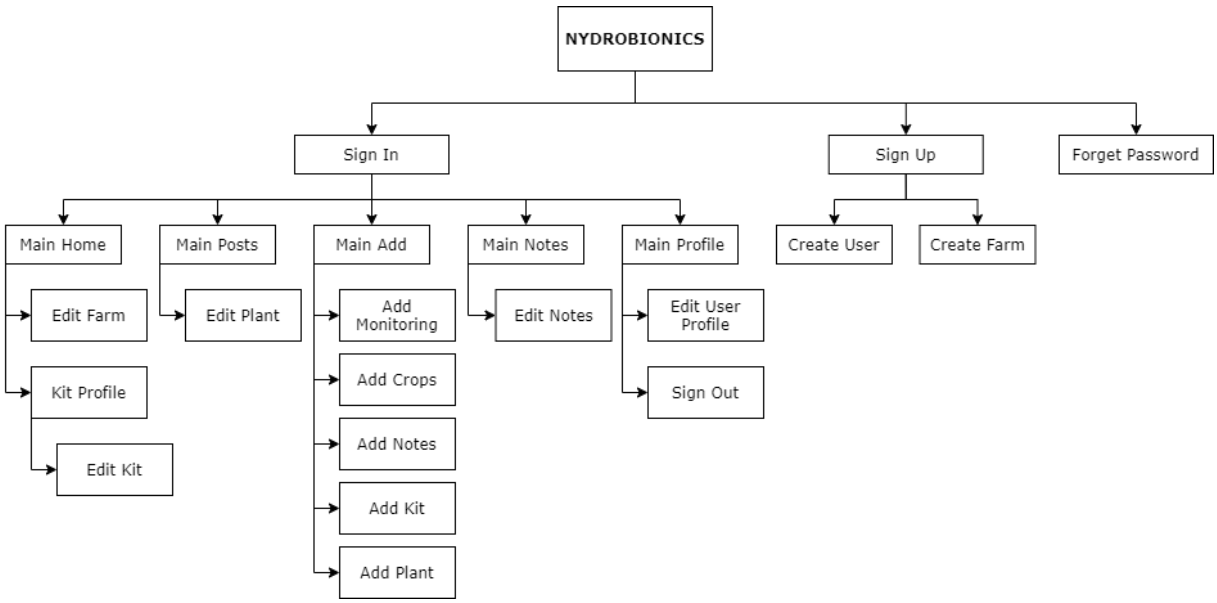
a) Pengguna melakukan autentikasi menggunakan email dan *password* yang telah didaftarkan sebelumnya.
b) Jika belum memiliki akun, maka pengguna melakukan *sign up* dengan email dan *password* kemudian membuat akun dengan memberikan data diri seperti nama, tanggal lahir, alamat, dsb. termasuk jenis akun yang akan dibuat, *owner* atau *staff.* Jika berjenis *owner*, pengguna diharuskan untuk membuat data kebun yang dimilikinya dan memberikan akses untuk *staff* yang sudah memiliki akun.

c) Jika lupa dengan *password*, maka pengguna dapat melakukan *forget password* dengan email yang terdaftar lalu melalukan verifikasi dan membuat *password* baru melalui email sebelum dapat menggunakan Nydrobionics.

d) Jika proses autentikasi berhasil dilakukan, maka aplikasi mengvalidasi apakah akun tersebut sudah melengkapi data pribadinya. Jika belum, maka diarahkan untuk mengaturnya seperti pada proses *sign up* (tanpa mendaftarkan email). Jika sudah, maka pengguna dapat menggunakan fitur-fitur yang dikembangkan.

2. DESAIN FRONT END

a) Struktur Menu Aplikasi



Gambar 6. Struktur menu aplikasi

Sumber : Dokumentasi pribadi

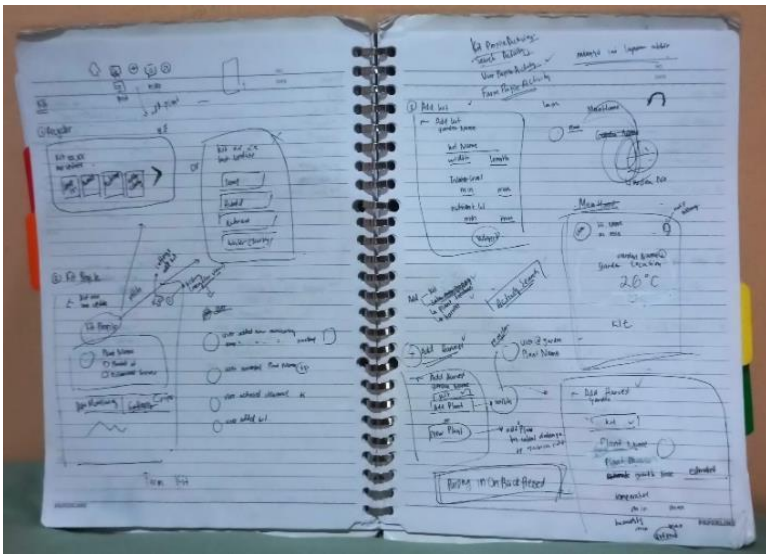
Struktur menu aplikasi Nydrobioncs ditunjukkan melalui Gambar 6. Saat pengguna membuka aplikasi, maka ada 3 menu yang dapat dipilih, yaitu *Sign In*, *Sign Up*, dan *Forget Password*. *Sign In* berguna untuk memproses autentikasi. *Sign Up* berguna memproses pembuatan akun yang dilanjutkan untuk membuat profil (*Create User*) dan membuat kebun (*Create Farm*) jika akun berjenis *Owner.* Sedangkan *Forget Password* memproses permintaan pergantian *password* yang dikirimkan melalui email yang terdaftar.

Menu utama aplikasi tediri dari 5 menu yang ditampilkan jika proses autentikasi berhasil dilakukan. *Main Home* sebagai menu awal menampilkan profil kebun, data terakhir monitoring kit serta memungkinkan pengguna untuk

mengubah data kebun (*Edit Farm*), melihat detail setiap kit (*Kit Profile*), dan mengedit setiap kit (*Edit Kit*). *Main Social* menampilkan tanaman yang tersimpan pada database dan memungkinkan pengguna yang membuat untuk mengubahnya (*Edit Plant*). *Main Add* menampilkan pilihan untuk menambahkan data-data yang terlibat pada aplikasi, seperti *Add Monitoring, Add Crops, Add Notes, Add Kit,* dan *Add Plant*. *Main Notes* menampilkan jadwal dan catatan yang dibuat oleh pengguna terkait. Sedangkan *Main Profile* menampilkan detail profil dari akun yang sedang aktif dan memungkinkan untuk mengubahnya (*Edit Profile*) maupun memutus akses akun terhadap aplikasi (*Sign Out*).

b) Kerangka Tampilan Aplikasi

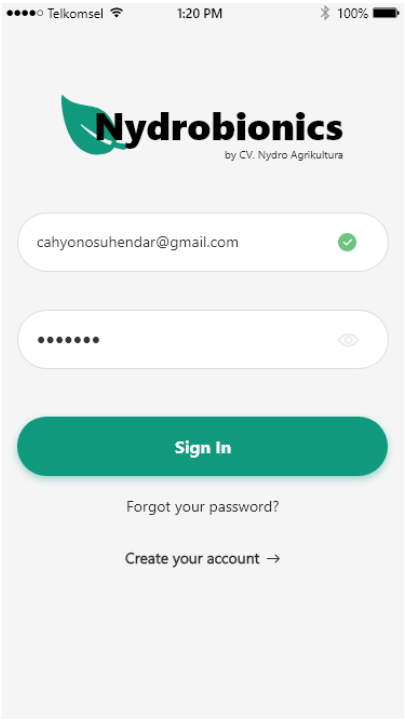
Berdasarkan struktur menu aplikasi yang dibuat, maka dapat dikembangkan kerangka tampilan aplikasi dari setiap menu yang telah ditentukan. Pada pengembangan kali ini, pembuatan kerangka tampilan aplikasi dilakukan secara manual gambar tangan sehingga hanya berupa *rough sketch* yang ditunjukkan pada Gambar 7. Seluruh proses desain kerangka tampilan hanya dilakukan dengan menggambar manual pada kertas dengan melakukan riset *front end* pada aplikasi-aplikasi yang memiliki fitur yang bersesuaian.



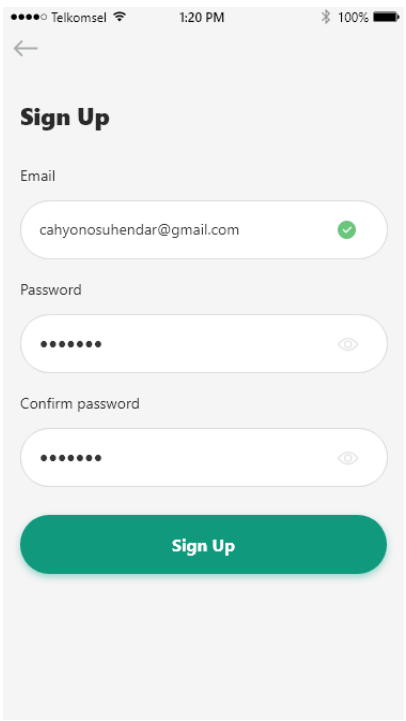
Gambar 7. *Mock up* aplikasi
Sumber : Dokumentasi pribadi

Selain itu digunakanlah *wireframe* (kerangka yang memiliki navigasi komponen) pada literatur terkait pada Gambar 8 sebagai pedoman yang disesuaikan terhadap fitur yang dibangun. *Wireframe* tersebut dibuat menggunakan Adobe XD memiliki 9 tampilan utama yang membangun fungsi

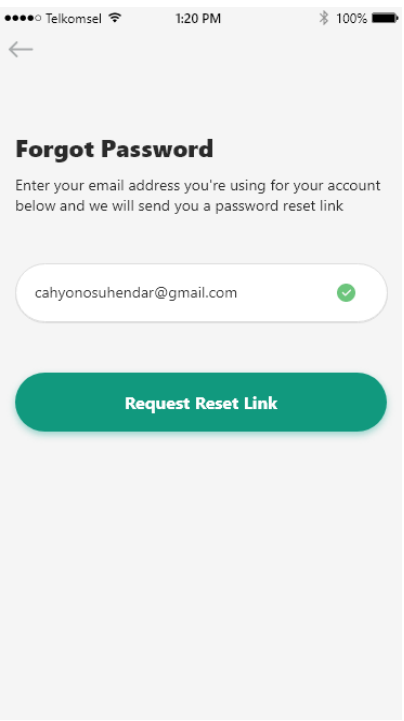
sign in (a), *sign up* (b), *forget password* (c), *main home* (d), *main farm* (e), *main schedule* (f), *main profile* (g), *kit profile* (h), dan *add schedule* (i). Desain *front end* final yang digunakan pada pengembangan kali ini langsung diimplementasikan melalui Layout XML yang bersangkutan.



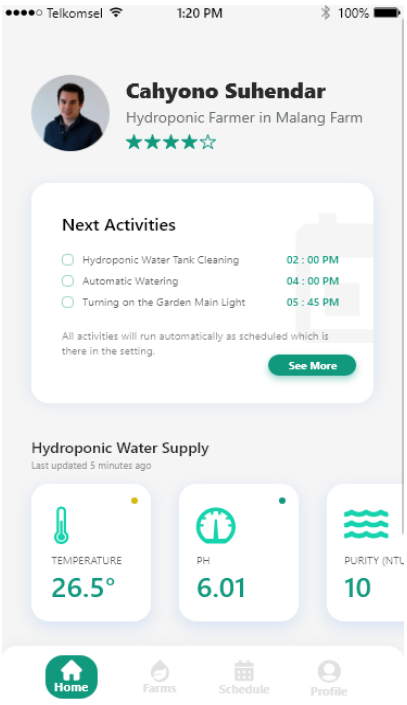
a) Menu *Sign In*



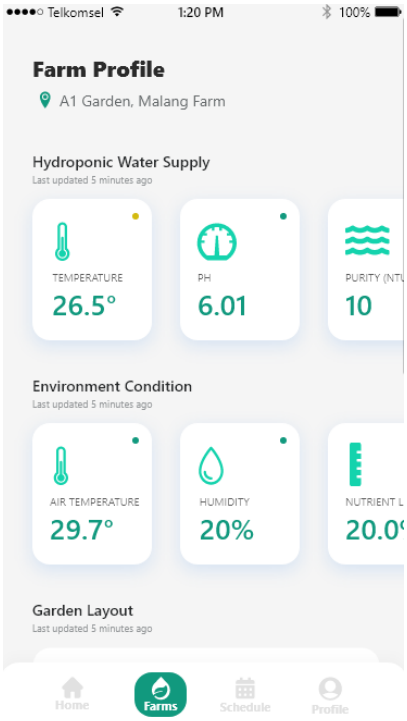
b) Menu *Sign Up*



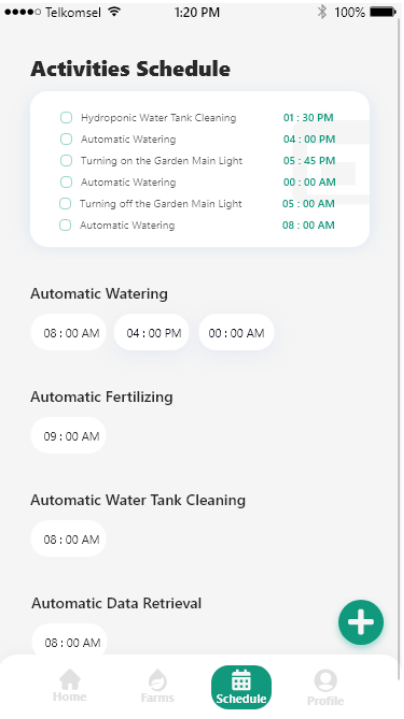
c) Menu Forget *Password*



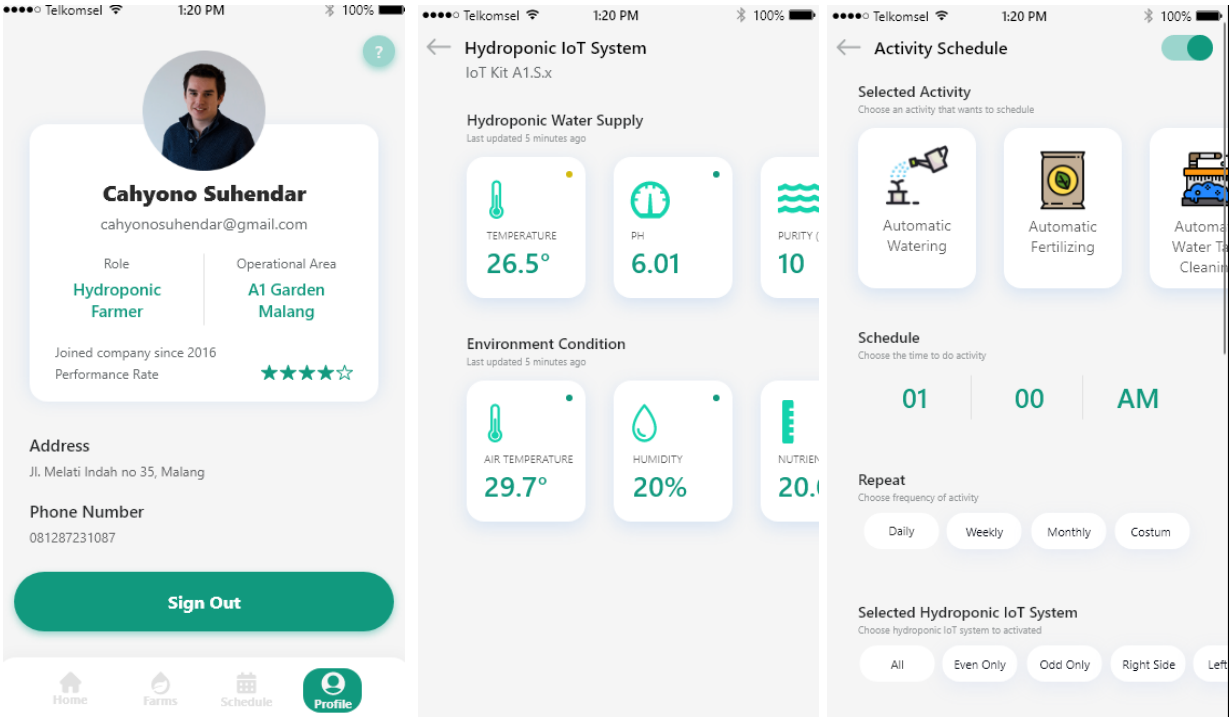
d) Menu *Main Home*



e) Menu *Main Farm*



f) Menu *Main Schedule*



g) Menu *Main Profile* h) Menu *Kit Profile* i) Menu *Add Schedule*

Gambar 8. *Wireframe* aplikasi yang diterapkan pada literatur

Sumber : Dokumentasi pribadi

3. DESAIN BACK END

a) *Entity Relationship Diagram* (ERD)

Berdasarkan desain *front end* kemudian ditentukan kebutuhan data dan atribut yang dapat memberikan fungsionalitas alur data yang diharapkan. Model atau rancangan untuk memenuhi kebutuhan data tersebut ditunjukkan melalui *entity relationship diagram* (ERD) pada Gambar 9. Nydrobionics membutuhkan 8 tabel data (nama tabel berbentuk prural dan data berbentuk singular, sebagai contoh tabel *users* dengan data *user*) dengan relasi antar tabel :

(1) Setiap *user* dapat memiliki banyak *note* dan *feedback*, *dataMonitoring*, *crops*, dan *plant* namun hanya memiliki 1 *farm*.

(2) Setiap *note* hanya memiliki 1 *user*.

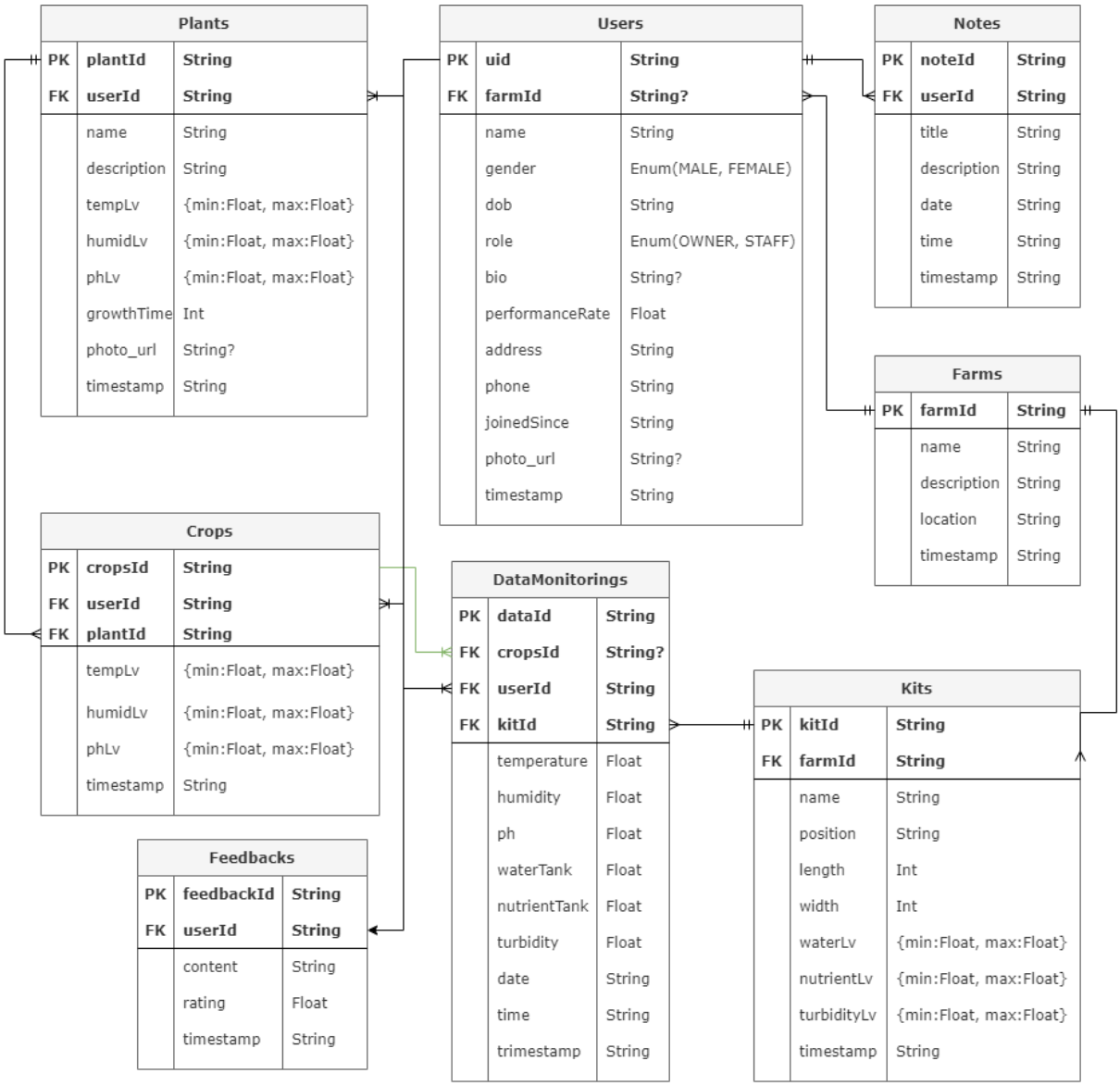
(3) Setiap *farm* dapat memiliki banyak *kit* dan *user.*

(4) Setiap *kit* dapat memiliki banyak *dataMonitoring* namun hanya memiliki 1 *farm.*

(5) Setiap *crops* dapat memiliki banyak *dataMonitoring* namun hanya memiliki 1 *user* dan 1 *plant*.

(6) Setiap *plant* dapat memiliki banyak *crop* dan *dataMonitoring* namun hanya memiliki 1 *user*.

(7) Setiap *feedback* hanya dimiliki oleh 1 *user*.



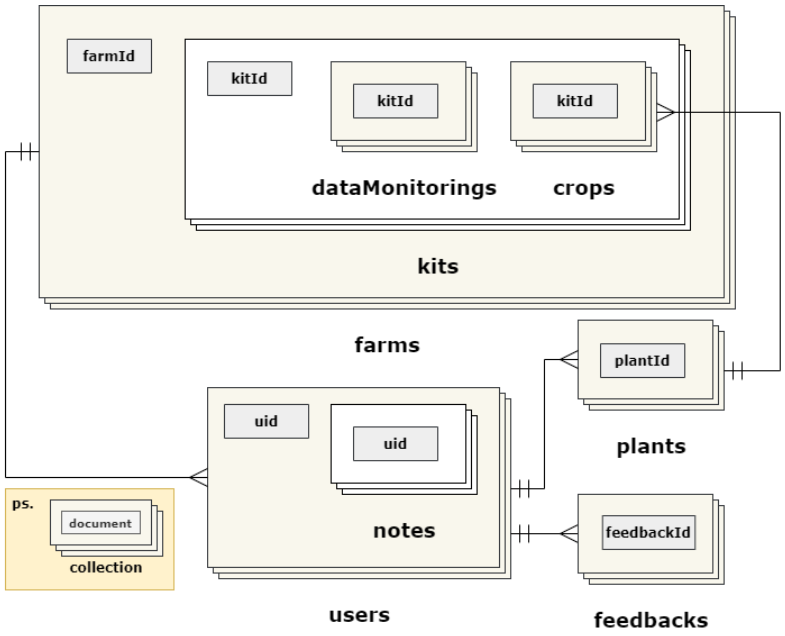
Gambar 9. *Entity Relationship Diagram* (ERD) aplikasi

Sumber : Dokumentasi pribadi

b) Struktur Database

Berdasarkan kebutuhan dan relasi data dari *Entity Relationship Diagram* (ERD), maka dapat ditentukan struktur database yang diterapkan sesuai dengan layanan penyedia penyimpanan yang digunakan. Nydrobionics menggunakan Firebase Cloud Firestore karena layanan tersebut menyediakan kemudahan untuk pengembangan *mobile application*. Karena Firebase Cloud Firestore menerapkan arsitektur *document collection*, maka atribut data yang ada pada

ERD disesuaikan ke dalam bentuk *document collection* dengan struktur database beserta relasi antar *collection* utamanya ditunjukkan pada Gambar 10.



Gambar 10. Struktur database *document collection* dan relasinya

Sumber : Dokumentasi pribadi

Proses yang dilakukan yaitu dengan menentukan tabel manakah yang dapat dijadikan sebagai *subcollection* sehingga *foreign key* yang menghubungkan antar tabel dapat dihilangkan. Relasi "*one to many*" dijadikan pedoman dalam penentuan tersebut hingga menghasilkan 3 *collection* utama, yaitu *users, farms* dan *plants* (nama *collection* berbentuk prural dan *document* berbentuk singular, sebagai contoh *document users* dengan data *user*) dengan sub-struktur yang berbeda seperti pada Gambar 11. Kegunaan dan relasi antar *collection-document* maupun *collection-collection* setara dengan relasi pada ERD, yaitu :

(1) Setiap *user* dapat memiliki banyak *plant* dan *feedback* dan hanya terikat dengan 1 *farm.*

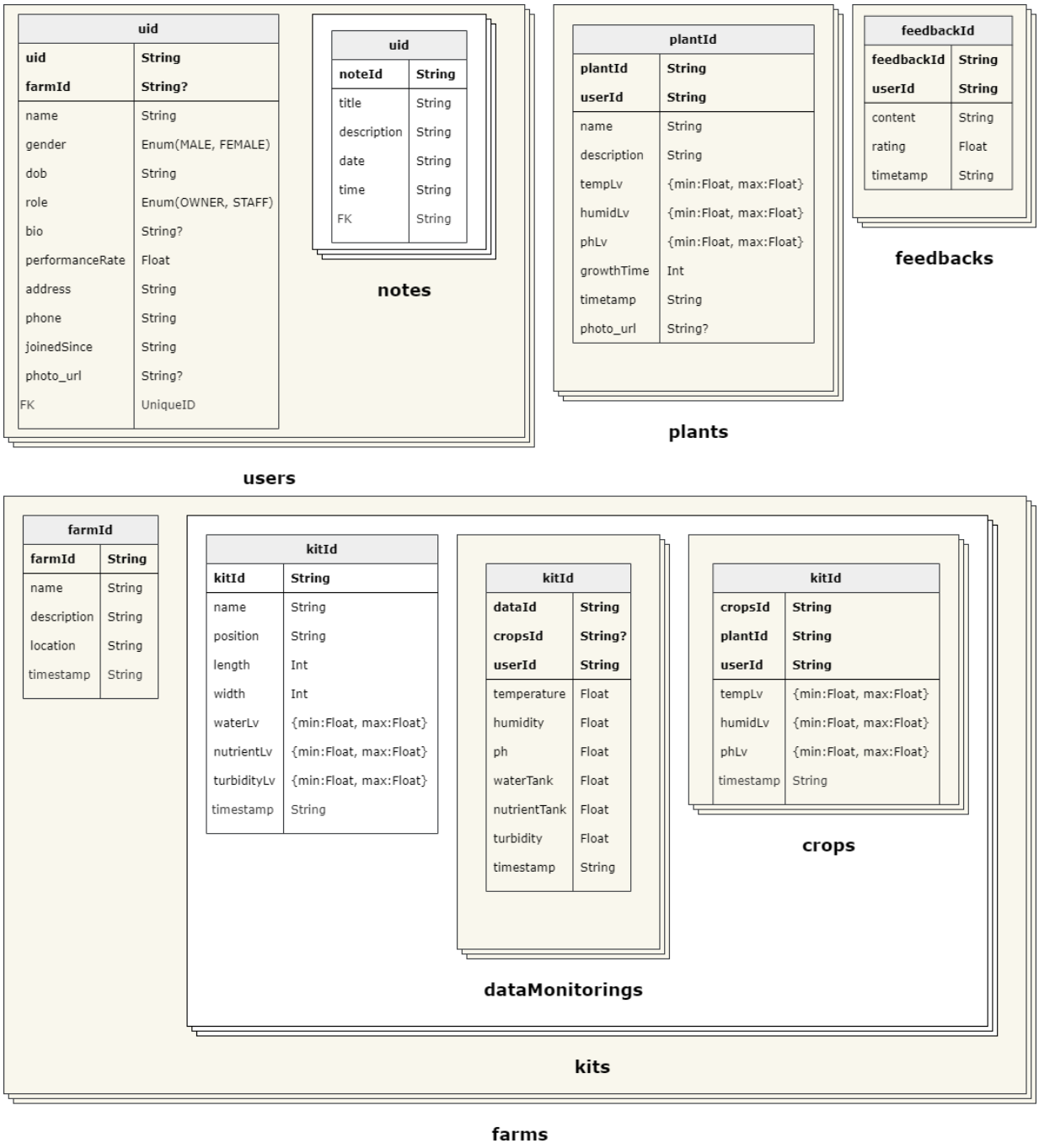
(2) Setiap *farm* dapat memiliki banyak *user*.

(3) Setiap *plant* dapat memiliki banyak *crops* (*subcollection farms*)dan hanya terikat dengan 1 *user*.

(4) Setiap *feedback* hanya terikat dengan 1 *user.*

Penghapus *foreign key* dari setiap table pada ERD saat menjadi *sub-collection* dari *collection* utama dilakukan untuk menghilangkan kemungkinan

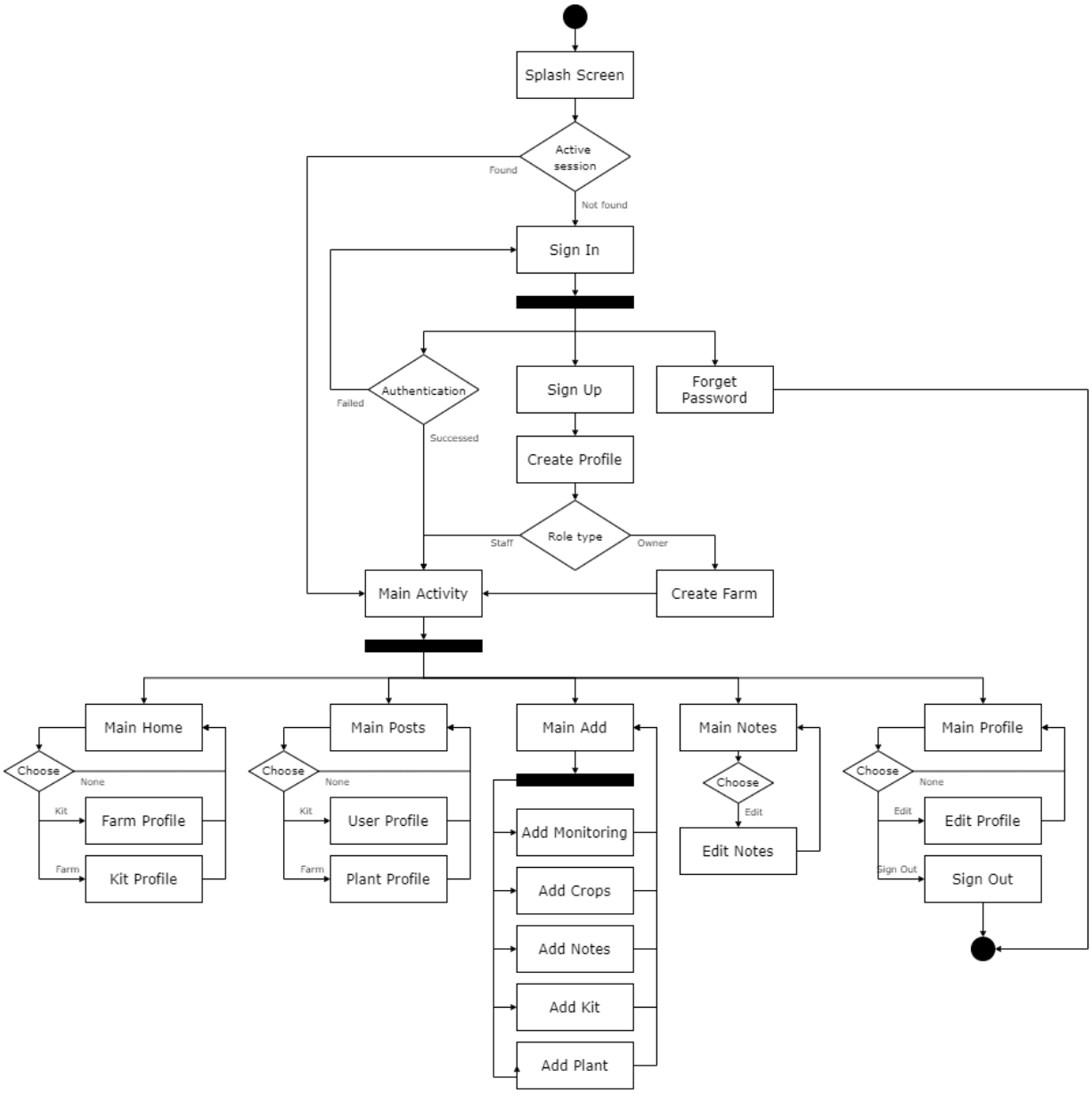
redudansi data. Sebagai contohnya untuk tabel *Notes* pada ERD (Gambar 9) memiliki *foreign key* `userId` yang terhubung dengan uid di tabel *Users,* disesuaikan sebagai *subcollection* dari *collection* utama *users* (Gambar 11) dengan *document id* yang digunakan adalah `uid` sehingga *foreign key* dapat dihilangkan.



Gambar 11. Detail struktur database beserta atribut datanya

Sumber : Dokumentasi pribadi

c) *Activity Diagram*



Gambar 12. *Activity Diagram* dari aplikasi

Sumber : Dokumentasi pribadi

Alur kerja dari aplikasi dijabarkan melalui *activity diagram* pada Gambar 12 yang dimulai pada *Splash Screen* dimana berfungsi untuk mengecek apakah ada akun yang sudah ter-*sign in* (*active session*) sebelumnya. Jika ada maka akan meminta *user profile* dari akun tersebut dan menuju ke *Main Activity*. Jika belum, maka menuju ke *Sign In*. Pada tahap ini, pengguna diberikan pilihan apakah ingin melalukan *Sign In* dengan email dan *password* (proses autentikasi), *Sign Up* yang dilanjut dengan *Create Profile*, atau *Forget Password* untuk meminta verifikasi *reset password* melalui email yang terdaftar. Pada *sign*

in dan *sign up* jika proses berhasil dilakukan maka diarahkan untuk ke *Main Activity*, sedangkan *forget password* mengharuskan pengguna untuk mengecek email sehingga mengakhiri alur kerja aplikasi.

Saat *Main Activity* ditampilkan, pengguna diberikan 5 pilihan *activity* yang bisa dilakukan dengan *Main Home* sebagai *default home*-nya. *Main Activity* bertindak sebagai pusat aplikasi yang memberikan pilihan untuk menampilkan seluruh data yang terlibat pada aplikasi. Pada *Main Home* melibatkan data tentang *farms* dan ringkasan data *kits* yang memungkinkan untuk menampilkan detail *kit*. *Main Social* memungkinkan pengguna untuk melihat profil *user* dan *plant* sesuai pilihannya. *Main Add* berfokus pada mekanisme penambahkan data sesuai dengan menu untuk menjalankan *activity* yang berkaitan. *Main Notes* menampilkan seluruh *note* yang pernah dibuat oleh pengguna dan bisa memodifikasi data tersebut. Dan *Main Profile* memberikan pengguna pilihan untuk mengedit *user profile*-nya atau melakukan *Sign Out* untuk mengakhiri aplikasi.

C. EXECUTING (DEVELOPMENT, TESTING, REVIEW & FEEDBACK)

Setelah *blueprint* aplikasi yang menyelesaikan solusi atas permasalahan ditentukan, maka implementasi desain menjadi kodingan dilakukan. Pada tahap ini memungkinkan adanya perubahan dari desain yang dibuat dikarenakan menyesuaikan kendala dan hambatan yang dialami. Agar pekerjaan yang dilakukan dapat dilacak keterselesaiannya demi mencapai target selesai tepat waktu, maka dilakukan *task break down* menjadi pengerjaan *View Layer* berupa Layout XML dan navigasi antar *activity*, pengerjaan *ViewModel* untuk menambah dan mengedit data Firebase, pengerjaan *ViewModel* untuk menampilkan data Firebase beserta adapter untuk data berbentuk list, dan *finishing* berupa menyesuaikan tema aplikasi dan supergrafis seperti ikon dan logo yang digunakan.

1. PROFIL APLIKASI

Nydrobionics adalah *mobile application* berbasis Android untuk monitoring kebun hidroponik agar memudahkan system pelaporan kondisi kebun dan meningkatkan akurasi data saat dicek dan dilaporkan. Aplikasi ini memungkinkan pengguna untuk memantau keadaan kebun hidroponik dengan tujuan dapat mengatur konsistensi variabel-variabel yang memengaruhi kualitas hasil panen. Terdiri dari 2 tipe akun, yaitu

Owner dan *Staff* yang memberikan batasan terhadap *jobdesk* setiap pengguna. Akun *Owner* sebagai pemilik kebun dapat mengatur anggota *staff* pada kebunnya dan mengalokasikan *staff* yang bertanggung jawab terhadap beban pekerjaannya. Dokumentasi dari keseluruhan project dapat diakses melalui GitHub maupun Google Drive.

Gambar 13. Logo aplikasi Nydrobionics
Sumber : Dokumentasi pribadi

Nydrobionics dibangun dengan minimum SDK 26 dan target SDK 30, sehingga dapat berjalan optimum pada *mobile device* Android 11 dan seminimalnya *mobile device* dengan *operating system* Android Oreo (8.0). Aplikasi ini juga membutuhkan penggunanya untuk memperbolehkan beberapa *permission* seperti :

a) Menggunakan jaringan internet (android.permission.INTERNET)
b) Mengakses media files (android.permission.READ_EXTERNAL_STORAGE)
c) Mengetahui kondisi jaringan (android.permission.ACCESS_NETWORK_STATE)
d) Mengakses lokasi yang tepat (android.permission.ACCESS_FINE_LOCATION)
e) Mengetahui perkiraan lokasi (android.permission.ACCESS_COARSE_LOCATION)
f) Mengetahui kondisi Wi-Fi(android.permission.ACCESS_WIFI_STATE)
g) Menggunakan kamera (android.permission.CAMERA)
h) Memulai telepon (android.permission.CALL_PHONE)

2. FITUR YANG DIKEMBANGKAN

Berikut ini beberapa fitur yang dikembangkan pada Nydrobionics :

a) Hydroponic Monitoring

Pengguna dapat menyimpan data keadaan kebun seperti kondisi suhu (°C), kelembaban udara (RH), tingkat keasaman air (pH), kejernihan air (NTU), kapasitas tanki air dan pupuk cair (L). Data yang telah tersimpan dapat dilihat dan dibandingkan dengan keseluruhan data sehingga dapat digunakan sebagai indicator untuk mengontrol kondisi tersebut sesuai dengan tanaman yang ditanam.

b) *Plant NydroDB*

Untuk mendukung monitoring keadaan kebun, Nydrobionics dilengkapi dengan system database tanaman, sehingga dapat memberikan informasi apakah kondisi kebun saat ini berada dalam batas ideal untuk tanaman yang ditanam. Pengguna dapat menyimpan data berupa profil tanaman seperti nama tanaman, nilai ideal (suhu, kelembaban udara, dan pH air), dan estimasi waktu yang dibutuhkan untuk siap panen. Selain itu juga memungkinkan untuk mengubah data tersebut jika diperlukan.

c) *Schedule and Notes*

Untuk mendukung pekerjaan petani kebun hidroponik, Nydrobionics dilengkapi dengan system yang memudahkan penggunanya untuk pengingat jadwal pekerjaan dan catatan.

