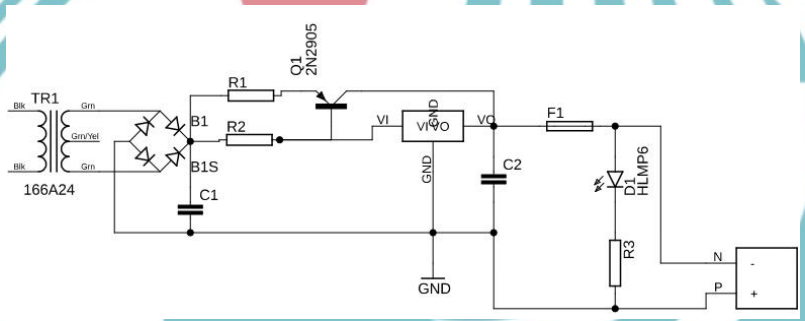
BAB II
TINJAUAN PUSTAKA

2.1 *Power Supply*

Power supply merupakan suatu rangkaian elektronika yang mengubah arus listrik bolak balik menjadi arus searah yang digunakan untuk mensuplai tegangan. Power supply menjadi bagian yang penting dalam elektronika yang berfungsi sebagai pemasok energi listrik untuk kebutuhan alat eleknonika. Komponen utama yang digunakan yaitu resistor 100 ohm, 0.1 ohm, 220ohm, LED, transistor TIP2955, Dioda Bridge, IC7805, Fuse 250V, dan Trafo 1A. (Saodah & Ramdani, 2020)



Gambar 2.1 Rangkaian Power Supply

2.2 *Internet Of Things* (IoT)

Internet Of Things, yang biasa disingkat IoT adalah sebuah konsep yang bertujuan untuk memperluas manfaat dari koneksi internet yang selalu aktif, menghubungkan mesin, perangkat, dan objek fisik lainnya ke sensor dan aktuator untuk memperoleh data dan mengelola kinerjanya sendiri. Memungkinkan mesin untuk bekerja sama dan beroperasi secara independen berdasarkan informasi yang baru diperoleh. (Efendi, 2018)

Teknologi IoT ini mendukung kerja sistem sebagai suatu kesatuan meliputi komponen/elemen dalam hal memudahkan proses aliran informasi data. Sistem pada penelitian ini mengabungkan tiga bagian penting, yaitu mekanik, *hardware* (elektronik) dan algoritma kontrol, dimana ketiga bagian tersebut saling berinteraksi dan tidak dapat dipisahkan dalam satu kesatuan sistem. (Abdullah, 2021)

2.3 ESP 32

Mikrokontroler ESP32 merupakan mikrokontroler SoC (System on Chip) terpadu dengan dilengkapi WiFi 802.11 b/g/n, Bluetooth versi 4.2, dan berbagai peripheral. ESP32 adalah chip yang cukup lengkap, terdapat prosesor, penyimpanan dan akses pada GPIO (General Purpose *Input Output*). ESP32 bisa digunakan untuk rangkaian pengganti pada Arduino, ESP32 memiliki kemampuan untuk mendukung terkoneksi ke WI-FI secara langsung (Wagyana & Rahmat, 2019).

Adapun spesifikasi dari ESP32 adalah sebagai berikut: Board ini memeliki dua versi, yaitu 30 GPIO dan 36 GPIO. Keduanya memiliki fungsi yang sama tetapi versi yang 30 GPIO dipilih karena memiliki dua pin GND. Semua pin diberi label dibagian atas board sehingga mudah untuk dikenali. Board ini memiliki interface USB to UART yang mudah diprogram dengan program pengembangan aplikasi seperti Arduino IDE. Sumber daya board bisa diberikan melalui konektor micro USB (Nizam, Yuana, & Wulansari, 2022)

