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.

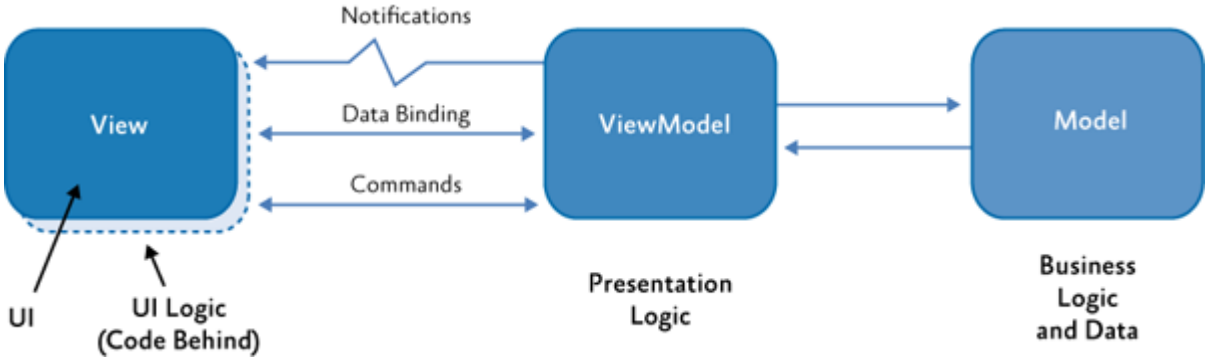
Dikenal sebagai produk yang dikembangkan oleh Google (GOOGL), namun pengembangan awalnya diinisasi oleh Android Inc. pada 2003 yaitu sebuah perusahaan software yang berbasis di Silicon Valley sebelum diakuisisi oleh Google pada 2005 [2]. Android dirilis pertama kali ke publik pada Oktober 2008 dengan 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

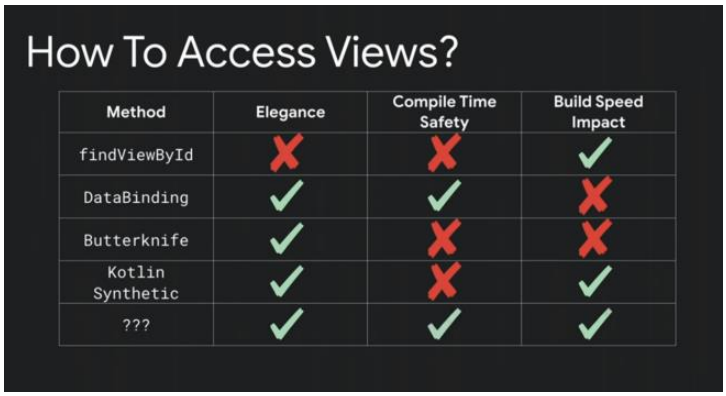


Gambar 1. Ilustrasi arsitektur MVVM

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

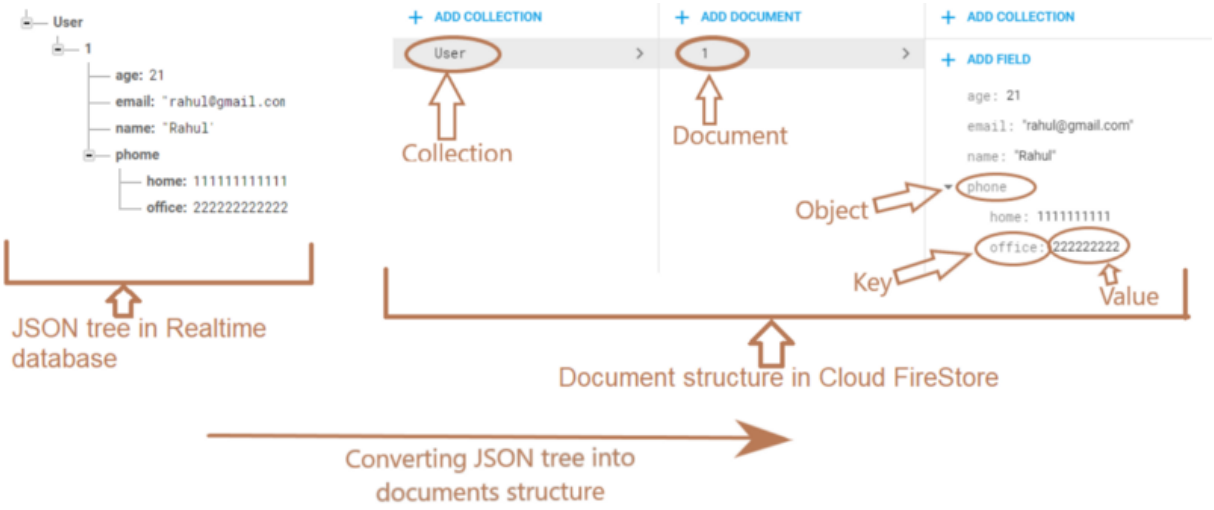
```
fun firebaseAuth(email:String, password:String){