3. IMPLEMENTASI APLIKASI

a) Struktur Data

Agar data yang terlibat dapat ditampilkan dan diolah dari aplikasi ke database maupun sebaliknya, maka diharuskan untuk menyiapkan struktur data pada alikasi, baik data yang secara langsung ditransmisikan ke database maupun variabel-variabel lain yang digunakan dalam pengolahannya. Terdiri dari 7 *data class* utama (model) untuk mengolah data dari database yang berisi fungsi untuk mengkonversi struktur yang terlibat, seperti *Data Class* ke *Hash Map* (`Model.toHashMap()`) untuk pengiriman data dan *Document Snapshot* ke *Data Class* (`DocumentSnapshot.toModel()`) untuk penerimaan data. Pada Kode 5 menunjukkan *data class* untuk *UserModel* yang berisi deklarasi atribut dan fungsi konversi untuk memudahkan proses pengolahan data. Untuk model lain dapat dilihat melalui lampiran.

```
@Parcelize
data class UserModel(
    var name : String? = null,
    var email: String? =null,
    var gender : String? = null,
    var dob : String? = null,
    var role : String? = null,
    var bio : String? = null,
    var uid : String? = null,
    var performanceRate : Float? = null,
    var address : String? = null,
    var phone : String? = null,
```

```
    var joinedSince : String? = null,
    var photo_url : String? = null,
    var farmId : String? = null,
    var timestamp : String? = null
) : Parcelable {
    companion object {
        fun DocumentSnapshot.toUserModel() : UserModel? {
            try{
                val name = getString("name")
                val email = getString("email")
                val gender = getString("gender")
                val dob = getString("dob")
                val role = getString("role")
                val bio = getString("bio")
                val uid = getString("uid")
                val performanceRate : Double? = get("performanceRate")
as Double?
                val address = getString("address")
                val phone = getString("phone")
                val joinedSince = getString("joinedSince")
                val photo_url = getString("photo_url")
                val farmId = getString("farmId")
                val timestamp = getString("timestamp")
                val output = UserModel(name, email, gender, dob,
                    role, bio, uid,
                    performanceRate?.toFloat(),
                    address, phone, joinedSince,
                    photo_url, farmId, timestamp)
                return output
            } catch (e: Exception) {
                Log.e(TAG, "Error converting $TAG", e)
                return null
            }
        }

        fun UserModel.toHashMap():HashMap<String,Any?>{
            val output = hashMapOf<String,Any?>()
            output["name"] = name
            output["email"] = email
            output["gender"] = gender
            output["dob"] = dob
            output["role"] = role
            output["bio"] = bio
            output["uid"] = uid
            output["performanceRate"] = performanceRate
            output["address"] = address
            output["phone"] = phone
            output["joinedSince"] = joinedSince
            output["photo_url"] = photo_url
            output["farmId"] = farmId
            output["timestamp"] = ViewUtility().getCurrentTimestamp()
            return output
        }

        private const val TAG = "UserModel"
    }
}
```

Kode 5. Struktur data *UserModel* beserta konverternya

Untuk memudahkan pengelolaan data enum pada aplikasi (*Gender* dan *Role*), maka dibuatlah *Enum Class* termasuk fungsi tambahan seperti `toString()` untuk mengubah *default* dari fungsi `toString()` yang ditunjukkan pada Kode 6. Pada *enum class Gender* (a) juga terdapat fungsi `getPosition()` karena *view component* yang digunakan untuk menampilkan data tersebut merupakan *SegmentedButton* yang menghasilkan *value* dari inputan pengguna berupa data posisi yang dipilih.

```
enum class Gender(var gender: String){
    MALE("male") {
        override fun getPosition(): Int {
            return 0
        }
    },
    FEMALE("female") {
        override fun getPosition() : Int {
            return 1
        }
    };

    override fun toString(): String {
        return gender
    }

    abstract fun getPosition():Int

    companion object{
        fun getType(type: String) : Gender?{
           return when(type){
                "male" -> MALE
                "female" -> FEMALE
                else -> null
            }
        }
    }
}
```

a) *Enum Class Gender*

```
enum class Role(var role: String){
    OWNER("owner"),
    STAFF("staff");

    override fun toString(): String {
        return role
    }

    companion object{
        fun getType(type: String) : Role? {
            return when(type){
                "owner" -> OWNER
                "female" -> STAFF
                else -> null
            }
        }
```

```
    }
}
```

b) *Enum Class Role*

Kode 6. Enum class *Gender* dan *Role* untuk memudahkan pengelolaan data terkait

b) Algoritma Input dan Output

```
interface LoadingInterface {
    var context: Context?
    var circularProgressButton: CircularProgressButton?
    var textInputEditTexts: ArrayList<TextInputEditText>?
    var viewsAsButton: ArrayList<View>?
    var numberPickers: ArrayList<ClickNumberPickerView>?
    var checkBoxes: ArrayList<CheckBox>?
    var actionBar: ActionBar?

    var initialLoadingState: Boolean
    var isLoading: Boolean
        get() = initialLoadingState
        set(value) {
            textInputEditTexts?.forEach {
                it.isCursorVisible = !value
                it.isFocusable = !value
                it.isFocusableInTouchMode = !value
            }
            viewsAsButton?.forEach {
                it.isEnabled = !value
            }
            numberPickers?.forEach {
                it.isEnabled = !value
            }
            circularProgressButton?.apply {
                this.isEnabled = true
                if (value) {
                    startAnimation()
                } else {
                    val handler = Handler(Looper.getMainLooper())
                    handler.postDelayed({
                        revertAnimation()
                    }, 3000)
                }
            }
            checkBoxes?.forEach {
                it.isClickable = !value
            }
            actionBar?.setDisplayShowHomeEnabled(value)
            onLoadingChangeListener { (initialLoadingState) }
            initialLoadingState = value
        }

    fun onLoadingChangeListener(function: (isLoading: Boolean) -> Unit)
}
```

Kode 7. LoadingInterface yang berfungsi untuk mengatur aktivasi *view* saat pengiriman data

Terdapat 3 pengelompokkan menu berdasarkan fungsi yang dijalankan oleh pengguna, yaitu menu awal aplikasi berupa *Splash Screen*, *Sign In, Sign Up*,

dan *Forget Password*, menu *Create Profile* yang terdiri *Create User Profile* dan *Create Farm Profile*, serta menu *Main* berupa *Main Home*, *Main Posts*, *Main Add*, *Main Notes*, dan *Main Profile*. Masing-masing memiliki fungsi yang berbeda-beda, namun penerimaan input *user* dan pemberian responnya memiliki mekanisme yang mirip, yaitu saat input yang diberikan belum sesuai dengan syarat, maka tombol untuk melanjutkan ke proses selanjutnya tidak diaktifkan sehingga pengguna diharuskan untuk melengkapi input tersebut. Selain itu saat proses pengiriman data ke Firebase dilakukan, pengguna juga tidak dapat memberikan input, baik dengan *soft keyboard* maupun dengan menekan *button* atau *view*. Hal ini dikarenakan adanya penggunaan *LoadingInterface* yang diberikan nilainya saat pengiriman dan penerimaan data dimulai serta selesai dilakukan seperti pada Kode 7.

Setiap *activity* yang dibuat menerapkan arsitektur MVVM dan dikombinasikan dengan ViewBinding, dimana membutuhkan *class* ViewModel untuk setiap *activity* yang dibuat. Dalam pengerjaan tugas akhir ini ada beberapa *activity* yang menggunakan ViewModel yang sama dikarenakan adanya kesamaan fungsi yang dibutuhkan, seperti *CreateProfileViewModel* yang digunakan untuk *CreateProfileActivity* (*CreateUserFragment* serta *CreateFarmFragment*), *EditProfileUserActivity*, dan *EditProfileFarmActivity*. Bagian yang membedakan penggunaan untuk *create* dan *edit* adalah pendefinisian *reference id* untuk pengiriman data ke Firestore seperti pada Kode 1 bagian `farmId`, dimana jika `farmId` bernilai `farmId` jika sudah memiliki nilai sebelumnya dan bernilai sesuai dengan *generate reference id* jika `farmId` masih bernilai null. Bagian inilah yang dimanfaatkan untuk mengubah fungsi dari *create farm* menjadi *edit farm*.

```
…
fun createFarmProfile(name:String, description:String,
location:String){
    try {
        createProfileError.value = ""
        val db = firestore.collection("farms")
        val ref : DocumentReference = db.document()

        farmModel.value?.apply {
            this.farmId = farmId ?: ref.id
            this.name = name
            this.description = description
            this.location = location
        }
```

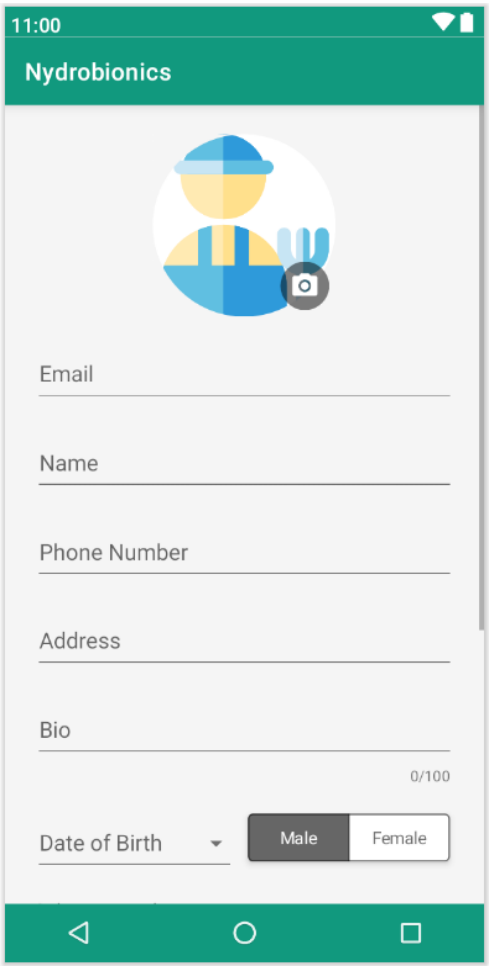
```
db.document(farmModel.value!!.farmId!!).set(farmModel.value!!.toHashMap
()).addOnCompleteListener {
            if(it.isSuccessful){
                userModel.value?.farmId = farmModel.value!!.farmId!!
                sendUserProfile()
                isFarmCreated.value = isUserCreated.value == true
            } else {
                createProfileError.value = it.exception.toString()
                isFarmCreated.value = false
            }
        }
    } catch (e:Exception){
        Log.e(TAG, "Error submit farm", e)
    }
}
...
```

Kode 8. Penggunaan CreateProfileViewModel untuk membuat dan mengubah data Farm

Selain penggunaan ViewModel untuk beberapa *activity,* penggunaan ulang *layout* juga dilakukan di beberapa bagian seperti untuk *EditProfileUserActivity* yang juga menggunakan layout untuk *CreateUserFragment* seperti pada Kode 9 dengan bentuk layout seperti pada Gambar 14 . Untuk mendefinisikan nilai awal data, maka dibutuhkan *observer* pada *activity* untuk mengetahui apakah ada nilai inisiasi awal yang diberikan dari *activity* sebelumnya (parsing data melalui intent seperti pada Kode 10). Selain itu, dilakukan beberapa penyesuaian seperti menghilangkan atau menambahkan beberapa komponen *view* yang dibutuhkan seperti pada b. Penerapan MVVM lebih lengkap lainnya dapat diakses pada lampiran maupun melalui dokumentasi project pada GitHub.

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".activity.EditProfileUserActivity">

    <include
        android:id="@+id/editProfileFragment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        layout="@layout/fragment_create_user" />
</ScrollView>
```

Kode 9. Penggunaan ulang layout *CreateUserFragment* untuk *EditProfileUserActivity*



Gambar 14. Hasil dari render untuk layout *CreateUserFragment* dan *EditProfileUserActivity*

Sumber : Dokumentasi pribadi

```
…
R.id.drawer_edit -> {
    val intent = Intent(this, EditProfileUserActivity::class.java)
    intent.putExtra(BuildConfig.CURRENT_USER,
viewModel.getCurrentUser().value)
    startActivity(intent)
    drawerLayout.closeDrawer(GravityCompat.END)
    true
}
…
```

a) Pengiriman data User dari *MainActivity* ke *EditProfileUserActivity*

```
…
viewModel.setCurrentUser(intent.getParcelableExtra<UserModel>(BuildCon
fig.CURRENT_USER))
bindingFragment.apply {
    createUserRoleGroup.visibility = View.GONE
    createUserSubmit.text = getString(R.string.save)

    viewModel.getCurrentUserModel()?.let {
        createUserEmail.setText(Firebase.auth.currentUser?.email)
        createUserName.setText(it.name ?: "")
        createUserPhone.setText(it.phone ?: "")
```

```
        createUserAddress.setText(it.address ?: "")
        createUserBio.setText(it.bio ?: "")
        createUserDOB.setText(it.dob ?: "")
        it.photo_url?.let {
            Glide.with(this@EditProfileUserActivity)
                .load(it)
                .centerCrop()
                .into(createUserPhoto)
        }

createUserGender.setPosition(Gender.getType(it.gender!!)!!.getPosition
(), false)

        strEdt[it.name ?: ""] = createUserName
        strEdt[it.phone ?: ""] = createUserPhone
        strEdt[it.address ?: ""] = createUserAddress
        strEdt[it.bio ?: ""] = createUserBio
        strEdt[it.dob ?: ""] = createUserDOB
    }
}
...
```

b) Penerimaan data di *EditProfileUserActivity* dan menampilkan data ke *view* yang bersesuaian

Kode 10. Pengiriman, penerimaan, dan pengaturan inisiali awal untuk *EditProfileUserActivity*

c) Utility Class

Saat melakukan implementasi *back end,* sangat memungkinkan adanya penambahan *class* selain *class* utama untuk *activity* maupun struktur data dikarenakan adanya *utility class* ini dapat memudahkan proses pengolahan dan pengelolaan data dengan mengurangi potensi redudansi *block code* dan menghindari adanya ketidak konsistenan algoritma pengolahan data. Sebagai contoh pada *IntentUtility* yang digunakan untuk melakukan navigasi di luar aplikasi (*implicit intent*). Beberapa fitur yang memanfaatkan *class* ini di antaranya untuk membuka *maps* sesuai dengan *key* yang ditentukan seperti pada Kode 11 di *MainProfileFragment* dan *ProfileUserActivivty* yang penulisan *code*-nya menjadi lebih ringkas (a) dibandingkan harus menulis keseluruhan *block code* (b) setiap kali *request permission* dibutuhkan.

```
...
IntentUtility(this).openMaps(mainProfileAddress.toString())
...
```

a) Penggunaan *IntentUtility* untuk membuka maps

```
class IntentUtility(val context: Context){
    fun openBrowser(url: String){
        val intent = Intent()
        intent.addCategory(Intent.CATEGORY_BROWSABLE)
```

```
        intent.action = Intent.ACTION_VIEW
        intent.data = Uri.parse(String.format(url))
        context.startActivity(intent)
    }

    fun openMaps(location:String){
        openBrowser(
            "https://www.google.com/maps?q=${
                location.replace(
                    ' ',
                    '+'
                )
            }"
        )
    }
}
```

b) *Code* dari membuka *maps* pada *IntentUtility*

Kode 11. Penggunaan *IntentUtility* untuk membuka *maps*

D. CLOSSURE (DEPLOYMENT & RELEASE)

Setelah aplikasi dapat menjalankan program sesuai dengan rancangan awal, maka dilakukan pembuatan laporan akhir yang mendokumentasikan keberlangsungan kegiatan pengembangan. Pengembangan aplikasi akan berakhir setelah melakukan presentasi untuk melaporkan hasil kegiatan yang telah dilaksanakan.

BAB V
PENUTUP

A. KESIMPULAN

Dari pelaksanaan pengembangan aplikasi yang sudah dilakukan, penulis dapat menemui beberapa kendala dan hambatan yang dapat ditarik beberapa kesimpulan sebagai berikut :

1. Beberapa *block code* yang dibutuhkan secara berulang dapat dijadikan satu *class* agar memudahkan modifikasi dan menjamin konsistensi pengkodean khususnya pada *class ViewUtility, IntentUtility*, dan *LoadingInterface*.
2. Penerapan *open source library* yang disediakan oleh developer lain melalui *github* untuk mencapai tujuan tertentu dapat mempermudah dan mempercepat proses pengembangan serta melatih diri dalam membaca *block code* dan memahami algoritma tersebut. Namun terdapat kekurangan jika membutuhkan sub-variable/resource namun di *private* maka membutuhkan waktu untuk explore lebih lanjut.

B. SARAN

Demi pengembangan lebih lanjut mengenai tugas akhir ini, disarankan beberapa langkah lanjutan sebagai berikut :

1. Jika aplikasi ini dikembangkan lebih lanjut, dapat memperbaiki dan menambah fitur-fitur lain seperti :
   a) Menambah jenis tipe postingan yang dapat dibuat oleh pengguna dan melengkapinya dengan *attach document* seperti gambar, video, maupun dokumen lainnya.
   b) Membedakan jenis *notes* untuk *reminder-schedule, notes*, maupun *to-do list* agar memberikan pengguna pilihan dalam memanfaatkan fitur yang sudah ada.
   c) Menambahkan fitur *deeplink* yang langsung terkoneksi dengan Firebase saat melakukan *forget password* agar tidak memerlukan pihak ketiga dalam mencapai fungsi tersebut (dari email konfirmasi langsung ter-*redirect* ke aplikasi).
2. Dapat memperbaiki *codingan back end* agar lebih efektif dan efisien serta dapat dimengerti lebih mudah oleh para pembaca.

DAFTAR PUSTAKA

[1] J. Chen, "Android Operating System," Dotdash, 03 Februari 2021. [Online]. Available: https://www.investopedia.com/terms/a/android-operating-system.asp. [Accessed 07 Oktober 2021].

[2] P. Christensson, "Android Definition," TecthTerms, 16 Mei 2016. [Online]. Available: https://techterms.com/definition/android. [Accessed 5 Oktober 2021].

[3] Wikipedia, "Android Version History," 04 Oktober 2021. [Online]. Available: https://en.wikipedia.org/wiki/Android_version_history. [Accessed 2021 Oktober 07].

[4] A. N. Anrini, "Apa Bedanya Android Studio dan Android SDK?," Definite, 31 Agustus 2020. [Online]. Available: https://definite.co.id/blogs/apa-bedanya-android-studio-dan-android-sdk/. [Accessed 07 Oktober 2021].

[5] S. Okta, "Mengenal Arsitektur Model View ViewModel (MVVM) di Android — Bagian 1," 21 Desember 2017. [Online]. Available: https://syafdia.medium.com/mengenal-arsitektur-model-view-viewmodel-mvvm-di-android-2e52ec98a74e. [Accessed 07 Oktober 2021].

[6] Android Developers, "What's New in Architecture Components (Google I/O'19)," 09 Mei 2019. [Online]. Available: https://www.youtube.com/watch?v=Qxj2eBmXLHg&t=446s&ab_channel=AndroidDevelopers. [Accessed 2021 Oktober 07].

[7] M. Firdaus, "View Binding; Membuat Aplikasi Android menjadi Lebih Paten — Bukan Kaleng-kaleng!," 31 Maret 2020. [Online]. Available: https://medium.com/gits-apps-insight/view-binding-membuat-aplikasi-android-menjadi-paten-bukan-kaleng-kaleng-7dca4b5f4739. [Accessed 07 Oktober 2021].

[8] Dicoding Intern, "Apa itu Firebase? Pengertian, Jenis-Jenis, dan Fungsi Kegunaannya," Dicoding, 25 November 2020. [Online]. Available: https://www.dicoding.com/blog/apa-itu-firebase-pengertian-jenis-jenis-dan-fungsi-kegunaannya/. [Accessed 2021 Oktober 2021].

[9] Google Developers, "Get Started with Firebase Authentication on Android," [Online]. Available: https://firebase.google.com/docs/auth/android/start.

[Accessed 22 Oktober 2021].

[10] R. Vyas, "Firebase Cloud Firestore v/s Firebase Realtime Database," 11 Oktober 2017. [Online]. Available: https://medium.com/zero-equals-false/firebase-cloud-firestore-v-s-firebase-realtime-database-931d4265d4b0. [Accessed 2021 Oktober 07].

[11] Google Developers, "Create a Cloud Storage reference on Android," [Online]. Available: https://firebase.google.com/docs/storage/android/create-reference. [Accessed 22 Oktober 2021].

LAMPIRAN

Untuk memudahkan pembuatan lampiran, maka kontennya di-*generate* menjadi PDF menggunakan LaTeX (TexStudio sebagai IDE-nya) dengan memanfaatkan *package* pdfpages dan minted. LaTeX adalah sebuah bahasa pemograman yang difokuskan untuk membuat sebuah dokumen dan banyak digunakan untuk membuat thesis maupun jurnal ilmiah. *Source code* dalam LaTeX untuk membuat lampiran ini ditunjukkan melalui Kode 1.

```latex
\documentclass{article}
\title{\vspace{-5ex} \bfseries {\sc{LAMPIRAN}}}
\date{\vspace{-10ex}}
\usepackage{geometry}
\geometry{
        a4paper,
        left=40mm,
        top=30mm,
        right=30mm,
        bottom=30mm
}
\usepackage{pdfpages}
\usepackage[bookmarks=false]{hyperref}
\usepackage{minted}
\usepackage{caption}
\usepackage{setspace}
\usepackage{fancyhdr}
\pagestyle{fancy}
\fancyhf{}
\fancyfoot[R]{Laporan Tugas Akhir Aplikasi Monitoring Kebun Hidroponik Berbasis
↪  \textit{Mobile} $\vert$ \thepage}
\fancypagestyle{plain}{
        \renewcommand{\headrulewidth}{0pt}
        \fancyhf{}%
        \fancyfoot[R]{Laporan Tugas Akhir Aplikasi Monitoring Kebun Hidroponik Berbasis
        ↪  \textit{Mobile} $\vert$ \thepage}
}

\setcounter{page}{43}
\setcounter{secnumdepth}{4}
\setminted{
        linenos,
        breaklines,
        breakanywhere,
        fontsize=\footnotesize,
        frame=single,
        framesep=10pt,
        baselinestretch=1
}
\newenvironment{multipageListing}{
        \setstretch{1}
        \captionsetup{type=listing, justification=centering, skip=-5pt}
}{}

\renewcommand{\listingscaption}{Kode}
\renewcommand{\thesection}{\Alph{section}.}
\renewcommand{\thesubsection}{\arabic{subsection}.}
\renewcommand{\thesubsubsection}{\alph{subsubsection})}
\newcommand{\subsectionpostlude}{\vspace{1em}}

\begin{document}
        \maketitle
        Untuk memudahkan pembuatan lampiran, maka kontennya di-\textit{generate}
        ↪  menjadi PDF menggunakan LaTeX (TexStudio sebagai IDE-nya) dengan
        ↪  memanfaatkan \textit{package} pdfpages dan minted.  LaTeX adalah sebuah
        ↪  bahasa pemograman yang difokuskan untuk membuat sebuah dokumen dan banyak
        ↪  digunakan untuk membuat thesis maupun jurnal ilmiah. \textit{Source code}
        ↪  dalam LaTeX untuk membuat lampiran ini ditunjukkan melalui Kode
        ↪  \ref*{code:latex}.
        \begin{multipageListing}
```

```
        \inputminted{latex}{generatePDF.tex}
        \captionof{listing}{\textit{Source code} LaTeX untuk membuat lampiran\label{code:latex}}
\end{multipageListing}

\includepdf[pages={1-3}, scale=0.75, nup=1x3, pagecommand=\section{PRODUCT MANUAL}]{codes/product_manual.pdf}
\includepdf[pages={4-}, scale=0.75, nup=1x3]{codes/product_manual.pdf}

\includepdf[pages={11-12}, scale=0.8, nup=1x2, offset=0 0.9cm, pagecommand=\section{FRONT END}]{codes/ppt.pdf}
\includepdf[pages={13-18}, scale=0.8, nup=1x2, offset=0 0.5cm]{codes/ppt.pdf}

\section{DATA MODEL}
        \subsection{USER MODEL}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/UserModel.kt}
                \captionof{listing}{\textit{Data class} untuk \textit{User}}
        \end{multipageListing}

        \subsection{NOTE MODEL}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/NoteModel.kt}
                \captionof{listing}{\textit{Data class} untuk \textit{Note}}
        \end{multipageListing}

        \subsection{FARM MODEL}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/FarmModel.kt}
                \captionof{listing}{\textit{Data class} untuk \textit{Farm}}
        \end{multipageListing}

        \subsection{KIT MODEL}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/KitModel.kt}
                \captionof{listing}{\textit{Data class} untuk \textit{Hydroponic Kit}}
        \end{multipageListing}

        \subsection{DATA MONITORING MODEL}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/DataMonitoringModel.kt}
                \captionof{listing}{\textit{Data class} untuk \textit{Data Monitoring}}
        \end{multipageListing}

        \subsection{CROPS MODEL}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/CropsModel.kt}
                \captionof{listing}{\textit{Data class} untuk \textit{Crops}}
        \end{multipageListing}

        \subsection{PLANT MODEL}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/PlantModel.kt}
                \captionof{listing}{\textit{Data class} untuk \textit{Plant}}
        \end{multipageListing}

        \subsection{FEEDBACK MODEL}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/FeedbackModel.kt}
                \captionof{listing}{\textit{Data class} untuk \textit{Feedback}}
        \end{multipageListing}

        \subsection{SCORE LEVEL MODEL}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/ScoreLevel.kt}
                \captionof{listing}{\textit{Data class} untuk data yang memiliki nilai minimum dan maksimum}
        \end{multipageListing}
```

```
117
118             \subsection{URI FILE EXTENSIONS MODEL}
119             \begin{multipageListing}
120                 \inputminted{kotlin}{codes/UriFileExtensions.kt}
121                 \captionof{listing}{\textit{Data class} untuk yang digunakan
                    ↪  untuk mengelola \textit{attach} gambar}
122             \end{multipageListing}
123
124         \section{VIEW MODEL}
125             \subsection{SPLASH SCREEN VIEW MODEL}
126             \begin{multipageListing}
127                 \inputminted{kotlin}{codes/SplashScreenViewModel.kt}
128                 \captionof{listing}{\textit{View Model} untuk \textit{Splash
                    ↪  Screen Activity}}
129             \end{multipageListing}
130
131             \subsection{SIGN IN VIEW MODEL}
132             \begin{multipageListing}
133                 \inputminted{kotlin}{codes/SignInViewModel.kt}
134                 \captionof{listing}{\textit{View Model} untuk \textit{Sign In
                    ↪  Activity}}
135             \end{multipageListing}
136
137             \subsection{SIGN UP VIEW MODEL}
138             \begin{multipageListing}
139                 \inputminted{kotlin}{codes/SignUpViewModel.kt}
140                 \captionof{listing}{\textit{View Model} untuk \textit{Sign Up
                    ↪  Activity}}
141             \end{multipageListing}
142
143             \subsection{FORGET PASSWORD VIEW MODEL}
144             \begin{multipageListing}
145                 \inputminted{kotlin}{codes/ForgetPasswordViewModel.kt}
146                 \captionof{listing}{\textit{View Model} untuk \textit{Forget
                    ↪  Password Activity}}
147             \end{multipageListing}
148
149             \subsection{CREATE PROFILE VIEW MODEL}
150             \begin{multipageListing}
151                 \inputminted{kotlin}{codes/CreateProfileViewModel.kt}
152                 \captionof{listing}{\textit{View Model} untuk \textit{Create
                    ↪  Profile Activity} beserta \textit{Create User} dan
                    ↪  \textit{Create Farm} \textit{fragments}. Termasuk juga
                    ↪  untuk \textit{Edit Profile User} dan \textit{Edit Profile
                    ↪  Farm} \textit{activities}}
153             \end{multipageListing}
154
155             \subsection{MAIN VIEW MODEL}
156             \begin{multipageListing}
157                 \inputminted{kotlin}{codes/MainViewModel.kt}
158                 \captionof{listing}{\textit{View Model} untuk \textit{Main
                    ↪  Activity} beserta \textit{Main Home}, \textit{Main Posts},
                    ↪  \textit{Main Add}, \textit{Main Notes}, dan \textit{Main
                    ↪  Profile} \textit{fragments}}
159             \end{multipageListing}
160
161             \subsection{ADD CROPS VIEW MODEL}
162             \begin{multipageListing}
163                 \inputminted{kotlin}{codes/AddCropsViewModel.kt}
164                 \captionof{listing}{\textit{View Model} untuk \textit{Add Crops
                    ↪  Activity}}
165             \end{multipageListing}
166
167             \subsection{ADD DATA MONITORING VIEW MODEL}
168             \begin{multipageListing}
169                 \inputminted{kotlin}{codes/AddDataMonitoringViewModel.kt}
170                 \captionof{listing}{\textit{View Model} untuk \textit{Add Data
                    ↪  Monitoring Activity}}
171             \end{multipageListing}
172
```

```
\subsection{ADD KIT VIEW MODEL}
\begin{multipageListing}
        \inputminted{kotlin}{codes/AddKitViewModel.kt}
        \captionof{listing}{\textit{View Model} untuk \textit{Add Kit Activity}}
\end{multipageListing}

\subsection{ADD NOTE VIEW MODEL}
\begin{multipageListing}
        \inputminted{kotlin}{codes/AddNoteViewModel.kt}
        \captionof{listing}{\textit{View Model} untuk \textit{Add Note Activity}}
\end{multipageListing}

\subsection{ADD PLANT VIEW MODEL}
\begin{multipageListing}
        \inputminted{kotlin}{codes/AddPlantViewModel.kt}
        \captionof{listing}{\textit{View Model} untuk \textit{Add Plant Activity}}
\end{multipageListing}

\subsection{PROFILE KIT VIEW MODEL}
\begin{multipageListing}
        \inputminted{kotlin}{codes/ProfileKitViewModel.kt}
        \captionof{listing}{\textit{View Model} untuk \textit{Profile Kit Activity} beserta \textit{Kit Overview}, \textit{Kit Monitoring}, dan \textit{Kit Crops} \textit{fragments}}
\end{multipageListing}

\subsection{PROFILE USER VIEW MODEL}
\begin{multipageListing}
        \inputminted{kotlin}{codes/ProfileUserViewModel.kt}
        \captionof{listing}{\textit{View Model} untuk \textit{Profile User Activity}}
\end{multipageListing}

\subsection{SEARCH VIEW MODEL}
\begin{multipageListing}
        \inputminted{kotlin}{codes/SearchViewModel.kt}
        \captionof{listing}{\textit{View Model} untuk \textit{Search Activity}}
\end{multipageListing}

\subsection{FEEDBACK VIEW MODEL}
\begin{multipageListing}
        \inputminted{kotlin}{codes/FeedbackViewModel.kt}
        \captionof{listing}{\textit{View Model} untuk \textit{Feedback Activity}}
\end{multipageListing}

\subsection{SHOW RECYCLER VIEW MODEL}
\begin{multipageListing}
        \inputminted{kotlin}{codes/ShowRecyclerViewModel.kt}
        \captionof{listing}{\textit{View Model} untuk \textit{Show Recycler Activity}}
\end{multipageListing}

\section{VIEW LAYER (ACTIVITY AND FRAGMENT)}
    \subsection{SPLASH SCREEN ACTIVITY}
    \begin{multipageListing}
        \inputminted{kotlin}{codes/SplashScreenActivity.kt}
        \captionof{listing}{\textit{Source Code} untuk \textit{Splash Screen Activity}}
    \end{multipageListing}

    \subsection{SIGN IN ACTIVITY}
    \begin{multipageListing}
        \inputminted{kotlin}{codes/SignInActivity.kt}
        \captionof{listing}{\textit{Source Code} untuk \textit{Sign In Activity}}
```

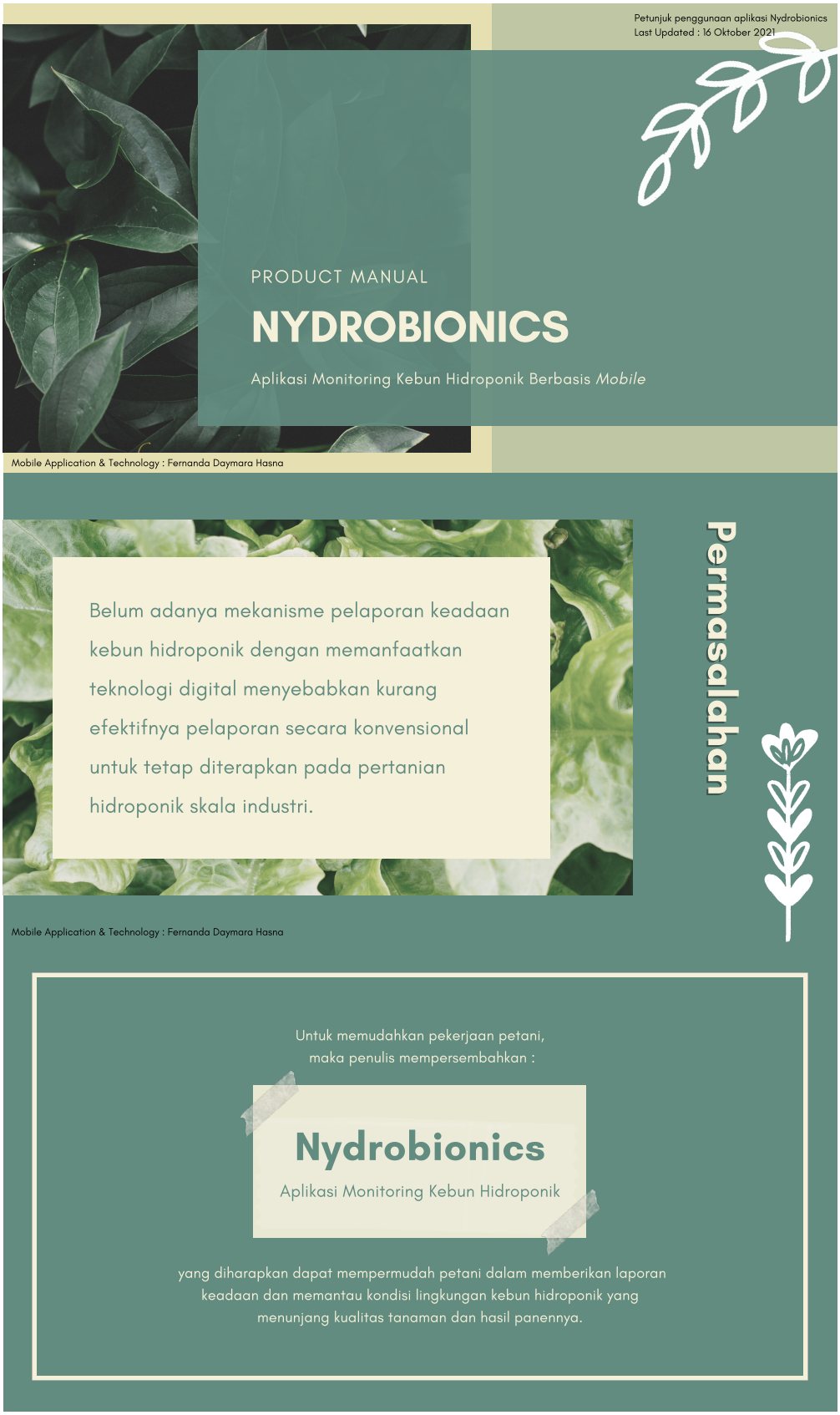
```
\end{multipageListing}

\subsection{SIGN UP ACTIVITY}
\begin{multipageListing}
        \inputminted{kotlin}{codes/SignUpActivity.kt}
        \captionof{listing}{\textit{Source Code} untuk \textit{Sign Up
        ↪  Activity}}
\end{multipageListing}

\subsection{FORGET PASSWORD ACTIVITY}
\begin{multipageListing}
        \inputminted{kotlin}{codes/ForgetPasswordActivity.kt}
        \captionof{listing}{\textit{Source Code} untuk \textit{Forget
        ↪  Password Activity}}
\end{multipageListing}

\subsection{CREATE PROFILE ACTIVITY}
\begin{multipageListing}
        \inputminted{kotlin}{codes/CreateProfileActivity.kt}
        \captionof{listing}{\textit{Source Code} untuk \textit{Create
        ↪  Profile Activity} beserta \textit{fragments} terkait}
\end{multipageListing}

        \subsubsection{CREATE USER FRAGMENT}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/CreateUserFragment.kt}
                \captionof{listing}{\textit{Source Code} untuk
                ↪  \textit{Create User Fragment}}
        \end{multipageListing}

        \subsubsection{CREATE FARM FRAGMENT}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/CreateFarmFragment.kt}
                \captionof{listing}{\textit{Source Code} untuk
                ↪  \textit{Create Farm Fragment}}
        \end{multipageListing}

\subsection{MAIN ACTIVITY}
\begin{multipageListing}
        \inputminted{kotlin}{codes/MainActivity.kt}
        \captionof{listing}{\textit{Source Code} untuk \textit{Main
        ↪  Activity} beserta \textit{fragments} terkait}
\end{multipageListing}

        \subsubsection{MAIN HOME FRAGMENT}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/MainHomeFragment.kt}
                \captionof{listing}{\textit{Source Code} untuk
                ↪  \textit{Main Home Fragment}}
        \end{multipageListing}

        \subsubsection{MAIN POSTS FRAGMENT}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/MainPostsFragment.kt}
                \captionof{listing}{\textit{Source Code} untuk
                ↪  \textit{Main Posts Fragment}}
        \end{multipageListing}

        \subsubsection{MAIN ADD FRAGMENT}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/MainAddFragment.kt}
                \captionof{listing}{\textit{Source Code} untuk
                ↪  \textit{Main Add Fragment}}
        \end{multipageListing}

        \subsubsection{MAIN NOTES FRAGMENT}
        \begin{multipageListing}
                \inputminted{kotlin}{codes/MainNotesFragment.kt}
                \captionof{listing}{\textit{Source Code} untuk
                ↪  \textit{Main Notes Fragment}}
```

```
\end{multipageListing}

\subsubsection{MAIN PROFILE FRAGMENT}
\begin{multipageListing}
	\inputminted{kotlin}{codes/MainProfileFragment.kt}
	\captionof{listing}{\textit{Source Code} untuk \textit{Main Profile Fragment}}
\end{multipageListing}

\subsection{ADD CROPS ACTIVITY}
\begin{multipageListing}
	\inputminted{kotlin}{codes/AddCropsActivity.kt}
	\captionof{listing}{\textit{Source Code} untuk \textit{Add Crops Activity}}
\end{multipageListing}

\subsection{ADD DATA MONITORING ACTIVITY}
\begin{multipageListing}
	\inputminted{kotlin}{codes/AddDataMonitoringActivity.kt}
	\captionof{listing}{\textit{Source Code} untuk \textit{Add Data Monitoring Activity}}
\end{multipageListing}

\subsection{ADD KIT ACTIVITY}
\begin{multipageListing}
	\inputminted{kotlin}{codes/AddKitActivity.kt}
	\captionof{listing}{\textit{Source Code} untuk \textit{Add Kit Activity}}
\end{multipageListing}

\subsection{ADD NOTE ACTIVITY}
\begin{multipageListing}
	\inputminted{kotlin}{codes/AddNoteActivity.kt}
	\captionof{listing}{\textit{Source Code} untuk \textit{Add Note Activity}}
\end{multipageListing}

\subsection{ADD PLANT ACTIVITY}
\begin{multipageListing}
	\inputminted{kotlin}{codes/AddPlantActivity.kt}
	\captionof{listing}{\textit{Source Code} untuk \textit{Add Plant Activity}}
\end{multipageListing}

\subsection{PROFILE KIT ACTIVITY}
\begin{multipageListing}
	\inputminted{kotlin}{codes/ProfileKitActivity.kt}
	\captionof{listing}{\textit{Source Code} untuk \textit{Profile Kit Activity} beserta \textit{fragments} terkait}
\end{multipageListing}

	\subsubsection{KIT OVERVIEW FRAGMENT}
	\begin{multipageListing}
		\inputminted{kotlin}{codes/KitOverviewFragment.kt}
		\captionof{listing}{\textit{Source Code} untuk \textit{KitOverview Fragment}}
	\end{multipageListing}

	\subsubsection{KIT MONITORING FRAGMENT}
	\begin{multipageListing}
		\inputminted{kotlin}{codes/KitMonitoringFragment.kt}
		\captionof{listing}{\textit{Source Code} untuk \textit{KitMonitoring Fragment}}
	\end{multipageListing}

	\subsubsection{KIT CROPS FRAGMENT}
	\begin{multipageListing}
		\inputminted{kotlin}{codes/KitCropsFragment.kt}
		\captionof{listing}{\textit{Source Code} untuk \textit{Kit Crops Fragment}}
```