Gambar 2.2 ESP 32

Sumber : Jurnal Ilmiah Setrum

2.4 *Ultrasonic* Sensor JSN SR04T

Sensor ini dilengkapi dengan kabel sepanjang 2,5 m yang menghubungkan ke papan breakout yang mengontrol sensor dan melakukan semua pemrosesan sinyal. Perhatikan bahwa hanya sensor dan kabelnya yang tahan air, jika Anda memasukkan air ke papan pelepas, sensor mungkin berhenti bekerja. Sensor jarak ultrasonik bekerja dengan mengirimkan gelombang ultrasonik. Gelombang ultrasonik ini dipantulkan kembali oleh suatu objek dan sensor ultrasonik mendeteksi jarak objek tersebut. (Purwanto, Riyadi, Astuti, & Kusuma, 2019)

Gambar 2.3 Sensor Ultrasonik JSN SR04T

Sumber : Jurnal Simetris

2.5 LCD OLED

LCD OLED adalah jenis layar yang menggunakan teknologi *Organic Light Emitting Diode* (OLED) untuk menghasilkan gambar. Tidak seperti LCD biasa yang memerlukan lampu latar, setiap piksel pada layar OLED memancarkan cahayanya sendiri, sehingga menghasilkan kontras yang lebih tinggi dan warna yang lebih cerah.

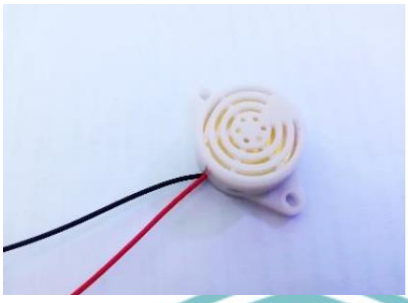
Gambar 2.4 LCD OLED

Sumber : Jurnal Ilmiah Setrum

Layar OLED juga lebih tipis dan fleksibel dibandingkan LCD konvensional, serta memiliki waktu respons yang lebih cepat, membuatnya ideal untuk perangkat elektronik seperti *smartphone*, televisi, dan *wearable devices*. Keunggulan lainnya termasuk konsumsi daya yang lebih rendah dan kemampuan untuk menampilkan warna hitam yang sejati karena piksel dapat sepenuhnya dimatikan. (Arafat & Alamsyah, 2018)

2.6 Buzzer

Buzzer adalah komponen elektronik yang mengubah sinyal listrik menjadi suara, menggunakan kristal piezoelektrik atau kumparan magnet. Digunakan dalam alarm, pengingat, dan indikator suara pada perangkat seperti jam, oven, mesin cuci, dan sistem keamanan.



Gambar 2.5 Buzzer

Sumber : Jurnal Mahasiswa Teknik Informatika

2.7 Dispenser Galon Bawah

Dispenser galon bawah adalah alat untuk menyediakan air minum yang dirancang dengan tempat galon air di bagian bawah unit. Desain ini memudahkan penggantian galon karena tidak perlu diangkat ke atas, sehingga lebih praktis dan ergonomis. Dispenser ini biasanya dilengkapi dengan pompa internal yang menarik air dari galon ke atas, menyediakan air panas, dingin, atau suhu ruangan sesuai kebutuhan. Dispenser galon bawah populer di rumah, kantor, dan ruang publik karena kemudahan penggunaannya dan desain yang lebih rapi.

2.8 Pompa Galon Elektrik

Pompa galon elektrik adalah perangkat praktis yang memudahkan pengeluaran air dari galon menggunakan tenaga listrik. Dilengkapi dengan selang dan tombol kontrol, pompa ini ideal untuk digunakan di rumah, kantor, atau luar ruangan, memberikan kemudahan dalam mengakses air minum.