    val auth : FirebaseAuth = Firebase.auth

    /** 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

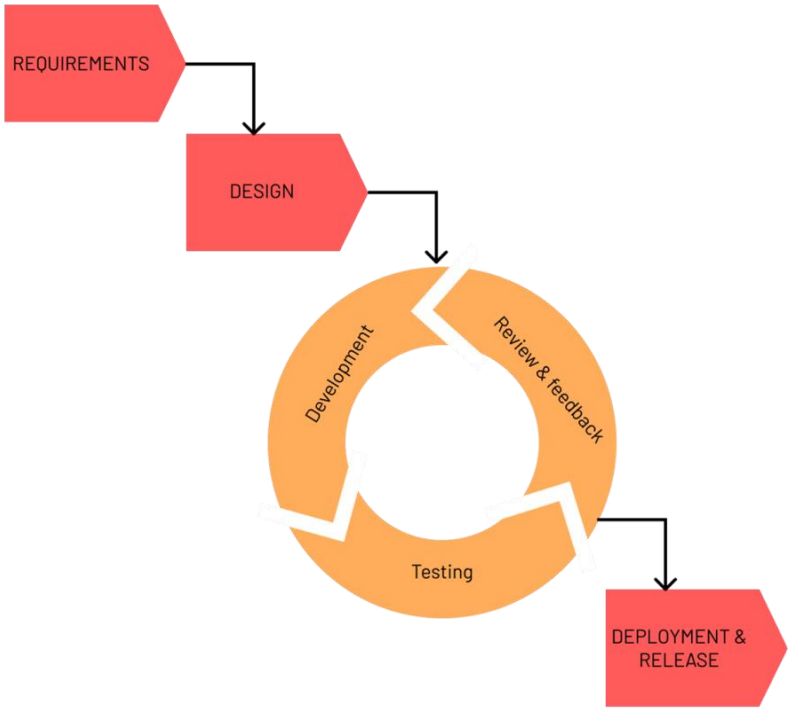
Firebase Storage adalah layanan untuk menyediakan penyimpanan file seperti gambar, dokumen, dan video. Layanan ini terintegrasi dengan Firebase Authentication, sehingga dapat digunakan untuk media penyimpanan berkas *client-based.* Setelah mengunggah berkas, maka akan mendapatkan url berkas sehingga 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 :

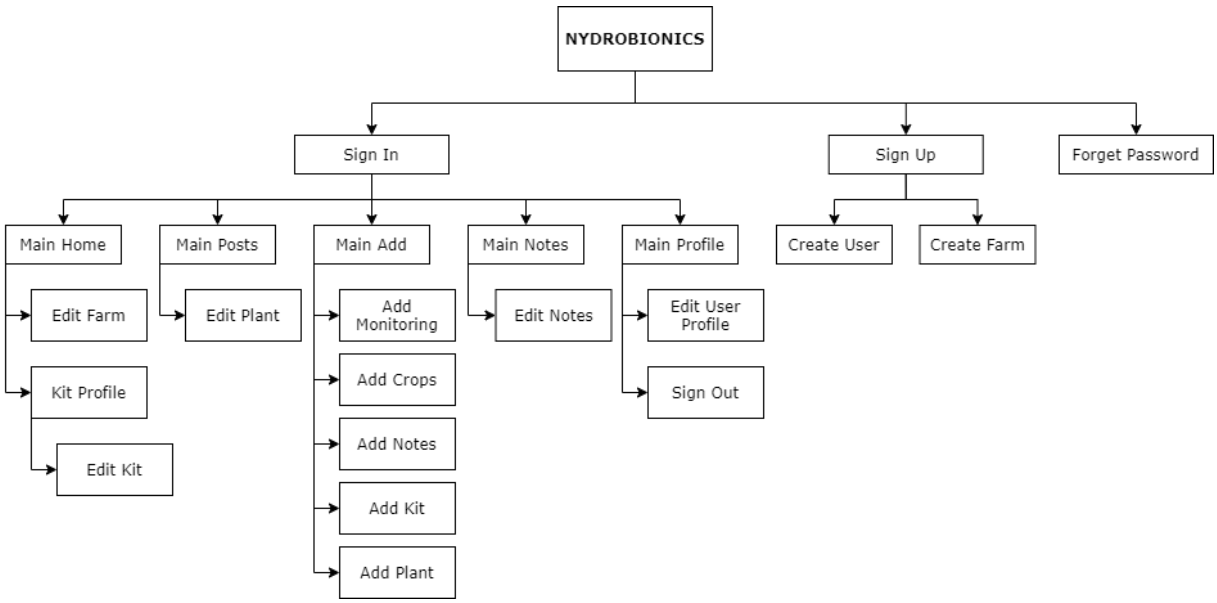
a) Pengguna melakukan autentikasi menggunakan email dan *password* yang telah didaftarkan sebelumnya.