```
352                         \end{multipageListing}
353
354                 \subsection{PROFILE USER ACTIVITY}
355                 \begin{multipageListing}
356                         \inputminted{kotlin}{codes/ProfileUserActivity.kt}
357                         \captionof{listing}{\textit{Source Code} untuk \textit{Profile
                            ↪  User Activity}}
358                 \end{multipageListing}
359
360                 \subsection{EDIT PROFILE FARM ACTIVITY}
361                 \begin{multipageListing}
362                         \inputminted{kotlin}{codes/EditProfileFarmActivity.kt}
363                         \captionof{listing}{\textit{Source Code} untuk \textit{Edit
                            ↪  Profile Farm Activity}}
364                 \end{multipageListing}
365
366                 \subsection{EDIT PROFILE USER ACTIVITY}
367                 \begin{multipageListing}
368                         \inputminted{kotlin}{codes/EditProfileUserActivity.kt}
369                         \captionof{listing}{\textit{Source Code} untuk \textit{Edit
                            ↪  Profile User Activity}}
370                 \end{multipageListing}
371
372                 \subsection{SEARCH MODEL}
373                 \begin{multipageListing}
374                         \inputminted{kotlin}{codes/SearchActivity.kt}
375                         \captionof{listing}{\textit{Source Code} untuk \textit{Search
                            ↪  Activity}}
376                 \end{multipageListing}
377
378                 \subsection{FEEDBACK ACTIVITY}
379                 \begin{multipageListing}
380                         \inputminted{kotlin}{codes/FeedbackActivity.kt}
381                         \captionof{listing}{\textit{Source Code} untuk \textit{Feedback
                            ↪  Activity}}
382                 \end{multipageListing}
383
384                 \subsection{SHOW RECYCLER ACTIVITY}
385                 \begin{multipageListing}
386                         \inputminted{kotlin}{codes/ShowRecyclerActivity.kt}
387                         \captionof{listing}{\textit{Source Code} untuk \textit{Show
                            ↪  Recycler Activity}}
388                 \end{multipageListing}
389
390                 \subsection{SHOW PICTURE ACTIVITY}
391                 \begin{multipageListing}
392                         \inputminted{kotlin}{codes/ShowPictureActivity.kt}
393                         \captionof{listing}{\textit{Source Code} untuk \textit{Show
                            ↪  Picture Activity}}
394                 \end{multipageListing}
395
396                 \subsection{ABOUT ME ACTIVITY}
397                 \begin{multipageListing}
398                         \inputminted{kotlin}{codes/AboutMeActivity.kt}
399                         \captionof{listing}{\textit{Source Code} untuk \textit{About Me
                            ↪  Activity}}
400                 \end{multipageListing}
401 \end{document}
```

Kode 1: *Source code* LaTeX untuk membuat lampiran

A. PRODUCT MANUAL



PENDAHULUAN

Nydrobionics adalah *mobile application* berbasis Android untuk monitoring kebun hidroponik agar memudahkan system pelaporan kondisi kebun dan meningkatkan akurasi data saat dicek dan dilaporkan.

Aplikasi ini memungkinkan pengguna untuk memantau keadaan kebun hidroponik dengan tujuan dapat mengatur konsistensi variabel-variabel yang memengaruhi kualitas hasil panen.

FITUR YANG DIKEMBANGKAN

HYDROPONIC MONITORING

Pemantauan keadaan kebun seperti kondisi suhu, kelembaban udara , tingkat keasaman air, kejernihan air, kapasitas tanki air dan pupuk cair

PLANT NYDRODB

Database tanaman seperti nama tanaman, nilai ideal (suhu, kelembaban udara, dan pH air), dan estimasi waktu yang dibutuhkan untuk siap panen.

SCHEDULE AND NOTES

Database catatan dan jadwal untuk petani hidroponik

Fitur Utama App.

Untuk menggunakan Nydrobionics, pengguna diharuskan untuk :

1. ***Memiliki akun Aktif***
   Terdaftar sebagai salah satu pengguna Nydrobionics dan akun terhubung dengan aplikasi
2. ***Menjadi anggota kebun***
   Terdaftar sebagai Owner maupun Staff dari sebuah kebun yang terdaftar
3. ***Mendaftarkan kit hidroponik***
   Terdapat seminimalnya 1 kit hidroponik untuk kebun yang terhubung dengan akun pengguna

Memiliki Akun Aktif

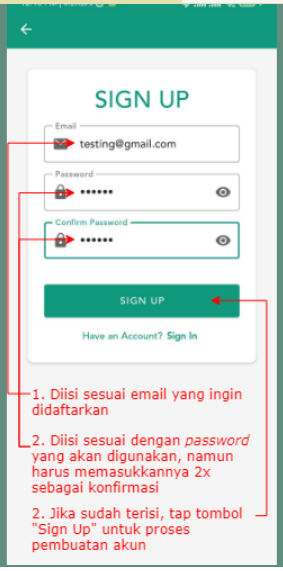
- **Sign In**
  Pengguna memiliki akun dan melakukan autentikasi untuk masuk aplikasi
- **Forget Password**
  Pengguna memiliki akun namun lupa dengan *password*-nya sehingga meminta url konfirmasi ke emailnya untuk mengganti *password*
- **Sign Up & Create User Profile**
  Pengguna belum memiliki akun sehingga mendaftarkan akun dan membuat *user profile*

Mobile Application & Technology : Fernanda Daymara Hasna

Memiliki Akun Aktif (1)

Sign Up

Membuat akun pengguna dengan memasukkan email dan *password* yang digunakan untuk proses autentikasi.



Memiliki Akun Aktif (2)

Create User Profile

Setelah melakukan *Sign Up*, pengguna membuat *user profile* dengan memasukkan data sesuai dengan kolom yang tersedia.

Memiliki Akun Aktif (3)

Menjadi Anggota Kebun

Create Farm Profile

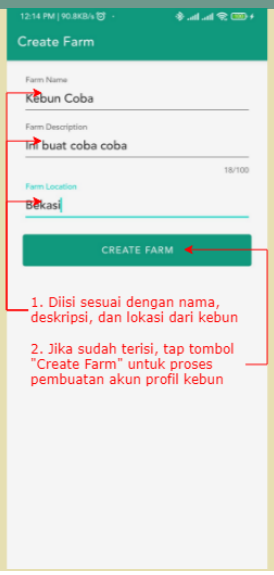
Pengguna yang berperan sebagai Owner akan otomatis diarahkan untuk mendaftarkan *Farm Profile* sehingga sudah termasuk sebagai anggota kebun

Edit Farm Profile

Pengguna yang berperan sebagai Owner dapat mengubah data *Farm*

Create Farm Profile

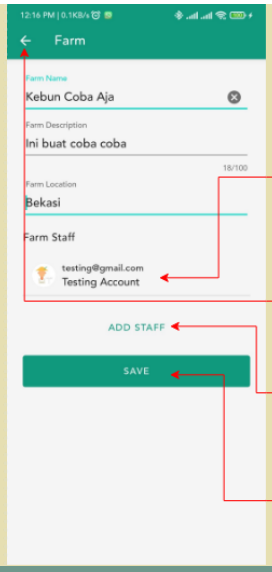
Setelah melakukan *Create User Profile*, pengguna yang berperan sebagai *Owner* membuat *Farm Profile*



Menjadi Anggota Kebun (1)

Edit Farm Profile

Saat menggunakan aplikasi, pengguna yang berperan sebagai Owner dapat mengubah data kebun maupun memodifikasi staff-nya, baik menambah maupun mengurangi staff



Muncul ketika tombol "Farm Profile" dipilih untuk melihat detail dan mengubah profil kebun

Bentuk layout dan fungsionalitas tidak jauh berbeda dengan Create Farm Profile, hanya saja ada Farm Staff untuk menampilkan dan mengelola list anggota kebun (Coming Soon)

Dapat dipilih jika ingin keluar dari Farm Profile. Jika ada data yang terubah, maka akan menyimpan data tersebut

Dapat dipilih jika ingin menambahkan akun lain sebagai staff kebun (Coming Soon)

Tombol "Save" hanya bisa dipilih jika ada perubahan dalam data, termasuk foto profil

Menjadi Anggota Kebun (2)

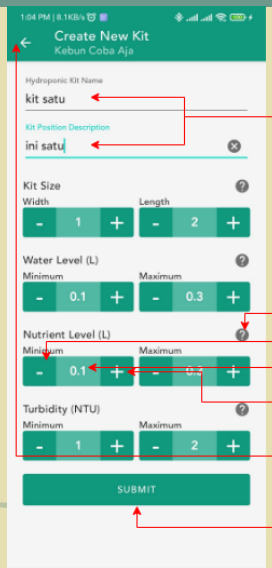
Mendaftarkan Kit Hidroponik

- **Add and Edit Kit**
  Pengguna yang sudah terdaftar pada kebun dapat menambah maupun memdifikasi kit hidroponik
- **Add Data Monitoring & Crops**
  Kit yang sudah didaftarkan dapat ditambahkan *data monitoring* maupun *crops*-nya
- **Show Kit Profile**
  Pengguna yang berperan sebagai Owner dapat mengubah data *Farm* temasuk mengelola anggota *staff*-nya

Mobile Application & Technology : Fernanda Daymara Hasna

Add and Edit Kit

Pengguna dapat menambahkan kit dari kebun yang di-*assign* dengan memasukkan data-data yang bersesuaian. Jika dibutuhkan, pengguna juga dapat memodifikasi data tersebut nantinya.



Muncul ketika menu "Add Kit" dipilih atau saat menampilkan setting kit untuk dilihat dan bisa diubah

Seluruh kolom dan counter harus diisi sesuai data yang berkaitan. Jika mengubah kit, maka data telah terisi sesuai kit yang dipilih

Terdiri dari 8 counter untuk memberikan nilai ukuran kit (panjang lebar) dan nilai min-max untuk kapasitas air, nutrisi, dan kejernihan air yang masing-masing ada 4 komponen

Tombol menampilkan info

Tombol mengurangi nilai

Menampilkan hasil dari klik tombol maupun slide

Tombol menambahkan nilai

Dapat dipilih jika ingin keluar dari Add/Edit Kit. Jika saat Edit Kit ada data yang terubah, maka akan menyimpan data tersebut

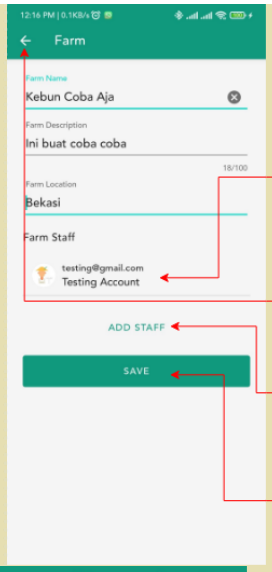
Tombol "Submit" hanya bisa dipilih jika semua data terisi dan memenuhi persyaratan (min < max)

Mendaftarkan Kit Hidroponik (1)

Add Data Monitoring

Pengguna dapat menambahkan data monitoring pada kit yang telah didaftarkan dengan memasukkan data-data yang bersesuaian

Mendaftarkan Kit Hidroponik (2)



Muncul ketika tombol "Farm Profile" dipilih untuk melihat detail dan mengubah profil kebun

Bentuk layout dan fungsionalitas tidak jauh berbeda dengan Create Farm Profile, hanya saja ada Farm Staff untuk menampilkan dan mengelola list anggota kebun (Coming Soon)

Dapat dipilih jika ingin keluar dari Farm Profile. Jika ada data yang terubah, maka akan menyimpan data tersebut

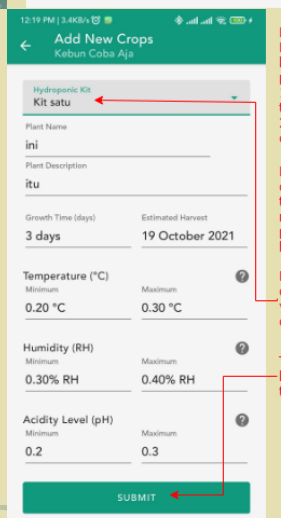
Dapat dipilih jika ingin menambahkan akun lain sebagai staff kebun (Coming Soon)

Tombol "Save" hanya bisa dipilih jika ada perubahan dalam data, termasuk foto profil

Add Crops

Jika kit tersebut belum terdapat tanaman yang ditanam, maka pengguna dapat menambahkan tanaman melalui tombol "Add Crops". Kemudian diberikan pilihan Select Plant dari database yang tersimpan dan Add New Plant

Mendaftarkan Kit Hidroponik (3)



Muncul ketika menu "Add Data Crops" dipilih, kemudian diberikan 2 pilihan :
1. Select Plant dari tanaman yang sudah ada
2. Add New Plant untuk diarahkan ke Add Plant

Ketika tanaman berhasil dipilih/dibuat, maka data tersebut dikembalikan ke menu ini untuk ditampilkan pada kolom yang bersesuaian

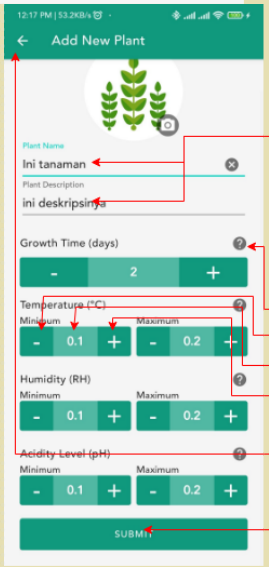
Diklik untuk membuka dropdown dan memilih kit yang ingin ditambah datanya

Tombol "Submit" hanya bisa dipilih jika kit dan tanaman sudah diisi

Add and Edit Plant

Pengguna dapat menambahkan data tanaman ke Plant NydroDB dengan memasukkan data yang bersesuaian. Jika pengguna menambahkannya saat Add Data Monitoring atau Add Crops, maka data tanaman tersebut akan langsung ditambahkan sebagai tanaman yang ditanam.

Mendaftarkan Kit Hidroponik (4)



Muncul ketika menu "Add Plant" dipilih atau saat melihat atau mengubah postingan akun

Selain foto, seluruh kolom dan counter harus diisi sesuai data yang berkaitan. Jika mengubah tanaman, maka data telah terisi sesuai tanaman yang dipilih

Terdiri dari 7 counter untuk memberikan nilai estimasi panen dan nilai min-max untuk suhu, kelembaban, dan pH yang masing-masing ada 4 komponen

Tombol menampilkan info

Tombol mengurangi nilai

Menampilkan hasil dari klik tombol maupun slide

Tombol menambahkan nilai

Dapat dipilih jika ingin keluar dari Add/Edit Plant. Jika saat Edit Kit ada data yang terubah, maka akan menyimpan data tersebut

Tombol "Submit" hanya bisa dipilih jika semua data terisi dan memenuhi persyaratan (min < max)

Show Kit Profile

Saat masuk ke aplikasi, pengguna dapat melihat data terbaru dari setiap kit pada kebunnya. Jika tombol "Show Detail" dipilih, maka akan menampilkan data lebih detailnya.

Muncul ketika autentikasi berhasil atau ketika "Home" dipilih

Dapat dipilih jika ingin melihat dan mengubah profil kebun

Menampilkan nama kebun yang terhubung dengan akun pengguna

Menampilkan data monitoring terakhir dari setiap kit yang dipilih :
1. Nama
2. Update terakhir
3. Nilai monitoring (suhu, kelembaban, pH, kapasitas air & pupuk, serta kejernihan air)

Dapat mengganti kit yang dilihat dengan tombol Next Previous

Dapat melihat profil kit lebih detail dengan tombol "Show More"

Digunakan untuk menuju ke menu lainnya pada Main Application

Mendaftarkan Kit Hidroponik (5)

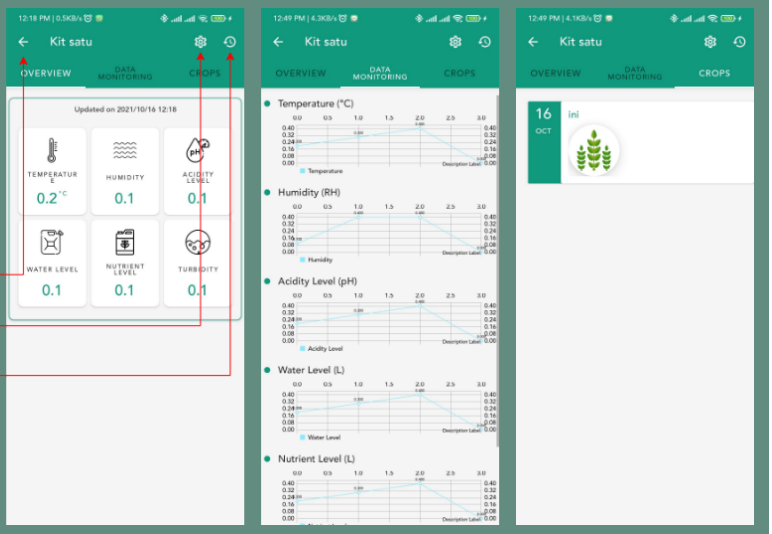
Muncul ketika memilih "Show More" pada kit yang bersangkutan. Secara default menampilkan bagian "Overview" data monitoring terbaru

Terdapat 3 tab :
1. Overview yaitu data monitoring terakhir
2. Data Monitoring yaitu rekapitulasi monitoring kebun dalam bentuk grafik
3. Crops yaitu rekapitulasi tanaman yang pernah ditanam

Dapat dipilih jika ingin keluar dari Kit Profile

Dapat dipilih jika ingin melihat detail dan mengubah data kit

Dapat dipilih jika ingin menampilkan historis mengenai kit (Coming Soon)



Mendaftarkan Kit Hidroponik (5)

Fitur Pendukung App.

Untuk mendukung fitur utama pada Nydrobionics, disediakan fitur-fitur tambahan seperti :

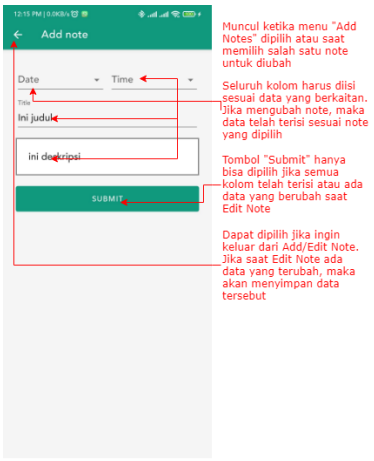
1. ***Add, Edit, and Show Notes***
   Menambah, memodifikasi, dan menampilkan catatan pengguna yang ter-sign-in

2. ***Show Plants***
   Menampilkan seluruh tanaman yang telah dibuat oleh pengguna lainnya

3. ***Show and Edit User Profile***
   Menampilkan data pengguna lain sesuai yang dipilih dan mengedit data pengguna ter-sign-in

Mobile Application & Technology : Fernanda Daymara Hasna

Add, Edit, & Show Notes

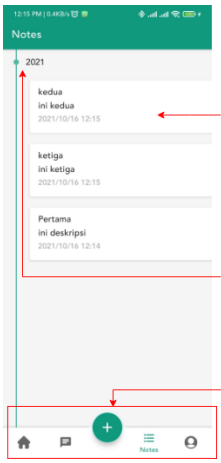


Muncul ketika menu "Add Notes" dipilih atau saat memilih salah satu note untuk diubah

Seluruh kolom harus diisi sesuai data yang berkaitan. Jika mengubah note, maka data telah terisi sesuai note yang dipilih

Tombol "Submit" hanya bisa dipilih jika semua kolom telah terisi atau ada data yang berubah saat Edit Note

Dapat dipilih jika ingin keluar dari Add/Edit Note. Jika saat Edit Note ada data yang terubah, maka akan menyimpan data tersebut



Muncul ketika ketika "Notes" dipilih

Menampilkan catatan yang pernah dibuat oleh pengguna yang sedang aktif yang diurutkan berdasarkan terbaru :
1. Judul catatan
2. Isi catatan
3. Update terakhir

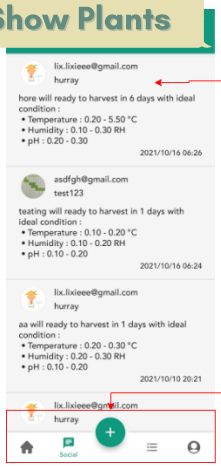
Jika di tap, akan menunjukkan menu Edit Notes yang bersangkutan

Mengelompokkan notes berdasarkan tahun pembuatan

Digunakan untuk menuju ke menu lainnya pada Main Application

Mobile Application & Technology : Fernanda Daymara Hasna

Show Plants

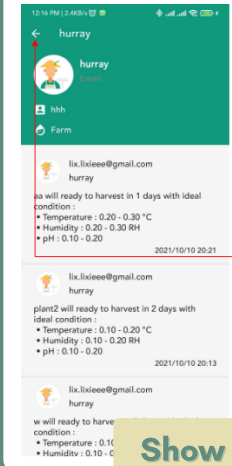


Muncul ketika "Social" dipilih

Menampilkan tanaman yang pernah dibuat oleh seluruh akun pengguna Nydrobionics yang diurutkan dari yang terbaru :
1. Nama dan email
2. Data tanaman
3. Timestamp pembaruan terakhir

Jika di tap, akan menunjukkan User Profile dari pembuat post

Digunakan untuk menuju ke menu lainnya pada Main Application



Muncul ketika memilih salah satu postingan untuk melihat detail akun pembuatnya

Berisi profil pengguna pemilik akun beserta postingan yang pernah dibuatnya. Jika berada dalam kebun yang sama, data pribadi seperti alamat, nomor telepon dapat terlihat (Coming Soon)

Dapat dipilih jika ingin keluar dari User Profile tersebut

Show User Profile

Mobile Application & Technology : Fernanda Daymara Hasna

Nydrobionics

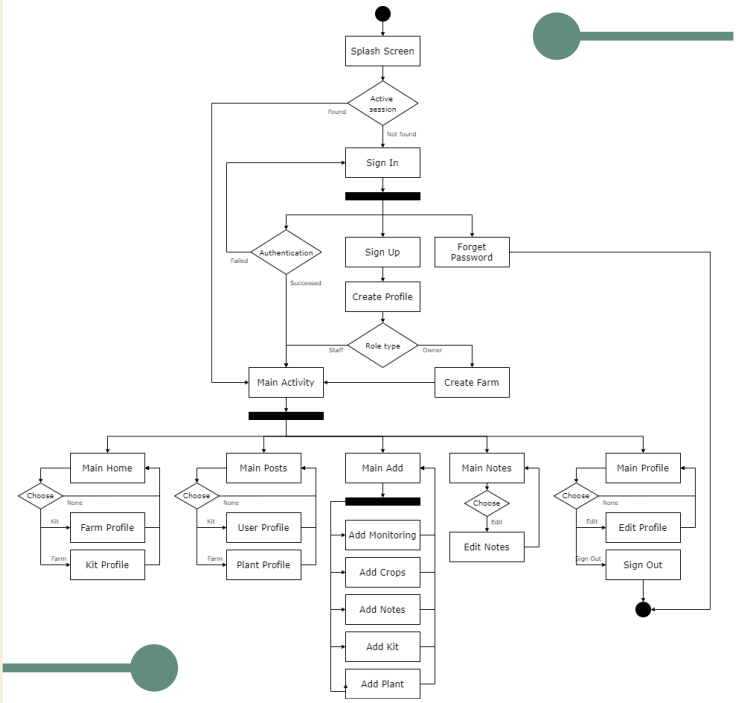
Mobile Application & Technology : Fernanda Daymara Hasna

B. FRONT END

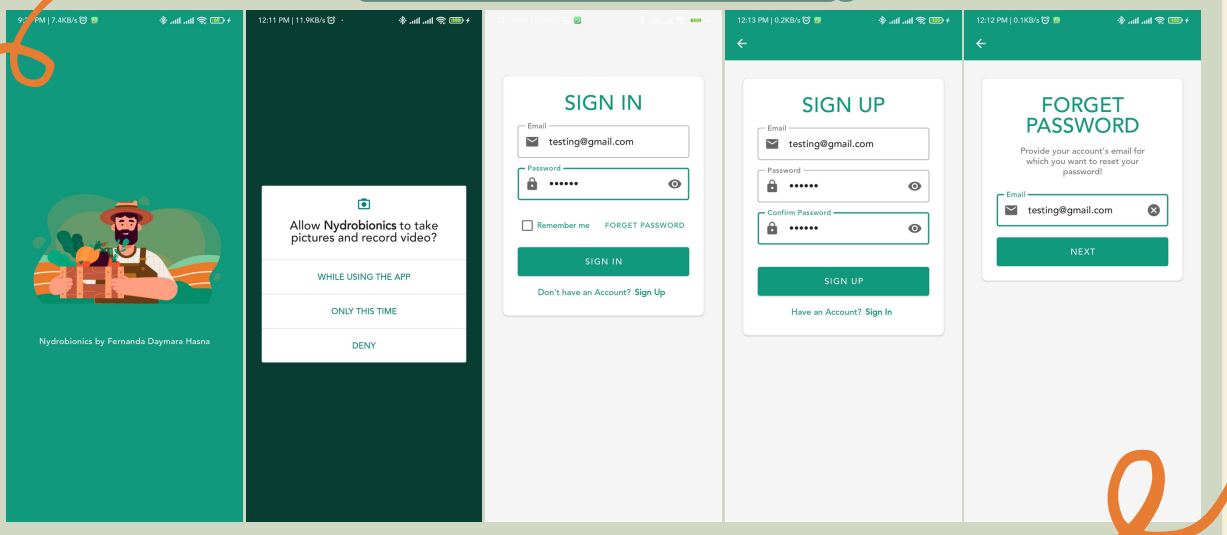
Activity Diagram

- ***Application Intro***
  Splash Screen, Sign In, Sign Up, Forget Password
- ***Create and Edit Profile***
  Create User Profile, Create Farm Profile
- ***Main Application***
  Main Home, Main Posts, Main Add, Main Notes, Main Profile
- ***Profile***
  User Profile, Farm Profile, Kit Profile
- ***Add and Edit***
  Data Monitoring, Crops, Note, Kit, Plant

Mobile Application & Technology : Fernanda Daymara Hasna

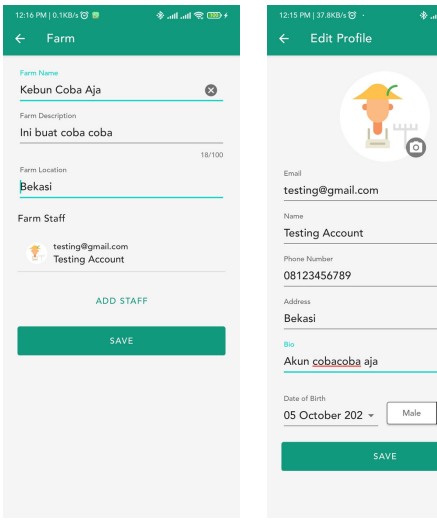


Application Intro

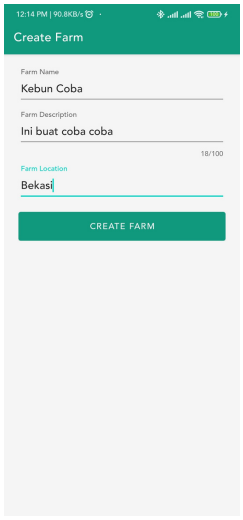


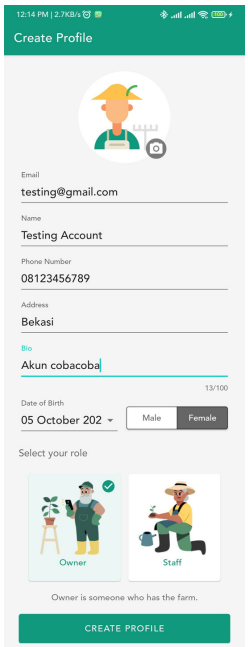
Mobile Application & Technology : Fernanda Daymara Hasna

Edit Profile



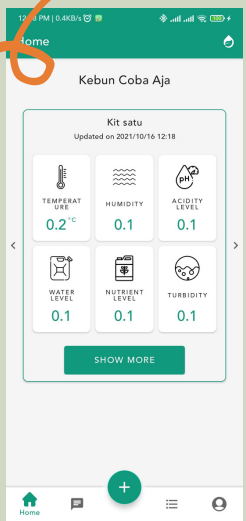
Create Profile

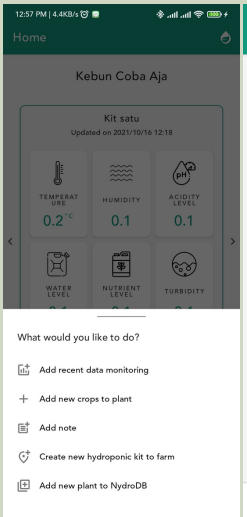


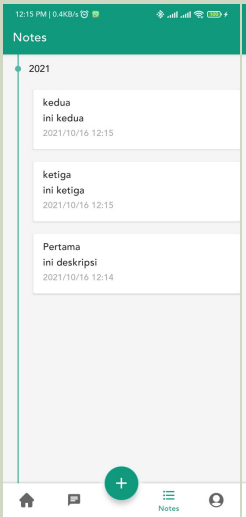


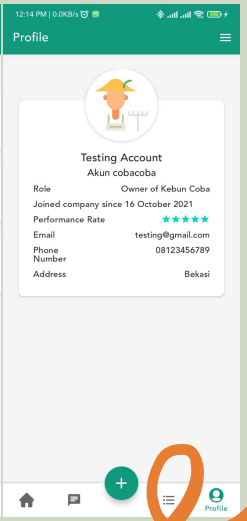
Mobile Application & Technology : Fernanda Daymara Hasna

Main Application



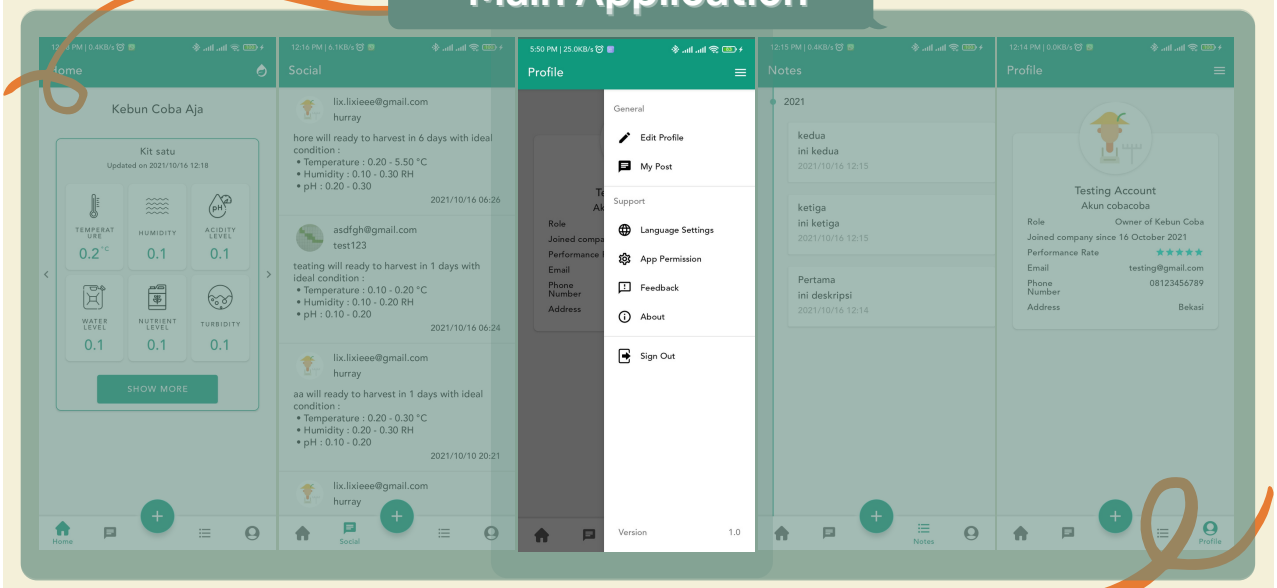






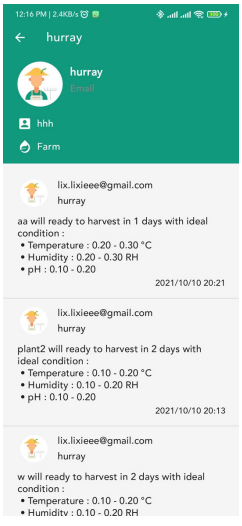
Mobile Application & Technology : Fernanda Daymara Hasna

Main Application

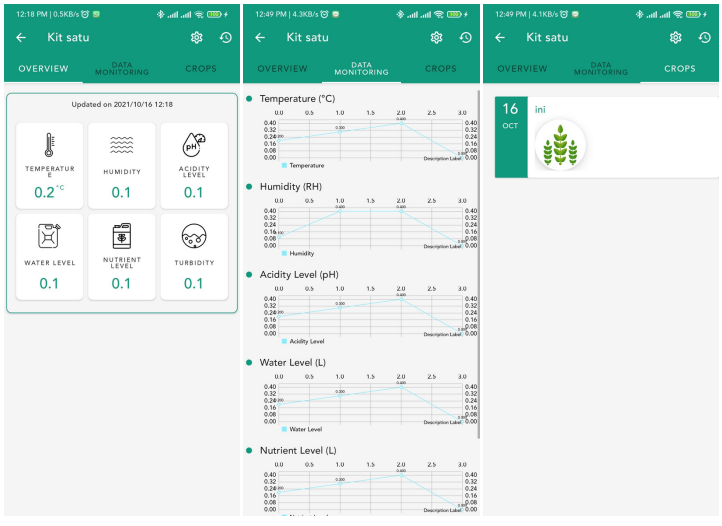


Mobile Application & Technology : Fernanda Daymara Hasna

User Profile

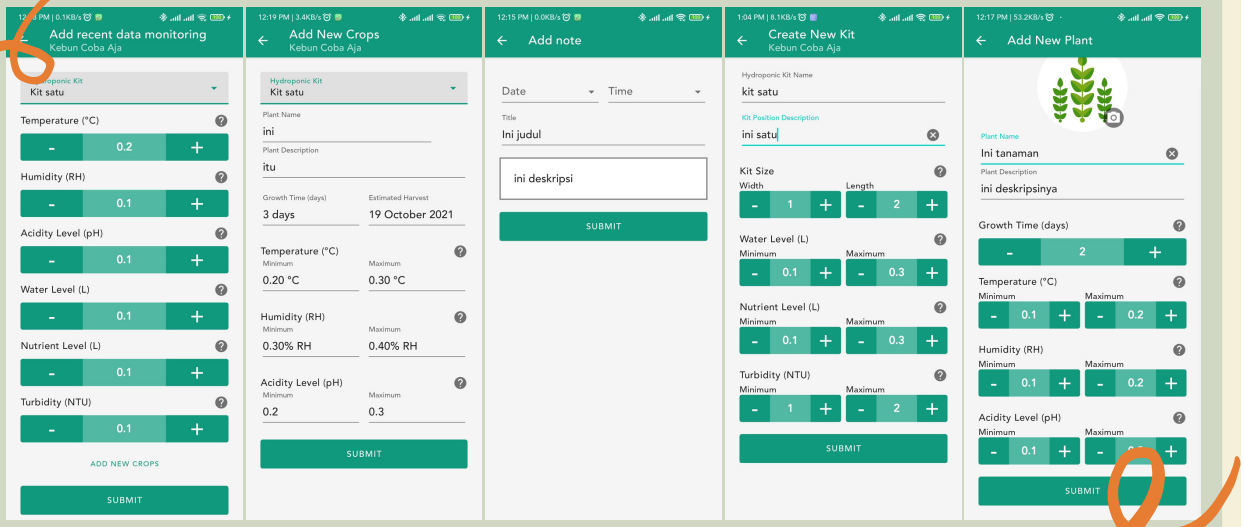


Kit Profile



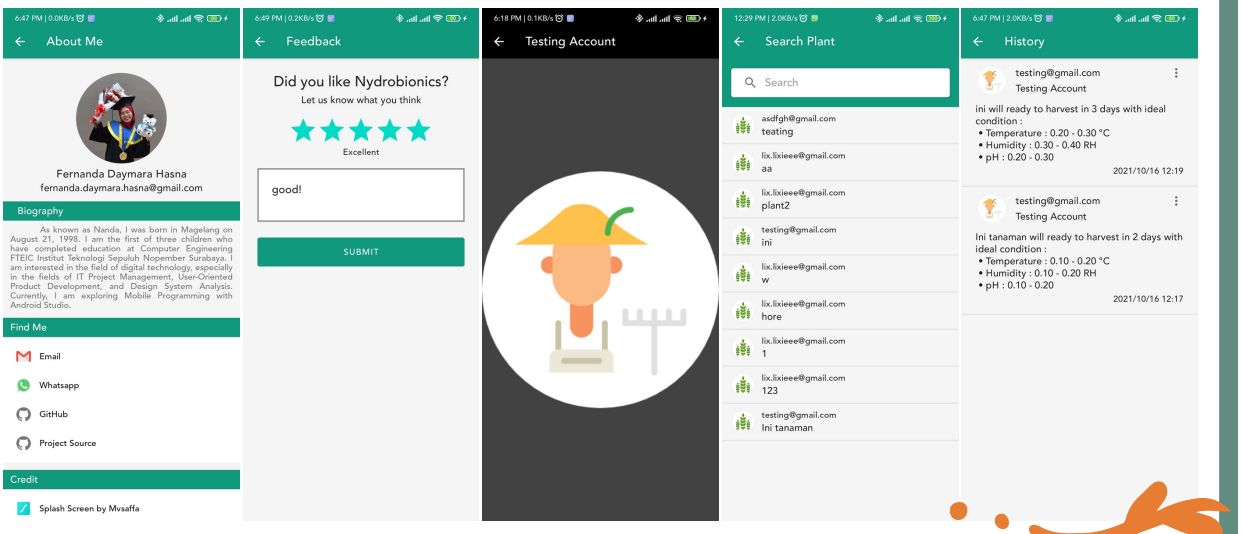
Mobile Application & Technology : Fernanda Daymara Hasna

Add and Edit Data



Mobile Application & Technology : Fernanda Daymara Hasna

Others



Mobile Application & Technology : Fernanda Daymara Hasna

C. DATA MODEL

1. USER MODEL

```
@Parcelize
data class UserModel(
    var name : String? = null,
    var email: String? =null,
    var gender : String? = null,
    var dob : String? = null,
    var role : String? = null,
    var bio : String? = null,
    var uid : String? = null,
    var performanceRate : Float? = null,
    var address : String? = null,
    var phone : String? = null,
    var joinedSince : String? = null,
    var photo_url : String? = null,
    var farmId : String? = null,
    var timestamp : String? = null
) : Parcelable
```

Kode 2: *Data class* untuk *User*

2. NOTE MODEL

```
@Parcelize
data class NoteModel(
    var noteId : String? = null,
    var title: String? = null,
    var description : String? = null,
    var timestamp : String?=null,
    var date : String? = null,
    var time : String? = null
) : Parcelable
```

Kode 3: *Data class* untuk *Note*

3. FARM MODEL

```
@Parcelize
data class FarmModel(
    var farmId : String? = null,
    var name : String? = null,
    var description : String? = null,
    var location : String? = null,
    var gps : String? = null,
    var timestamp : String? = null,
    var kitModels : ArrayList<KitModel>? = null
) : Parcelable
```

Kode 4: *Data class* untuk *Farm*

4. KIT MODEL

```
@Parcelize
data class KitModel(
    var kitId : String? = null,
    var name : String?= null,
    var position : String? = null,
    var length : Int? = null,
    var width : Int? = null,
    var waterLv : @RawValue ScoreLevel? = null,
```

```
    var nutrientLv : @RawValue ScoreLevel? = null,
    var turbidityLv: @RawValue ScoreLevel? = null,
    var isPlanted : Boolean? = false,
    var timestamp : String? = null,
    var lastMonitoring : DataMonitoringModel? = null,
    var lastCrops : CropsModel? = null
)  : Parcelable
```

Kode 5: *Data class* untuk *Hydroponic Kit*

5. DATA MONITORING MODEL

```
@Parcelize
data class DataMonitoringModel(
    var dataId : String? = null,
    var temperature : Float? = 0f,
    var humidity : Float? = 0f,
    var waterTank : Float? = 0f,
    var nutrientTank : Float? = 0f,
    var turbidity : Float? =0f,
    var ph : Float? = 0f,
    var userId : String? = null,
    var cropsId : String? = null,
    var timestamp : String? = null
) : Parcelable
```

Kode 6: *Data class* untuk *Data Monitoring*

6. CROPS MODEL

```
@Parcelize
data class CropsModel(
    var cropsId : String? = null,
    var plantId : String? = null,
    var userId : String? = null,
    var tempLv : @RawValue ScoreLevel? = null,
    var humidLv : @RawValue ScoreLevel? = null,
    var phLv : @RawValue ScoreLevel? = null,
    var timestamp : String? = null,
    var plantModel: PlantModel? = null
) : Parcelable
```

Kode 7: *Data class* untuk *Crops*

7. PLANT MODEL

```
@Parcelize
data class PlantModel(
    var plantId : String? = null,
    var name : String? = null,
    var description : String? = null,
    var photo_url : String? = null,
    var tempLv : @RawValue ScoreLevel? = null,
    var humidLv : @RawValue ScoreLevel? = null,
    var phLv : @RawValue ScoreLevel? = null,
    var userId : String? = null,
    var growthTime : Int? = 0,
    var timestamp : String? = null
) : Parcelable
```

Kode 8: *Data class* untuk *Plant*

8. FEEDBACK MODEL

```
@Parcelize
data class FeedbackModel(
    var feedbackId : String? = null,
    var userId : String? = null,
    var content : String? = null,
    var rating : Float? = null,
    var timestamp: String? = null
) : Parcelable
```

Kode 9: *Data class* untuk *Feedback*

9. SCORE LEVEL MODEL

```
@Parcelize
data class ScoreLevel(
    var min : Float? = 0f,
    var max : Float? = 0f,
    var score : Float? = 0f
) : Parcelable {
    companion object {
        fun getLevelModel(string: String?) : ScoreLevel? {
            return try {
                val output = Gson().fromJson<ScoreLevel>(string, ScoreLevel::class.java)
                ScoreLevel(output.min, output.max)
            } catch (e:Exception){
                Log.e(TAG, "Error converting $TAG", e)
                null
            }
        }

        private const val TAG = "ScoreLevel"
    }
}
```

Kode 10: *Data class* untuk data yang memiliki nilai minimum dan maksimum

10. URI FILE EXTENSIONS MODEL

```
data class UriFileExtensions(
    var uri : Uri,
    var fileExtensions : String
)
```

Kode 11: *Data class* untuk yang digunakan untuk mengelola *attach* gambar

D. VIEW MODEL

1. SPLASH SCREEN VIEW MODEL