2.9 Android

Android merupakan sistem operasi terbuka di bawah Lisensi Apache, sehingga Android dimungkinkan untuk dimodifikasi secara bebas oleh para pembuat perangkat, operator nirkabel, dan pengembang aplikasi. Fungsionalitas perangkat Android diperluas oleh sejumlah besar komunitas pengembang aplikasi (apps). Umumnya Android ditulis dalam versi bahasa pemrograman Java. (Gunadi, Tanone, & Beeh, 2020)

2.10 *Firebase*

Firebase adalah platform pengembangan aplikasi yang menyediakan berbagai layanan, termasuk autentikasi, penyimpanan database, analitik, dan

lainnya. Untuk menggunakan Firebase dalam pengembangan aplikasi, langkah pertama adalah mendaftarkan proyek di *Firebase Console*, di mana pengembang dapat menambahkan aplikasi sesuai platform yang digunakan, seperti Android, iOS, atau Web. Setelah aplikasi ditambahkan, SDK *Firebase* perlu diintegrasikan ke dalam proyek. Misalnya, untuk Android, konfigurasi dilakukan dengan menambahkan SDK pada file *build.gradle*. Layanan *Firebase* seperti Autentikasi memungkinkan pengaturan metode login, sementara *Realtime Database* atau *Firestore* menyediakan fasilitas penyimpanan dan pengelolaan data secara *real-time*. Selain itu, *Firebase Storage* dapat digunakan untuk menyimpan file, dan *Cloud Messaging* untuk mengirim notifikasi push. Setelah implementasi selesai, aplikasi dapat di *deploy*, dan performa serta penggunaan dapat dipantau melalui *Firebase Console*. *Firebase* menyediakan fleksibilitas yang tinggi dengan dokumentasi yang lengkap, memudahkan integrasi dalam berbagai jenis aplikasi. (Sonita & Fardianitama, 2018)

2.11 Android Studio

Android Studio merupakan sebuah Integrated Development Environment (IDE) khusus untuk membangun aplikasi yang berjalan pada platform android. Bahasa pemrograman utama yang digunakan adalah Kotlin, sedangkan untuk membuat tampilan atau layout, digunakan bahasa XML. Android studio juga terintegrasi dengan Android *Software* Development Kit (SDK) untuk deploy ke perangkat android. (Sibuea, Saputro, Annan, & Widodo, 2022)

Android Studio adalah Integrated Development Environment (IDE) yang digunakan untuk mengembangkan aplikasi Android. Penggunaannya dimulai dengan mengunduh dan menginstal Android Studio, yang sudah dilengkapi dengan Android SDK. Setelah terinstal, pengembang dapat membuat proyek baru dengan menentukan nama proyek, paket, dan target SDK. Proyek ini terdiri dari modul yang mencakup kode sumber, file antarmuka pengguna (layout), dan sumber daya seperti gambar dan string.

Bahasa pemrograman utama yang digunakan dalam Android Studio adalah *Java* dan *Kotlin*. Pengembang dapat menulis logika aplikasi dalam bahasa ini, sementara antarmuka pengguna diatur menggunakan XML. Android Studio

mendukung fitur seperti *debugging*, emulasi perangkat, dan pengelolaan dependensi melalui *Gradle*, memungkinkan pengembang untuk membangun, menguji, dan mendistribusikan aplikasi dengan efisien. Misalnya, pengembang bisa menulis logika seperti *onCreate()* di *Java* atau *Kotlin*, serta mengatur tampilan dengan kode XML seperti <*LinearLayout*> untuk membuat antarmuka pengguna.

2.12 *Wireshark*

Wireshark merupakan sebuah *software packet analyzer* gratis dan open *source*. *Software* Wireshark digunakan untuk menemukan masalah pada jaringan, pengembangan perangkat lunak dan protokol komunikasi. Fungsi Wireshark yaitu menganalisa data yang melintas pada media transmisi dan mempresentasikan informasi yang didapat secara logis sesuai dengan model OSI referensi model (Rika Wulandari, 2016).