b) Jika belum memiliki akun, maka pengguna melakukan *sign up* dengan email dan *password* kemudian membuat akun dengan memberikan data diri seperti nama, tanggal lahir, alamat, dsb. termasuk jenis akun 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

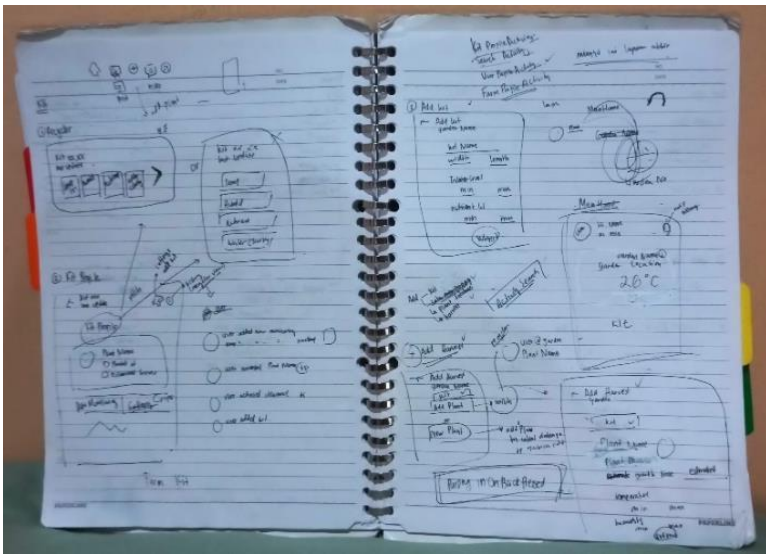
Struktur menu aplikasi Nydrobioncs ditunjukkan melalui Gambar 6. Saat pengguna membuka aplikasi, maka ada 3 menu yang dapat dipilih, yaitu *Sign In*, *Sign Up*, dan *Forget Password*. *Sign In* berguna untuk memproses autentikasi. *Sign Up* berguna memproses pembuatan akun yang dilanjutkan untuk membuat 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

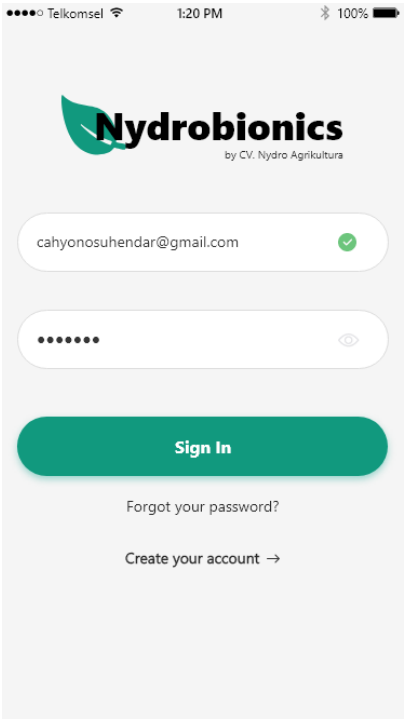
Berdasarkan struktur menu aplikasi yang dibuat, maka dapat dikembangkan kerangka tampilan aplikasi dari setiap menu yang telah ditentukan. Pada pengembangan kali ini, pembuatan kerangka tampilan aplikasi dilakukan secara manual gambar tangan sehingga hanya berupa *rough sketch* yang ditunjukkan pada 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.