```
class SplashScreenViewModel : ViewModel() {
    private var auth : FirebaseAuth = Firebase.auth
    private var firestore : FirebaseFirestore = Firebase.firestore
    var currentUserModel : MutableLiveData<UserModel?> = MutableLiveData(null)
    var currentFarmModel : MutableLiveData<FarmModel?> = MutableLiveData(null)
    var isCurrentUserExist : MutableLiveData<Boolean?> = MutableLiveData(null)
    var isCurrentFarmExist : MutableLiveData<Boolean?> = MutableLiveData(null)

    companion object {
        const val TAG = "splashScreen"
    }

```

```
    fun getCurrentData(){
        try{
            auth.uid?.let {
                firestore.collection("users").document(auth.uid!!)
                    .get().addOnSuccessListener {
                        if(it.exists()){
                            currentUserModel.value = it.toUserModel()
                            isCurrentUserExist.value = true
                            currentUserModel.value?.farmId?.let {
                                firestore.collection("farms").document(it)
                                    .get().addOnSuccessListener {
                                        if(it.exists()){
                                            currentFarmModel.value = it.toFarmModel()
                                            Log.i(
                                                TAG,
                                                "getCurrentData:
                                                ↪  ${currentUserModel.value}\n" +
                                                        "${currentFarmModel.value}"
                                            )
                                            isCurrentFarmExist.value = true
                                        } else {
                                            isCurrentFarmExist.value = false
                                        }
                                    }
                            } ?: kotlin.run {
                                isCurrentFarmExist.value = false
                            }

                        } else {
                            isCurrentUserExist.value = false
                        }
                    }
            } ?: kotlin.run {
                isCurrentUserExist.value = false
            }
        } catch (e:Exception){
            Log.e(TAG, "getCurrentData ${auth.uid}", e)
        }
    }
}
```

Kode 12: *View Model* untuk *Splash Screen Activity*

2. SIGN IN VIEW MODEL

```
class SignInViewModel : ViewModel() {
    private var auth = Firebase.auth
    private var isRememberChecked : MutableLiveData<Boolean> = MutableLiveData(false)
    private var isNotEmpties : MutableLiveData<Boolean> = MutableLiveData(false)
    private var localDataModel : MutableLiveData<DatabaseModel> =
    ↪  MutableLiveData(DatabaseModel())
    var isUserSignIn : MutableLiveData<Boolean> = MutableLiveData(false)
    var signInError : MutableLiveData<String?> = MutableLiveData(null)

    fun checkNotEmpty(boolean: Boolean) : MutableLiveData<Boolean> {
        isNotEmpties.value = boolean
        return isNotEmpties
    }

    fun getDataModel() : DatabaseModel = localDataModel.value!!

    fun setRememberChecked(boolean: Boolean){
        isRememberChecked.value = boolean

    }

    fun signIn(email:String, password:String){
        auth.signInWithEmailAndPassword(email, password).addOnCompleteListener {
            if(it.isSuccessful){
```

```
                localDataModel.value = DatabaseModel(
                    uid = auth.uid,
                    email = email,
                    remember = isRememberChecked.value!!
                )
                isUserSignIn.value = true
            } else {
                signInError.value = it.exception.toString()
                isUserSignIn.value = false
            }
        }
    }
}
```

Kode 13: *View Model* untuk *Sign In Activity*

3. SIGN UP VIEW MODEL

```
class SignUpViewModel : ViewModel() {
    private var auth = Firebase.auth
    private var isNotEmpties : MutableLiveData<Boolean> = MutableLiveData(false)
    var isUserSignUp : MutableLiveData<Boolean> = MutableLiveData(false)
    var signUpError : MutableLiveData<String?> = MutableLiveData(null)

    fun checkNotEmpty(boolean: Boolean) : MutableLiveData<Boolean> {
        isNotEmpties.value = boolean
        return isNotEmpties
    }

    fun signUp(email:String, password:String){
        signUpError.value = null
        auth.createUserWithEmailAndPassword(email, password).addOnCompleteListener {
            if(it.isSuccessful){
                isUserSignUp.value = true
            } else {
                signUpError.value = it.exception.toString()
                isUserSignUp.value = false
            }
        }
    }
}
```

Kode 14: *View Model* untuk *Sign Up Activity*

4. FORGET PASSWORD VIEW MODEL

```
class ForgetPasswordViewModel : ViewModel() {
    private var auth = Firebase.auth
    private var isEmailNotEmpty : MutableLiveData<Boolean> = MutableLiveData(false)
    var isUserForgetPassword : MutableLiveData<Boolean> = MutableLiveData(null)
    var forgetPasswordError : MutableLiveData<String> = MutableLiveData("")

    fun checkNotEmpty(boolean: Boolean) : MutableLiveData<Boolean>  {
        isEmailNotEmpty.value = boolean
        return isEmailNotEmpty
    }

    fun sendEmailReset(email:String) {
        forgetPasswordError.value = ""
        auth.sendPasswordResetEmail(email).addOnCompleteListener {
            if(it.isSuccessful){
                isUserForgetPassword.value = true
            } else {
                forgetPasswordError.value = it.exception.toString()
                isUserForgetPassword.value = false
            }
```

```
        }
    }
}
```

Kode 15: *View Model* untuk *Forget Password Activity*

5. CREATE PROFILE VIEW MODEL

```
class CreateProfileViewModel : ViewModel() {
    private val today = ViewUtility().getCurrentDate()
    private var auth : FirebaseAuth = Firebase.auth
    private var storage : FirebaseStorage = Firebase.storage
    private var firestore : FirebaseFirestore = Firebase.firestore
    private var userImageUri : MutableLiveData<UriFileExtensions> =
    ↪ MutableLiveData(null)

    private var userModel : MutableLiveData<UserModel> = MutableLiveData(
        UserModel(gender = Gender.MALE.toString(),
             dob = today)
    )
    private var farmModel : MutableLiveData<FarmModel> = MutableLiveData(FarmModel())
    private var updateStaff : MutableLiveData<ArrayList<UserModel>> =
    ↪ MutableLiveData(arrayListOf())
    private var currentStaff : MutableLiveData<ArrayList<UserModel>> =
    ↪ MutableLiveData(null)

    var isNotEmpties : MutableLiveData<Boolean> = MutableLiveData(false)
    var isUserCreated : MutableLiveData<Boolean> = MutableLiveData(false)
    var isFarmCreated : MutableLiveData<Boolean> = MutableLiveData(false)
    var isStaffAdded : MutableLiveData<Boolean> = MutableLiveData(false)
    var createProfileError : MutableLiveData<String> = MutableLiveData("")

    private var updateUser : MutableLiveData<UserModel> = MutableLiveData(null)

    companion object{
        const val TAG = "createProfileViewModel"
    }

    fun checkNotEmpty(boolean: Boolean) : MutableLiveData<Boolean>{
        isNotEmpties.value = boolean
        return isNotEmpties
    }

    /** SET MODEL **/
    fun setCurrentUser(userModel: UserModel?){
        userModel?.let {
            this.userModel.value = userModel
            isUserCreated.value = true
            updateUserGender.value  =
            ↪ Gender.getType(userModel.gender!!)?.getPosition()!!
        }
    }

    fun setCurrentFarm(farmModel : FarmModel?, userModel: UserModel?){
        this.farmModel.value = farmModel
        this.userModel.value = userModel
        isFarmCreated.value = true
        isUserCreated.value = true
    }

    fun getCurrentUserModel() : UserModel? = userModel.value

    fun getCurrentFarmModel() : FarmModel? = farmModel.value


    /** CREATE USER FRAGMENT **/
    fun setPhotoProfile(uri : Uri?, fileExtension: String?=null){
        userImageUri.value = if(uri != null){
             UriFileExtensions(uri, fileExtension!!)
```

```
58          } else {
59              null
60          }
61          Log.i(TAG, "setPhotoProfile: $uri")
62      }
63
64      fun getPhotoProfile() : MutableLiveData<UriFileExtensions>{
65          return userImageUri
66      }
67
68      fun setRole(role:Role){
69          userModel.value?.role = role.toString()
70      }
71
72      fun setGender(gender: Gender){
73          userModel.value?.gender = gender.toString()
74      }
75
76      fun setDOB(date:Long?) : String? {
77          val temp : UserModel? = userModel.value
78          temp?.dob =  ViewUtility().formatDateToString(date)
79          userModel.value = temp
80          Log.i(TAG, "setDOB: $date")
81          return userModel.value?.dob
82      }
83
84      fun getDOB() : Long{
85          if(userModel.value?.dob == null){
86              val temp : UserModel? = userModel.value
87              temp?.dob = ViewUtility().getCurrentDate()
88              userModel.value = temp
89          }
90          return ViewUtility().formatStringToDate(userModel.value?.dob)
91      }
92
93      fun createUserProfile(name:String, phone:String, address:String, bio:String) {
94          try {
95              userModel.value?.apply {
96                  this.name = name
97                  this.email = auth.currentUser?.email
98                  this.phone = phone
99                  this.address = address
100                 this.bio = bio
101                 this.uid = auth.uid
102                 this.joinedSince = today
103                 this.performanceRate = 5.0f
104             }
105
106             if(userImageUri.value != null){
107                 val storageReference : StorageReference =
                    ↪  storage.getReference("profile_images")
108                     .child(System.currentTimeMillis().toString() +
                        ↪  ".${userImageUri.value?.fileExtensions!!}")
109                 val uploadTask = storageReference.putFile(userImageUri.value?.uri!!)
110                 uploadTask.continueWithTask {
111                     if(!it.isSuccessful){
112                         throw it.exception!!.cause!!
113                     }
114                     storageReference.downloadUrl
115                 }.addOnCompleteListener {
116                     if (it.isSuccessful) {
117                         it.result.let {
118                             userModel.value?.photo_url = it.toString()
119                             sendUserProfile()
120                         }
121                     }
122                 }
123             } else {
124                 sendUserProfile()
125             }
```

```
126             } catch (e:Exception){
127                 Log.e(TAG, "Error submit user", e)
128             }
129         }
130
131         private fun sendUserProfile(){
132             createProfileError.value = ""
133             try {
134                 firestore.collection("users").document(auth.uid!!).set(userModel.value!!.to
                    ↪ HashMap())
135                     .addOnCompleteListener {
136                         if (it.isSuccessful) {
137                             isUserCreated.value = true
138                             isNotEmpties.value = false
139                         } else {
140                             createProfileError.value = it.exception.toString()
141                             isUserCreated.value = false
142                         }
143                     }
144             } catch (e:Exception){
145                 Log.e(TAG, "sendUserProfile failed", e)
146             }
147         }
148
149
150         /** CREATE FARM FRAGMENT **/
151         fun createFarmProfile(name:String, description:String, location:String){
152             try {
153                 createProfileError.value = ""
154                 val db = firestore.collection("farms")
155                 val ref : DocumentReference = db.document()
156
157                 farmModel.value?.apply {
158                     this.farmId = farmId ?: ref.id
159                     this.name = name
160                     this.description = description
161                     this.location = location
162                 }
163
164                 db.document(farmModel.value!!.farmId!!).set(farmModel.value!!.toHashMap()).
                    ↪ addOnCompleteListener
                    ↪ {
165                     if(it.isSuccessful){
166                         userModel.value?.farmId = farmModel.value!!.farmId!!
167                         sendUserProfile()
168                         isFarmCreated.value = isUserCreated.value == true
169                     } else {
170                         createProfileError.value = it.exception.toString()
171                         isFarmCreated.value = false
172                     }
173                 }
174             } catch (e:Exception){
175                 Log.e(TAG, "Error submit farm", e)
176             }
177         }
178
179
180         /** EDIT FARM **/
181         fun getCurrentStaff() : MutableLiveData<ArrayList<UserModel>>  = currentStaff
182
183         fun getStaff() : MutableLiveData<ArrayList<UserModel>> = updateStaff
184
185         fun addStaff(userModel : UserModel){
186             updateStaff.value?.let {
187                 val array : ArrayList<UserModel> = it
188                 array.add(userModel)
189                 updateStaff.value = array
190                 Log.i(TAG, "addStaff: ${updateStaff.value}")
191             }
192         }
```

```
193
194     fun removeStaff(position : Int){
195         updateStaff.value?.let {
196             val array : ArrayList<UserModel> = it
197             array.removeAt(position)
198             updateStaff.value = array
199             Log.i(TAG, "removeStaff: ${updateStaff.value}")
200         }
201     }
202
203     fun createStaff(){
204         try {
205             farmModel.value?.let {
206                 firestore.collection("farms").document(it.farmId!!)
207                     .set(it.toHashMap())
208             }
209
210             currentStaff.value?.let {
211                 for (staff in it) {
212                     firestore.collection("users").document(staff.uid!!)
213                         .update("farmId", null)
214                 }
215             }
216
217             updateStaff.value?.let {
218                 for (staff in it) {
219                     firestore.collection("users").document(staff.uid!!)
220                         .update("farmId", farmModel.value?.farmId)
221                 }
222                 isStaffAdded.value = true
223             }
224         } catch (e:Exception){
225             Log.e(TAG, "createStaff: ", e)
226         }
227     }
228
229     fun updateStaff() {
230         try {
231             firestore.collection("users")
232                 .get().addOnCompleteListener {
233                     if(it.isSuccessful) {
234                         val staffs: ArrayList<UserModel> = arrayListOf()
235                         for (staffDocument in it.result.documents) {
236                             staffDocument?.let { staff ->
237                                 staff.toUserModel()?.let { staffModel ->
238                                     staffModel.farmId?.let {
239                                         if (it == farmModel.value?.farmId) {
240                                             staffs.add(staffModel)
241                                         }
242                                     }
243                                 }
244                             }
245                         }
246                         currentStaff.value = staffs
247                         updateStaff.value = staffs
248                     }
249                 }
250         } catch (e:Exception){
251             Log.e(TAG, "updateCurrentStaff for farm ${farmModel.value?.farmId}: ", e)
252         }
253     }
254
255     /** EDIT USER **/
256     private var updateUserGender : MutableLiveData<Int> = MutableLiveData(0)
257     fun getUpdateGender() : Int? = updateUserGender.value
258 }
```

Kode 16: *View Model* untuk *Create Profile Activity* beserta *Create User* dan *Create Farm fragments*. Termasuk juga untuk *Edit Profile User* dan *Edit Profile Farm activities*

6. MAIN VIEW MODEL

```
class MainViewModel : ViewModel() {
    private var auth : FirebaseAuth = Firebase.auth
    private var firestore : FirebaseFirestore = Firebase.firestore
    private var currentUserModel : MutableLiveData<UserModel?> = MutableLiveData(null)
    private var currentFarmModel : MutableLiveData<FarmModel?> = MutableLiveData(null)
    private var currentKitModels : MutableLiveData<ArrayList<KitModel>?> =
    ↪  MutableLiveData(null)
    private var currentNoteModels : MutableLiveData<ArrayList<NoteModel>?> =
    ↪  MutableLiveData(null)
    private var allUserModels : MutableLiveData<ArrayList<UserModel>?> =
    ↪  MutableLiveData(null)
    private var allPlantModels : MutableLiveData<ArrayList<PlantModel>?> =
    ↪  MutableLiveData(null)

    companion object{
        const val TAG = "mainViewModel"
    }

    fun setCurrentData(userModel: UserModel?, farmModel: FarmModel?){
        getUsersUpdate()
        getPlantsUpdate()
        userModel?.let {
            currentUserModel.value = it
            farmModel?.let {
                currentFarmModel.value = it
            } ?: kotlin.run {
                getFarmsUpdate(currentFarmModel.value?.farmId)
            }
        } ?: kotlin.run {
            getUsersUpdate()
        }
    }

    /** GET DATA **/
    fun getCurrentUser() : MutableLiveData<UserModel?> = currentUserModel
    fun getCurrentFarm() : MutableLiveData<FarmModel?> = currentFarmModel
    fun getCurrentKits() : MutableLiveData<ArrayList<KitModel>?> = currentKitModels
    fun getCurrentNotes() : MutableLiveData<ArrayList<NoteModel>?> = currentNoteModels
    fun getAllUsers() : MutableLiveData<ArrayList<UserModel>?> = allUserModels
    fun getAllPosts() : MutableLiveData<ArrayList<PlantModel>?> = allPlantModels

    /** UPDATE DATABASE LISTENER **/
    private lateinit var usersListener : ListenerRegistration
    private lateinit var farmsListener : ListenerRegistration
    private lateinit var kitsListener : ListenerRegistration
    private var monitoringsListeners : ArrayList<ListenerRegistration> = arrayListOf()
    private var cropsListeners : ArrayList<ListenerRegistration> = arrayListOf()
    private lateinit var plantsListener : ListenerRegistration

    private fun getUsersUpdate(){
        val db = firestore.collection("users")
        try {
            usersListener =
                db.orderBy("timestamp", Query.Direction.DESCENDING)
                    .addSnapshotListener { usersSnapshot, error ->
                        usersSnapshot?.let {
                            val users : ArrayList<UserModel> = arrayListOf()
                            for(userDocument in usersSnapshot.documents){
                                userDocument?.let { user ->
                                    user.toUserModel()?.let { userModel ->
                                        users.add(userModel)
                                        val userId = userDocument.id
                                        if(auth.uid == userId) {
                                            currentUserModel.value = userModel
                                            getFarmsUpdate(userModel.farmId)

                                            val userRef = db.document(userId)
                                            userRef.collection("notes")
```

```
                                        .orderBy("timestamp",
                                        ↪  Query.Direction.DESCENDING)
                                        .addSnapshotListener { notesSnapshot,
                                        ↪  error ->
                                            notesSnapshot?.let {
                                                val notes: ArrayList<NoteModel>
                                                ↪  = arrayListOf()
                                                for (noteDocument in
                                                ↪  notesSnapshot.documents) {
                                                    noteDocument?.let { note ->
                                                        note.toNoteModel()?.let
                                                        ↪  { noteModel ->
                                                            notes.add(noteModel)
                                                        }
                                                    }
                                                }
                                                currentNoteModels.value = notes
                                                Log.i(TAG, "getUsersUpdate ${cu
                                                ↪  rrentUserModel.value?.uid}"
                                                ↪  +
                                                        " \tnotes:${currentNote
                                                        ↪  Models.value?.size}
                                                        ↪  \n"
                                                        ↪  +
                                                        "${currentUserModel.val
                                                        ↪  ue}\n")
                                            }

                                            error?.let {
                                                Log.e(TAG, "Listen notes from
                                                ↪  $userId failed", it)
                                            }
                                        }
                                    }
                                }
                            }
                            allUserModels.value = users
                            Log.i(TAG, "getUsersUpdate: ${allUserModels.value?.size}\n")
                        }

                        error?.let {
                            Log.e(TAG, "Listen users failed", it)
                        }
                    }
        } catch (e:Exception){
            Log.e(TAG, "getUsersUpdate() error", e)
        }
    }

    private fun getFarmsUpdate(farmId: String?){
        if(farmId == null){
            currentFarmModel.value = null
        } else {
            try{
                val db = firestore.collection("farms").document(farmId)
                farmsListener = db.addSnapshotListener { farmSnapshot, error ->
                    farmSnapshot?.let { farm ->
                        farm.toFarmModel()?.let { farmModel ->
                            currentFarmModel.value = farmModel

                            val kitsRef = db.collection("kits")
                            kitsListener = kitsRef.addSnapshotListener{ kitSnapshot,
                            ↪  error ->
                                kitSnapshot?.let { kitDocuments ->
                                    val kits : ArrayList<KitModel> = arrayListOf()
                                    kitDocuments.documents.forEachIndexed { index,
                                    ↪  kitDocument ->
                                        kitDocument?.toKitModel().let { kit ->
                                            kit?.let { kitModel ->
```

```
                                val kitId = kitDocument.id
                                val kitRef = kitsRef.document(kitId)
                                kits.add(kitModel)

                                val monitorListener =
                                ↪  kitRef.collection("dataMonitorings")
                                    .orderBy("timestamp",
                                    ↪  Query.Direction.DESCENDING)
                                    .addSnapshotListener {
                                    ↪  monitoringSnapshot, error ->
                                        monitoringSnapshot?.let {
                                        ↪  monitoringDocuments ->
                                            if(monitoringDocuments.docu
                                            ↪  ments.size >
                                            ↪  0){
                                                val lastMonitoring : Da
                                                ↪  taMonitoringModel?
                                                ↪  = monitoringDocumen
                                                ↪  ts.documents[0].toD
                                                ↪  ataMonitoringModel()
                                                updateKitModel(index,
                                                ↪  monitoringModel =
                                                ↪  lastMonitoring)
                                            }
                                        }

                                        error?.let {
                                            Log.e(TAG, "Listen
                                            ↪  dataMonitoring from kit
                                            ↪  $kitId failed", it)
                                        }
                                    }

                                val cropsListener =
                                ↪  kitRef.collection("crops")
                                    .orderBy("timestamp",
                                    ↪  Query.Direction.DESCENDING)
                                    .addSnapshotListener {
                                    ↪  cropsSnapshot, error ->
                                        cropsSnapshot?.let {
                                        ↪  cropsDocuments ->
                                            if(cropsDocuments.documents
                                            ↪  .size >
                                            ↪  0){
                                                val lastCropsModel :
                                                ↪  CropsModel? = crops
                                                ↪  Documents.documents
                                                ↪  [0].toCropsModel()
                                                updateKitModel(index,
                                                ↪  cropsModel =
                                                ↪  lastCropsModel)
                                            }
                                        }

                                        error?.let {
                                            Log.e(TAG, "Listen crops
                                            ↪  from kit $kitId
                                            ↪  failed", it)
                                        }
                                    }

                                monitoringsListeners.add(monitorListene
                                ↪  r)
                                cropsListeners.add(cropsListener)
                            }
                        }
                    }
                    currentKitModels.value = kits
                    updateKitModel(0)
                }
```

```
163
164                                 error?.let {
165                                     Log.e(TAG, "Listen kits from farm $farmId failed",
                                        ↪ it)
166                                 }
167                             }
168                         }
169
170                         error?.let {
171                             Log.e(TAG, "Listen farm $farmId failed", it)
172                         }
173                     }
174                 }
175             } catch (e:Exception){
176                 Log.e(TAG, "real time farms error", e)
177             }
178         }
179
180     }
181
182     private fun getPlantsUpdate(){
183         val db = firestore.collection("plants").orderBy("timestamp",
            ↪ Query.Direction.DESCENDING)
184         try {
185             plantsListener = db.addSnapshotListener { plantsSnapshot, error ->
186                 plantsSnapshot?.let {
187                     val plants : ArrayList<PlantModel> = arrayListOf()
188                     for(plantDocument in plantsSnapshot.documents){
189                         plantDocument?.let { plant ->
190                             plant.toPlantModel()?.let { plantModel ->
191                                 plants.add(plantModel)
192                             }
193                         }
194                     }
195                     allPlantModels.value = plants
196                     Log.i(TAG, "getPlantsUpdate: ${allPlantModels.value?.size}")
197                 }
198
199                 error?.let {
200                     Log.e(TAG, "Listen plants failed", it)
201                 }
202             }
203         } catch (e:Exception){
204             Log.e(TAG, "real time plant error", e)
205         }
206     }
207
208     private fun updateKitModel(index:Int, cropsModel: CropsModel?=null,
        ↪ monitoringModel: DataMonitoringModel? = null){
209         try {
210             currentKitModels.value?.let {
211                 val kitModels = it
212                 kitModels.forEachIndexed { kitIndex, kitModel ->
213                     if(kitIndex == index){
214                         cropsModel?.let {
215                             kitModel.lastCrops = it
216                         }
217                         monitoringModel?.let {
218                             kitModel.lastMonitoring = it
219                         }
220                     }
221                 }
222                 currentKitModels.value = kitModels
223             }
224
225             currentFarmModel.value?.let {
226                 val farmModel = it
227                 farmModel.kitModels = currentKitModels.value
228                 currentFarmModel.value = farmModel
229             }
```

```
230             }catch (e:Exception) {
231                 Log.e(TAG, "updateKitModel ${currentKitModels.value!![index]}\n", e)
232             }
233 
234             Log.i(TAG, "getFarmsUpdate ${currentFarmModel.value?.farmId}" +
235                     " \tkits:${currentKitModels.value?.size}\n" +
236                     "${currentFarmModel.value}\n")
237         }
238 
239         /** NOTES **/
240         var isNoteDeleted : MutableLiveData<Boolean?> = MutableLiveData(null)
241         var deleteNoteError : MutableLiveData<String> = MutableLiveData(null)
242 
243         fun getNote(position:Int) : NoteModel?{
244             return currentNoteModels.value?.get(position)
245         }
246 
247         fun deleteNote(position:Int){
248             isNoteDeleted.value = null
249             val noteId = getNote(position)?.noteId
250             try {
251                 noteId?.let{
252                     firestore.collection("users").document(auth.uid!!)
253                         .collection("notes").document(noteId)
254                         .delete().addOnCompleteListener {
255                             if (it.isSuccessful) {
256                                 isNoteDeleted.value = true
257                             } else {
258                                 isNoteDeleted.value = false
259                                 deleteNoteError.value = it.exception.toString()
260                                 Log.e(TAG, "deleteNote $noteId\n ${it.exception}")
261                             }
262                         }
263                 }
264             } catch (e:Exception){
265                 Log.e(TAG, "deleteNote $noteId failed", e)
266                 isNoteDeleted.value = false
267                 deleteNoteError.value = e.toString()
268             }
269         }
270 
271         fun refreshNotes(){
272             usersListener.remove()
273             currentNoteModels.value?.clear()
274             getUsersUpdate()
275         }
276 
277         /** POSTS **/
278         var isPostDeleted : MutableLiveData<Boolean?> = MutableLiveData(null)
279         var deletePostError : MutableLiveData<String> = MutableLiveData(null)
280 
281         fun getPostUser(position: Int) : UserModel? {
282             var output : UserModel? = null
283             val post = getPost(position)?.userId
284             post?.let { postId ->
285                 allUserModels.value?.forEach {
286                     if (it.uid == postId){
287                         output = it
288                     }
289                 }
290             }
291             return output
292         }
293 
294         fun getPost(position: Int) : PlantModel?{
295             return allPlantModels.value?.get(position)
296         }
297 
298         fun deletePost(position: Int){
299             isPostDeleted.value = null
```

```
        val plantModel = getPost(position)
        val plantId = plantModel?.plantId
        try {
            if(plantModel?.userId == auth.uid) {
                firestore.collection("plants").document(plantId!!)
                    .delete().addOnCompleteListener {
                        if (it.isSuccessful) {
                            isPostDeleted.value = true
                        } else {
                            isPostDeleted.value = false
                            deletePostError.value = it.exception.toString()
                            Log.e(TAG, "deletePost $plantId\n ${it.exception}")
                        }
                    }
            }
        } catch (e:Exception){
            Log.e(TAG, "deletePost $plantId failed", e)
        }
    }

    fun refreshPosts(){
        plantsListener.remove()
        allPlantModels.value?.clear()
        getPlantsUpdate()
    }

    /** FARM DASHBOARD **/
    fun getKit(position: Int) : KitModel? {
        return currentKitModels.value?.get(position)
    }

    fun refreshFarm(){
        farmsListener.remove()
        kitsListener.remove()
        monitoringsListeners.forEach { it.remove() }
        cropsListeners.forEach { it.remove() }

        currentFarmModel.value?.let {
            val farmModel = it
            farmModel.kitModels?.clear()
            currentFarmModel.value = farmModel
        }
        currentKitModels.value?.clear()
        getFarmsUpdate(currentUserModel.value?.farmId)
    }

    /** SIGN OUT **/
    var isUserSignOut : MutableLiveData<Boolean?> = MutableLiveData(null)
    var signOutError : MutableLiveData<String?> = MutableLiveData(null)

    fun signOut(){
        try {
            auth.signOut()
            isUserSignOut.value = true
        } catch (e:Exception){
            isUserSignOut.value = false
            signOutError.value = e.toString()
        }
    }
}
```

Kode 17: *View Model* untuk *Main Activity* beserta *Main Home*, *Main Posts*, *Main Add*, *Main Notes*, dan *Main Profile fragments*

7. ADD CROPS VIEW MODEL

```
class AddCropsViewModel : ViewModel() {
    var isPlantExist: MutableLiveData<Boolean> = MutableLiveData(false)
```

```
    private var plantModel : MutableLiveData<PlantModel> = MutableLiveData(PlantModel())

    fun setPlantModel(plantModel: PlantModel) {
        this.plantModel.value = plantModel
        isPlantExist.value = true
    }

    fun getPlantModel() : MutableLiveData<PlantModel> = plantModel
}
```

Kode 18: *View Model* untuk *Add Crops Activity*

8. ADD DATA MONITORING VIEW MODEL

```
class AddDataMonitoringViewModel : ViewModel() {
    private var auth : FirebaseAuth = Firebase.auth
    private var firestore : FirebaseFirestore = Firebase.firestore
    var currentUserModel : MutableLiveData<UserModel> = MutableLiveData(UserModel())
    var currentFarmModel : MutableLiveData<FarmModel> = MutableLiveData(FarmModel())
    private var currentKitModel : MutableLiveData<KitModel> = MutableLiveData(null)
    private var currentPlantModel : MutableLiveData<PlantModel> = MutableLiveData(null)
    private var currentCropsModel : MutableLiveData<CropsModel> = MutableLiveData(null)
    private var dataMonitoringModel : MutableLiveData<DataMonitoringModel> =
    ↪  MutableLiveData(DataMonitoringModel())
    private var kitSelectors : MutableLiveData<MutableList<String>> =
    ↪  MutableLiveData(mutableListOf<String>())
    private var availableKits : MutableLiveData<ArrayList<KitModel>> =
    ↪  MutableLiveData(arrayListOf())

    private var water : MutableLiveData<Float> = MutableLiveData(0f)
    private var nutrient : MutableLiveData<Float> = MutableLiveData(0f)
    private var turb : MutableLiveData<Float> = MutableLiveData(0f)
    private var temp : MutableLiveData<Float> = MutableLiveData(0f)
    private var humid : MutableLiveData<Float> = MutableLiveData(0f)
    private var acid : MutableLiveData<Float> = MutableLiveData(0f)
    private var isNotEmpties : MutableLiveData<Boolean> = MutableLiveData(false)
    var isDataMonitoringAdd: MutableLiveData<Boolean> = MutableLiveData(false)
    var addDataMonitoringError : MutableLiveData<String> = MutableLiveData("")
    var isCropsAdd: MutableLiveData<Boolean> = MutableLiveData(false)
    var addCropsError : MutableLiveData<String> = MutableLiveData("")
    private var isMenuAddCrops : MutableLiveData<Boolean> = MutableLiveData(false)

    companion object {
        const val TAG = "addDataMonitoringViewModel"
    }

    fun checkNotEmpty(boolean: Boolean) : MutableLiveData<Boolean> {
        isNotEmpties.value = boolean
        return isNotEmpties
    }

    fun setCurrentData(userModel: UserModel?, farmModel: FarmModel?,
    ↪  isAddCrops:Boolean){
        isMenuAddCrops.value = isAddCrops
        userModel?.let {
            this.currentUserModel.value = it
        }
        farmModel?.let {
            this.currentFarmModel.value = it
            val kitSelectors : MutableList<String> = mutableListOf()
            currentFarmModel.value?.kitModels?.let {
                val plantTrue : ArrayList<Int> = arrayListOf()
                it.forEachIndexed { index, kit ->
                    if((isAddCrops && kit.isPlanted != true)) {
                        kitSelectors.add(kit.name.toString())
                        availableKits.value?.add(kit)
                    }
                    if(!isAddCrops){
                        kitSelectors.add(kit.name.toString())
```

```
                    }
                }
                this.kitSelectors.value = kitSelectors
            }
            Log.i(TAG, "$farmModel")
        }
    }

    fun setCurrentKit(position:Int){
        currentKitModel.value = if(isMenuAddCrops.value == true){
            availableKits.value?.get(position)
        } else {
            currentFarmModel.value?.kitModels?.get(position)
        }
        val cropsModel = currentFarmModel.value?.kitModels?.get(position)?.lastCrops
        currentCropsModel.value = CropsModel()
        Log.i(TAG, "setCurrentKit: ${currentKitModel.value}")
    }

    fun getKitSelector() : MutableLiveData<MutableList<String>> = kitSelectors

    fun getCurrentKit() : MutableLiveData<KitModel> = currentKitModel

    fun getCurrentCrops() : MutableLiveData<CropsModel> = currentCropsModel

    fun setCurrentPlant(plantModel: PlantModel?) {
        currentPlantModel.value = plantModel
    }

    fun getCurrentPlant() : MutableLiveData<PlantModel> = currentPlantModel

    fun setNumberPickerValue(currentValue : Float, type: NumberPickerType?){
        when(type){
            NumberPickerType.WATER_TANK -> water.value = currentValue
            NumberPickerType.NUTRIENT_TANK -> nutrient.value = currentValue
            NumberPickerType.TURBIDITY -> turb.value = currentValue
            NumberPickerType.TEMPERATURE -> temp.value = currentValue
            NumberPickerType.HUMIDITY -> humid.value = currentValue
            NumberPickerType.ACIDITY -> acid.value = currentValue
            else -> { }
        }
    }

    fun getNumberPickerValue(type: NumberPickerType?) : MutableLiveData<Float>?{
        return when(type){
            NumberPickerType.WATER_TANK -> water
            NumberPickerType.NUTRIENT_TANK -> nutrient
            NumberPickerType.TURBIDITY -> turb
            NumberPickerType.TEMPERATURE -> temp
            NumberPickerType.HUMIDITY -> humid
            NumberPickerType.ACIDITY -> acid
            else -> null
        }
    }

    /** ADD DATA MONITORING **/
    fun addDataMonitoring(){
        val db =
        ↪  firestore.collection("farms").document(currentFarmModel.value?.farmId!!)
            .collection("kits").document(currentKitModel.value?.kitId!!)
            .collection("dataMonitorings")
        val ref : DocumentReference = db.document()

        try {
            dataMonitoringModel.value?.apply {
                this.dataId = ref.id
                this.userId = auth.uid
                this.timestamp = ViewUtility().getCurrentTimestamp()
                this.temperature = temp.value
                this.humidity = humid.value
```

```
                this.turbidity = turb.value
                this.waterTank = water.value
                this.nutrientTank = nutrient.value
                this.ph = acid.value
            }

            db.document(ref.id).set(dataMonitoringModel.value!!.toHashMap()).addOnCompl
            ↪  eteListener
            ↪  {
                if(it.isSuccessful){
                    isDataMonitoringAdd.value = true
                } else {
                    addDataMonitoringError.value = it.exception.toString()
                    isDataMonitoringAdd.value = false
                }
            }

        } catch (e:Exception){
            Log.e(TAG, "Error submit data monitoring", e)
        }
    }

    /** ADD CROPS **/
    fun addCrops(){
        val db =
        ↪  firestore.collection("farms").document(currentFarmModel.value?.farmId!!)
            .collection("kits").document(currentKitModel.value?.kitId!!)
            .collection("crops")
        val ref : DocumentReference = db.document()

        try{
            currentCropsModel.value.apply {
                currentCropsModel.value = CropsModel(
                    cropsId = ref.id,
                    userId = auth.uid,
                    plantId = currentPlantModel.value?.plantId,
                    tempLv = currentPlantModel.value?.tempLv,
                    humidLv = currentPlantModel.value?.humidLv,
                    phLv = currentPlantModel.value?.phLv
                )

                db.document(ref.id).set(currentCropsModel.value!!.toHashMap()).addOnCom
                ↪  pleteListener
                ↪  {
                    if(it.isSuccessful){
                        updateKit(ref.id)
                    } else {
                        addCropsError.value = it.exception.toString()
                        isCropsAdd.value = false
                    }
                }
            }

        } catch (e:Exception){
            Log.e(TAG, "Error add crops to ${currentKitModel.value?.kitId}", e)
        }
    }

    private fun updateKit(cropsId : String){
        val db =
        ↪  firestore.collection("farms").document(currentFarmModel.value?.farmId!!)
            .collection("kits").document(currentKitModel.value?.kitId!!)
        try {
            currentKitModel.value?.isPlanted = true
            db.set(currentKitModel.value!!.toHashMap()).addOnCompleteListener {
                if(it.isSuccessful){
                    isCropsAdd.value = true
                    if(isMenuAddCrops.value == false) {
                        dataMonitoringModel.value?.cropsId = cropsId
                        addDataMonitoring()
```

```
                    }
                } else {
                    addCropsError.value = it.exception.toString()
                    isCropsAdd.value = false
                }
            }
        } catch (e:Exception){
            Log.e(TAG, "Error update kit ${currentKitModel.value?.kitId}", e)
        }
    }
}
```

Kode 19: *View Model* untuk *Add Data Monitoring Activity*

9. ADD KIT VIEW MODEL

```
class AddKitViewModel : ViewModel() {
    private var firestore : FirebaseFirestore = Firebase.firestore
    var isNotEmpties : MutableLiveData<Boolean> = MutableLiveData(false)
    var isKitAdd : MutableLiveData<Boolean> = MutableLiveData(false)
    var addKitError : MutableLiveData<String> = MutableLiveData("")

    private var currentUserModel : MutableLiveData<UserModel> = MutableLiveData(null)
    private var currentFarmModel : MutableLiveData<FarmModel> = MutableLiveData(null)
    private var currentKitModel : MutableLiveData<KitModel> =
    ↪  MutableLiveData(KitModel())

    private var kitWidth : MutableLiveData<Float> = MutableLiveData(0f)
    private var kitLength : MutableLiveData<Float> = MutableLiveData(0f)
    private var waterMin : MutableLiveData<Float> = MutableLiveData(0f)
    private var waterMax : MutableLiveData<Float> = MutableLiveData(0f)
    private var nutrientMin : MutableLiveData<Float> = MutableLiveData(0f)
    private var nutrientMax : MutableLiveData<Float> = MutableLiveData(0f)
    private var turbidityMin : MutableLiveData<Float> = MutableLiveData(0f)
    private var turbidityMax : MutableLiveData<Float> = MutableLiveData(0f)


    companion object {
        const val TAG = "addKitViewModel"
    }

    fun checkNotEmpty(boolean: Boolean) : MutableLiveData<Boolean> {
        isNotEmpties.value = boolean
        return isNotEmpties
    }

    fun setCurrentData(userModel: UserModel?, farmModel: FarmModel?, kitModel:
    ↪  KitModel?){
        currentUserModel.value = userModel
        currentFarmModel.value = farmModel
        kitModel?.let{
            this.currentKitModel.value = it
        }
    }

    fun getCurrentFarm():MutableLiveData<FarmModel> = currentFarmModel
    fun getCurrentKit() : MutableLiveData<KitModel> = currentKitModel


    fun setNumberPickerValue(currentValue : Float, type: NumberPickerType?){
        when(type){
            NumberPickerType.KIT_WIDTH -> kitWidth.value = currentValue
            NumberPickerType.KIT_LENGTH -> kitLength.value = currentValue
            NumberPickerType.WATER_MIN -> waterMin.value = currentValue
            NumberPickerType.WATER_MAX -> waterMax.value = currentValue
            NumberPickerType.NUTRIENT_MIN -> nutrientMin.value = currentValue
            NumberPickerType.NUTRIENT_MAX -> nutrientMax.value = currentValue
            NumberPickerType.TURBIDITY_MIN -> turbidityMin.value = currentValue
            NumberPickerType.TURBIDITY_MAX -> turbidityMax.value = currentValue
```

```
52              else -> { }
53          }
54      }
55
56      fun getNumberPickerValue(type: NumberPickerType?) : MutableLiveData<Float>?{
57          return when(type){
58              NumberPickerType.KIT_WIDTH -> kitWidth
59              NumberPickerType.KIT_LENGTH -> kitLength
60              NumberPickerType.WATER_MIN -> waterMin
61              NumberPickerType.WATER_MAX -> waterMax
62              NumberPickerType.NUTRIENT_MIN -> nutrientMin
63              NumberPickerType.NUTRIENT_MAX ->nutrientMax
64              NumberPickerType.TURBIDITY_MIN -> turbidityMin
65              NumberPickerType.TURBIDITY_MAX -> turbidityMax
66              else -> null
67          }
68      }
69
70      fun createKit(name:String, position:String){
71          val db = firestore.collection("farms").document(currentUserModel.value?.farmId!
            ↪  !).collection("kits")
72          val ref : DocumentReference = db.document()
73
74          try {
75              currentKitModel.value?.apply {
76                  this.kitId = kitId ?: ref.id
77                  this.isPlanted = isPlanted ?: false
78                  this.name = name
79                  this.position = position
80                  this.length = kitLength.value?.toInt()
81                  this.width = kitWidth.value?.toInt()
82                  this.waterLv = ScoreLevel(waterMin.value, waterMax.value)
83                  this.nutrientLv = ScoreLevel(nutrientMin.value, nutrientMax.value)
84                  this.turbidityLv = ScoreLevel(turbidityMin.value, turbidityMax.value)
85                  this.timestamp = ViewUtility().getCurrentTimestamp()
86              }
87
88              val kitRef = db.document(currentKitModel.value!!.kitId!!)
89                  kitRef.set(currentKitModel.value!!.toHashMap()).addOnCompleteListener {
90                  if(it.isSuccessful){
91                      if(currentKitModel.value!!.kitId == ref.id){
92                          val dataRef = kitRef.collection("dataMonitorings")
93                          val dataId = dataRef.document().id
94                          val initialData = DataMonitoringModel(userId =
                            ↪  Firebase.auth.uid, dataId = dataId)
95                          dataRef.document(dataId).set(initialData.toHashMap()).addOnComp
                            ↪  leteListener
                            ↪  {
96                              if(it.isSuccessful){
97                                  isKitAdd.value = true
98                              }
99                          }
100                     } else {
101                         isKitAdd.value = true
102                     }
103                 } else {
104                     addKitError.value = it.exception.toString()
105                     isKitAdd.value = false
106                 }
107             }
108         } catch (e:Exception){
109             Log.e(TAG, "Error submit kit", e)
110         }
111     }
112 }
```

Kode 20: *View Model* untuk *Add Kit Activity*

10. ADD NOTE VIEW MODEL