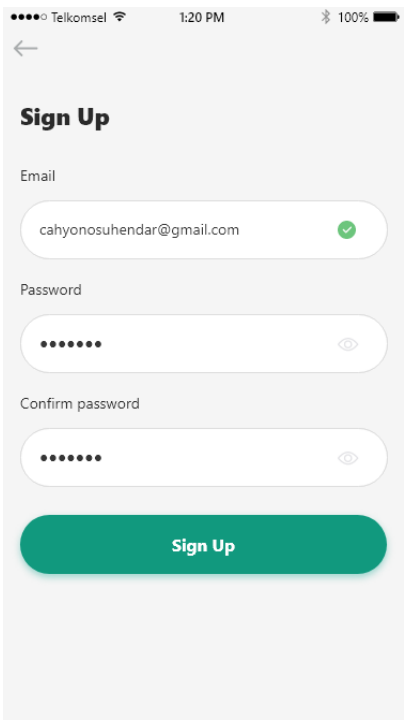
Gambar 7. *Mock up* aplikasi
Sumber : Dokumentasi pribadi

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

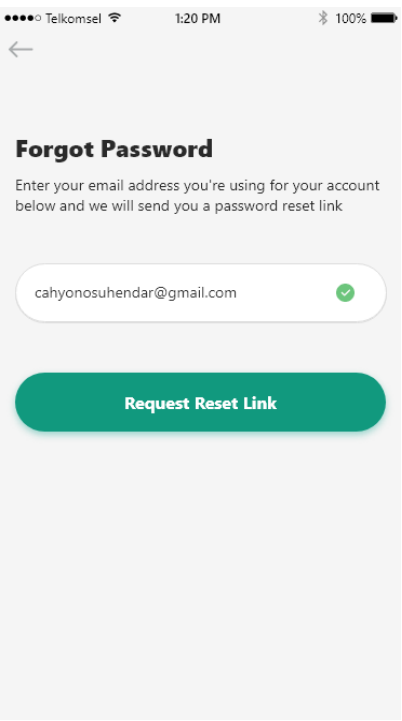
sign in (a), *sign up* (b), *forget password* (c), *main home* (d), *main farm* (e), *main schedule* (f), *main profile* (g), *kit profile* (h), dan *add schedule* (i). Desain *front end* final yang digunakan pada pengembangan kali ini langsung diimplementasikan melalui Layout XML yang bersangkutan.



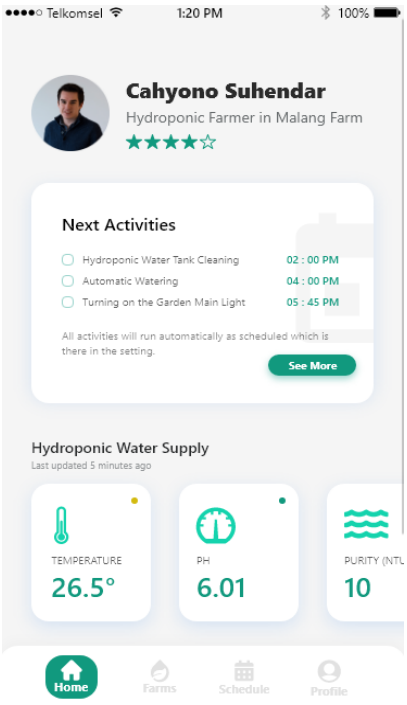
a) Menu *Sign In*



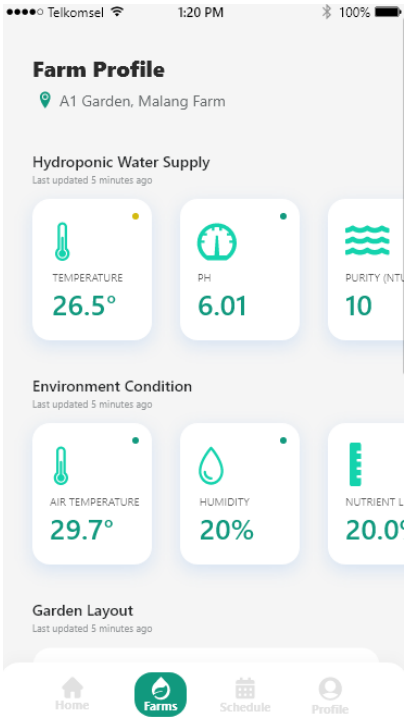
b) Menu *Sign Up*



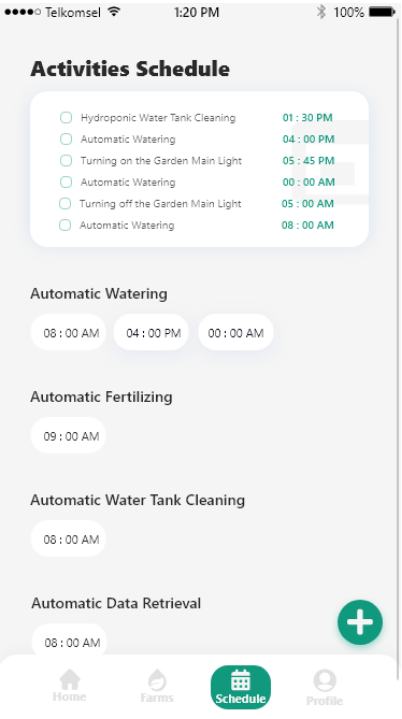
c) Menu Forget *Password*



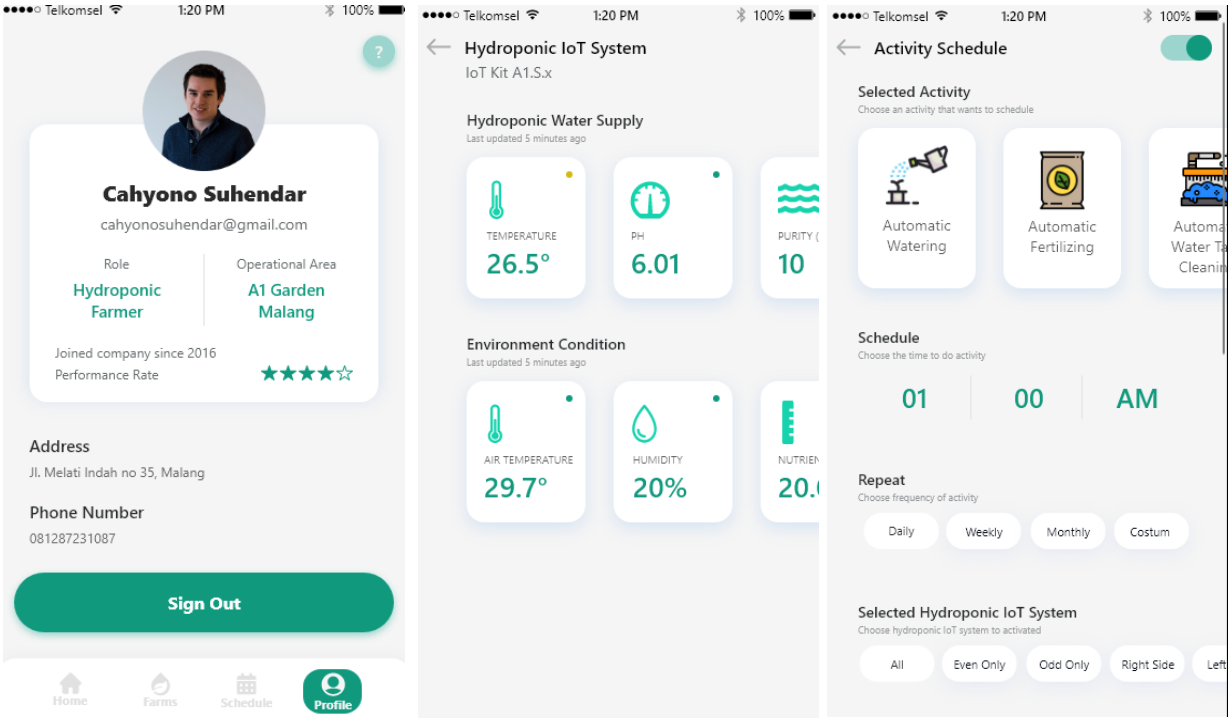
d) Menu *Main Home*



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*.