```
class AddNoteViewModel : ViewModel(){
    private var auth : FirebaseAuth = Firebase.auth
    private var firestore : FirebaseFirestore = Firebase.firestore
    private var noteModel : MutableLiveData<NoteModel> = MutableLiveData(NoteModel())
    private var timeHour : MutableLiveData<Int> = MutableLiveData(0)
    private var timeMinute : MutableLiveData<Int> = MutableLiveData(0)
    var isNotEmpties : MutableLiveData<Boolean> = MutableLiveData(false)
    var isNoteAdd: MutableLiveData<Boolean> = MutableLiveData(false)
    var addNoteError : MutableLiveData<String> = MutableLiveData("")

    companion object {
        const val TAG = "addNote"
    }

    fun setCurrentData(noteModel: NoteModel?){
        noteModel?.let{
            this.noteModel.value = it
        }
    }

    fun getCurrentNoteModel() : MutableLiveData<NoteModel> = noteModel

    fun checkNotEmpty(boolean: Boolean) : MutableLiveData<Boolean> {
        isNotEmpties.value = boolean
        return isNotEmpties
    }

    fun getDate(): Long {
        if(noteModel.value?.date == null){
            noteModel.value?.date = ViewUtility().getCurrentDate()
        }
        return ViewUtility().formatStringToDate(noteModel.value?.date)
    }

    fun setDate(date: Long?): String? {
        noteModel.value?.date = ViewUtility().formatDateToString(date)
        return noteModel.value?.date
    }

    fun setTime(hour: Int, minute: Int) : String?{
        noteModel.value?.time = ViewUtility().formatTimeToString(hour, minute)
        return noteModel.value?.time
    }

    fun getTime(){
        if(noteModel.value?.time == null){
            noteModel.value?.time = ViewUtility().getCurrentTime()
        }
        val (hour, minute) = ViewUtility().formatStringToTime(noteModel.value?.time)
        timeHour.value = hour
        timeMinute.value = minute
    }

    fun getTimeHour() : Int {
        timeHour.value?.let {
            return it
        }
        return 0
    }

    fun getTimeMinute() : Int {
        timeMinute.value?.let {
            return it
        }
        return 0
    }

    fun createNote(title:String, description:String){
```

```
        val db = firestore.collection("users").document(auth.uid!!).collection("notes")
        val ref : DocumentReference = db.document()

        try {
            noteModel.value?.apply{
                this.noteId = noteId ?: ref.id
                this.title = title
                this.description = description
            }

            db.document(noteModel.value!!.noteId!!).set(noteModel.value!!.toHashMap()).
            ↪  addOnCompleteListener
            ↪  {
                if(it.isSuccessful){
                    isNoteAdd.value = true
                } else {
                    addNoteError.value = it.exception.toString()
                    isNoteAdd.value = false
                }
            }

        } catch (e:Exception){
            Log.e(TAG, "Error submit note", e)
        }
    }
}
```

Kode 21: *View Model* untuk *Add Note Activity*

11. ADD PLANT VIEW MODEL

```
class AddPlantViewModel : ViewModel() {
    private var auth : FirebaseAuth = Firebase.auth
    private var firestore : FirebaseFirestore = Firebase.firestore
    private var storage : FirebaseStorage = Firebase.storage
    private var growth : MutableLiveData<Float> = MutableLiveData(0f)
    private var tempMin : MutableLiveData<Float> = MutableLiveData(0f)
    private var tempMax : MutableLiveData<Float> = MutableLiveData(0f)
    private var humidMin : MutableLiveData<Float> = MutableLiveData(0f)
    private var humidMax : MutableLiveData<Float> = MutableLiveData(0f)
    private var acidMin : MutableLiveData<Float> = MutableLiveData(0f)
    private var acidMax : MutableLiveData<Float> = MutableLiveData(0f)
    private var imageUri : MutableLiveData<UriFileExtensions> = MutableLiveData(null)

    var plantModel : MutableLiveData<PlantModel> = MutableLiveData(PlantModel())

    var isNotEmpties : MutableLiveData<Boolean> = MutableLiveData(false)
    var isPlantAdd : MutableLiveData<Boolean> = MutableLiveData(false)
    var addPlantError : MutableLiveData<String> = MutableLiveData("")

    companion object {
        const val TAG = "addPlantViewModel"
    }

    fun checkNotEmpty(boolean: Boolean) : MutableLiveData<Boolean> {
        isNotEmpties.value = boolean
        return isNotEmpties
    }

    fun setCurrentData(plantModel: PlantModel?){
        plantModel?.let{
            this.plantModel.value = it
        }
    }

    fun setNumberPickerValue(currentValue : Float, type: NumberPickerType?){
        when(type){
            NumberPickerType.GROWTH_TIME -> growth.value = currentValue
            NumberPickerType.TEMP_MIN -> tempMin.value = currentValue
```

```
            NumberPickerType.TEMP_MAX -> tempMax.value = currentValue
            NumberPickerType.HUMID_MIN -> humidMin.value = currentValue
            NumberPickerType.HUMID_MAX -> humidMax.value = currentValue
            NumberPickerType.ACID_MIN -> acidMin.value = currentValue
            NumberPickerType.ACID_MAX -> acidMax.value = currentValue
            else -> { }
        }
    }

    fun getNumberPickerValue(type: NumberPickerType?) : MutableLiveData<Float>?{
        return when(type){
            NumberPickerType.GROWTH_TIME -> growth
            NumberPickerType.TEMP_MIN -> tempMin
            NumberPickerType.TEMP_MAX -> tempMax
            NumberPickerType.HUMID_MIN -> humidMin
            NumberPickerType.HUMID_MAX -> humidMax
            NumberPickerType.ACID_MIN -> acidMin
            NumberPickerType.ACID_MAX -> acidMax
            else -> null
        }
    }

    fun createPlant(name:String, description:String){
        try{
            plantModel.value?.apply {
                this.name = name
                this.description = description
                this.growthTime = growth.value?.toInt()
                this.tempLv = ScoreLevel(tempMin.value, tempMax.value)
                this.humidLv = ScoreLevel(humidMin.value, humidMax.value)
                this.phLv = ScoreLevel(acidMin.value, acidMax.value)
                this.userId = auth.uid!!
            }

            if(imageUri.value != null){
                val storageReference : StorageReference =
                ↪  storage.getReference("profile_images")
                    .child(System.currentTimeMillis().toString() +
                    ↪  ".${imageUri.value?.fileExtensions!!}")
                val uploadTask = storageReference.putFile(imageUri.value?.uri!!)
                uploadTask.continueWithTask {
                    if(!it.isSuccessful){
                        throw it.exception!!.cause!!
                    }
                    storageReference.downloadUrl
                }.addOnCompleteListener {
                    if (it.isSuccessful) {
                        it.result.let {
                            plantModel.value?.photo_url = it.toString()
                            sendPlantProfile()
                        }
                    }
                }
            } else {
                sendPlantProfile()
            }
        } catch (e:Exception){
            Log.e(TAG, "Error submit plant ", e)
        }
    }

    private fun sendPlantProfile(){
        val db = firestore.collection("plants")
        val ref : DocumentReference = db.document()

        plantModel.value?.apply {
            this.plantId = plantId ?: ref.id
        }

```

```
106         db.document(plantModel.value!!.plantId!!).set(plantModel.value!!.toHashMap()).a
            ↪ ddOnCompleteListener
            ↪ {
107             if(it.isSuccessful){
108                 isPlantAdd.value = true
109             } else {
110                 addPlantError.value = it.exception.toString()
111                 isPlantAdd.value = false
112             }
113         }
114     }
115
116     fun setPhotoPlant(uri : Uri?, fileExtension: String?=null){
117         imageUri.value = if(uri != null){
118             UriFileExtensions(uri, fileExtension!!)
119         } else {
120             null
121         }
122         Log.i(TAG, "setPhotoPlant: $uri")
123     }
124
125     fun getPhotoProfile() : MutableLiveData<UriFileExtensions>{
126         return imageUri
127     }
128
129     fun getCurrentPlant() : MutableLiveData<PlantModel>{
130         return plantModel
131     }
132 }
```

Kode 22: *View Model* untuk *Add Plant Activity*

12. PROFILE KIT VIEW MODEL

```
1   class ProfileKitViewModel : ViewModel() {
2       private var firestore : FirebaseFirestore = Firebase.firestore
3       private var currentUserModel : MutableLiveData<UserModel> = MutableLiveData(null)
4       private var currentKitModel : MutableLiveData<KitModel> = MutableLiveData(null)
5       private var currentFarmModel : MutableLiveData<FarmModel> = MutableLiveData(null)
6
7       private var monitoringArray : MutableLiveData<ArrayList<DataMonitoringModel>> =
        ↪ MutableLiveData(null)
8       private var cropsArray : MutableLiveData<ArrayList<CropsModel>> =
        ↪ MutableLiveData(null)
9       private lateinit var kitListener : ListenerRegistration
10      private lateinit var monitoringListener : ListenerRegistration
11      private lateinit var cropsListener : ListenerRegistration
12
13      companion object {
14          const val TAG = "profileKitViewModel"
15      }
16
17      fun setCurrentKit(userModel: UserModel?, farmModel: FarmModel?, kitModel:
        ↪ KitModel?) {
18          userModel?.let { user ->
19              currentUserModel.value = user
20              farmModel?.let {
21                  currentFarmModel.value = it
22              }
23              kitModel?.let { kit ->
24                  currentKitModel.value = kit
25                  getKitUpdate()
26              }
27          }
28          Log.i(TAG, "setCurrentKit: $userModel $kitModel")
29      }
30
31      private fun getKitUpdate() {
32          try{
```

```
            val db =
            ↪  firestore.collection("farms").document(currentUserModel.value?.farmId!!)
                .collection("kits").document(currentKitModel.value?.kitId!!)

            kitListener = db.addSnapshotListener { kitSnapshot, error ->
                kitSnapshot?.let {
                    currentKitModel.value = it.toKitModel()
                }

                error?.let {
                    Log.e(TAG, "Listen kits failed", it)
                }
            }

            monitoringListener = db.collection("dataMonitorings")
                .orderBy("timestamp", Query.Direction.DESCENDING)
                .addSnapshotListener { monitoringSnapshot, error ->
                    monitoringSnapshot?.let {
                        val monitorings : ArrayList<DataMonitoringModel> = arrayListOf()
                        for(each in it.documents){
                            each.toDataMonitoringModel()?.let {
                                monitorings.add(it)
                            }
                        }
                        monitoringArray.value = monitorings
                    }
                    error?.let {
                        Log.e(TAG, "Listen monitoring failed", it)
                    }
            }

            cropsListener = db.collection("crops")
                .orderBy("timestamp", Query.Direction.DESCENDING)
                .addSnapshotListener { cropsSnapshot, error ->
                    cropsSnapshot?.let {
                        val crops : ArrayList<CropsModel> = arrayListOf()
                        for(each in it.documents){
                            each.toCropsModel()?.let { cropsModel ->
                                cropsModel.plantId?.let{ plantId ->
                                    firestore.collection("plants").document(plantId)
                                        .get().addOnSuccessListener {
                                            cropsModel.plantModel = it.toPlantModel()
                                        }
                                }
                                crops.add(cropsModel)
                            }
                        }
                        cropsArray.value = crops
                    }

                    error?.let {
                        Log.e(TAG, "Listen crops failed", it)
                    }
            }
        } catch (e:Exception){
            Log.e(TAG, "getKitUpdate: ",e )
        }
    }

    fun getCurrentKit() = currentKitModel
    fun getCurrentMonitoring() = monitoringArray
    fun getCurrentCrops() = cropsArray
    fun getCurrentUser() = currentUserModel
    fun getCurrentFarm() = currentFarmModel
}
```

Kode 23: *View Model* untuk *Profile Kit Activity* beserta *Kit Overview*, *Kit Monitoring*, dan *Kit Crops fragments*

13. PROFILE USER VIEW MODEL

```
class ProfileUserViewModel : ViewModel(){
    private var firestore : FirebaseFirestore = Firebase.firestore
    private var userModel : MutableLiveData<UserModel> = MutableLiveData(null)
    private var plantModels : MutableLiveData<ArrayList<PlantModel>> =
    ↪  MutableLiveData(null)
    private var allUsers : MutableLiveData<ArrayList<UserModel>> = MutableLiveData(null)

    companion object {
        const val TAG = "profileUserViewModel"
    }

    fun setUserModel(userModel: UserModel?){
        this.userModel.value = userModel
        try {
            this.userModel.value?.let { user ->
                firestore.collection("plants")
                    .whereEqualTo("userId", user.uid)
                    .addSnapshotListener { plantSnapshot, error ->
                        plantSnapshot?.let {
                            val plants : ArrayList<PlantModel> = arrayListOf()
                            for(plantDoc in it.documents){
                                plantDoc.toPlantModel()?.let {
                                    plants.add(it)
                                }
                            }
                            plantModels.value = plants
                        }
                    }
            }
        } catch (e:Exception){
            Log.e(TAG, "setUserModel: ", )
        }
    }

    fun getUserModel() : MutableLiveData<UserModel> = userModel
    fun getUserPosts() : MutableLiveData<ArrayList<PlantModel>> = plantModels
}
```

Kode 24: *View Model* untuk *Profile User Activity*

14. SEARCH VIEW MODEL

```
class SearchViewModel : ViewModel(){
    private var firestore : FirebaseFirestore = Firebase.firestore
    private var userModels : MutableLiveData<ArrayList<UserModel>> = MutableLiveData()
    private var plantModels : MutableLiveData<ArrayList<PlantModel>> = MutableLiveData()
    private var searchUsers : MutableLiveData<ArrayList<UserModel>> = MutableLiveData()
    private var searchPlants : MutableLiveData<ArrayList<PlantModel>> =
    ↪  MutableLiveData()
    private var exceptUsers : MutableLiveData<ArrayList<UserModel>> =
    ↪  MutableLiveData(null)

    var isSearchSuccess : MutableLiveData<Boolean> = MutableLiveData(false)
    var searchError : MutableLiveData<String> = MutableLiveData("")

    companion object {
        const val TAG = "searchViewModel"
    }
    init {
        getAllUsers(null)
    }

    fun setExceptUsers(userModels : ArrayList<UserModel>?){
        exceptUsers.value = userModels
        Log.i(TAG, "setExceptUsers: ${exceptUsers.value}")
    }
```

```
23
24      fun getAllUsers() : MutableLiveData<ArrayList<UserModel>> = userModels
25
26      fun getAllUsers(lastKey : String? = null){
27          val db = firestore.collection("users")
28          try {
29              db.addSnapshotListener { snapshot, error ->
30                  snapshot?.let {
31                      val users : ArrayList<UserModel> = arrayListOf()
32                      for(plant in it.documents){
33                          plant.toUserModel()?.let { it1 -> users.add(it1) }
34                      }
35                      userModels.value = users
36                      searchUsers(lastKey)
37                  }
38
39                  error?.let {
40                      Log.e(TAG, "Listen data users failed", it)
41                      return@addSnapshotListener
42                  }
43              }
44          } catch (e:Exception){
45              Log.e(TAG, "Error get plants", e)
46          }
47      }
48      fun getAllPlants(lastKey : String? = null){
49          val db = firestore.collection("plants")
50          try {
51              db.addSnapshotListener { snapshot, error ->
52                  snapshot?.let {
53                      val plants : ArrayList<PlantModel> = arrayListOf()
54                      for(plant in it.documents){
55                          plant.toPlantModel()?.let { it1 -> plants.add(it1) }
56                      }
57                      plantModels.value = plants
58                      searchPlants(lastKey)
59                  }
60
61                  error?.let {
62                      Log.e(TAG, "Listen data plants failed", it)
63                      return@addSnapshotListener
64                  }
65              }
66          } catch (e:Exception){
67              Log.e(TAG, "Error get plants", e)
68          }
69      }
70
71      fun getUsers() :  MutableLiveData<ArrayList<UserModel>> = searchUsers
72      fun getPlants() : MutableLiveData<ArrayList<PlantModel>> = searchPlants
73
74      fun getUser(position: Int) : UserModel? {
75          return searchUsers.value?.get(position)
76      }
77      fun getPlant(position:Int) : PlantModel?{
78          return searchPlants.value?.get(position)
79      }
80
81      fun searchUsers(key:String?) {
82          if(key != null){
83              val results : ArrayList<UserModel> = arrayListOf()
84              userModels.value?.let {
85                  for (user in it) {
86                      if (user.name?.contains(key) == true || user.email?.contains(key)
                        ↪  == true) {
87                          exceptUsers.value?.let {
88                              for (exUser in it) {
89                                  if (exUser.uid != user.uid){ //|| user.role !=
                                    ↪  Role.OWNER.toString()) {
90                                      results.add(user)
```

```
                                }
                            }
                        } ?: kotlin.run {
                            results.add(user)
                        }
                    }
                }
                searchUsers.value = results
            }
        }
        else {
            userModels.value?.let { allUsers ->
                exceptUsers.value?.let {
                    val results : ArrayList<UserModel> = arrayListOf()
                    for(user in allUsers) {
                        for (exUser in it) {
                            if (exUser.uid != user.uid) {
                                results.add(user)
                            }
                        }
                    }
                    searchUsers.value = results
                } ?: kotlin.run {
                    searchUsers.value = userModels.value
                }
            }
        }
    }
    fun searchPlants(key:String?) {
        if(key != null && plantModels.value != null){
            val results : ArrayList<PlantModel> = arrayListOf()
            plantModels.value?.let {
                for (user in it) {
                    if (user.name?.contains(key) == true) {
                        results.add(user)
                    }
                }
                searchPlants.value = results
            }
        }
        else {
            searchPlants.value = plantModels.value
        }
    }

}
```

Kode 25: *View Model* untuk *Search Activity*

15. FEEDBACK VIEW MODEL

```
class FeedbackViewModel : ViewModel(){
    private var auth : FirebaseAuth = Firebase.auth
    private val db = Firebase.firestore.collection("feedbacks")
    private var ref : DocumentReference = db.document()
    private var feedbackModel : MutableLiveData<FeedbackModel> =
    ↪  MutableLiveData(FeedbackModel())
    private var isNotEmpties : MutableLiveData<Boolean> = MutableLiveData(false)
    var isFeedbackSubmit : MutableLiveData<Boolean> = MutableLiveData(false)
    var feedbackSubmitError : MutableLiveData<String> = MutableLiveData("")

    companion object {
        const val TAG = "feedback"
    }

    fun checkNotEmpty(boolean: Boolean) : MutableLiveData<Boolean> {
        isNotEmpties.value = boolean
        return isNotEmpties
    }
```

```
18
19      fun setRating(rating : Float) : MutableLiveData<FeedbackModel>? {
20          feedbackModel.value?.rating = rating
21          return feedbackModel
22      }
23
24      fun getRating() = feedbackModel.value?.rating
25
26      fun submitFeedback(content:String){
27          try{
28              feedbackModel.value?.let {
29                  it.feedbackId = ref.id
30                  it.content = content
31                  it.userId = auth.uid
32              }
33
34              db.document(ref.id).set(feedbackModel.value!!.toHashMap()).addOnCompleteLis
                ↪  tener
                ↪  {
35                  if(it.isSuccessful){
36                      isFeedbackSubmit.value = true
37                  } else {
38                      feedbackSubmitError.value = it.exception.toString()
39                      isFeedbackSubmit.value = false
40                  }
41              }
42          } catch (e:Exception){
43              Log.e(TAG, "Error submit feedback", e)
44          }
45      }
46  }
```

Kode 26: *View Model* untuk *Feedback Activity*

16. SHOW RECYCLER VIEW MODEL

```
1   class ShowRecyclerViewModel : ViewModel() {
2       private var auth : FirebaseAuth = Firebase.auth
3       private var firestore : FirebaseFirestore = Firebase.firestore
4       private var plantModels : MutableLiveData<ArrayList<PlantModel>> =
        ↪  MutableLiveData(null)
5       private var userModel : MutableLiveData<UserModel> = MutableLiveData(null)
6
7       fun setCurrentPosts(plantModel: ArrayList<PlantModel>?, userModel : UserModel?) {
8           this.userModel.value = userModel
9           plantModel?.let {
10              val plants : ArrayList<PlantModel> = arrayListOf()
11              it.forEach {
12                  if(it.userId == auth.uid){
13                      plants.add(it)
14                  }
15              }
16              plantModels.value = plants
17              getPlantsUpdate()
18          } ?: kotlin.run {
19              plantModels.value = arrayListOf()
20              getPlantsUpdate()
21          }
22      }
23
24      fun getCurrentPosts() : MutableLiveData<ArrayList<PlantModel>> = plantModels
25      fun getCurrentUser() : MutableLiveData<UserModel> = userModel
26
27      var isPostDeleted : MutableLiveData<Boolean?> = MutableLiveData(null)
28      var deletePostError : MutableLiveData<String> = MutableLiveData(null)
29
30      fun getPost(position: Int) : PlantModel?{
31          return plantModels.value?.get(position)
32      }
```

```
33
34      fun deletePost(position: Int){
35          isPostDeleted.value = null
36          val plantModel = getPost(position)
37          val plantId = plantModel?.plantId
38          try {
39              if(plantModel?.userId == auth.uid) {
40                  firestore.collection("plants").document(plantId!!)
41                      .delete().addOnCompleteListener {
42                          if (it.isSuccessful) {
43                              isPostDeleted.value = true
44                          } else {
45                              isPostDeleted.value = false
46                              deletePostError.value = it.exception.toString()
47                              Log.e(MainViewModel.TAG, "deletePost $plantId\n
                                ↪  ${it.exception}")
48                          }
49                      }
50              }
51          } catch (e:Exception){
52              Log.e(MainViewModel.TAG, "deletePost $plantId failed", e)
53          }
54      }
55
56      private fun getPlantsUpdate(){
57          val db = firestore.collection("plants")
58              .whereEqualTo("userId", auth.uid)
59              .orderBy("timestamp", Query.Direction.DESCENDING)
60          try {
61              db.addSnapshotListener { plantsSnapshot, error ->
62                  plantsSnapshot?.let {
63                      val plants : ArrayList<PlantModel> = arrayListOf()
64                      for(plantDocument in plantsSnapshot.documents){
65                          plantDocument?.let { plant ->
66                              plant.toPlantModel()?.let { plantModel ->
67                                  plants.add(plantModel)
68                              }
69                          }
70                      }
71                      plantModels.value = plants
72                      Log.i(MainViewModel.TAG, "getPlantsUpdate:
                        ↪  ${plantModels.value?.size}")
73                  }
74
75                  error?.let {
76                      Log.e(MainViewModel.TAG, "Listen plants failed", it)
77                  }
78              }
79          } catch (e:Exception){
80              Log.e(MainViewModel.TAG, "real time plant error", e)
81          }
82      }
83  }
```

Kode 27: *View Model* untuk *Show Recycler Activity*

E. VIEW LAYER (ACTIVITY AND FRAGMENT)

1. SPLASH SCREEN ACTIVITY

```
1   class SplashScreenActivity : AppCompatActivity() {
2       private lateinit var binding : ActivitySplashScreenBinding
3       private lateinit var viewModel : SplashScreenViewModel
4       private var auth : FirebaseAuth = Firebase.auth
5
6       companion object{
7           const val TAG = "splashScreenActivity"
8       }
```

```
9
10      override fun onCreate(savedInstanceState: Bundle?) {
11          super.onCreate(savedInstanceState)
12          binding = ActivitySplashScreenBinding.inflate(layoutInflater)
13          setContentView(binding.root)
14          viewModel = ViewModelProvider(this).get(SplashScreenViewModel::class.java)
15          RequestPermission().requestAllPermissions(this)
16      }
17
18      fun splashIsDone(){
19          if(auth.currentUser != null){
20              viewModel.getCurrentData()
21              viewModel.isCurrentUserExist.observe(this,{
22                  when(it){
23                      true -> {
24                          Log.i(TAG, "user ${viewModel.currentUserModel.value}")
25                          viewModel.isCurrentFarmExist.observe(this, {
26                              when(it){
27                                  true -> {
28                                      Log.i(TAG, "farm
                                        ↪  ${viewModel.currentFarmModel.value}")
29                                      val intent = Intent(this, MainActivity::class.java)
30                                      intent.putExtra(BuildConfig.CURRENT_USER,
                                        ↪  viewModel.currentUserModel.value)
31                                      intent.putExtra(BuildConfig.CURRENT_FARM,
                                        ↪  viewModel.currentFarmModel.value)
32                                      startActivity(intent)
33                                      finish()
34                                  }
35                                  false -> {
36                                      Log.i(TAG, "currentFarm not exist")
37                                      if(Role.getType(viewModel.currentUserModel.value?.r
                                        ↪  ole!!) ==
                                        ↪  Role.OWNER){
38                                          val intent = Intent(this,
                                            ↪  CreateProfileActivity::class.java)
39                                          intent.putExtra(BuildConfig.SELECTED_USER,
                                            ↪  viewModel.currentUserModel.value)
40                                          startActivity(intent)
41                                          finish()
42                                      } else {
43                                          Toast.makeText(this, "wait for owner add you",
                                            ↪  Toast.LENGTH_SHORT).show()
44                                          Log.i(TAG, "wait for owner add you")
45                                      }
46                                  }
47                                  null -> {
48                                      Log.i(TAG, "currentFarm loading")
49                                  }
50                              }
51                          })
52                      }
53                      false -> {
54                          Log.i(TAG, "currentUser not exist")
55                          startActivity(Intent(this, CreateProfileActivity::class.java))
56                          finish()
57                      }
58                      null -> {
59                          Log.i(TAG, "currentUser loading")
60                      }
61                  }
62              })
63
64          } else {
65              val handler = Handler(Looper.getMainLooper())
66              handler.postDelayed({
67                  startActivity(Intent(this, SignInActivity::class.java))
68              }, 3000)
69          }
70      }
```

```
}
```

Kode 28: *Source Code* untuk *Splash Screen Activity*

2. SIGN IN ACTIVITY

```
class SignInActivity : AppCompatActivity(), View.OnClickListener, TextWatcher,
↪  CompoundButton.OnCheckedChangeListener {
    private lateinit var binding: ActivitySignInBinding
    private lateinit var viewModel : SignInViewModel
    private lateinit var utility : ViewUtility
    private lateinit var editTexts : ArrayList<TextInputEditText>
    private lateinit var viewsAsButton : ArrayList<View>

    companion object{
        const val TAG = "signIn"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivitySignInBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider(this).get(SignInViewModel::class.java)
        binding.apply {
            editTexts = arrayListOf(signinEmail, signinPassword)
            viewsAsButton = arrayListOf(signinSubmit,
                signinSignUp,
                signinForgetPassword)
            utility = ViewUtility(
                context = this@SignInActivity,
                circularProgressButton = signinSubmit,
                textInputEditTexts = editTexts,
                viewsAsButton = viewsAsButton,
                checkBoxes = arrayListOf(signinRemember),
                actionBar = supportActionBar)

            signinRemember.setOnCheckedChangeListener(this@SignInActivity)
            viewsAsButton.forEach { it.setOnClickListener(this@SignInActivity) }
            editTexts.forEach { it.addTextChangedListener(this@SignInActivity) }
            checkEmpty()
        }
    }

    override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
        if (currentFocus != null) {
            val imm: InputMethodManager =
                getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
            imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
        }
        return super.dispatchTouchEvent(ev)
    }

    override fun onClick(v: View?) {
        when(v){
            binding.signinSubmit -> {
                utility.isLoading = true
                viewModel.signIn(binding.signinEmail.text.toString(),
                ↪  binding.signinPassword.text.toString())
                viewModel.isUserSignIn.observe(this@SignInActivity, {
                    if(it){
                        DatabaseHandler(this).signIn(viewModel.getDataModel())
                        utility.isLoading = false
                        startActivity(Intent(this@SignInActivity,
                        ↪  MainActivity::class.java))
                        finish()
                    } else {
                        utility.isLoading = false
                        viewModel.signInError.observe(this, {
```

```
                            it?.let {
                                if (it.isNotEmpty()) {
                                    Toast.makeText(this@SignInActivity, it,
                                    ↪  Toast.LENGTH_SHORT)
                                        .show()
                                    Log.i(TAG, it)
                                    viewModel.signInError.value = null
                                }
                            }
                        })
                    }
                })
            }
            binding.signinSignUp -> startActivity(Intent(this,
            ↪  SignUpActivity::class.java))
            binding.signinForgetPassword -> startActivity(Intent(this,
            ↪  ForgetPasswordActivity::class.java))
        }
    }

    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after:
    ↪  Int) {}

    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
        checkEmpty()
    }

    override fun afterTextChanged(s: Editable?) {}

    private fun checkEmpty() {
        viewModel.checkNotEmpty(utility.isEmpties(editTexts)).observe(this, {
            binding.signinSubmit.isEnabled = it
        })
    }

    override fun onCheckedChanged(buttonView: CompoundButton?, isChecked: Boolean) {
        viewModel.setRememberChecked(isChecked)
    }
}
```

Kode 29: *Source Code* untuk *Sign In Activity*

3. SIGN UP ACTIVITY

```
class SignUpActivity : AppCompatActivity(), View.OnClickListener, TextWatcher {
    private lateinit var binding : ActivitySignUpBinding
    private lateinit var viewModel : SignUpViewModel
    private lateinit var utility : ViewUtility
    private lateinit var editTexts : ArrayList<TextInputEditText>
    private lateinit var viewsAsButton : ArrayList<View>

    companion object{
        const val TAG = "signUp"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivitySignUpBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider(this).get(SignUpViewModel::class.java)
        supportActionBar?.title = null
        supportActionBar?.elevation= 0f
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowHomeEnabled(false)

        binding.apply {
            viewsAsButton = arrayListOf(signupSignIn, signupSubmit)
            editTexts = arrayListOf(signupEmail, signupPassword, signupConfirmPassword)
```

```
            utility = ViewUtility(
                context = this@SignUpActivity,
                circularProgressButton = signupSubmit,
                textInputEditTexts = editTexts,
                viewsAsButton = viewsAsButton,
                actionBar = supportActionBar
            )

            viewsAsButton.forEach { it.setOnClickListener(this@SignUpActivity) }
            editTexts.forEach { it.addTextChangedListener(this@SignUpActivity) }
            checkEmpty()
        }
    }

    override fun onSupportNavigateUp(): Boolean {
        super.onBackPressed()
        return true
    }

    override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
        if (currentFocus != null) {
            val imm: InputMethodManager =
                getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
            imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
        }
        return super.dispatchTouchEvent(ev)
    }

    override fun onClick(v: View?) {
        when(v){
            binding.signupSubmit -> {
                    utility.isLoading = true
                    viewModel.signUp(binding.signupEmail.text.toString(),
                    ↪  binding.signupPassword.text.toString())
                    viewModel.isUserSignUp.observe(this@SignUpActivity, {
                        if(it){
                            utility.isLoading = false
                            startActivity(Intent(this,
                            ↪  CreateProfileActivity::class.java))
                            finish()
                        } else {

                            viewModel.signUpError.observe(this, {
                                it?.let {
                                    if (it.isNotEmpty()) {
                                        Toast.makeText(this@SignUpActivity, it,
                                        ↪  Toast.LENGTH_SHORT)
                                            .show()
                                        Log.i(TAG, it)
                                        viewModel.signUpError.value = ""
                                        utility.isLoading = false
                                    }
                                }
                            })
                        }
                    })
            }
            binding.signupSignIn -> super.onBackPressed()
        }
    }

    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after:
    ↪  Int) {}

    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
        checkEmpty()
    }

    override fun afterTextChanged(s: Editable?) {}

```

```
92      private fun checkEmpty() {
93          val passwordMatch = binding.signupPassword.text.toString() ==
            ↪  binding.signupConfirmPassword.text.toString()
94          binding.signupConfirmPasswordLayout.error = if(passwordMatch){
95              null
96          } else {
97              "Confirm password should be same"
98          }
99
100         viewModel.apply {
101             checkNotEmpty(utility.isEmpties(editTexts) &&
                ↪  passwordMatch).observe(this@SignUpActivity, {
102                 binding.signupSubmit.isEnabled = it
103             })
104         }
105     }
106 }
```

Kode 30: *Source Code* untuk *Sign Up Activity*

4. FORGET PASSWORD ACTIVITY

```
1   class ForgetPasswordActivity : AppCompatActivity(), View.OnClickListener, TextWatcher {
2       private lateinit var binding : ActivityForgetPasswordBinding
3       private lateinit var viewModel : ForgetPasswordViewModel
4       private lateinit var utility : ViewUtility
5       private lateinit var viewsAsButton : ArrayList<View>
6
7       companion object{
8           const val TAG = "forgetPasswordActivity"
9       }
10
11      override fun onCreate(savedInstanceState: Bundle?) {
12          super.onCreate(savedInstanceState)
13          binding = ActivityForgetPasswordBinding.inflate(layoutInflater)
14          setContentView(binding.root)
15
16          viewModel = ViewModelProvider(this).get(ForgetPasswordViewModel::class.java)
17          supportActionBar?.title = null
18          supportActionBar?.elevation= 0f
19          supportActionBar?.setDisplayHomeAsUpEnabled(true)
20          supportActionBar?.setDisplayShowHomeEnabled(false)
21
22          binding.apply {
23              viewsAsButton = arrayListOf(forgetSubmit, forgetResend)
24              utility = ViewUtility(
25                  context = this@ForgetPasswordActivity,
26                  circularProgressButton = forgetSubmit,
27                  textInputEditTexts = arrayListOf(forgetEmail),
28                  viewsAsButton = viewsAsButton,
29                  actionBar = supportActionBar
30              )
31              utility.onLoadingChangeListener{
32                  if(it){
33                      binding.forgetSuccess.visibility = View.GONE
34                  }
35              }
36
37              viewsAsButton.forEach { it.setOnClickListener(this@ForgetPasswordActivity) }
38              forgetEmail.addTextChangedListener(this@ForgetPasswordActivity)
39              checkEmpty()
40          }
41      }
42
43      override fun onSupportNavigateUp(): Boolean {
44          super.onBackPressed()
45          return true
46      }
47
```

```
48      override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
49          if (currentFocus != null) {
50              val imm: InputMethodManager =
51                  getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
52              imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
53          }
54          return super.dispatchTouchEvent(ev)
55      }
56
57      override fun onClick(v: View?) {
58          when(v){
59              binding.forgetSubmit, binding.forgetResend -> {
60                  utility.isLoading = true
61                  viewModel.sendEmailReset(binding.forgetEmail.text.toString())
62                  viewModel.isUserForgetPassword.observe(this, {
63                      if(it){
64                          utility.isLoading = false
65                          handleDynamicLink()
66                      } else {
67                          utility.isLoading = false
68                          viewModel.forgetPasswordError.observe(this, {
69                              if (it.isNotEmpty()) {
70                                  Toast.makeText(this@ForgetPasswordActivity, it,
                                    ↪  Toast.LENGTH_SHORT)
71                                      .show()
72                                  viewModel.forgetPasswordError.value = ""
73                              }
74                          })
75                      }
76                  })
77              }
78          }
79      }
80
81      override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after:
        ↪  Int) {}
82
83      override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
84          checkEmpty()
85      }
86
87      override fun afterTextChanged(s: Editable?) {}
88
89      private fun checkEmpty() {
90          viewModel.checkNotEmpty(utility.isEmpty(binding.forgetEmail)).observe(this, {
91              binding.forgetSubmit.isEnabled = it
92          })
93      }
94
95      fun handleDynamicLink() {
96  //        FirebaseDynamicLinks.getInstance().getDynamicLink(intent)
97  //            .addOnSuccessListener(this) { pendingDynamicLinkData:
    ↪  PendingDynamicLinkData ->
98  //                Log.i("forgetPasswordActivity", "Dynamic link detected")
99  //                val oobCode =
100 //                    pendingDynamicLinkData.link!!.getQueryParameter("oobCode")
101 //                if (oobCode != null) {
102 //                    auth.checkActionCode(oobCode).addOnSuccessListener { result ->
103 //                        when (result.getOperation()) {
104 //                            ActionCodeResult.VERIFY_EMAIL -> {
105 //                                auth.applyActionCode(oobCode)
106 //                                    .addOnSuccessListener { resultCode ->
107 //                                        Log.i("forgetPasswordActivity", "Verified
    ↪  email")
108 //                                        finish()
109 //                                    }.addOnFailureListener { resultCode ->
110 //                                        Log.w(
111 //                                            "forgetPasswordActivity",
112 //                                            "Failed to Verified Email",
113 //                                            resultCode
```

```
114 //                                      )
115 //                                  }
116 //                              }
117 //                              ActionCodeResult.PASSWORD_RESET -> {
118 //                                  val passWordResetInetemnt =
119 //                                      Intent(this, ResetPasswordActivity::class.java)
120 //                                  passWordResetInetemnt.putExtra("oobCode", oobCode)
121 //                                  startActivity(passWordResetInetemnt)
122 //                                  finish()
123 //                              }
124 //                          }
125 //                      }.addOnFailureListener { result ->
126 //                          Log.w("forgetPasswordActivity", "Invalid code sent")
127 //                          finish()
128 //                      }
129 //                  }
130 //              }.addOnFailureListener { result: Exception? ->
131 //                  Log.w("forgetPasswordActivity", "Failed to get dynamic link")
132 //                  finish()
133 //              }
134     }
135 }
```

Kode 31: *Source Code* untuk *Forget Password Activity*

5. CREATE PROFILE ACTIVITY

```
1  class CreateProfileActivity : AppCompatActivity() {
2      private lateinit var binding : ActivityCreateProfileBinding
3      lateinit var viewModel : CreateProfileViewModel
4
5      companion object{
6          const val TAG = "createProfile"
7      }
8
9      override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         binding = ActivityCreateProfileBinding.inflate(layoutInflater)
12         setContentView(binding.root)
13
14         viewModel = ViewModelProvider(this).get(CreateProfileViewModel::class.java)
15         viewModel.setCurrentUser(intent.getParcelableExtra<UserModel>(BuildConfig.CURRE
           ↪  NT_USER))
16     }
17
18     override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
19         if (currentFocus != null) {
20             val imm: InputMethodManager =
21                 getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
22             imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
23         }
24         return super.dispatchTouchEvent(ev)
25     }
26
27     val startForResult =
       ↪  registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
       ↪  data ->
28         Log.i(TAG, "$data")
29         try{
30             if(data?.resultCode == Activity.RESULT_OK){
31                 CropImage.getActivityResult(data.data)?.let{
32                     val mimeTypeMap = MimeTypeMap.getSingleton()
33                     val fileExtension = mimeTypeMap.getExtensionFromMimeType(contentRes
                       ↪  olver.getType(it.uriContent!!))
34                     viewModel.setPhotoProfile(it.uriContent!!, fileExtension)
35                 }
36
```

```
                data.data?.getParcelableExtra<UserModel>(BuildConfig.SELECTED_USER)?.le
                ↪ t
                ↪ {
                    viewModel.addStaff(it)
                }
            }

        } catch (e: Exception){
            Toast.makeText(this, e.message.toString(), Toast.LENGTH_SHORT).show()
        }
    }

    fun changeImageProfile(){
        val intent = CropImage.activity()
            .setActivityTitle("Edit Photo")
            .setAspectRatio(1,1)
            .setGuidelines(CropImageView.Guidelines.ON)
            .setAutoZoomEnabled(true)
            .setAllowFlipping(false)
            .getIntent(this)
        startForResult.launch(intent)
    }

    fun searchUser(){
        val intent = Intent(this, SearchActivity::class.java)
        intent.putExtra("search", TAG)
        startForResult.launch(intent)
    }
}
```

Kode 32: *Source Code* untuk *Create Profile Activity* beserta *fragments* terkait

a) CREATE USER FRAGMENT