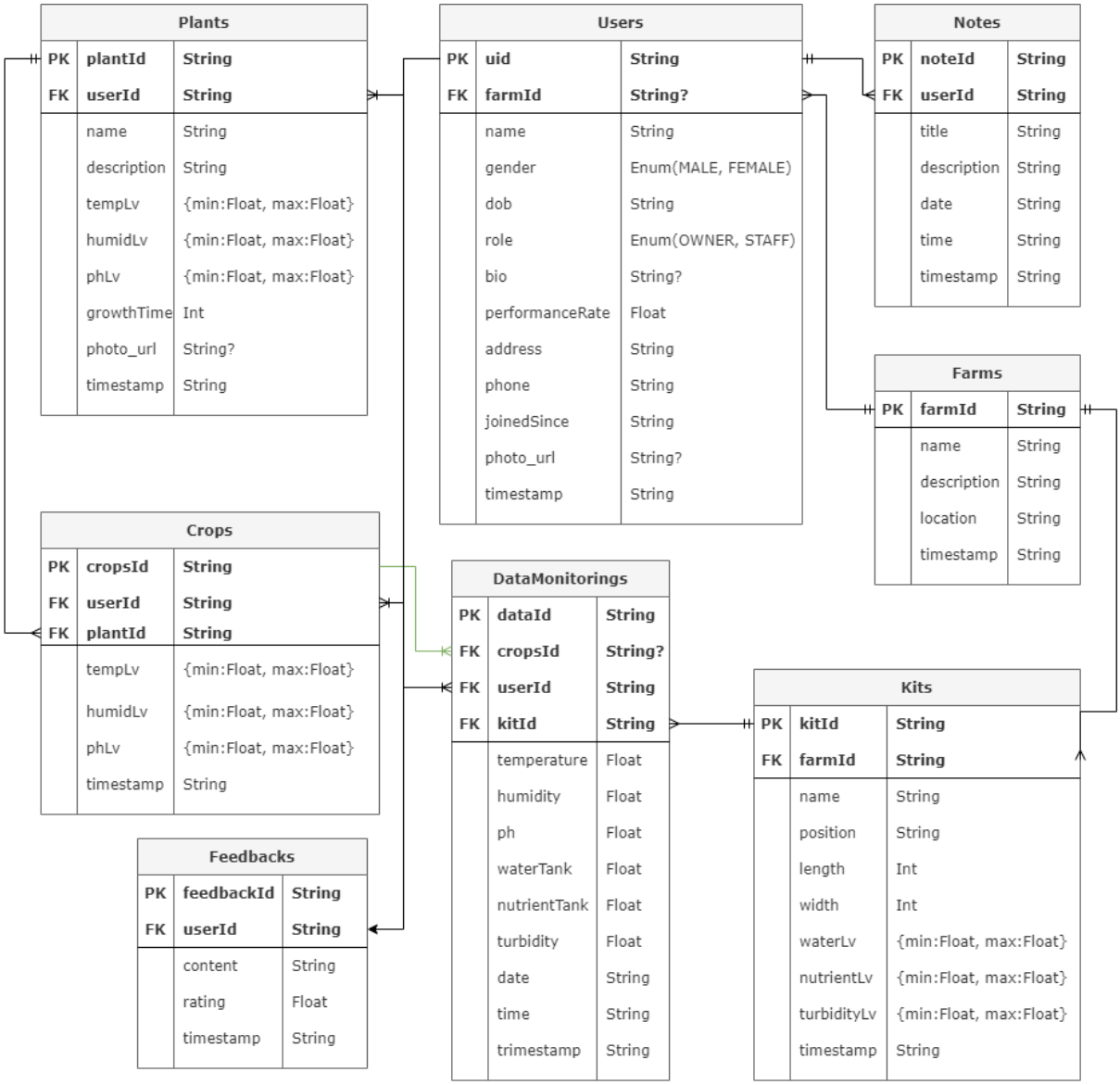
(3) Setiap *farm* dapat memiliki banyak *kit* dan *user.*

(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*.



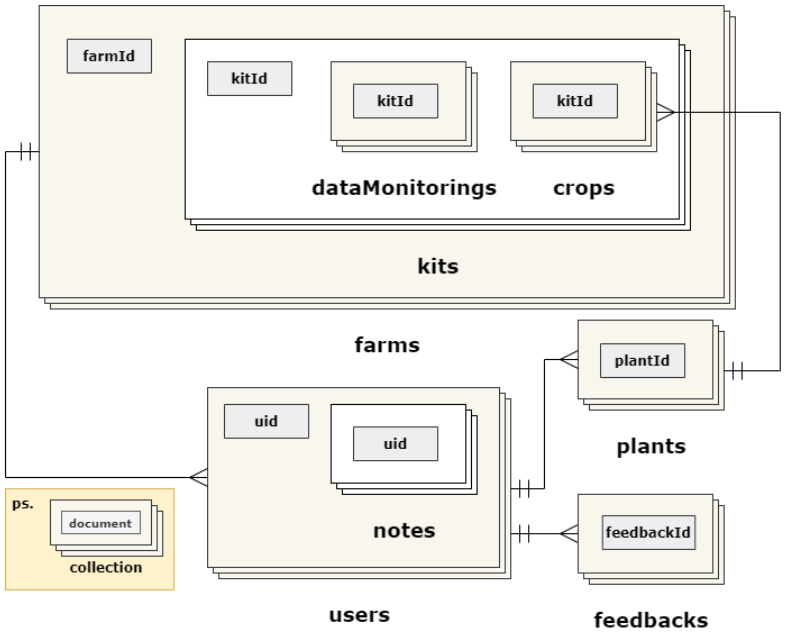
Gambar 9. *Entity Relationship Diagram* (ERD) aplikasi

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.*

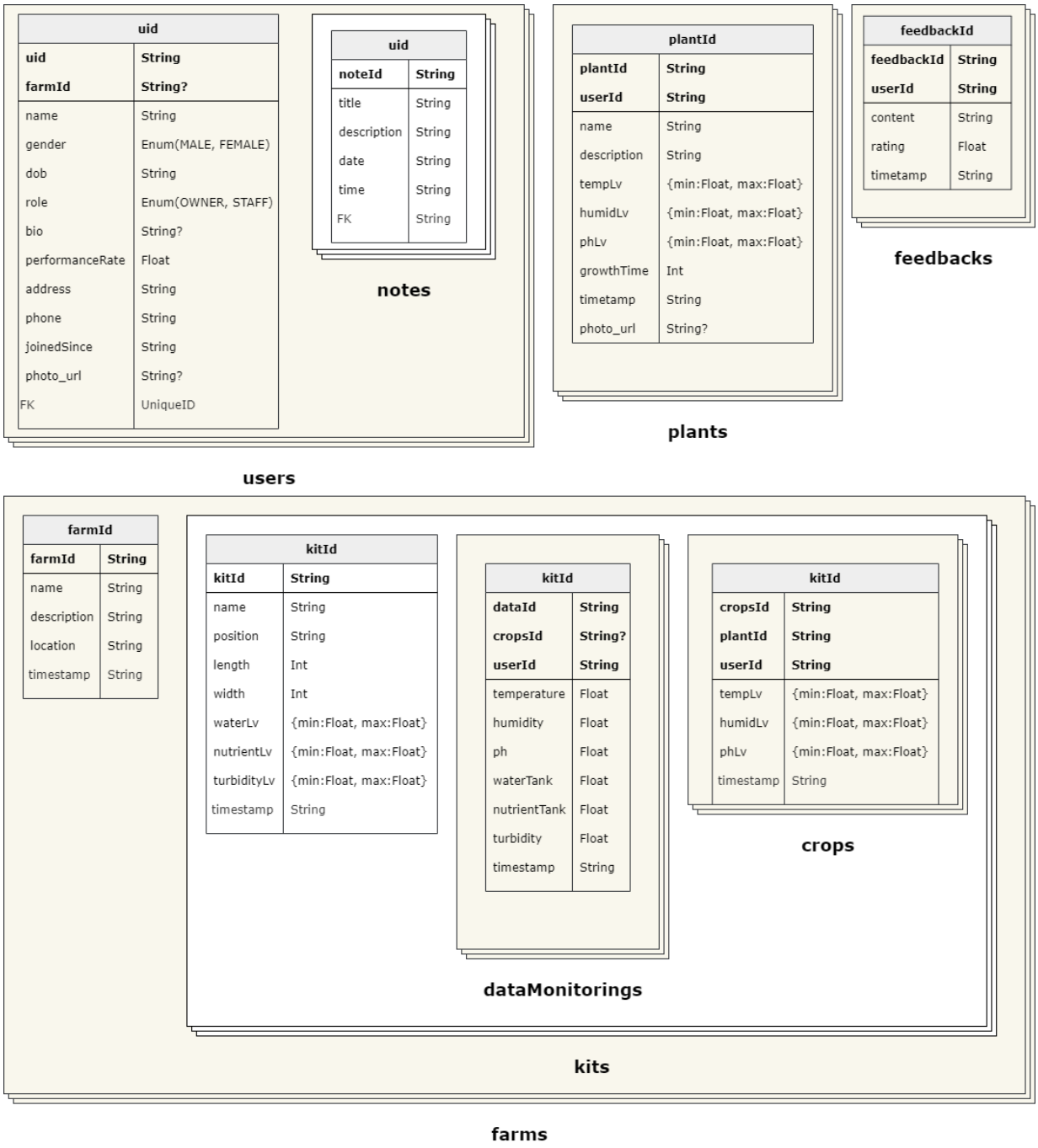
(2) Setiap *farm* dapat memiliki banyak *user*.

(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

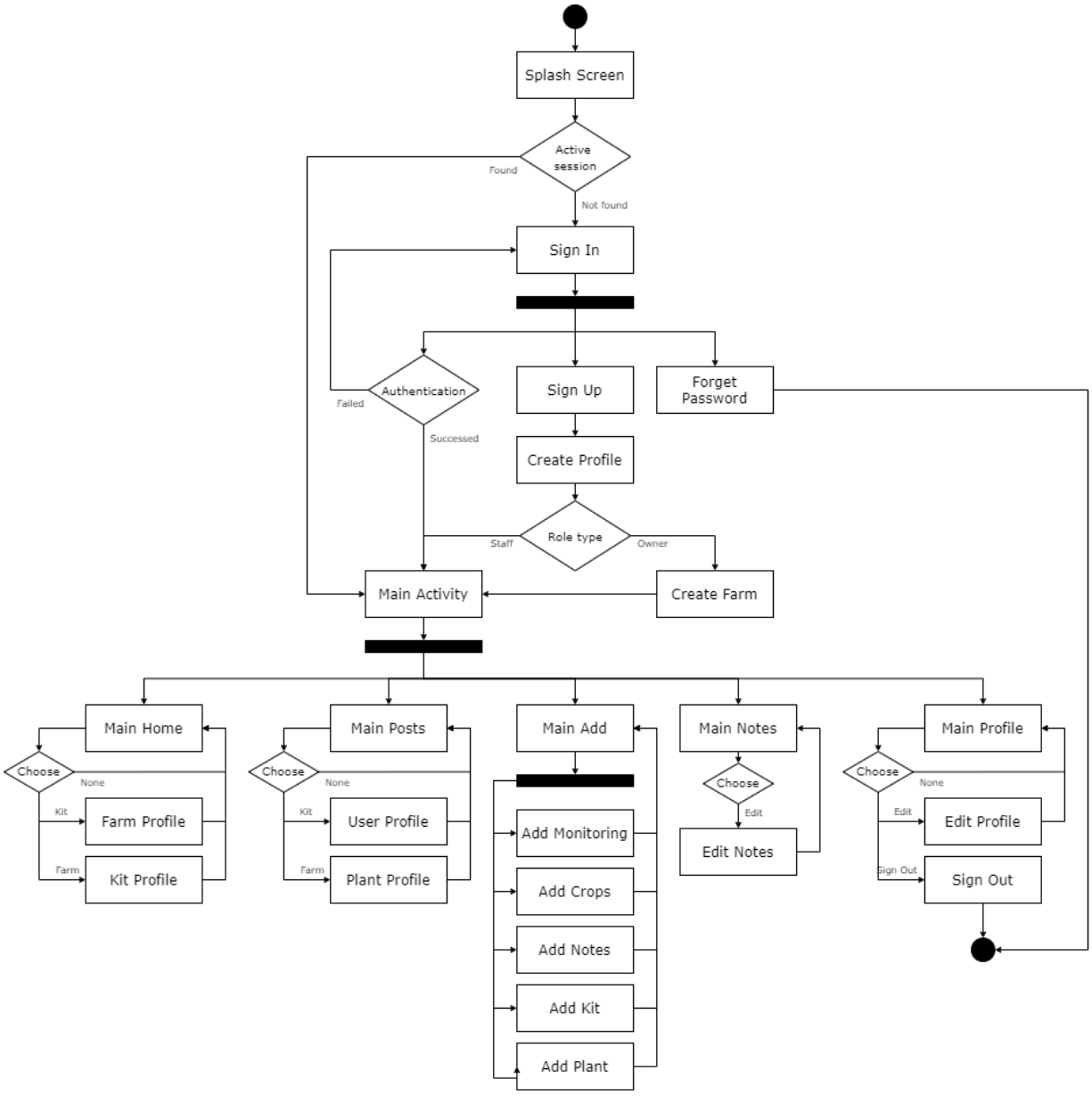
redudansi data. Sebagai contohnya untuk tabel *Notes* pada ERD (Gambar 9) memiliki *foreign key* `userId` yang terhubung dengan uid di tabel *Users,* disesuaikan sebagai *subcollection* dari *collection* utama *users* (Gambar 11) dengan *document id* yang digunakan adalah `uid` sehingga *foreign key*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