```
class CreateUserFragment : Fragment(), View.OnClickListener,
↪ SegmentedButtonGroup.OnPositionChangedListener, TextWatcher {
    private var _binding : FragmentCreateUserBinding? = null
    private val binding get() = _binding!!
    private lateinit var viewModel : CreateProfileViewModel
    private lateinit var utility: ViewUtility
    private lateinit var editTexts : ArrayList<TextInputEditText>
    private lateinit var viewsAsButton : ArrayList<View>

    companion object {
        const val TAG = "createUserFragment"
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentCreateUserBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        requireActivity().title = getString(R.string.create_profile)
        viewModel = (requireActivity() as CreateProfileActivity).viewModel

        binding.apply {
            viewsAsButton = arrayListOf(roleOwner,
                                        createUserDOB,
                                        createUserSubmit,
                                        createUserEditPhoto,
                                        createUserPhoto)
            editTexts = arrayListOf(
                    createUserName,
                    createUserPhone,
```

```
                    createUserAddress,
                    createUserBio)
            utility = ViewUtility(
                context = requireContext(),
                circularProgressButton = createUserSubmit,
                textInputEditTexts = editTexts,
                viewsAsButton = viewsAsButton,
                actionBar = (requireActivity() as
                ↪  CreateProfileActivity).supportActionBar)
            utility.onLoadingChangeListener {
                binding.roleOwner.isClickable = it
                binding.roleStaff.isClickable = it
            }

            editTexts.forEach { it.addTextChangedListener(this@CreateUserFragment) }
            viewsAsButton.forEach { it.setOnClickListener(this@CreateUserFragment) }
            createUserGender.onPositionChangedListener = this@CreateUserFragment
            createUserPhoto.setOnLongClickListener { createUserEditPhoto.performClick()
            ↪  }
            checkEmpty()

            createUserEmail.setText(Firebase.auth.currentUser?.email)
            roleOwner.performClick()
        }

        viewModel.getPhotoProfile().observe(requireActivity(), {
            it?.let {
                Glide.with(this)
                    .load(it.uri)
                    .circleCrop()
                    .into(binding.createUserPhoto)
            }
        })

        viewModel.isUserCreated.observe(viewLifecycleOwner, {
            if(it){
                utility.isLoading = false
                Toast.makeText(requireContext(), "User created",
                ↪  Toast.LENGTH_SHORT).show()
                if (Role.getType(viewModel.getCurrentUserModel()?.role!!) ==
                ↪  Role.OWNER) {
                    Navigation.findNavController(binding.root).navigate(R.id.action_cre
                    ↪  ateUserFragment_to_createFarmFragment)
                } else {
                    val intent = Intent(requireContext(), MainActivity::class.java)
                    intent.putExtra(BuildConfig.CURRENT_USER,
                    ↪  viewModel.getCurrentUserModel())
                    startActivity(intent)
                    requireActivity().finish()
                }
            } else {
                utility.isLoading = false
                viewModel.createProfileError.observe(viewLifecycleOwner, {
                    if(it.isNotEmpty()){
                        Toast.makeText(requireContext(), it, Toast.LENGTH_SHORT).show()
                        Log.i(TAG, it)
                        viewModel.createProfileError.value = ""
                    }
                })
            }
        })
    }

    override fun onClick(v: View?) {
        when(v){
            binding.roleOwner -> setRoleType(Role.OWNER)
            binding.roleStaff -> setRoleType(Role.STAFF)
            binding.createUserDOB -> {
                val calendar = Calendar.getInstance(TimeZone.getTimeZone("WIB"))
                calendar[Calendar.MONTH] = Calendar.DECEMBER
```

```
                val constraintBuilder = CalendarConstraints.Builder()
                constraintBuilder.setValidator(DateValidatorPointBackward.now()).setEnd
                ↪  (calendar.timeInMillis)

                val datePicker = MaterialDatePicker.Builder.datePicker()
                    .setTitleText("Date of Birth")
                    .setSelection(viewModel.getDOB())
                    .setInputMode(MaterialDatePicker.INPUT_MODE_CALENDAR)
                    .setCalendarConstraints(constraintBuilder.build())
                    .build()
                datePicker.show(childFragmentManager, "DATE_PICKER")
                datePicker.addOnPositiveButtonClickListener{
                    binding.createUserDOB.setText(viewModel.setDOB(datePicker.selection
                    ↪  ))
                    checkEmpty()
                }
                datePicker.addOnNegativeButtonClickListener{}
                datePicker.isCancelable = false
            }
            binding.createUserEditPhoto ->
            ↪  utility.openEditPhoto(binding.createUserPhoto, ProfileType.USER)
            binding.createUserPhoto -> utility.openImage(binding.createUserPhoto)
            binding.createUserSubmit -> {
                utility.isLoading = true
                viewModel.createUserProfile(
                    binding.createUserName.text.toString(),
                    binding.createUserPhone.text.toString(),
                    binding.createUserAddress.text.toString(),
                    binding.createUserBio.text.toString()
                )
            }
        }
    }

    override fun onPositionChanged(position: Int) {
        when(position){
            0 -> viewModel.setGender(Gender.MALE)
            1 -> viewModel.setGender(Gender.FEMALE)
        }
        checkEmpty()
    }

    private fun setRoleType(role : Role){
        viewModel.setRole(role)
        when(role){
            Role.OWNER -> {
                binding.roleOwner.isChecked = true
                binding.roleStaff.isChecked = false
                binding.createUserRoleDescription.text =
                ↪  getString(R.string.owner_description)
            }
            Role.STAFF -> {
                binding.roleStaff.isChecked = true
                binding.roleOwner.isChecked = false
                binding.createUserRoleDescription.text =
                ↪  getString(R.string.staff_description)
            }
        }
        checkEmpty()
    }

    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after:
    ↪  Int) {}

    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
        checkEmpty()
    }

    override fun afterTextChanged(s: Editable?) {}

```

```
    private fun checkEmpty() {
        val checkerEditText = arrayListOf<TextInputEditText>(
            binding.createUserName,
            binding.createUserPhone,
            binding.createUserAddress,
            binding.createUserDOB)
        viewModel.checkNotEmpty(
            utility.isEmpties(checkerEditText) && (binding.roleOwner.isChecked)
        ).observe(viewLifecycleOwner, {
            binding.createUserSubmit.isEnabled = it
        })
    }
}
```

Kode 33: *Source Code* untuk *Create User Fragment*

b) CREATE FARM FRAGMENT

```
class CreateFarmFragment : Fragment(), View.OnClickListener, TextWatcher {
    private var _binding : FragmentCreateFarmBinding? = null
    private val binding get() = _binding!!
    private lateinit var viewModel : CreateProfileViewModel
    private lateinit var utility: ViewUtility
    private lateinit var editTexts : ArrayList<TextInputEditText>

    companion object {
        const val TAG = "createFarm"
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentCreateFarmBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        requireActivity().title = getString(R.string.create_farm)
        viewModel =
        ↪  ViewModelProvider(requireActivity()).get(CreateProfileViewModel::class.java)
        binding.apply {
            editStaff.visibility = View.GONE
            editTexts = arrayListOf(
                createFarmName,
                createFarmLoc,
                createFarmDesc
            )
            utility = ViewUtility(
                context = requireContext(),
                circularProgressButton = createFarmSubmit,
                textInputEditTexts = editTexts,
                actionBar = (requireActivity() as
                ↪  CreateProfileActivity).supportActionBar
            )

            createFarmSubmit.setOnClickListener(this@CreateFarmFragment)
            editTexts.forEach { it.addTextChangedListener(this@CreateFarmFragment) }
            checkEmpty()
        }

        val mapFragment = childFragmentManager.findFragmentById(R.id.map) as
        ↪  SupportMapFragment?
        mapFragment?.getMapAsync(callback)
    }

    override fun onClick(v: View?) {
        when(v){
```

```
            binding.createFarmSubmit -> {
                utility.isLoading = true
                viewModel.createFarmProfile(
                    binding.createFarmName.text.toString(),
                    binding.createFarmDesc.text.toString(),
                    binding.createFarmLoc.text.toString()
                )
                viewModel.isFarmCreated.observe(this, {
                    if(it){
                        utility.isLoading = false
                        Toast.makeText(requireContext(), "Farm created",
                        ↪  Toast.LENGTH_SHORT).show()
                        val intent = Intent(requireContext(), MainActivity::class.java)
                        intent.putExtra(BuildConfig.CURRENT_USER,
                        ↪  viewModel.getCurrentUserModel())
                        intent.putExtra(BuildConfig.CURRENT_FARM,
                        ↪  viewModel.getCurrentFarmModel())
                        startActivity(intent)
                        Navigation.findNavController(binding.root).navigate(R.id.action
                        ↪  _createGardenFragment_to_createStaffFragment)
                    } else {
                        utility.isLoading = false
                        viewModel.createProfileError.observe(this, {
                            if(it.isNotEmpty()){
                                Toast.makeText(requireContext(), it,
                                ↪  Toast.LENGTH_SHORT).show()
                                Log.i(TAG, it)
                                viewModel.createProfileError.value = ""
                            }
                        })
                    }
                })
            }
        }
    }

    private val callback = OnMapReadyCallback { googleMap ->
        val sydney = LatLng(-34.0, 151.0)
        googleMap.addMarker(MarkerOptions().position(sydney).title("Marker in Sydney"))
        googleMap.moveCamera(CameraUpdateFactory.newLatLng(sydney))
        //todo : Maps. posisi awal sesuai current location, tambahin zoom
    }

    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after:
    ↪  Int) {}

    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
        checkEmpty()
    }

    override fun afterTextChanged(s: Editable?) {}

    private fun checkEmpty() {
        viewModel.checkNotEmpty(utility.isEmpties(editTexts)).observe(viewLifecycleOwne
        ↪  r,
        ↪  {
            binding.createFarmSubmit.isEnabled = it
        })
    }
}
```

Kode 34: *Source Code* untuk *Create Farm Fragment*

6. MAIN ACTIVITY

```
class MainActivity : AppCompatActivity(), View.OnClickListener,
    SwipeRefreshLayout.OnRefreshListener,
    NavigationView.OnNavigationItemSelectedListener {
```

```
4       private lateinit var binding : ActivityMainBinding
5       private lateinit var viewModel : MainViewModel
6       private lateinit var navHostFragment : NavHostFragment
7       private lateinit var navController : NavController
8       private lateinit var actionBarToggle : ActionBarDrawerToggle
9       lateinit var swipeRefresh: SwipeRefreshLayout
10      lateinit var drawerLayout : DrawerLayout
11
12      companion object{
13          const val TAG = "mainActivity"
14      }
15
16      override fun onCreate(savedInstanceState: Bundle?) {
17          super.onCreate(savedInstanceState)
18          binding = ActivityMainBinding.inflate(layoutInflater)
19          setContentView(binding.root)
20
21          viewModel = ViewModelProvider(this).get(MainViewModel::class.java)
22          viewModel.setCurrentData(
23              intent.getParcelableExtra<UserModel>(BuildConfig.CURRENT_USER),
24              intent.getParcelableExtra<FarmModel>(BuildConfig.CURRENT_FARM))
25
26          navHostFragment =
            ↪  supportFragmentManager.findFragmentById(R.id.mainFragmentContainer) as
            ↪  NavHostFragment
27          navController = navHostFragment.navController
28
29          binding.apply {
30              mainBottomNavigation.setupWithNavController(navController)
31              mainVersion.text = BuildConfig.VERSION_NAME
32              swipeRefresh = mainSwipeRefresh
33              mainAddFragment.setOnClickListener(this@MainActivity)
34              mainSwipeRefresh.setOnRefreshListener(this@MainActivity)
35              mainDrawerNavigation.setNavigationItemSelectedListener(this@MainActivity)
36          }
37
38          setupDrawer()
39      }
40
41      private fun setupDrawer() {
42          drawerLayout = binding.mainDrawerLayout
43          actionBarToggle = ActionBarDrawerToggle(this, drawerLayout, 0,0)
44          drawerLayout.addDrawerListener(actionBarToggle)
45          actionBarToggle.syncState()
46
47      }
48
49      override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
50          if (currentFocus != null) {
51              val imm: InputMethodManager =
52                  getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
53              imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
54          }
55          return super.dispatchTouchEvent(ev)
56      }
57
58      override fun onClick(v: View?) {
59          when(v){
60              binding.mainAddFragment -> MainAddFragment().show(supportFragmentManager,
                ↪  TAG+"FAB")
61          }
62      }
63
64      override fun onRefresh() {
65          when(binding.mainBottomNavigation.selectedItemId){
66              R.id.mainHomeFragment -> viewModel.refreshFarm()
67              R.id.mainSocialFragment -> viewModel.refreshPosts()
68              R.id.mainNotesFragment -> viewModel.refreshNotes()
69              R.id.mainProfileFragment -> {}
70          }
```

```
71      }
72
73      override fun onNavigationItemSelected(item: MenuItem): Boolean {
74          return when(item.itemId) {
75              R.id.drawer_edit -> {
76                  val intent = Intent(this, EditProfileUserActivity::class.java)
77                  intent.putExtra(BuildConfig.CURRENT_USER,
                    ↪  viewModel.getCurrentUser().value)
78                  Log.i(TAG, "onNavigationItemSelected:
                    ↪  ${viewModel.getCurrentUser().value}")
79                  startActivity(intent)
80                  drawerLayout.closeDrawer(GravityCompat.END)
81                  true
82              }
83              R.id.drawer_signout -> {
84                  AlertDialog.Builder(this)
85                      .setTitle(getString(R.string.sign_out))
86                      .setMessage(getString(R.string.sign_out_warning))
87                      .setPositiveButton(getString(R.string.sign_out)){ _,_ ->
88                          val uid = Firebase.auth.uid!!
89                          viewModel.signOut()
90                          viewModel.isUserSignOut.observe(this@MainActivity, {
91                              it?.let {
92                                  if(it){
93                                      drawerLayout.closeDrawer(GravityCompat.END)
94                                      DatabaseHandler(this@MainActivity).signOut(uid)
95                                      startActivity(Intent(this@MainActivity,
                                        ↪  SignInActivity::class.java))
96                                      finish()
97                                  } else {
98  //                                      dialogButton.forEach { it.isEnabled = true }
99                                      viewModel.signOutError.observe(this@MainActivity, {
100                                         it?.let {
101                                             if (it.isNotEmpty()) {
102                                                 Toast.makeText(this@MainActivity, it,
                                                    ↪  Toast.LENGTH_SHORT).show()
103                                                 Log.i(TAG, it)
104                                                 viewModel.signOutError.value = null
105                                             }
106                                         }
107                                     })
108                                 }
109                             }
110
111                         })
112                     }
113                     .setNegativeButton(getString(R.string.cancel)){_,_ ->}
114                     .create()
115                     .show()
116                 true
117             }
118             R.id.drawer_feedback -> {
119                 val intent = Intent(this, FeedbackActivity::class.java)
120                 intent.putExtra(BuildConfig.CURRENT_USER,
                    ↪  viewModel.getCurrentUser().value)
121                 startActivity(intent)
122                 drawerLayout.closeDrawer(GravityCompat.END)
123                 true
124             }
125             R.id.drawer_info -> {
126                 IntentUtility(this).openAppInfo()
127                 drawerLayout.closeDrawer(GravityCompat.END)
128                 true
129             }
130             R.id.drawer_language ->{
131                 IntentUtility(this).openLanguageSettings()
132                 drawerLayout.closeDrawer(GravityCompat.END)
133                 true
134             }
135             R.id.drawer_about -> {
```

```
                startActivity(Intent(this, AboutMeActivity::class.java))
                drawerLayout.closeDrawer(GravityCompat.END)
                true
            }
            R.id.drawer_my_post -> {
                val intent = Intent(this, ShowRecyclerActivity::class.java)
                intent.putExtra(BuildConfig.CURRENT_USER,
                ↪  viewModel.getCurrentUser().value)
                intent.putExtra(BuildConfig.ALL_PLANTS, viewModel.getAllPosts().value)
                startActivity(intent)
                true
            }
            else -> false
        }
    }
}
```

Kode 35: *Source Code* untuk *Main Activity* beserta *fragments* terkait

a) MAIN HOME FRAGMENT

```
class MainHomeFragment : Fragment(), View.OnClickListener, CardStackListener{
    private var _binding : FragmentMainHomeBinding? = null
    private val binding get() = _binding!!
    private lateinit var viewModel : MainViewModel
    private lateinit var viewsAsButton : ArrayList<View>
    private lateinit var cardStackLayoutManager: CardStackLayoutManager
    private lateinit var cardStackAdapter : RecyclerView.Adapter<*>

    companion object {
        const val TAG = "mainHomeFragment"
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentMainHomeBinding.inflate(layoutInflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        setHasOptionsMenu(true)
        requireActivity().title = getString(R.string.home)
        viewModel = ViewModelProvider(requireActivity()).get(MainViewModel::class.java)

        binding.apply {
            viewsAsButton = arrayListOf(mainCardPrevious, mainCardNext)
            viewsAsButton.forEach { it.setOnClickListener(this@MainHomeFragment) }

            viewModel.getCurrentFarm().observe(viewLifecycleOwner,{
                it?.let {
                    mainHomeFarmName.text = it.name.toString()
                    Log.i(TAG, "onViewCreated: update")
                }
            })
        }

        setupCardStack()
    }

    private fun setupCardStack() {
        val data : ArrayList<KitModel> = arrayListOf()
        cardStackAdapter = AdapterType.DATA_MONITORING.getAdapter(requireContext(),
        ↪  data)
        viewModel.getCurrentKits().observe(viewLifecycleOwner, {
            val temp = it ?: arrayListOf()
            data.clear()
```

```
48              data.addAll(temp)
49               cardStackAdapter.notifyDataSetChanged()
50              (cardStackAdapter as KitModelAdapter).setOnItemClickListener(
51                  object : KitModelAdapter.OnItemClickListener{
52                      override fun onItemClicked(position: Int) {
53                          val intent = Intent(requireContext(),
                            ↪  ProfileKitActivity::class.java)
54                          intent.putExtra(BuildConfig.CURRENT_USER,
                            ↪  viewModel.getCurrentUser().value)
55                          intent.putExtra(BuildConfig.CURRENT_FARM,
                            ↪  viewModel.getCurrentFarm().value)
56                          intent.putExtra(BuildConfig.SELECTED_KIT,
                            ↪  viewModel.getKit(position))
57                          startActivity(intent)
58                      }
59                  }
60              )
61              Log.i(TAG, "adapter count ${cardStackAdapter.itemCount}")
62              (requireActivity() as MainActivity).swipeRefresh.isRefreshing = false
63
64              if(temp.size ==0){
65                  binding.mainCardPrevious.visibility = View.GONE
66                  binding.mainCardNext.visibility = View.GONE
67              } else {
68                  binding.mainCardPrevious.visibility = View.VISIBLE
69                  binding.mainCardNext.visibility = View.VISIBLE
70              }
71          })
72
73
74          val swipeSetting = SwipeAnimationSetting.Builder()
75              .setDirection(Direction.Right)
76              .setDuration(Duration.Normal.duration)
77              .build()
78
79          val rewindSetting = RewindAnimationSetting.Builder()
80              .setDirection(Direction.Left)
81              .setDuration(Duration.Normal.duration)
82              .build()
83
84          cardStackLayoutManager = CardStackLayoutManager(requireContext(),
            ↪  this@MainHomeFragment).apply {
85              setStackFrom(StackFrom.None)
86              setDirections(Direction.HORIZONTAL)
87              setCanScrollHorizontal(true)
88              setCanScrollVertical(false)
89              setSwipeAnimationSetting(swipeSetting)
90              setRewindAnimationSetting(rewindSetting)
91          }
92
93          binding.mainHomeCardStack.apply {
94              adapter = cardStackAdapter
95              layoutManager = cardStackLayoutManager
96          }
97      }
98
99      override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
100         super.onCreateOptionsMenu(menu, inflater)
101         menu.clear()
102         inflater.inflate(R.menu.menu_main_home, menu)
103     }
104
105     override fun onOptionsItemSelected(item: MenuItem): Boolean {
106         when(item.itemId){
107             R.id.farmProfile -> {
108                 val intent = Intent(activity, EditProfileFarmActivity::class.java)
109                 intent.putExtra(BuildConfig.CURRENT_FARM,
                    ↪  viewModel.getCurrentFarm().value)
110                 intent.putExtra(BuildConfig.CURRENT_USER,
                    ↪  viewModel.getCurrentUser().value)
```

```
                startActivity(intent)
            }
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onClick(v: View?) {
        when(v){
            binding.mainCardNext -> {
                if(cardStackLayoutManager.topPosition < cardStackAdapter.itemCount-1){
                    binding.mainHomeCardStack.swipe()
                } else {
                    binding.mainHomeCardStack.smoothScrollToPosition(0)
                }
            }
            binding.mainCardPrevious -> {
                if(cardStackLayoutManager.topPosition != 0){
                    binding.mainHomeCardStack.rewind()
                } else {
                    binding.mainHomeCardStack.smoothScrollToPosition(cardStackAdapter.i
                    ↪  temCount-1)
                }
            }
        }
    }

    override fun onCardDragging(direction: Direction?, ratio: Float) {}

    override fun onCardSwiped(direction: Direction?) {
        if(cardStackLayoutManager.topPosition == cardStackAdapter.itemCount){
            binding.mainHomeCardStack.scrollToPosition(0)
        }
    }

    override fun onCardRewound() {}

    override fun onCardCanceled() {}

    override fun onCardAppeared(view: View?, position: Int) {
        Log.i(TAG, "appear $position")
    }

    override fun onCardDisappeared(view: View?, position: Int) {
    }
}
```

Kode 36: *Source Code* untuk *Main Home Fragment*

b) MAIN POSTS FRAGMENT

```
class MainPostsFragment : Fragment() {
    private var _binding : FragmentMainPostsBinding? = null
    private val binding get() = _binding!!
    private lateinit var viewModel : MainViewModel

    companion object {
        const val TAG = "mainSocialFragment"
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentMainPostsBinding.inflate(layoutInflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
```

```
19          super.onViewCreated(view, savedInstanceState)
20          requireActivity().title = getString(R.string.social)
21          viewModel = ViewModelProvider(requireActivity()).get(MainViewModel::class.java)
22          setupRecyclerView()
23      }
24
25      private fun setupRecyclerView() {
26          val plantModels : ArrayList<PlantModel> = arrayListOf()
27          val userModels : ArrayList<UserModel> = arrayListOf()
28          val rowAdapter = AdapterType.POST_PLANT.getAdapter(requireContext(),
            ↪  plantModels, allUsers = userModels)
29          viewModel.getAllPosts().observe(viewLifecycleOwner,{
30              plantModels.clear()
31              plantModels.addAll(it ?: arrayListOf())
32              rowAdapter.notifyDataSetChanged()
33              (rowAdapter as PostModelAdapter).setOnItemClickListener(
34                  object : PostModelAdapter.OnItemClickListener{
35                      override fun onItemClicked(
36                          position: Int,
37                          itemView: View,
38                          v: RowItemPostBinding
39                      ) {
40                          when(itemView){
41                              v.postImageContent -> {
42                                  IntentUtility(requireContext()).openImage(itemView,
                                    ↪  viewModel.getPostUser(position)?.name ?: "Photo
                                    ↪  Profile" )
43                              }
44                              v.postRoot -> {
45                                  val intent = Intent(requireContext(),
                                    ↪  ProfileUserActivity::class.java)
46                                  intent.putExtra(BuildConfig.SELECTED_USER,
                                    ↪  viewModel.getPostUser(position))
47                                  startActivity(intent)
48                              }
49                              v.postOptions -> {
50                                  val items = arrayOf("Edit", "Delete")
51                                  MaterialAlertDialogBuilder(context!!)
52                                      .setItems(items){_, which ->
53                                          when(which){
54                                              0 -> gotoPost(position)
55                                              1 -> {
56                                                  viewModel.deletePost(position)
57                                                  viewModel.isPostDeleted.observe(require
                                                    ↪  Activity(),
                                                    ↪  {
58                                                      when(it){
59                                                          true -> Toast.makeText(requireA
                                                            ↪  ctivity(), "Post deleted.",
                                                            ↪  Toast.LENGTH_SHORT).show()
60                                                      }
61                                                  })
62                                              }
63                                          }
64                                      }
65                                      .show()
66                              }
67                          }
68                      }
69
70                  }
71              )
72              Log.i(TAG, "${rowAdapter.itemCount}")
73          })
74
75          viewModel.getAllUsers().observe(viewLifecycleOwner, {
76              userModels.clear()
77              userModels.addAll(it ?: arrayListOf())
78              rowAdapter.notifyDataSetChanged()
79          })
```

```
80
81          binding.mainSocialsRecyclerView.apply {
82              adapter = rowAdapter
83              layoutManager = LinearLayoutManager(requireContext())
84              addItemDecoration(object : DividerItemDecoration(requireContext(),
                ↪  VERTICAL) {})
85              setHasFixedSize(true)
86          }
87      }
88
89      private fun gotoPost(position:Int){
90          val intent = Intent(requireContext(), AddPlantActivity::class.java)
91          intent.putExtra(BuildConfig.SELECTED_PLANT, viewModel.getPost(position))
92          startActivity(intent)
93      }
94  }
```

Kode 37: *Source Code* untuk *Main Posts Fragment*

c) MAIN ADD FRAGMENT

```
1   class MainAddFragment : BottomSheetDialogFragment(),
    ↪  NavigationView.OnNavigationItemSelectedListener {
2       private var _binding : FragmentMainAddBinding? = null
3       private val binding get() = _binding!!
4       private lateinit var viewModel : MainViewModel
5
6       override fun onCreateView(
7           inflater: LayoutInflater, container: ViewGroup?,
8           savedInstanceState: Bundle?
9       ): View {
10          _binding = FragmentMainAddBinding.inflate(layoutInflater, container, false)
11          return binding.root
12      }
13
14      override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
15          super.onViewCreated(view, savedInstanceState)
16          viewModel = ViewModelProvider(requireActivity()).get(MainViewModel::class.java)
17          binding.mainAddMenu.itemIconTintList = null
18          binding.mainAddMenu.setNavigationItemSelectedListener(this)
19      }
20
21      override fun onNavigationItemSelected(item: MenuItem): Boolean {
22          return when(item.itemId){
23              R.id.addDataMonitoring  ->
                ↪  gotoActivity(AddDataMonitoringActivity::class.java)
24              R.id.addCrops           -> gotoActivity(AddCropsActivity::class.java)
25              R.id.addNote            -> gotoActivity(AddNoteActivity::class.java)
26              R.id.addKit             -> gotoActivity(AddKitActivity::class.java)
27              R.id.addPlant           -> gotoActivity(AddPlantActivity::class.java)
28              else                    -> false
29          }
30      }
31
32      private fun gotoActivity(destination : Class<*>) : Boolean{
33          dismiss()
34          val intent = Intent(requireActivity(), destination)
35          intent.putExtra(BuildConfig.CURRENT_USER, viewModel.getCurrentUser().value)
36          intent.putExtra(BuildConfig.CURRENT_FARM, viewModel.getCurrentFarm().value)
37          startActivity(intent)
38          return true
39      }
40  }
```

Kode 38: *Source Code* untuk *Main Add Fragment*

d) MAIN NOTES FRAGMENT

```
class MainNotesFragment : Fragment(), SwipeRefreshLayout.OnRefreshListener {
    private var _binding : FragmentMainNotesBinding? = null
    private val binding get() = _binding!!
    private lateinit var viewModel : MainViewModel

    companion object {
        const val TAG = "mainNotesFragment"
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentMainNotesBinding.inflate(layoutInflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        requireActivity().title = getString(R.string.notes)
        viewModel = ViewModelProvider(requireActivity()).get(MainViewModel::class.java)
        setupRecyclerView()
    }

    private fun setupRecyclerView() {
        val data : ArrayList<NoteModel> = arrayListOf()
        val rowAdapter = AdapterType.NOTE.getAdapter(requireContext(), data)
        viewModel.getCurrentNotes().observe(viewLifecycleOwner,{
            val currentData : ArrayList<NoteModel> = it ?: arrayListOf()
            data.clear()
            data.addAll(currentData)
            rowAdapter.notifyDataSetChanged()
            (rowAdapter as NoteModelAdapter).setOnItemClickListener(
                object : NoteModelAdapter.OnItemClickListener{
                    override fun onItemClicked(
                        position: Int,
                        itemView: View,
                        v: RowItemNoteBinding
                    ) {
                        gotoNote(position)
                    }

                    override fun onItemLongClicked(position: Int) {
                        val items = arrayOf("Edit", "Delete")
                        MaterialAlertDialogBuilder(context!!)
                            .setItems(items){_, which ->
                                when(which){
                                    0 -> gotoNote(position)
                                    1 -> {
                                        viewModel.deleteNote(position)
                                        viewModel.isNoteDeleted.observe(requireActivity
                                        ↪  (),
                                        ↪  {
                                            when(it){
                                                true ->
                                                ↪  Toast.makeText(requireContext(),
                                                ↪  "Note deleted.",
                                                ↪  Toast.LENGTH_SHORT).show()
                                            }
                                        })
                                    }
                                }
                            }
                            .show()
                    }
                }
            )
            Log.i(TAG, "${rowAdapter.itemCount}")
```

```
            (requireActivity() as MainActivity).swipeRefresh.isRefreshing = false
            binding.noteLine.visibility = if(currentData.size==0){
                 View.GONE
            } else {
                View.VISIBLE
            }
        })

        binding.mainNotesRecyclerView.apply {
            adapter = rowAdapter
            layoutManager = LinearLayoutManager(requireContext())
        }
    }

    override fun onRefresh() {
        viewModel.refreshNotes()
    }

    private fun gotoNote(position:Int){
        val intent = Intent(requireContext(), AddNoteActivity::class.java)
        intent.putExtra(BuildConfig.SELECTED_NOTE, viewModel.getNote(position))
        startActivity(intent)
    }
}
```

Kode 39: *Source Code* untuk *Main Notes Fragment*

e) MAIN PROFILE FRAGMENT

```
class MainProfileFragment : Fragment(), View.OnClickListener {
    private var _binding : FragmentMainProfileBinding? = null
    private val  binding get() = _binding!!
    private lateinit var viewModel : MainViewModel
    private lateinit var drawerLayout : DrawerLayout
    private lateinit var viewsAsButton : ArrayList<View>

    companion object {
        const val TAG = "mainProfile"
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentMainProfileBinding.inflate(layoutInflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        setHasOptionsMenu(true)
        requireActivity().title = getString(R.string.profile)
        viewModel = ViewModelProvider(requireActivity()).get(MainViewModel::class.java)
        drawerLayout = (requireActivity() as MainActivity).drawerLayout

        binding.apply {
            viewsAsButton = arrayListOf(mainProfileEmailGroup,
                                        mainProfilePhoneGroup,
                                        mainProfileAddressGroup,
                                        mainProfilePhoto)
            viewsAsButton.forEach { it.setOnClickListener(this@MainProfileFragment) }

            viewModel.getCurrentUser().observe(requireActivity(),{
                try {
                    it?.let {
                        mainProfileName.text = it.name
                        mainProfilePhone.text = it.phone
                        mainProfileAddress.text = it.address
                        mainProfileJoinedSince.text =
```

```
                            getString(R.string.joined_company_since, it.joinedSince)
                        mainProfileBio.text = it.bio
                        it.performanceRate?.let {
                            mainProfileRate.rating = it
                        }
                        mainProfileEmail.text = it.email
                        mainProfileRole.text = getString(
                            R.string.role_in_profile,
                            it.role!!.replaceFirstChar { it.uppercase() },
                            viewModel.getCurrentFarm().value?.name.toString()
                        )
                        Glide.with(requireActivity())
                            .load(it.photo_url ?: R.drawable.bg_farmer)
                            .centerCrop()
                            .into(mainProfilePhoto)
                    }
                } catch (e:Exception){
                    Log.e(TAG, "currentUser value null in some field", e)
                }
            })
        }
    }

    override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
        super.onCreateOptionsMenu(menu, inflater)
        menu.clear()
        inflater.inflate(R.menu.menu_main_profile, menu)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when(item.itemId){
            R.id.settings -> {
                if(drawerLayout.isDrawerOpen(GravityCompat.END)){
                    drawerLayout.closeDrawer(GravityCompat.END)
                }
                else{
                    drawerLayout.openDrawer(GravityCompat.END)
                }
            }
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onClick(v: View?) {
        when(v){
            binding.mainProfileEmail -> IntentUtility(requireContext()).openEmail(bindi
            ↪  ng.mainProfileEmail.text.toString())
            binding.mainProfilePhoneGroup ->
            ↪  RequestPermission().requestPermission(requireActivity(),
            ↪  Manifest.permission.CALL_PHONE, "Phone call",
            ↪  binding.mainProfilePhone.text.toString())
            binding.mainProfileAddressGroup -> IntentUtility(requireContext()).openMaps
            ↪  (binding.mainProfileAddress.text.toString())
            binding.mainProfilePhoto ->
            ↪  IntentUtility(requireContext()).openImage(binding.mainProfilePhoto,
            ↪  binding.mainProfileName.text.toString())
        }
    }
}
```

Kode 40: *Source Code* untuk *Main Profile Fragment*

7. ADD CROPS ACTIVITY

```
class AddCropsActivity : AppCompatActivity(), View.OnClickListener {
    private lateinit var binding : ActivityAddCropsBinding
    private lateinit var viewModel : AddDataMonitoringViewModel
    private lateinit var utility: ViewUtility
```

```
5       private lateinit var viewsAsButton : ArrayList<View>
6
7       companion object{
8           const val TAG = "addCrops"
9       }
10
11      override fun onCreate(savedInstanceState: Bundle?) {
12          super.onCreate(savedInstanceState)
13          binding = ActivityAddCropsBinding.inflate(layoutInflater)
14          setContentView(binding.root)
15
16          viewModel = ViewModelProvider(this).get(AddDataMonitoringViewModel::class.java)
17          viewModel.setCurrentData(intent.getParcelableExtra<UserModel>(BuildConfig.CURRE
            ↪ NT_USER),
18              intent.getParcelableExtra<FarmModel>(BuildConfig.CURRENT_FARM),
19              isAddCrops = true)
20          supportActionBar?.title = getString(R.string.add_new_crops)
21          supportActionBar?.subtitle = viewModel.currentFarmModel.value?.name
22          supportActionBar?.setDisplayHomeAsUpEnabled(true)
23          supportActionBar?.setDisplayShowHomeEnabled(false)
24
25          binding.apply {
26              viewsAsButton = arrayListOf(acSelectPlant, acNewPlant, acCropsSubmit)
27              viewsAsButton.forEach { it.setOnClickListener(this@AddCropsActivity) }
28
29              utility = ViewUtility(
30                  context = this@AddCropsActivity,
31                  circularProgressButton = acCropsSubmit,
32                  viewsAsButton = viewsAsButton,
33                  actionBar = supportActionBar
34              )
35
36              viewModel.getKitSelector().observe(this@AddCropsActivity, {
37                  it?.let{
38                      val adapter = ArrayAdapter(this@AddCropsActivity,
                        ↪ R.layout.row_item_list, it)
39                      (acSelectKit.editText as? AutoCompleteTextView)?.setAdapter(adapter)
40                      Log.i(TAG, "${adapter.count} ${it}")
41
42                      (acSelectKit.editText as?
                        ↪ AutoCompleteTextView)?.setOnItemClickListener { parent, view,
                        ↪ position, id ->
43                          viewModel.setCurrentKit(position)
44                          checkEmpty()
45                      }
46                  }
47              })
48
49              viewModel.getCurrentPlant().observe(this@AddCropsActivity, {
50                  if(it != null){
51                      try{
52                          val estimated : String =
                            ↪ utility.formatDateToString(utility.formatStringToDate(est =
                            ↪ it.growthTime!!)).toString()
53                          acPlantName.setText(it.name)
54                          acPlantDescription.setText(it.description)
55                          acPlantGrowth.setText(getString(R.string.d_days, it.growthTime))
56                          acPlantEstimated.setText(estimated)
57                          acPlantTempMin.setText(getString(R.string.temp_value,
                            ↪ it.tempLv?.min))
58                          acPlantTempMax.setText(getString(R.string.temp_value,
                            ↪ it.tempLv?.max))
59                          acPlantHumidMin.setText(getString(R.string.humid_value,
                            ↪ it.humidLv?.min))
60                          acPlantHumidMax.setText(getString(R.string.humid_value,
                            ↪ it.humidLv?.max))
61                          acPlantAcidMin.setText(it.phLv?.min.toString())
62                          acPlantAcidMax.setText(it.phLv?.max.toString())
63                          Glide.with(this@AddCropsActivity)
64                              .load(it.photo_url)
```

```
                            .circleCrop()
                            .into(acPlantPhoto)
                    } catch (e:Exception){
                        Log.e(AddDataMonitoringActivity.TAG, "cannot parse plant data",
                        ↪ e)
                    }
                } else {
                    acPlantSelector.visibility = View.VISIBLE
                    acCropsContent.visibility = View.GONE
                }
                checkEmpty()
            })
            checkEmpty()
        }
    }

    private fun checkEmpty() {
        viewModel.checkNotEmpty(
            viewModel.getCurrentKit().value != null &&
                    viewModel.getCurrentPlant().value != null
        ).observe(this, {
            binding.acCropsSubmit.isEnabled = it
        })
    }

    override fun onSupportNavigateUp(): Boolean {
        super.onBackPressed()
        finish()
        return true
    }

    override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
        if (currentFocus != null) {
            val imm: InputMethodManager =
                getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
            imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
        }
        return super.dispatchTouchEvent(ev)
    }

    override fun onClick(v: View?) {
        when(v){
            binding.acNewPlant -> {
                val intent = Intent(this, AddPlantActivity::class.java)
                intent.putExtra("from", TAG)
                startForResult.launch(intent)
            }
            binding.acSelectPlant -> {
                val intent = Intent(this, SearchActivity::class.java)
                intent.putExtra("search", TAG)
                startForResult.launch(intent)
            }
            binding.acCropsSubmit -> {
                utility.isLoading = true
                viewModel.addCrops()
                viewModel.isCropsAdd.observe(this, {
                    if(it){
                        utility.isLoading = false
                        Toast.makeText(this, "New crops added",
                        ↪ Toast.LENGTH_SHORT).show()
                        super.onBackPressed()
                        finish()
                    } else {
                        utility.isLoading = false
                        viewModel.addCropsError.observe(this, {
                            if(it.isNotEmpty()){
                                Toast.makeText(this, it, Toast.LENGTH_SHORT).show()
                                Log.i(CreateFarmFragment.TAG, it)
                                viewModel.addCropsError.value = ""
                            }
```

```
                        })
                    }
                })
            }
        }
    }

    private val startForResult =
    ↪ registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
    ↪ data ->
        Log.i(CreateProfileActivity.TAG, "$data")
        try{
            if(data?.resultCode == Activity.RESULT_OK){
                data.data?.getParcelableExtra<PlantModel>(BuildConfig.SELECTED_PLANT)?.
                ↪ let
                ↪ {
                    binding.acPlantSelector.visibility = View.GONE
                    binding.acCropsContent.visibility = View.VISIBLE
                    viewModel.setCurrentPlant(it)
                }
            }
        } catch (e: Exception){
            Toast.makeText(this, e.message.toString(), Toast.LENGTH_SHORT).show()
        }
    }

}
```

Kode 41: *Source Code* untuk *Add Crops Activity*

8. ADD DATA MONITORING ACTIVITY

```
class AddDataMonitoringActivity : AppCompatActivity(), View.OnClickListener {
    private lateinit var binding : ActivityAddDataMonitoringBinding
    private lateinit var bindingFragment : ActivityAddCropsBinding
    private lateinit var viewModel : AddDataMonitoringViewModel
    private lateinit var utility: ViewUtility
    private lateinit var tooltips : ArrayList<ImageButton>
    private lateinit var viewsAsButton : ArrayList<View>
    private lateinit var numberPickers : ArrayList<ClickNumberPickerView>

    companion object{
        const val TAG = "addDataMonitoring"
    }

    //todo : kurang tombol buat panen, nampilin yg msh ada tanemannya
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityAddDataMonitoringBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider(this).get(AddDataMonitoringViewModel::class.java)
        supportActionBar?.title = getString(R.string.add_recent_data_monitoring)
        viewModel.setCurrentData(intent.getParcelableExtra<UserModel>(BuildConfig.CURRE
        ↪ NT_USER),
            intent.getParcelableExtra<FarmModel>(BuildConfig.CURRENT_FARM),
            isAddCrops = false)
        supportActionBar?.subtitle = viewModel.currentFarmModel.value?.name
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowHomeEnabled(false)

        //todo : tooltip semuanya
        binding.apply {
            bindingFragment = binding.addMonitoringCropsCreate
            bindingFragment.apply {
                acSelectKit.visibility = View.GONE
                acCropsSubmit.visibility = View.GONE

```