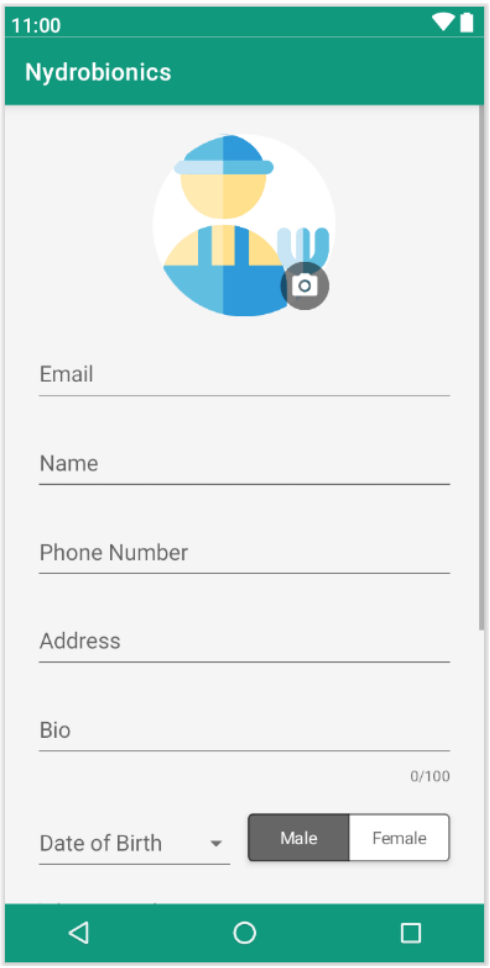
```
db.document(farmModel.value!!.farmId!!).set(farmModel.value!!.toHashMap
()).addOnCompleteListener {
            if(it.isSuccessful){
                userModel.value?.farmId = farmModel.value!!.farmId!!
                sendUserProfile()
                isFarmCreated.value = isUserCreated.value == 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