```
                viewsAsButton = arrayListOf(
                    addMonitoringSubmit,
                    addMonitoringCropsButton,
                    acNewPlant,
                    acSelectPlant
                )
                numberPickers = arrayListOf(
                    addMonitoringTemperature,
                    addMonitoringHumidity,
                    addMonitoringAcidity,
                    addMonitoringWaterTank,
                    addMonitoringNutrientTank,
                    addMonitoringWaterTurbidity
                )
                tooltips = arrayListOf(admTempInfo,
                    admHumidInfo,
                    admAcidityInfo,
                    admWaterLevelInfo,
                    admNutrientLevelInfo,
                    admWaterTurbidityInfo)

                utility = ViewUtility(
                    context = this@AddDataMonitoringActivity,
                    circularProgressButton = addMonitoringSubmit,
                    viewsAsButton = viewsAsButton,
                    actionBar = supportActionBar
                )

                numberPickers.forEachIndexed { i, view ->
                    val numberPickerDataType = getNumberPickerType(view)
                    viewModel.getNumberPickerValue(numberPickerDataType)?.observe(this@
                    ↪  AddDataMonitoringActivity,{
                        if(it != view.value){
                            view.setPickerValue(it)
                        }
                    })

                    numberPickers[i].setClickNumberPickerListener { previousValue,
                    ↪  currentValue, pickerClickType ->
                        viewModel.setNumberPickerValue(currentValue,
                        ↪  numberPickerDataType)
                        checkEmpty()
                    }
                }
                viewsAsButton.forEach {
                ↪  it.setOnClickListener(this@AddDataMonitoringActivity) }
                tooltips.forEach {
                    it.setOnClickListener {
                        it.performLongClick()
                    }
                }
                admHumidInfo.tooltipText = "Min : \n" +
                        "Max : "

                viewModel.getCurrentCrops().observe(this@AddDataMonitoringActivity, {
                    if(it == null){ //crops not found
                        addMonitoringCropsPlanted.root.visibility = View.GONE
                        addMonitoringNewCrops.visibility = View.VISIBLE
                    } else { //crops found
                        if(it.cropsId != null) {
                            addMonitoringCropsPlanted.root.visibility = View.VISIBLE
                            addMonitoringNewCrops.visibility = View.GONE
                            addMonitoringCropsPlanted.apply {
                                //todo kurang moreday
//                              val plantedDate : String =
↪  utility.formatTimestampToDate(it.timestamp)
//                              val estimateDate : Long =
↪  utility.formatStringToDate(plantedDate, it.plantModel?.growthTime)
                                this.kitPlantName.text = it.plantModel?.name.toString()
//                              this.kitPlantDate.text = plantedDate
```

```
                        this.kitPlantHarvest.text = getString(
                            R.string.can_be_harvested_at_s,
                            "now"
                        )//utility.formatDateToString(estimateDate))
                        Glide.with(this@AddDataMonitoringActivity)
                            .load(it.plantModel?.photo_url)
                            .circleCrop()
                            .into(this.kitPlantPhoto)
                    }
                }
            }
        })

        viewModel.getCurrentPlant().observe(this@AddDataMonitoringActivity, {
            if(it != null){
                try{
                    acPlantSelector.visibility = View.GONE
                    acCropsContent.visibility = View.VISIBLE
                    addMonitoringCropsButton.visibility = View.GONE

                    val estimated : String = utility.formatDateToString(utility
                    ↪  .formatStringToDate(est =
                    ↪  it.growthTime!!)).toString()
                    acPlantName.setText(it.name)
                    acPlantDescription.setText(it.description)
                    acPlantGrowth.setText(getString(R.string.d_days,
                    ↪  it.growthTime))
                    acPlantEstimated.setText(estimated)
                    acPlantTempMin.setText(getString(R.string.temp_value,
                    ↪  it.tempLv?.min))
                    acPlantTempMax.setText(getString(R.string.temp_value,
                    ↪  it.tempLv?.max))
                    acPlantHumidMin.setText(getString(R.string.humid_value,
                    ↪  it.humidLv?.min))
                    acPlantHumidMax.setText(getString(R.string.humid_value,
                    ↪  it.humidLv?.max))
                    acPlantAcidMin.setText(it.phLv?.min.toString())
                    acPlantAcidMax.setText(it.phLv?.max.toString())
                    Glide.with(this@AddDataMonitoringActivity)
                        .load(it.photo_url)
                        .circleCrop()
                        .into(acPlantPhoto)
                } catch (e:Exception){
                    Log.e(TAG, "cannot parse plant data", e)
                }
            } else {
                acPlantSelector.visibility = View.VISIBLE
                acCropsContent.visibility = View.GONE
                addMonitoringCropsButton.visibility = View.VISIBLE
            }
            checkEmpty()
        })

        viewModel.getKitSelector().observe(this@AddDataMonitoringActivity, {
            it?.let{
                val adapter = ArrayAdapter(this@AddDataMonitoringActivity,
                ↪  R.layout.row_item_list, it)
                (admSelectKit.editText as?
                ↪  AutoCompleteTextView)?.setAdapter(adapter)
                Log.i(TAG, "${adapter.count} ${it}")

                (admSelectKit.editText as?
                ↪  AutoCompleteTextView)?.setOnItemClickListener { parent,
                ↪  view, position, id ->
                    viewModel.setCurrentKit(position)
                    checkEmpty()
                }
            }
        })
        checkEmpty()
```

```
159             }
160         }
161     }
162
163     private fun checkEmpty() {
164         val currTemp : Float = viewModel.getNumberPickerValue(getNumberPickerType(bindi
            ↪ ng.addMonitoringTemperature))?.value
            ↪ ?:0f
165         val currHumid : Float = viewModel.getNumberPickerValue(getNumberPickerType(bind
            ↪ ing.addMonitoringHumidity))?.value
            ↪ ?:0f
166         val currAcid : Float =  viewModel.getNumberPickerValue(getNumberPickerType(bind
            ↪ ing.addMonitoringAcidity))?.value
            ↪ ?:0f
167         val currWaterTank : Float =  viewModel.getNumberPickerValue(getNumberPickerType
            ↪ (binding.addMonitoringWaterTank))?.value
            ↪ ?:0f
168         val currNutrientTank : Float =  viewModel.getNumberPickerValue(getNumberPickerT
            ↪ ype(binding.addMonitoringNutrientTank))?.value ?:
            ↪ 0f
169         val currentTurbidity : Float =  viewModel.getNumberPickerValue(getNumberPickerT
            ↪ ype(binding.addMonitoringWaterTurbidity))?.value
            ↪ ?:0f
170
171         viewModel.checkNotEmpty(
172             viewModel.getCurrentKit().value != null &&
173                     currTemp > 0f &&
174                     currHumid > 0f &&
175                     currAcid > 0f &&
176                     currWaterTank > 0f &&
177                     currNutrientTank > 0f &&
178                     currentTurbidity > 0f
179         ).observe(this, {
180             binding.addMonitoringSubmit.isEnabled = it
181         })
182     }
183
184     override fun onSupportNavigateUp(): Boolean {
185         super.onBackPressed()
186         finish()
187         return true
188     }
189
190     override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
191         if (currentFocus != null) {
192             val imm: InputMethodManager =
193                 getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
194             imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
195         }
196         return super.dispatchTouchEvent(ev)
197     }
198
199     override fun onClick(v: View?) {
200         when(v){
201             binding.addMonitoringSubmit -> {
202                 utility.isLoading = true
203                 when(binding.addCropsLayout.visibility){
204                     View.GONE -> viewModel.addDataMonitoring()
205                     View.VISIBLE -> viewModel.addCrops()
206                 }
207                 viewModel.isDataMonitoringAdd.observe(this,{
208                     if(it){
209                         utility.isLoading = false
210                         Toast.makeText(this, "New data monitoring added",
                            ↪ Toast.LENGTH_SHORT).show()
211                         super.onBackPressed()
212                         finish()
213                     } else {
214                         utility.isLoading = false
215                         viewModel.addDataMonitoringError.observe(this, {
```

```
                            if(it.isNotEmpty()){
                                Toast.makeText(this, it, Toast.LENGTH_SHORT).show()
                                Log.i(CreateFarmFragment.TAG, it)
                                viewModel.addDataMonitoringError.value = ""
                            }
                        })
                    }
                })
            }
            binding.addMonitoringCropsButton ->{
                when(binding.addCropsLayout.visibility){
                    View.GONE -> {
                        binding.addCropsLayout.visibility = View.VISIBLE
                        binding.addMonitoringCropsButton.text =
                        ↪  getString(R.string.cancel)
                    }
                    View.VISIBLE -> {
                        binding.addCropsLayout.visibility = View.GONE
                        binding.addMonitoringCropsButton.text =
                        ↪  getString(R.string.add_new_crops)
                    }
                }
            }
            bindingFragment.acNewPlant -> {
                val intent = Intent(this, AddPlantActivity::class.java)
                intent.putExtra("from", TAG)
                startForResult.launch(intent)
            }
            bindingFragment.acSelectPlant -> {
                val intent = Intent(this, SearchActivity::class.java)
                intent.putExtra("search", TAG)
                startForResult.launch(intent)
            }
        }
    }

    private val startForResult =
    ↪  registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
    ↪  data ->
        Log.i(CreateProfileActivity.TAG, "$data")
        try{
            if(data?.resultCode == Activity.RESULT_OK){
                data.data?.getParcelableExtra<PlantModel>(BuildConfig.SELECTED_PLANT)?.
                ↪  let
                ↪  {
                    bindingFragment.acPlantSelector.visibility = View.GONE
                    bindingFragment.acCropsContent.visibility = View.VISIBLE
                    viewModel.setCurrentPlant(it)
                }
            }
        } catch (e: Exception){
            Toast.makeText(this, e.message.toString(), Toast.LENGTH_SHORT).show()
        }
    }

    private fun getNumberPickerType(view: ClickNumberPickerView) : NumberPickerType?{
        return when (view) {
            binding.addMonitoringTemperature -> NumberPickerType.TEMPERATURE
            binding.addMonitoringHumidity -> NumberPickerType.HUMIDITY
            binding.addMonitoringAcidity -> NumberPickerType.ACIDITY
            binding.addMonitoringWaterTank -> NumberPickerType.WATER_TANK
            binding.addMonitoringNutrientTank -> NumberPickerType.NUTRIENT_TANK
            binding.addMonitoringWaterTurbidity -> NumberPickerType.TURBIDITY
            else -> null
        }
    }
}
```

Kode 42: *Source Code* untuk *Add Data Monitoring Activity*

9. ADD KIT ACTIVITY

```
class AddKitActivity : AppCompatActivity(), TextWatcher{
    private lateinit var binding : ActivityAddKitBinding
    private lateinit var viewModel : AddKitViewModel
    private lateinit var utility : ViewUtility
    private lateinit var editTexts : ArrayList<TextInputEditText>
    private lateinit var tooltips : ArrayList<ImageButton>
    private lateinit var numberPickers : ArrayList<ClickNumberPickerView>
    private var strEdt : HashMap<String, TextInputEditText> = hashMapOf()

    companion object {
        const val TAG = "addKitActivity"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityAddKitBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider(this).get(AddKitViewModel::class.java)
        viewModel.setCurrentData(intent.getParcelableExtra<UserModel>(BuildConfig.CURRE
        ↪  NT_USER),
            intent.getParcelableExtra<FarmModel>(BuildConfig.CURRENT_FARM),
            intent.getParcelableExtra<KitModel>(BuildConfig.SELECTED_KIT))

        supportActionBar?.title = viewModel.getCurrentKit().value?.name ?:
        ↪  getString(R.string.create_new_kit)
        supportActionBar?.subtitle = viewModel.getCurrentFarm().value?.name
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowHomeEnabled(false)

        binding.apply {
            setupDefaultData()
            editTexts = arrayListOf(addKitName, addKitPosition)
            tooltips = arrayListOf(akSizeInfo,
                                   akWaterLevelInfo,
                                   akNutrientLevelInfo,
                                   akTurbidityLevelInfo)
            numberPickers = arrayListOf(addKitWidth,
                addKitLength,
                addKitWaterMin,
                addKitWaterMax,
                addKitNutrientMin,
                addKitNutrientMax,
                addKitTurbidityMin,
                addKitTurbidityMax)
            utility = ViewUtility(context = this@AddKitActivity,
                circularProgressButton = addKitSubmit,
                textInputEditTexts = editTexts,
                numberPickers = numberPickers,
                actionBar = supportActionBar)

            tooltips.forEach {
                it.setOnClickListener {
                    it.performLongClick()
                }
            }
            akSizeInfo.tooltipText = "How many holes on each side length and width
            ↪  (must more than 0)."
            akWaterLevelInfo.tooltipText = getString(R.string.level_tooltip)
            akNutrientLevelInfo.tooltipText = getString(R.string.level_tooltip)
            akTurbidityLevelInfo.tooltipText = "belum ditentuin"
            //todo : turbidity tooltip

            editTexts.forEach { it.addTextChangedListener(this@AddKitActivity) }
            numberPickers.forEachIndexed { i, view ->
                val numberPickerDataType = getNumberPickerType(view)
                viewModel.getNumberPickerValue(numberPickerDataType)?.observe(this@AddK
                ↪  itActivity,{
```

```
                    if(it != view.value){
                        view.setPickerValue(it)
                    }
                })

                numberPickers[i].setClickNumberPickerListener { previousValue,
                ↪  currentValue, pickerClickType ->
                    viewModel.setNumberPickerValue(currentValue, numberPickerDataType)
                    checkEmpty()
                }
            }

            addKitSubmit.setOnClickListener {
                utility.isLoading = true
                viewModel.createKit(binding.addKitName.text.toString(),
                                    binding.addKitPosition.text.toString())
                viewModel.isKitAdd.observe(this@AddKitActivity, {
                    if(it){
                        utility.isLoading = false
                        val toastTxt = if(intent.getParcelableExtra<NoteModel>(BuildCon
                        ↪  fig.SELECTED_NOTE) !=
                        ↪  null){
                            "Kit updated."
                        } else {
                            "New kit created."
                        }
                        Toast.makeText(this@AddKitActivity, toastTxt,
                        ↪  Toast.LENGTH_SHORT).show()
                        super.onBackPressed()
                        finish()
                    } else {
                        utility.isLoading = false
                        viewModel.addKitError.observe(this@AddKitActivity, {
                            if(it.isNotEmpty()){
                                Toast.makeText(this@AddKitActivity, it,
                                ↪  Toast.LENGTH_SHORT).show()
                                Log.i(CreateFarmFragment.TAG, it)
                                viewModel.addKitError.value = ""
                            }
                        })
                    }
                })
            }
            checkEmpty()
        }
    }

    private fun setupDefaultData(){
        binding.apply {
            viewModel.getCurrentKit().observe(this@AddKitActivity, { it ->
                it.kitId?.let { kitId ->
                    addKitName.setText(it.name)
                    addKitPosition.setText(it.position)
                    addKitWidth.setPickerValue(it.width?.toFloat() ?: 0f)
                    addKitLength.setPickerValue(it.length?.toFloat() ?: 0f)
                    addKitWaterMin.setPickerValue(it.waterLv?.min ?: 0f)
                    addKitWaterMax.setPickerValue(it.waterLv?.max ?: 0f)
                    addKitNutrientMin.setPickerValue(it.nutrientLv?.min ?: 0f)
                    addKitNutrientMax.setPickerValue(it.nutrientLv?.max ?: 0f)
                    addKitTurbidityMin.setPickerValue(it.turbidityLv?.min ?: 0f)
                    addKitTurbidityMax.setPickerValue(it.turbidityLv?.max ?: 0f)

                    strEdt[it.name ?: ""] = addKitName
                    strEdt[it.position ?: ""] = addKitPosition
                }
            })
        }
    }

    override fun onSupportNavigateUp(): Boolean {
```

```
130             if(intent.getParcelableExtra<KitModel>(BuildConfig.SELECTED_KIT) != null){
131                 viewModel.isNotEmpties.observe(this, {
132                     when(it){
133                         true -> binding.addKitSubmit.performClick()
134                         else -> super.onBackPressed()
135                     }
136                 })
137             } else {
138                 super.onBackPressed()
139             }
140             return true
141         }
142 
143         override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
144             if (currentFocus != null) {
145                 val imm: InputMethodManager =
146                     getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
147                 imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
148             }
149             return super.dispatchTouchEvent(ev)
150         }
151 
152         override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after:
        ↪  Int) {}
153 
154         override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
155             checkEmpty()
156         }
157 
158         override fun afterTextChanged(s: Editable?) {}
159 
160         private fun checkEmpty(){
161             val currWidth : Float = viewModel.getNumberPickerValue(getNumberPickerType(bind
                ↪  ing.addKitWidth))?.value ?:
                ↪  0f
162             val currLength : Float = viewModel.getNumberPickerValue(getNumberPickerType(bin
                ↪  ding.addKitLength))?.value ?:
                ↪  0f
163             val currWaterMin : Float? =  viewModel.getNumberPickerValue(getNumberPickerType
                ↪  (binding.addKitWaterMin))?.value
164             val currWaterMax : Float? =  viewModel.getNumberPickerValue(getNumberPickerType
                ↪  (binding.addKitWaterMax))?.value
165             val currNutrientMin : Float? =  viewModel.getNumberPickerValue(getNumberPickerT
                ↪  ype(binding.addKitNutrientMin))?.value
166             val currentNutrientMax : Float? =  viewModel.getNumberPickerValue(getNumberPick
                ↪  erType(binding.addKitNutrientMax))?.value
167             val currentTurbidityMin : Float? = viewModel.getNumberPickerValue(getNumberPick
                ↪  erType(binding.addKitTurbidityMin))?.value
168             val currentTurbidityMax : Float? = viewModel.getNumberPickerValue(getNumberPick
                ↪  erType(binding.addKitTurbidityMax))?.value
169             val hashMap : HashMap<Float?, Float?> = hashMapOf()
170             hashMap[currWaterMin] = currWaterMax
171             hashMap[currNutrientMin] = currentNutrientMax
172             hashMap[currentTurbidityMin] = currentTurbidityMax
173 
174             if(intent.getParcelableExtra<KitModel>(BuildConfig.SELECTED_KIT) != null){
175                 viewModel.getCurrentKit().value?.let {
176                     val floNumPick : HashMap<Float, Float> = hashMapOf()
177                     floNumPick[it.width?.toFloat() ?: 0f] = currWidth
178                     floNumPick[it.length?.toFloat() ?: 0f] = currLength
179                     floNumPick[it.waterLv?.min ?: 0f] = currWaterMin ?:0f
180                     floNumPick[it.waterLv?.max ?: 0f] = currWaterMax ?:0f
181                     floNumPick[it.nutrientLv?.min ?: 0f] = currNutrientMin ?:0f
182                     floNumPick[it.nutrientLv?.max ?: 0f] = currentNutrientMax ?:0f
183                     floNumPick[it.turbidityLv?.min ?: 0f] = currentTurbidityMin ?:0f
184                     floNumPick[it.turbidityLv?.max ?: 0f] = currentTurbidityMax ?:0f
185 
186                     viewModel.checkNotEmpty(utility.isEmpties(editTexts) &&
187                             currWidth > 0f &&
188                             currLength > 0f &&
```

```
                        utility.isInRanges(hashMap) &&
                        (utility.isChanges(strEdt) ||
                        ↪  utility.isNumberPickerChanges(floNumPick))
                ).observe(this@AddKitActivity, {
                    binding.addKitSubmit.isEnabled = it
                })

                Log.i(
                    TAG,
                    "checkEmpty: ${utility.isEmpties(editTexts)} ${currWidth > 0f}
                    ↪  ${currLength > 0f} ${utility.isInRanges(hashMap)}
                    ↪  ${utility.isChanges(strEdt)}
                    ↪  ${utility.isNumberPickerChanges(floNumPick)}"
                )
            }
        } else {
            viewModel.checkNotEmpty(
                utility.isEmpties(editTexts) &&
                        currWidth > 0f &&
                        currLength > 0f &&
                        utility.isInRanges(hashMap)
            ).observe(this@AddKitActivity, {
                binding.addKitSubmit.isEnabled = it
            })
        }
    }

    private fun getNumberPickerType(view: ClickNumberPickerView) : NumberPickerType?{
        return when (view) {
            binding.addKitWidth -> NumberPickerType.KIT_WIDTH
            binding.addKitLength -> NumberPickerType.KIT_LENGTH
            binding.addKitWaterMin -> NumberPickerType.WATER_MIN
            binding.addKitWaterMax -> NumberPickerType.WATER_MAX
            binding.addKitNutrientMin -> NumberPickerType.NUTRIENT_MIN
            binding.addKitNutrientMax -> NumberPickerType.NUTRIENT_MAX
            binding.addKitTurbidityMin -> NumberPickerType.TURBIDITY_MIN
            binding.addKitTurbidityMax -> NumberPickerType.TURBIDITY_MAX
            else -> null
        }
    }
}
```

Kode 43: *Source Code* untuk *Add Kit Activity*

10. ADD NOTE ACTIVITY

```
class AddNoteActivity : AppCompatActivity(), View.OnClickListener, TextWatcher {
    private lateinit var binding : ActivityAddNoteBinding
    private lateinit var viewModel : AddNoteViewModel
    private lateinit var utility : ViewUtility
    private lateinit var editTexts : ArrayList<TextInputEditText>
    private var strEdt : HashMap<String, TextInputEditText> = hashMapOf()

    companion object {
        const val TAG = "addNoteActivity"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityAddNoteBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider(this).get(AddNoteViewModel::class.java)
        viewModel.setCurrentData(intent.getParcelableExtra<NoteModel>(BuildConfig.SELEC
        ↪  TED_NOTE))

        supportActionBar?.title =
            if(intent.getParcelableExtra<NoteModel>(BuildConfig.SELECTED_NOTE) == null)
            ↪  {
```

```
                getString(R.string.add_notes)
            } else {
                getString(R.string.edit_notes)
            }
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowHomeEnabled(false)

        binding.apply {
            editTexts = arrayListOf(addNoteTitle, addNoteDesc)
            utility = ViewUtility(
                context = this@AddNoteActivity,
                circularProgressButton = addNoteSubmit,
                textInputEditTexts = editTexts,
                actionBar = supportActionBar
            )

            editTexts.forEach {
                it.setOnClickListener(this@AddNoteActivity)
                it.addTextChangedListener(this@AddNoteActivity)
            }
            addNoteSubmit.setOnClickListener(this@AddNoteActivity)

            viewModel.getCurrentNoteModel().observe(this@AddNoteActivity, {
                if(it!=null) {
                    addNoteTitle.setText(it.title)
                    addNoteDesc.setText(it.description)
                    it.date?.let { date->
                        addNoteDate.setText(date)
                    }
                    it.time?.let{ time->
                        addNoteTime.setText(time)
                    }
                    strEdt[it.title ?: ""] = addNoteTitle
                    strEdt[it.description ?: ""] = addNoteDesc
                    strEdt[it.date ?: ""] = addNoteDate
                    strEdt[it.time ?:""] = addNoteTime
                }
            })
            checkEmpty()
        }
    }

    override fun onSupportNavigateUp(): Boolean {
        if(intent.getParcelableExtra<NoteModel>(BuildConfig.SELECTED_NOTE) != null){
            viewModel.isNotEmpties.observe(this, {
                when(it){
                    true -> binding.addNoteSubmit.performClick()
                    else -> super.onBackPressed()
                }
            })
        } else {
            super.onBackPressed()
        }
        return true
    }

    override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
        if (currentFocus != null) {
            val imm: InputMethodManager =
                getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
            imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
        }
        return super.dispatchTouchEvent(ev)
    }

    override fun onClick(v: View?) {
        when(v){
            binding.addNoteDate -> {
                val datePicker = MaterialDatePicker.Builder.datePicker()
                    .setTitleText("Date of Birth")
```

```
                    .setSelection(viewModel.getDate())
                    .setInputMode(MaterialDatePicker.INPUT_MODE_CALENDAR)
                    .build()
                datePicker.show(supportFragmentManager, "DATE_PICKER")
                datePicker.addOnPositiveButtonClickListener{
                    binding.addNoteDate.setText(viewModel.setDate(datePicker.selection))
                }
                datePicker.addOnNegativeButtonClickListener{}
                datePicker.isCancelable = false
            }
            binding.addNoteTime -> {
                viewModel.getTime()
                val timePicker = MaterialTimePicker.Builder()
                    .setTimeFormat(TimeFormat.CLOCK_24H)
                    .setHour(viewModel.getTimeHour())
                    .setMinute(viewModel.getTimeMinute())
                    .setTitleText("Select a time")
                    .build()
                timePicker.show(supportFragmentManager, "TIME_PICKER")
                timePicker.isCancelable = false

                timePicker.addOnPositiveButtonClickListener {
                    binding.addNoteTime.setText(viewModel.setTime(timePicker.hour,
                    ↪  timePicker.minute))
                }
                timePicker.addOnNegativeButtonClickListener {}
            }
            binding.addNoteSubmit -> {
                utility.isLoading = true
                viewModel.createNote(binding.addNoteTitle.text.toString(),
                    binding.addNoteDesc.text.toString())
                viewModel.isNoteAdd.observe(this, {
                    if(it){
                        utility.isLoading = false
                        val toastTxt = if(intent.getParcelableExtra<NoteModel>(BuildCon
                        ↪  fig.SELECTED_NOTE) !=
                        ↪  null){
                            "Note updated."
                        } else {
                            "New note created."
                        }
                        Toast.makeText(this, toastTxt, Toast.LENGTH_SHORT).show()
                        super.onBackPressed()
                        finish()
                    } else {
                        utility.isLoading = false
                        viewModel.addNoteError.observe(this, {
                            if(it.isNotEmpty()){
                                Toast.makeText(this, it, Toast.LENGTH_SHORT).show()
                                Log.i(CreateFarmFragment.TAG, it)
                                viewModel.addNoteError.value = ""
                            }
                        })
                    }
                })
            }
        }
    }

    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after:
    ↪  Int) {}

    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
        checkEmpty()
    }

    override fun afterTextChanged(s: Editable?) {}

    private fun checkEmpty(){
        if(intent.getParcelableExtra<NoteModel>(BuildConfig.SELECTED_NOTE) != null){
```

```
            viewModel.apply {
                checkNotEmpty(utility.isEmpties(editTexts) && utility.isChanges(strEdt))
                    .observe(this@AddNoteActivity, {
                        binding.addNoteSubmit.isEnabled = it
                    })
            }
        } else {
            viewModel.apply {
                checkNotEmpty(utility.isEmpties(editTexts)).observe(this@AddNoteActivit
                ↪ y,
                ↪ {
                    binding.addNoteSubmit.isEnabled = it
                })
            }
        }
    }
}
```

Kode 44: *Source Code* untuk *Add Note Activity*

11. ADD PLANT ACTIVITY

```
class AddPlantActivity : AppCompatActivity(), TextWatcher, View.OnClickListener {
    private lateinit var binding : ActivityAddPlantBinding
    private lateinit var viewModel : AddPlantViewModel
    private lateinit var utility: ViewUtility
    private lateinit var editTexts : ArrayList<TextInputEditText>
    private lateinit var tooltips : ArrayList<ImageButton>
    private lateinit var numberPickers : ArrayList<ClickNumberPickerView>
    private lateinit var viewsAsButton : ArrayList<View>
    private var strEdt : HashMap<String, TextInputEditText> = hashMapOf()

    companion object{
        const val TAG = "addPlantActivity"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityAddPlantBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider(this).get(AddPlantViewModel::class.java)
        viewModel.setCurrentData(intent.getParcelableExtra<PlantModel>(BuildConfig.SELE
        ↪ CTED_PLANT))

        supportActionBar?.title =
        ↪ if(intent.getParcelableExtra<PlantModel>(BuildConfig.SELECTED_PLANT) ==
        ↪ null) {
            getString(R.string.add_new_plant)
        } else {
            getString(R.string.edit_plant)
        }
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowHomeEnabled(false)

        binding.apply {
            tooltips = arrayListOf(apGrowthInfo,
                                   apTempInfo,
                                   apHumidInfo)
            viewsAsButton = arrayListOf(addPlantPhoto, addPlantEditPhoto,
            ↪ addPlantSubmit)
            editTexts = arrayListOf(addPlantName, addPlantDescription)
            numberPickers = arrayListOf(addPlantGrowthTime,
                addPlantTempMin,
                addPlantTempMax,
                addPlantHumidMin,
                addPlantHumidMax,
                addPlantAcidMin,
```

```
            addPlantAcidMax)
        utility = ViewUtility(
            context = this@AddPlantActivity,
            circularProgressButton = addPlantSubmit,
            textInputEditTexts = editTexts,
            viewsAsButton = viewsAsButton,
            numberPickers = numberPickers,
            actionBar = supportActionBar)

        tooltips.forEach {
            it.setOnClickListener {
                it.performLongClick()
            }
        }
        //todo : tooltip

        viewsAsButton.forEach { it.setOnClickListener(this@AddPlantActivity) }
        addPlantPhoto.setOnLongClickListener { addPlantEditPhoto.performClick() }
        editTexts.forEach { it.addTextChangedListener(this@AddPlantActivity) }
        numberPickers.forEachIndexed { i, view ->
            val numberPickerDataType = getNumberPickerType(view)
            viewModel.getNumberPickerValue(numberPickerDataType)?.observe(this@AddP
            ↪ lantActivity,{
                if(it != view.value){
                    view.setPickerValue(it)
                }
            })

            numberPickers[i].setClickNumberPickerListener { previousValue,
            ↪ currentValue, pickerClickType ->
                viewModel.setNumberPickerValue(currentValue, numberPickerDataType)
                checkEmpty()
            }
        }

        viewModel.getPhotoProfile().observe(this@AddPlantActivity, {
            it?.let {
                Glide.with(this@AddPlantActivity)
                    .load(it.uri)
                    .circleCrop()
                    .into(addPlantPhoto)
            }
        })

        viewModel.getCurrentPlant().observe(this@AddPlantActivity, { it ->
            it.plantId?.let { plantId ->
                supportActionBar?.title = it.name
                addPlantName.setText(it.name)
                addPlantDescription.setText(it.description)
                addPlantGrowthTime.setPickerValue(it.growthTime?.toFloat() ?: 0f)
                addPlantTempMin.setPickerValue(it.tempLv?.min ?: 0f)
                addPlantTempMax.setPickerValue(it.tempLv?.max ?: 0f)
                addPlantHumidMin.setPickerValue(it.humidLv?.min ?: 0f)
                addPlantHumidMax.setPickerValue(it.humidLv?.max ?: 0f)
                addPlantAcidMin.setPickerValue(it.phLv?.min ?: 0f)
                addPlantAcidMax.setPickerValue(it.phLv?.max ?: 0f)

                Glide.with(this@AddPlantActivity)
                    .load(it.photo_url ?: R.drawable.bg_plant)
                    .circleCrop()
                    .into(addPlantPhoto)

                if (it.userId != Firebase.auth.uid){
                    addPlantSubmit.visibility = View.GONE
                    utility.isLoading = true
                    supportActionBar?.setDisplayShowHomeEnabled(false)
                }
                Log.i(TAG, "onCreate: ${it.userId} ${Firebase.auth.uid}")

                strEdt[it.name ?: ""] = addPlantName
```

```
                    strEdt[it.description ?: ""] = addPlantDescription
                }
            })

            checkEmpty()
        }
    }

    override fun onSupportNavigateUp(): Boolean {
        if(intent.getParcelableExtra<PlantModel>(BuildConfig.SELECTED_PLANT) != null &&
                viewModel.getCurrentPlant().value?.userId == Firebase.auth.uid){
            viewModel.isNotEmpties.observe(this, {
                when(it){
                    true -> binding.addPlantSubmit.performClick()
                    else -> super.onBackPressed()
                }
            })
        } else {
            super.onBackPressed()
        }
        return true
    }

    override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
        if (currentFocus != null) {
            val imm: InputMethodManager =
                getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
            imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
        }
        return super.dispatchTouchEvent(ev)
    }

    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after:
    ↪  Int) {}

    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
        checkEmpty()
    }

    override fun afterTextChanged(s: Editable?) {}

    private fun checkEmpty(){
        val currGrowthTime : Float = viewModel.getNumberPickerValue(getNumberPickerType
        ↪  (binding.addPlantGrowthTime))?.value ?:
        ↪  0f
        val currTempMin : Float? =  viewModel.getNumberPickerValue(getNumberPickerType(
        ↪  binding.addPlantTempMin))?.value
        val currTempMax : Float? =  viewModel.getNumberPickerValue(getNumberPickerType(
        ↪  binding.addPlantTempMax))?.value
        val currHumidMin : Float? =  viewModel.getNumberPickerValue(getNumberPickerType
        ↪  (binding.addPlantHumidMin))?.value
        val currHumidMax : Float? =  viewModel.getNumberPickerValue(getNumberPickerType
        ↪  (binding.addPlantHumidMax))?.value
        val currAcidMin : Float? = viewModel.getNumberPickerValue(getNumberPickerType(b
        ↪  inding.addPlantAcidMin))?.value
        val currAcidMax : Float? = viewModel.getNumberPickerValue(getNumberPickerType(b
        ↪  inding.addPlantAcidMax))?.value
        val hashMap : HashMap<Float?, Float?> = hashMapOf()
        hashMap[currTempMin] = currTempMax
        hashMap[currHumidMin] = currHumidMax
        hashMap[currAcidMin] = currAcidMax

        if(intent.getParcelableExtra<PlantModel>(BuildConfig.SELECTED_PLANT) != null){
            viewModel.getCurrentPlant().value?.let {
                val floNumPick: HashMap<Float, Float> = hashMapOf()
                floNumPick[it.growthTime?.toFloat() ?: 0f] = currGrowthTime
                floNumPick[it.tempLv?.min ?: 0f] = currTempMin ?: 0f
                floNumPick[it.tempLv?.max ?: 0f] = currTempMax ?: 0f
                floNumPick[it.humidLv?.min ?: 0f] = currHumidMin ?: 0f
                floNumPick[it.humidLv?.max ?: 0f] = currHumidMax ?: 0f
```

```
                floNumPick[it.phLv?.min ?: 0f] = currAcidMin ?: 0f
                floNumPick[it.phLv?.max ?: 0f] = currAcidMax ?: 0f

                viewModel.checkNotEmpty(utility.isEmpties(editTexts) &&
                            currGrowthTime > 0f &&
                            utility.isInRanges(hashMap) &&
                            (utility.isChanges(strEdt) ||
                            ↪  utility.isNumberPickerChanges(floNumPick))
                ).observe(this@AddPlantActivity, {
                    binding.addPlantSubmit.isEnabled = it
                })
                Log.i(TAG, "checkEmpty: ${utility.isEmpties(editTexts)}
                ↪  ${currGrowthTime > 0f} ${utility.isInRanges(hashMap)}" +
                        " ${utility.isChanges(strEdt)}
                        ↪  ${utility.isNumberPickerChanges(floNumPick)}")
            }
        } else {
            viewModel.checkNotEmpty(
                utility.isEmpties(editTexts) &&
                        currGrowthTime > 0f &&
                        utility.isInRanges(hashMap)
            ).observe(this, {
                binding.addPlantSubmit.isEnabled = it
            })
        }
    }

    private fun getNumberPickerType(view: ClickNumberPickerView) : NumberPickerType?{
        return when (view) {
            binding.addPlantGrowthTime -> NumberPickerType.GROWTH_TIME
            binding.addPlantTempMin -> NumberPickerType.TEMP_MIN
            binding.addPlantTempMax -> NumberPickerType.TEMP_MAX
            binding.addPlantHumidMin -> NumberPickerType.HUMID_MIN
            binding.addPlantHumidMax -> NumberPickerType.HUMID_MAX
            binding.addPlantAcidMin -> NumberPickerType.ACID_MIN
            binding.addPlantAcidMax -> NumberPickerType.ACID_MAX
            else -> null
        }
    }

    override fun onClick(v: View?) {
        when(v){
            binding.addPlantPhoto -> utility.openImage(binding.addPlantPhoto)
            binding.addPlantEditPhoto -> utility.openEditPhoto(imageView =
            ↪  binding.addPlantPhoto, profileType = ProfileType.PLANT)
            binding.addPlantSubmit -> {
                utility.isLoading = true
                viewModel.createPlant(binding.addPlantName.text.toString(),
                    binding.addPlantDescription.text.toString())
                viewModel.isPlantAdd.observe(this@AddPlantActivity, {
                    if(it){
                        utility.isLoading = false
                        if(intent.getStringExtra("from") != null){
                            val intent = Intent()
                            intent.putExtra(BuildConfig.SELECTED_PLANT,
                            ↪  viewModel.plantModel.value)
                            setResult(RESULT_OK, intent)
                        } else {
                            val toastTxt = if(intent.getParcelableExtra<PlantModel>(Bui
                            ↪  ldConfig.SELECTED_PLANT) !=
                            ↪  null){
                                "Plant updated."
                            } else {
                                "New plant added."
                            }
                            Toast.makeText(this, toastTxt, Toast.LENGTH_SHORT).show()
                            super.onBackPressed()
                            finish()
                        }
                    } else {
```

```
                        utility.isLoading = false
                        viewModel.addPlantError.observe(this@AddPlantActivity, {
                            if(it.isNotEmpty()){
                                Toast.makeText(this@AddPlantActivity, it,
                                ↪  Toast.LENGTH_SHORT).show()
                                Log.i(CreateFarmFragment.TAG, it)
                                viewModel.addPlantError.value = ""
                            }
                        })
                    }
                })
            }
        }
    }

    val startForResult =
    ↪  registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
    ↪  data ->
        Log.i(CreateProfileActivity.TAG, "$data")
        try{
            if(data?.resultCode == Activity.RESULT_OK){
                CropImage.getActivityResult(data.data)?.let{
                    val mimeTypeMap = MimeTypeMap.getSingleton()
                    val fileExtension = mimeTypeMap.getExtensionFromMimeType(contentRes
                    ↪  olver.getType(it.uriContent!!))
                    viewModel.setPhotoPlant(it.uriContent!!, fileExtension)
                }
            }

        } catch (e: Exception){
            Toast.makeText(this, e.message.toString(), Toast.LENGTH_SHORT).show()
        }
    }
}
```

Kode 45: *Source Code* untuk *Add Plant Activity*

12. PROFILE KIT ACTIVITY

```
class ProfileKitActivity : AppCompatActivity() {
    private lateinit var binding : ActivityProfileKitBinding
    lateinit var viewModel : ProfileKitViewModel

    companion object {
        const val TAG = "kitProfileActivity"
        val TAB_LAYOUT = arrayListOf<String>("Overview", "Data Monitoring", "Crops")
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityProfileKitBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider(this).get(ProfileKitViewModel::class.java)
        viewModel.setCurrentKit(intent.getParcelableExtra<UserModel>(BuildConfig.CURREN
        ↪  T_USER),
            intent.getParcelableExtra<FarmModel>(BuildConfig.CURRENT_FARM),
            intent.getParcelableExtra<KitModel>(BuildConfig.SELECTED_KIT))
        supportActionBar?.title = getString(R.string.hydroponic_kit)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowHomeEnabled(false)
        supportActionBar?.elevation = 0f

        val tabLayoutAdapter = object : FragmentStateAdapter(supportFragmentManager,
        ↪  lifecycle){
            override fun getItemCount(): Int {
                return TAB_LAYOUT.size
            }
```

```
28
29              override fun createFragment(position: Int): Fragment {
30                  return when(position){
31                      1 -> KitMonitoringFragment()
32                      2 -> KitCropsFragment()
33                      else -> KitOverviewFragment()
34                  }
35              }
36          }
37
38          binding.profileKitViewPager.adapter = tabLayoutAdapter
39          TabLayoutMediator(binding.profileKitTabLayout, binding.profileKitViewPager) {
            ↪  tab, position ->
40              tab.text = TAB_LAYOUT[position]
41          }.attach()
42      }
43
44      override fun onSupportNavigateUp(): Boolean {
45          super.onBackPressed()
46          finish()
47          return true
48      }
49
50      override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
51          if (currentFocus != null) {
52              val imm: InputMethodManager =
53                  getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
54              imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
55          }
56          return super.dispatchTouchEvent(ev)
57      }
58
59      override fun onCreateOptionsMenu(menu: Menu?): Boolean {
60          menuInflater.inflate(R.menu.menu_profile_kit, menu)
61          return super.onCreateOptionsMenu(menu)
62      }
63
64      override fun onOptionsItemSelected(item: MenuItem): Boolean {
65          when(item.itemId){
66              R.id.kitSettings -> {
67                  val intent = Intent(this, AddKitActivity::class.java)
68                  intent.putExtra(BuildConfig.CURRENT_USER,
                    ↪  viewModel.getCurrentUser().value)
69                  intent.putExtra(BuildConfig.CURRENT_FARM,
                    ↪  viewModel.getCurrentFarm().value)
70                  intent.putExtra(BuildConfig.SELECTED_KIT,
                    ↪  viewModel.getCurrentKit().value)
71                  startActivity(intent)
72              }
73              R.id.kitHistory -> startActivity(Intent(this,
                ↪  ShowRecyclerActivity::class.java))
74          }
75          return super.onOptionsItemSelected(item)
76      }
77  }
```

Kode 46: *Source Code* untuk *Profile Kit Activity* beserta *fragments* terkait

a) KIT OVERVIEW FRAGMENT

```
1   class KitOverviewFragment : Fragment() {
2       private var _binding : FragmentKitOverviewBinding? = null
3       private val binding get() = _binding!!
4       private lateinit var bindingCondition : RowItemKitBinding
5       private lateinit var bindingPlanted : LayoutPlantedCropsBinding
6       private lateinit var viewModel : ProfileKitViewModel
7
8       companion object {
9           const val TAG = "kitOverViewFragment"
```

```
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentKitOverviewBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        viewModel = (requireActivity() as ProfileKitActivity).viewModel
        bindingCondition = binding.kitCondition
        bindingPlanted = binding.kitPlantedCrops
        bindingCondition.apply {
            kitName.visibility = View.GONE
            kitShowDetail.visibility = View.GONE
        }

        binding.apply {
            bindingCondition.apply {
                bindingPlanted.apply {
                    viewModel.getCurrentKit().observe(viewLifecycleOwner, { kit ->
                        kit?.let {
                            (requireActivity() as
                            ↪  ProfileKitActivity).supportActionBar?.title =
                            ↪  kit.name.toString()
                            if(it.isPlanted == true){
                                kitPlantedRoot.visibility = View.GONE
                            } else {
                                kitPlantedRoot.visibility = View.GONE
                            }
                        }
                    })

                    viewModel.getCurrentMonitoring().observe(viewLifecycleOwner, {
                    ↪  monitorings ->
                        monitorings?.let {
                            if(monitorings.size > 0)
                            {
                                monitorings[0].apply {
                                    kitLastMonitoring.visibility = View.VISIBLE
                                    kitTimestamp.visibility = View.VISIBLE
                                    kitTimestamp.text =
                                    ↪  getString(R.string.updated_on_s, timestamp)
                                    kitTemp.text = temperature.toString()
                                    kitHumid.text = humidity.toString()
                                    kitAcidity.text = ph.toString()
                                    kitWaterTank.text = waterTank.toString()
                                    kitNutrientTank.text = nutrientTank.toString()
                                    kitTurbidity.text = turbidity.toString()
                                }
                            }
                        }
                    })

                    viewModel.getCurrentCrops().observe(viewLifecycleOwner, { crops ->
                        crops?.let{
                            if(crops.size > 0){
                                crops[0].apply {
                                    plantModel?.let {
                                        kitPlantTitle.text = it.name.toString()
                                        kitPlantDate.text = this.timestamp.toString()

                                    }
                                }
                            }
                        } ?: kotlin.run {
                            kitPlantedRoot.visibility = View.GONE
```

```
                        }
                    })
                }
            }
        }
    }
}
```

Kode 47: *Source Code* untuk *KitOverview Fragment*

b) KIT MONITORING FRAGMENT

```
class KitMonitoringFragment : Fragment() {
    private var _binding : FragmentKitMonitoringBinding? = null
    private val binding get() = _binding!!
    private lateinit var viewModel : ProfileKitViewModel

    companion object {
        const val TAG = "kitMonitoringFragment"
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentKitMonitoringBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        viewModel = (requireActivity() as ProfileKitActivity).viewModel

        viewModel.getCurrentMonitoring().observe(viewLifecycleOwner, {
            val tempList : ArrayList<Entry> = arrayListOf()
            val humidList : ArrayList<Entry> = arrayListOf()
            val acidList : ArrayList<Entry> = arrayListOf()
            val waterList : ArrayList<Entry> = arrayListOf()
            val nutrientList : ArrayList<Entry> = arrayListOf()
            val turbidityList : ArrayList<Entry> = arrayListOf()

            it?.forEachIndexed { index, dataMonitoringModel ->
                tempList.add(Entry(index.toFloat(), dataMonitoringModel.temperature ?:
                ↪ 0f))
                humidList.add(Entry(index.toFloat(), dataMonitoringModel.humidity ?:
                ↪ 0f))
                acidList.add(Entry(index.toFloat(), dataMonitoringModel.ph ?: 0f))
                waterList.add(Entry(index.toFloat(), dataMonitoringModel.waterTank ?:
                ↪ 0f))
                nutrientList.add(Entry(index.toFloat(),
                ↪ dataMonitoringModel.nutrientTank ?: 0f))
                turbidityList.add(Entry(index.toFloat(), dataMonitoringModel.turbidity
                ↪ ?: 0f))
            }

            binding.apply {
                kitCropsTemp.data = LineData(LineDataSet(tempList,
                ↪ getString(R.string.temperature_only)))
                kitCropsHumid.data = LineData(LineDataSet(humidList,
                ↪ getString(R.string.humidity_only)))
                kitCropsAcid.data = LineData(LineDataSet(tempList,
                ↪ getString(R.string.acidity_only)))
                kitCropsWaterTank.data = LineData(LineDataSet(tempList,
                ↪ getString(R.string.water_level_only)))
                kitCropsNutrientTank.data = LineData(LineDataSet(tempList,
                ↪ getString(R.string.nutrient_level_only)))
                kitCropsTurbidity.data = LineData(LineDataSet(tempList,
                ↪ getString(R.string.turbidity_only)))

```

```
                kitCropsTemp.invalidate()
                kitCropsHumid.invalidate()
                kitCropsAcid.invalidate()
                kitCropsWaterTank.invalidate()
                kitCropsNutrientTank.invalidate()
                kitCropsTurbidity.invalidate()
            }
        })
    }
}
```

Kode 48: *Source Code* untuk *KitMonitoring Fragment*

c) KIT CROPS FRAGMENT

```
class KitCropsFragment : Fragment() {
    private var _binding : FragmentKitCropsBinding? = null
    private val binding get() = _binding!!
    private lateinit var viewModel : ProfileKitViewModel

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentKitCropsBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        viewModel = (requireActivity() as ProfileKitActivity).viewModel

        val data : ArrayList<CropsModel> = arrayListOf()
        val rowAdapter = AdapterType.PROFILE_CROPS.getAdapter(requireContext(), data)
        viewModel.getCurrentCrops().observe(viewLifecycleOwner, {
            data.clear()
            data.addAll(it ?: arrayListOf())
            rowAdapter.notifyDataSetChanged()
        })

        binding.kitCropsRecyclerView.apply {
            adapter = rowAdapter
            layoutManager = LinearLayoutManager(requireContext())
        }
    }
}
```

Kode 49: *Source Code* untuk *Kit Crops Fragment*

13. PROFILE USER ACTIVITY

```
class ProfileUserActivity : AppCompatActivity() {
    private lateinit var binding : ActivityProfileUserBinding
    private lateinit var viewModel : ProfileUserViewModel

    companion object {
        const val TAG = "profileUserActivity"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityProfileUserBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider(this).get(ProfileUserViewModel::class.java)
        viewModel.setUserModel(intent.getParcelableExtra<UserModel>(BuildConfig.SELECTE
        ↪ D_USER))
```

```
16          supportActionBar?.setDisplayHomeAsUpEnabled(true)
17          supportActionBar?.setDisplayShowHomeEnabled(false)
18          supportActionBar?.elevation= 0f
19
20          binding.apply {
21              viewModel.getUserModel().observe(this@ProfileUserActivity, {
22                  it?.let {
23                      supportActionBar?.title = it.name
24                      profileUserName.text = it.name
25                      it.bio?.let{
26                          profileUserBio.text = it
27                      } ?: kotlin.run {
28                          profileUserBio.visibility = View.GONE
29                      }
30                      it.photo_url?.let {
31                          Glide.with(this@ProfileUserActivity)
32                              .load(it)
33                              .circleCrop()
34                              .into(profileUserPhoto)
35                      }
36                  }
37              })
38
39              profileUserPhoto.setOnClickListener {
40                  IntentUtility(this@ProfileUserActivity).openImage(profileUserPhoto,
                    ↪  viewModel.getUserModel().value?.name ?: "Photo Profile")
41              }
42          }
43          setupRecyclerView()
44      }
45
46      private fun setupRecyclerView() {
47          val userModel : ArrayList<UserModel> = arrayListOf()
48          val plantModels : ArrayList<PlantModel> = arrayListOf()
49          val rowAdapter = AdapterType.POST_PLANT.getAdapter(this, plantModels, allUsers
            ↪  = userModel)
50          viewModel.getUserPosts().observe(this,{
51              plantModels.clear()
52              plantModels.addAll(it ?: arrayListOf())
53              rowAdapter.notifyDataSetChanged()
54          })
55
56          viewModel.getUserModel().observe(this, {
57              userModel.clear()
58              userModel.addAll(arrayListOf(it))
59              rowAdapter.notifyDataSetChanged()
60          })
61
62          binding.profileUserRecyclerView.apply {
63              adapter = rowAdapter
64              layoutManager = LinearLayoutManager(this@ProfileUserActivity)
65              addItemDecoration(object : DividerItemDecoration(this@ProfileUserActivity,
                ↪  VERTICAL) {})
66              setHasFixedSize(true)
67          }
68      }
69
70      override fun onSupportNavigateUp(): Boolean {
71          super.onBackPressed()
72          finish()
73          return true
74      }
75
76      override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
77          if (currentFocus != null) {
78              val imm: InputMethodManager =
79                  getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
80              imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
81          }
82          return super.dispatchTouchEvent(ev)
```

```
    }
}
```

Kode 50: *Source Code* untuk *Profile User Activity*

14. EDIT PROFILE FARM ACTIVITY

```
class EditProfileFarmActivity : AppCompatActivity(), View.OnClickListener, TextWatcher {
    private lateinit var binding : ActivityEditProfileFarmBinding
    private lateinit var bindingFragment : FragmentCreateFarmBinding
    private lateinit var viewModel : CreateProfileViewModel
    private lateinit var viewsAsButton : ArrayList<View>
    private lateinit var utility: ViewUtility
    private lateinit var editTexts : java.util.ArrayList<TextInputEditText>
    private var strEdt : HashMap<String, TextInputEditText> = hashMapOf()

    companion object {
        const val TAG = "editProfileFarm"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityEditProfileFarmBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider(this).get(CreateProfileViewModel::class.java)
        viewModel.setCurrentFarm(intent.getParcelableExtra<FarmModel>(BuildConfig.CURRE
        ↪ NT_FARM),
                                 intent.getParcelableExtra<UserModel>(BuildConfig.CURRE
                                 ↪ NT_USER))
        supportActionBar?.title = getString(R.string.farm)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowHomeEnabled(false)

        bindingFragment = binding.editFarmFragment
        bindingFragment.apply {
            setupDefaultData()
            viewsAsButton = arrayListOf(esAddStaff, createFarmSubmit)
            editTexts = arrayListOf(
                createFarmName,
                createFarmLoc,
                createFarmDesc
            )
            utility = ViewUtility(
                context = this@EditProfileFarmActivity,
                circularProgressButton = createFarmSubmit,
                textInputEditTexts = editTexts,
                viewsAsButton = viewsAsButton,
                actionBar = supportActionBar
            )

            viewsAsButton.forEach { it.setOnClickListener(this@EditProfileFarmActivity)
            ↪ }
            editTexts.forEach { it.addTextChangedListener(this@EditProfileFarmActivity)
            ↪ }
            setupRecyclerView()
            checkUpdate()
        }

        val mapFragment = supportFragmentManager.findFragmentById(R.id.map) as
        ↪ SupportMapFragment?
        mapFragment?.getMapAsync(callback)
    }

    private fun setupDefaultData() {
        bindingFragment.apply {
            createFarmSubmit.text = getString(R.string.save)
            editStaff.visibility = View.VISIBLE
```

```
57
58              viewModel.getCurrentFarmModel()?.let {
59                  createFarmName.setText(it.name ?: "")
60                  createFarmDesc.setText(it.description ?: "")
61                  createFarmLoc.setText(it.location ?: "")
62
63                  strEdt[it.name ?: ""] = createFarmName
64                  strEdt[it.description ?: ""] = createFarmDesc
65                  strEdt[it.location ?: ""] = createFarmLoc
66              }
67          }
68      }
69
70      private fun setupRecyclerView(){
71          val data : ArrayList<UserModel> = arrayListOf()
72          val rowAdapter = AdapterType.SEARCH_USER.getAdapter(this, data, type =
            ↪  AdapterType.Companion.SearchSelectType.SELECT)
73          viewModel.updateStaff()
74          viewModel.getStaff().observe(this, {
75              data.clear()
76              data.addAll(it ?: arrayListOf())
77              rowAdapter.notifyDataSetChanged()
78              (rowAdapter as UserModelAdapter).setOnItemClickListener(
79                  object : UserModelAdapter.OnItemClickListener{
80                      override fun onItemClicked(
81                          userModel: UserModel,
82                          position: Int,
83                          itemView: View,
84                          v: RowItemSearchBinding
85                      ) {
86                          when(itemView){
87                              v.searchRoot -> {
88                                  val intent = Intent(this@EditProfileFarmActivity,
                                    ↪  ProfileUserActivity::class.java)
89                                  intent.putExtra(BuildConfig.SELECTED_USER, userModel)
90                                  startActivity(intent)
91                              }
92                              v.searchClose -> viewModel.removeStaff(position)
93                          }
94                      }
95
96                  }
97              )
98          })
99          bindingFragment.esRecyclerView.adapter = rowAdapter
100         bindingFragment.esRecyclerView.layoutManager = LinearLayoutManager(this)
101         bindingFragment.esRecyclerView.addItemDecoration(object :
            ↪  DividerItemDecoration(this, VERTICAL) {})
102         bindingFragment.esRecyclerView.setHasFixedSize(true)
103     }
104
105     private fun checkUpdate(){
106         viewModel.checkNotEmpty(
107             utility.isChanges(strEdt) && utility.isEmpties(editTexts)
108         ).observe(this, {
109             bindingFragment.createFarmSubmit.isEnabled = it
110         })
111     }
112
113     override fun onSupportNavigateUp(): Boolean {
114         viewModel.isNotEmpties.observe(this, {
115             when(it){
116                 true -> bindingFragment.createFarmSubmit.performClick()
117                 else -> super.onBackPressed()
118             }
119         })
120         return true
121     }
122
123     override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
```

```
124             if (currentFocus != null) {
125                 val imm: InputMethodManager =
126                     getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
127                 imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
128             }
129             return super.dispatchTouchEvent(ev)
130         }
131 
132         override fun onClick(v: View?) {
133             when(v){
134                 bindingFragment.esAddStaff -> {
135                     val intent = Intent(this, SearchActivity::class.java)
136                     intent.putExtra("search", TAG)
137                     intent.putExtra(BuildConfig.EXCEPT_USERS, viewModel.getStaff().value)
138                     startForResult.launch(intent)
139                 }
140                 bindingFragment.createFarmSubmit -> {
141                     utility.isLoading = true
142                     viewModel.createFarmProfile(bindingFragment.createFarmName.text.toStrin
                        ↪ g(),
143                         bindingFragment.createFarmDesc.text.toString(),
144                         bindingFragment.createFarmLoc.text.toString())
145                     viewModel.isFarmCreated.observe(this, {
146                         if(it){
147                             utility.isLoading = false
148                             Toast.makeText(this, "Farm updated.", Toast.LENGTH_SHORT).show()
149                             super.onBackPressed()
150                         } else {
151                             utility.isLoading = false
152                             viewModel.createProfileError.observe(this, {
153                                 if(it.isNotEmpty()){
154                                     Toast.makeText(this, it, Toast.LENGTH_SHORT).show()
155                                     Log.i(CreateFarmFragment.TAG, it)
156                                     viewModel.createProfileError.value = ""
157                                 }
158                             })
159                         }
160                     })
161 //                  viewModel.createStaff()
162 //                  viewModel.isStaffAdded.observe(this, {
163 //                      if(it){
164 //                          utility.isLoading = false
165 //                          Toast.makeText(this, "Farm updated", Toast.LENGTH_SHORT).show()
166 //                          super.onBackPressed()
167 //                          finish()
168 //                      }
169 //                  })
170                 }
171             }
172         }
173 
174         private val callback = OnMapReadyCallback { googleMap ->
175             val sydney = LatLng(-34.0, 151.0)
176             googleMap.addMarker(MarkerOptions().position(sydney).title("Marker in Sydney"))
177             googleMap.moveCamera(CameraUpdateFactory.newLatLng(sydney))
178             //todo posisi awal sesuai dari data starter
179         }
180 
181         private val startForResult =
            ↪ registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
            ↪ data ->
182             Log.i(CreateProfileActivity.TAG, "$data")
183             try{
184                 if(data?.resultCode == Activity.RESULT_OK){
185                     data.data?.getParcelableExtra<UserModel>(BuildConfig.SELECTED_USER)?.le
                        ↪ t
                        ↪ {
186                         viewModel.addStaff(it)
187                         checkUpdate()
188                     }
```

```
            }

        } catch (e: Exception){
            Toast.makeText(this, e.message.toString(), Toast.LENGTH_SHORT).show()
        }
    }

    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after:
    ↪  Int) {}

    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
        checkUpdate()
    }

    override fun afterTextChanged(s: Editable?) {}
}
```

Kode 51: *Source Code* untuk *Edit Profile Farm Activity*

15. EDIT PROFILE USER ACTIVITY

```
class EditProfileUserActivity : AppCompatActivity(), View.OnClickListener,
↪  SegmentedButtonGroup.OnPositionChangedListener, TextWatcher {
    private lateinit var binding : ActivityEditProfileUserBinding
    private lateinit var bindingFragment : FragmentCreateUserBinding
    private lateinit var viewModel : CreateProfileViewModel
    private lateinit var utility: ViewUtility
    private lateinit var viewsAsButton : ArrayList<View>
    private lateinit var editTexts : ArrayList<TextInputEditText>
    private var strEdt : HashMap<String, TextInputEditText> = hashMapOf()

    companion object {
        const val TAG = "editProfileUserActivity"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityEditProfileUserBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider(this).get(CreateProfileViewModel::class.java)
        viewModel.setCurrentUser(intent.getParcelableExtra<UserModel>(BuildConfig.CURRE
        ↪  NT_USER))
        supportActionBar?.title = getString(R.string.edit_profile)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowHomeEnabled(false)

        bindingFragment = binding.editProfileFragment
        bindingFragment.apply {
            setupDefaultData()
            editTexts = arrayListOf(
                createUserName,
                createUserPhone,
                createUserAddress,
                createUserDOB,
                createUserBio)
            viewsAsButton = arrayListOf(createUserDOB,
                                        createUserSubmit,
                                        createUserEditPhoto,
                                        createUserPhoto)
            utility = ViewUtility(
                context = this@EditProfileUserActivity,
                circularProgressButton = createUserSubmit,
                textInputEditTexts = editTexts,
                viewsAsButton = viewsAsButton,
                actionBar = supportActionBar
            )

```

```
            createUserGender.onPositionChangedListener = this@EditProfileUserActivity
            editTexts.forEach { it.addTextChangedListener(this@EditProfileUserActivity) }
            viewsAsButton.forEach { it.setOnClickListener(this@EditProfileUserActivity) }
            createUserPhoto.setOnLongClickListener { createUserEditPhoto.performClick() }
            checkUpdate()
        }

        viewModel.getPhotoProfile().observe(this, {
            it?.let {
                Glide.with(this)
                    .load(it.uri)
                    .circleCrop()
                    .into(bindingFragment.createUserPhoto)
            }
        })
    }

    override fun onSupportNavigateUp(): Boolean {
        viewModel.isNotEmpties.observe(this, {
            when(it){
                true -> bindingFragment.createUserSubmit.performClick()
                else -> super.onBackPressed()
            }
        })
        return true
    }

    override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
        if (currentFocus != null) {
            val imm: InputMethodManager =
                getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
            imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
        }
        return super.dispatchTouchEvent(ev)
    }

    override fun onClick(v: View?) {
        when(v){
            bindingFragment.createUserEditPhoto ->
                utility.openEditPhoto(bindingFragment.createUserPhoto, ProfileType.USER)
            bindingFragment.createUserPhoto ->
                utility.openImage(bindingFragment.createUserPhoto)
            bindingFragment.createUserDOB -> {
                val calendar = Calendar.getInstance(TimeZone.getTimeZone("WIB"))
                calendar[Calendar.MONTH] = Calendar.DECEMBER
                val constraintBuilder = CalendarConstraints.Builder()
                constraintBuilder.setValidator(DateValidatorPointBackward.now()).setEnd
                    (calendar.timeInMillis)

                val datePicker = MaterialDatePicker.Builder.datePicker()
                    .setTitleText("Date of Birth")
                    .setSelection(viewModel.getDOB())
                    .setInputMode(MaterialDatePicker.INPUT_MODE_CALENDAR)
                    .setCalendarConstraints(constraintBuilder.build())
                    .build()
                datePicker.show(supportFragmentManager, "DATE_PICKER")
                datePicker.addOnPositiveButtonClickListener{
                    bindingFragment.createUserDOB.setText(viewModel.setDOB(datePicker.s
                        election))
                }
                datePicker.addOnNegativeButtonClickListener{}
                datePicker.isCancelable = false
            }
            bindingFragment.createUserSubmit -> {
                utility.isLoading = true
                viewModel.createUserProfile(
                    bindingFragment.createUserName.text.toString(),
```

```
109                     bindingFragment.createUserPhone.text.toString(),
110                     bindingFragment.createUserAddress.text.toString(),
111                     bindingFragment.createUserBio.text.toString()
112                 )
113                 viewModel.isUserCreated.observe(this, {
114                     if(it){
115                         utility.isLoading = false
116                         Toast.makeText(this, "Profile updated.",
                            ↪  Toast.LENGTH_SHORT).show()
117                         super.onBackPressed()
118                         finish()
119                     } else {
120                         viewModel.createProfileError.observe(this, {
121                             if(it.isNotEmpty()){
122                                 Toast.makeText(this, it, Toast.LENGTH_SHORT).show()
123                                 viewModel.createProfileError.value = ""
124                             }
125                         })
126                     }
127                 })
128 
129             }
130         }
131     }
132 
133     override fun onPositionChanged(position: Int) {
134         when(position){
135             0 -> viewModel.setGender(Gender.MALE)
136             1 -> viewModel.setGender(Gender.FEMALE)
137         }
138         checkUpdate()
139     }
140 
141     private fun setupDefaultData(){
142         bindingFragment.apply {
143             createUserRoleGroup.visibility = View.GONE
144             createUserSubmit.text = getString(R.string.save)
145 
146             viewModel.getCurrentUserModel()?.let {
147                 createUserEmail.setText(Firebase.auth.currentUser?.email)
148                 createUserName.setText(it.name ?: "")
149                 createUserPhone.setText(it.phone ?: "")
150                 createUserAddress.setText(it.address ?: "")
151                 createUserBio.setText(it.bio ?: "")
152                 createUserDOB.setText(it.dob ?: "")
153                 Glide.with(this@EditProfileUserActivity)
154                     .load(it.photo_url ?: R.drawable.bg_farmer)
155                     .centerCrop()
156                     .into(createUserPhoto)
157                 createUserGender.setPosition(Gender.getType(it.gender!!)!!.getPosition(
                    ↪  ),
                    ↪  false)
158 
159                 strEdt[it.name ?: ""] = createUserName
160                 strEdt[it.phone ?: ""] = createUserPhone
161                 strEdt[it.address ?: ""] = createUserAddress
162                 strEdt[it.bio ?: ""] = createUserBio
163                 strEdt[it.dob ?: ""] = createUserDOB
164             }
165         }
166     }
167 
168     override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after:
        ↪  Int) {}
169 
170     override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
171         checkUpdate()
172     }
173 
174     override fun afterTextChanged(s: Editable?) {}
```

```
175
176     private fun checkUpdate() {
177         viewModel.checkNotEmpty(
178             utility.isEmpties(editTexts) &&
179                     (utility.isChanges(strEdt) ||
180                     viewModel.getUpdateGender() !=
                        ↪  bindingFragment.createUserGender.position ||
181                     viewModel.getPhotoProfile().value != null)
182         ).observe(this, {
183             bindingFragment.createUserSubmit.isEnabled = it
184         })
185     }
186
187     val startForResult =
        ↪  registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
        ↪  data ->
188         Log.i(TAG, "$data")
189         try{
190             if(data?.resultCode == Activity.RESULT_OK){
191                 CropImage.getActivityResult(data.data)?.let{
192                     val mimeTypeMap = MimeTypeMap.getSingleton()
193                     val fileExtension = mimeTypeMap.getExtensionFromMimeType(contentRes
                        ↪  olver.getType(it.uriContent!!))
194                     viewModel.setPhotoProfile(it.uriContent!!, fileExtension)
195                     checkUpdate()
196                 }
197             }
198         } catch (e: Exception){
199             Toast.makeText(this, e.message.toString(), Toast.LENGTH_SHORT).show()
200         }
201     }
202 }
```

Kode 52: *Source Code* untuk *Edit Profile User Activity*

16. SEARCH MODEL

```
1   class SearchActivity : AppCompatActivity(), SearchView.OnQueryTextListener,
    ↪  SwipeRefreshLayout.OnRefreshListener {
2       private lateinit var binding : ActivitySearchBinding
3       private lateinit var viewModel : SearchViewModel
4       private lateinit var objectSearch : ProfileType
5       private lateinit var rowAdapter : RecyclerView.Adapter<*>
6
7       companion object {
8           const val TAG = "searchActivity"
9       }
10
11      override fun onCreate(savedInstanceState: Bundle?) {
12          super.onCreate(savedInstanceState)
13          binding = ActivitySearchBinding.inflate(layoutInflater)
14          setContentView(binding.root)
15
16          objectSearch = ProfileType.getType(intent.getStringExtra("search").toString())
17
18          viewModel = ViewModelProvider(this).get(SearchViewModel::class.java)
19          supportActionBar?.title = getString(R.string.search) + " $objectSearch"
20          supportActionBar?.elevation = 0f
21          supportActionBar?.setDisplayHomeAsUpEnabled(true)
22          supportActionBar?.setDisplayShowHomeEnabled(false)
23
24          val searchManager : SearchManager = getSystemService(Context.SEARCH_SERVICE) as
            ↪  SearchManager
25          binding.searchView.setSearchableInfo(searchManager.getSearchableInfo(componentN
            ↪  ame))
26          binding.searchView.setOnQueryTextListener(this)
27          binding.searchRefresh.setOnRefreshListener(this)
28
```

```
29          when(objectSearch){
30              ProfileType.PLANT -> setupRecyclerViewPlant()
31              ProfileType.USER -> setupRecyclerViewUser()
32          }
33      }
34
35      private fun setupRecyclerViewPlant(){
36          val plantModels : ArrayList<PlantModel> = arrayListOf()
37          val allUsers : ArrayList<UserModel> = viewModel.getAllUsers().value ?:
            ↪  arrayListOf()
38          viewModel.getAllPlants()
39          rowAdapter = AdapterType.SEARCH_PLANT.getAdapter(this, plantModels,
40              allUsers = allUsers,
41              type = AdapterType.Companion.SearchSelectType.SEARCH)
42          viewModel.getPlants().observe(this, {
43              plantModels.clear()
44              plantModels.addAll(it ?: arrayListOf())
45              rowAdapter.notifyDataSetChanged()
46              (rowAdapter as PlantModelAdapter).setOnItemClickListener(
47                  object : PlantModelAdapter.OnItemClickListener{
48                      override fun onItemClicked(
49                          position: Int,
50                          itemView: View,
51                          v: RowItemSearchBinding
52                      ) {
53                          when(itemView){
54                              v.searchRoot -> {
55                                  val intent = Intent()
56                                  intent.putExtra(BuildConfig.SELECTED_PLANT,
                                    ↪  viewModel.getPlant(position))
57                                  setResult(RESULT_OK, intent)
58                                  this@SearchActivity.onBackPressed()
59                                  finish()
60                              }
61                          }
62
63                      }
64                  }
65              )
66              binding.searchRefresh.isRefreshing = false
67              Log.i(TAG, "setupRecyclerViewPlant: ${rowAdapter.itemCount}")
68          })
69          viewModel.getAllUsers().observe(this, {
70              allUsers.clear()
71              allUsers.addAll(it ?: arrayListOf())
72              rowAdapter.notifyDataSetChanged()
73          })
74          setupRecyclerView()
75      }
76
77      private fun setupRecyclerViewUser(){
78          viewModel.setExceptUsers(intent.getParcelableArrayListExtra<UserModel>(BuildCon
            ↪  fig.EXCEPT_USERS))
79          val data : ArrayList<UserModel> = arrayListOf()
80          rowAdapter = AdapterType.SEARCH_USER.getAdapter(this, data,
81              type = AdapterType.Companion.SearchSelectType.SEARCH)
82          viewModel.getUsers().observe(this, {
83              data.clear()
84              data.addAll(it ?: arrayListOf())
85              rowAdapter.notifyDataSetChanged()
86              (rowAdapter as UserModelAdapter).setOnItemClickListener(
87                  object : UserModelAdapter.OnItemClickListener{
88                      override fun onItemClicked(
89                          userModel: UserModel,
90                          position: Int,
91                          itemView: View,
92                          v: RowItemSearchBinding
93                      ) {
94                          when(itemView){
95                              v.searchRoot -> {
```

```
                                val intent = Intent()
                                intent.putExtra(BuildConfig.SELECTED_USER,
                                ↪  viewModel.getUser(position))
                                setResult(RESULT_OK, intent)
                                this@SearchActivity.onBackPressed()
                                finish()
                            }
                        }
                    }
                })
            binding.searchRefresh.isRefreshing = false
            Log.i(TAG, "setupRecyclerViewUser: ${rowAdapter.itemCount}")
        })
        setupRecyclerView()
    }

    private fun setupRecyclerView(){
        binding.searchRecyclerView.adapter = rowAdapter
        binding.searchRecyclerView.layoutManager = LinearLayoutManager(this)
        binding.searchRecyclerView.addItemDecoration(object :
        ↪  DividerItemDecoration(this, VERTICAL) {})
        binding.searchRecyclerView.setHasFixedSize(true)
    }

    override fun onSupportNavigateUp(): Boolean {
        super.onBackPressed()
        return true
    }

    override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
        if (currentFocus != null) {
            val imm: InputMethodManager =
                getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
            imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
        }
        return super.dispatchTouchEvent(ev)
    }

    private fun searchKeyword(key:String?) : Boolean{
        when(objectSearch){
            ProfileType.PLANT -> viewModel.searchPlants(key)
            ProfileType.USER -> viewModel.searchUsers(key)
            else -> Log.i(TAG, "enum class not found")
        }
        return true
    }

    override fun onQueryTextSubmit(query: String?): Boolean {
        binding.searchView.clearFocus()
        return searchKeyword(query)
    }

    override fun onQueryTextChange(newText: String?): Boolean {
        return searchKeyword(newText)
    }

    override fun onRefresh() {
        when(objectSearch){
            ProfileType.PLANT -> {
                viewModel.getAllPlants(binding.searchView.query.toString())
            }
            ProfileType.USER ->
            ↪  viewModel.getAllUsers(binding.searchView.query.toString())
            else -> Log.i(TAG, "enum class not found")
        }
    }
}
```

Kode 53: *Source Code* untuk *Search Activity*

17. FEEDBACK ACTIVITY

```
class FeedbackActivity : AppCompatActivity(), TextWatcher,
↪  RatingBar.OnRatingBarChangeListener {
    private lateinit var binding : ActivityFeedbackBinding
    private lateinit var viewModel : FeedbackViewModel
    private lateinit var utility : ViewUtility

    companion object {
        const val TAG = "feedback"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityFeedbackBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider(this).get(FeedbackViewModel::class.java)
        supportActionBar?.title = getString(R.string.feedback)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowHomeEnabled(false)

        binding.apply {
            utility = ViewUtility(context = this@FeedbackActivity,
                circularProgressButton = feedbackSubmit,
                textInputEditTexts = arrayListOf(feedbackContent),
                actionBar = supportActionBar)

            feedbackRatingbar.rating = 5f
            feedbackRatingbar.onRatingBarChangeListener = this@FeedbackActivity
            feedbackContent.addTextChangedListener(this@FeedbackActivity)
            checkEmpty()

            feedbackSubmit.setOnClickListener {
                utility.isLoading = true
                viewModel.submitFeedback(feedbackContent.text.toString())
                viewModel.isFeedbackSubmit.observe(this@FeedbackActivity, {
                    if(it){
                        utility.isLoading = false
                        Toast.makeText(this@FeedbackActivity, "Thank you for your
                        ↪  feedback.", Toast.LENGTH_SHORT).show()
                        super.onBackPressed()
                        finish()
                    } else {
                        utility.isLoading = false
                        viewModel.feedbackSubmitError.observe(this@FeedbackActivity, {
                            if(it.isNotEmpty()){
                                Toast.makeText(this@FeedbackActivity, it,
                                ↪  Toast.LENGTH_SHORT).show()
                                Log.i(SignInActivity.TAG, it)
                                viewModel.feedbackSubmitError.value = ""
                            }
                        })
                    }
                })
            }
        }
    }

    override fun onSupportNavigateUp(): Boolean {
        super.onBackPressed()
        return true
    }

    override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
        if (currentFocus != null) {
            val imm: InputMethodManager =
                getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
            imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
        }
```

```
66          return super.dispatchTouchEvent(ev)
67      }
68
69      override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after:
        ↪ Int) {}
70
71      override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
72          checkEmpty()
73      }
74
75      override fun afterTextChanged(s: Editable?) {}
76
77      private fun checkEmpty() {
78          viewModel.checkNotEmpty(
79              utility.isEmpty(binding.feedbackContent) &&
80              viewModel.getRating() ?: 0f > 0f
81          ).observe(this, {
82              binding.feedbackSubmit.isEnabled = it
83          })
84      }
85
86      override fun onRatingChanged(ratingBar: RatingBar?, rating: Float, fromUser:
        ↪ Boolean) {
87          viewModel.setRating(rating)?.observe(this, {
88              it.rating?.let {
89                  binding.feedbackRatingtxt.text = Rating.getType(rating).toString()
90              }
91          })
92      }
93  }
```

Kode 54: *Source Code* untuk *Feedback Activity*

18. SHOW RECYCLER ACTIVITY

```
1   class ShowRecyclerActivity : AppCompatActivity() {
2       private lateinit var binding : ActivityShowRecyclerBinding
3       private lateinit var viewModel : ShowRecyclerViewModel
4
5       override fun onCreate(savedInstanceState: Bundle?) {
6           super.onCreate(savedInstanceState)
7           binding = ActivityShowRecyclerBinding.inflate(layoutInflater)
8           setContentView(binding.root)
9
10          viewModel = ViewModelProvider(this).get(ShowRecyclerViewModel::class.java)
11          viewModel.setCurrentPosts(intent.getParcelableArrayListExtra(BuildConfig.ALL_PL
            ↪ ANTS),
12              intent.getParcelableExtra<UserModel>(BuildConfig.CURRENT_USER))
13          supportActionBar?.title = getString(R.string.history)
14          supportActionBar?.setDisplayHomeAsUpEnabled(true)
15          supportActionBar?.setDisplayShowHomeEnabled(false)
16
17          setupRecyclerView()
18      }
19
20      private fun setupRecyclerView() {
21          val plantModels : ArrayList<PlantModel> = arrayListOf()
22          val userModels : ArrayList<UserModel> = arrayListOf()
23          val rowAdapter = AdapterType.POST_PLANT.getAdapter(this, plantModels, allUsers
            ↪ = userModels)
24          viewModel.getCurrentPosts().observe(this,{
25              plantModels.clear()
26              plantModels.addAll(it ?: arrayListOf())
27              rowAdapter.notifyDataSetChanged()
28              (rowAdapter as PostModelAdapter).setOnItemClickListener(
29                  object : PostModelAdapter.OnItemClickListener{
30                      override fun onItemClicked(
31                          position: Int,
```

```
                        itemView: View,
                        v: RowItemPostBinding
                    ) {
                        when(itemView){
                            v.postRoot -> {
                                gotoPost(position)
                            }
                            v.postOptions -> {
                                val items = arrayOf("Edit", "Delete")
                                MaterialAlertDialogBuilder(this@ShowRecyclerActivity)
                                    .setItems(items){_, which ->
                                        when(which){
                                            0 -> gotoPost(position)
                                            1 -> {
                                                viewModel.deletePost(position)
                                                viewModel.isPostDeleted.observe(this@Sh
                                                ↪  owRecyclerActivity,
                                                ↪  {
                                                    when(it){
                                                        true -> Toast.makeText(this@Sho
                                                        ↪  wRecyclerActivity, "Post
                                                        ↪  deleted.",
                                                        ↪  Toast.LENGTH_SHORT).show()
                                                    }
                                                })
                                            }
                                        }
                                    }
                                    .show()
                            }
                        }
                    }

                }
            )
            Log.i(MainPostsFragment.TAG, "${rowAdapter.itemCount}")
        })

        viewModel.getCurrentUser().observe(this, {
            userModels.clear()
            userModels.addAll(arrayListOf(it))
            rowAdapter.notifyDataSetChanged()
        })

        binding.recyclerView.apply {
            adapter = rowAdapter
            layoutManager = LinearLayoutManager(this@ShowRecyclerActivity)
            addItemDecoration(object : DividerItemDecoration(this@ShowRecyclerActivity,
            ↪  VERTICAL) {})
            setHasFixedSize(true)
        }
    }

    override fun onSupportNavigateUp(): Boolean {
        super.onBackPressed()
        return true
    }

    override fun dispatchTouchEvent(ev: MotionEvent?): Boolean {
        if (currentFocus != null) {
            val imm: InputMethodManager =
                getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
            imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
        }
        return super.dispatchTouchEvent(ev)
    }

    private fun gotoPost(position:Int){
        val intent = Intent(this, AddPlantActivity::class.java)
        intent.putExtra(BuildConfig.SELECTED_PLANT, viewModel.getPost(position))
```

```
        startActivity(intent)
    }
}
```

Kode 55: *Source Code* untuk *Show Recycler Activity*

19. SHOW PICTURE ACTIVITY

```
class ShowPictureActivity : AppCompatActivity() {
    private lateinit var binding : ActivityShowPictureBinding

    private var currentImage : Bitmap? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityShowPictureBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val titleBar = intent.getStringExtra("actionBarTitle")

        window.statusBarColor = Color.BLACK
        supportActionBar?.setDisplayShowHomeEnabled(true)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setBackgroundDrawable(ColorDrawable(ContextCompat.getColor(th
        ↪  is,
        ↪  R.color.black)))
        supportActionBar?.title = if(titleBar.isNullOrEmpty()){"Profile Picture"}
        else{titleBar}

        val filename = intent.getStringExtra("photoContent")
        try {
            val `is`: FileInputStream = openFileInput(filename)
            currentImage = BitmapFactory.decodeStream(`is`)
            `is`.close()
        } catch (e: Exception) {
            e.printStackTrace()
        }
        binding.showPicture.setImageBitmap(currentImage)
    }

    override fun onSupportNavigateUp(): Boolean {
        super.onBackPressed()
        return true
    }
}
```

Kode 56: *Source Code* untuk *Show Picture Activity*

20. ABOUT ME ACTIVITY

```
class AboutMeActivity : AppCompatActivity() {
    private lateinit var binding: ActivityAboutMeBinding
    private var intentData = IntentUtility(this)

    private fun setupToolbar(){
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowHomeEnabled(false)
        supportActionBar?.title = getString(R.string.about_me)
    }

    private fun setupHeader(){
        binding.aboutEmail.text = BuildConfig.CREATOR_EMAIL
        binding.aboutDescription.text = listOf(getString(R.string.tab),
        ↪  getString(R.string.profile_description)).joinToString(" ")
        Glide.with(this)
            .load(R.drawable.profile_photo)
            .circleCrop()
```

```
            .into(binding.profilePicture)
    }

    private fun setupFindMe(){
        binding.aboutFindMe.itemIconTintList = null
        binding.aboutFindMe.setNavigationItemSelectedListener {
            when(it.itemId) {
                R.id.about_whatsapp -> {
                    try{
                        val intent = Intent()
                        applicationContext.packageManager.getPackageInfo("com.whatsapp"
                        ↪ ,
                        ↪ PackageManager.GET_META_DATA)
                        intent.action = Intent.ACTION_VIEW
                        intent.type = "text/plain"
                        intent.data = Uri.parse(BuildConfig.CREATOR_WHATSAPP)
                        intent.setPackage("com.whatsapp")
                        startActivity(intent)
                    } catch (e : PackageManager.NameNotFoundException) {
                        Toast.makeText(this, e.toString(), Toast.LENGTH_SHORT).show()
                    }
                    true }
                R.id.about_email ->{
                    intentData.openEmail(BuildConfig.CREATOR_EMAIL)
                    true }
                R.id.about_github -> {
                    intentData.openBrowser(BuildConfig.CREATOR_GITHUB)
                    true }
                R.id.about_project -> {
                    intentData.openBrowser(BuildConfig.PROJECT_URL)
                    true }
                else -> false
            }
        }
    }

    private fun setupCredit() {
        binding.aboutCredit.itemIconTintList = null
        binding.aboutCredit.setNavigationItemSelectedListener {
            when (it.itemId) {
                R.id.about_lottie ->
                ↪ intentData.openBrowser("https://lottiefiles.com/45869-farmers")
                R.id.about_freepik -> intentData.openBrowser("https://www.freepik.com/")
            }
            true
        }
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityAboutMeBinding.inflate(layoutInflater)
        setContentView(binding.root)
        setupToolbar()
        setupHeader()
        setupFindMe()
        setupCredit()
    }

    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return true
    }
}
```

Kode 57: *Source Code* untuk *About Me Activity*