2.13 *Quality of Service* (QoS)

QoS adalah konsep yang digunakan untuk mengelola lalu lintas jaringan dan memastikan bahwa aplikasi yang memerlukan performa tinggi, seperti VoIP dan video *streaming*, mendapatkan prioritas yang tepat untuk memastikan kualitas pengalaman pengguna. Adapun beberapa parameter yang dihitung pada QoS adalah : *Packet Loss*, *Throughput*, *Delay*, dan *Jitter* (Wijaya, 2024)

Tabel 2.1 Rumus Perhitungan QoS

No	Parameter	Rumus
1	*Packet Loss*	$Packet\ Loss = \frac{(paket\ yang\ dikirim - paket\ yang\ diterima)}{paket\ yang\ dikirim} \times 100\%$
2	*Throughput*	$Throughput = \frac{\text{Jumlah Data Yang Dikirim}}{\text{Waktu Pengiriman Data}}$
3	*Delay*	$Delay = \frac{\text{Waktu pengiriman}}{\text{Paket diterima}}$

No	Parameter	Rumus
4	*Jitter*	$\frac{\text{Total Variasi } Delay \text{ (-1)}}{\text{Total Paket Diterima}}$

BAB III
PERENCANAAN DAN REALISASI

3.1 Perencanaan Alat dan Aplikasi

Pada bab ini akan dijelaskan mengenai perencanaan dan realisasi alat dan aplikasi yang dibuat dalam tugas akhir. Bagian ini akan dijelaskan tentang deskripsi alat, cara kerja alat, spesifikasi alat, dan diagram blok yang menunjukan cara kerja alat. Untuk perancangan alat terdiri dari perancangan *hardware* yaitu alat sistem, dan catu daya serta perancangan *software* yaitu aplikasi android.

3.1.1 Deskripsi Alat

Monitoring yang dilakukan yaitu berapa tinggi air dari mulut galon dan statusnya. Komponen utama yang digunakan ada sensor *Ultrasonic* JSN SR04T untuk membaca ketinggian air, dan ESP 32 sebagai mikrokontrollernya. ESP 32 akan terkoneksikan dengan *WiFi* agar terhubung ke internet sehingga *data* pembacaan dapat dikirimkan ke *database*.

Gambar 3.1 Ilustrasi Alat Sistem Monitoring Level Air dalam Galon

Dispenser terdiri dari dua jenis desain, salah satunya dengan galon tertutup di bagian bawah, sehingga air tidak terlihat habis atau belum. Untuk mengatasi masalah ini, sensor ditempatkan di mulut galon guna mendeteksi ketinggian air dengan akurat. Gambar 3.1 menunjukkan gambaran dari dispenser

galon bawah yang sudah dimodifikasi dan pada box casing terdapat beberapa komponen seperti ESP32, sensor ultrasonik, LCD OLED, dan buzzer.

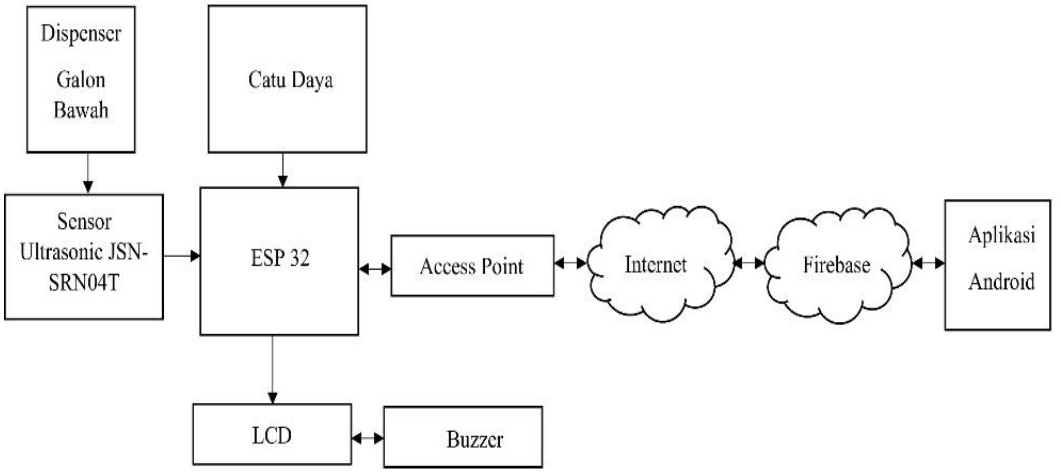
Untuk memastikan bahwa alat dapat berfungsi dengan baik dan memiliki sumber tegangan yang stabil, direncanakan penggunaan catu daya sebagai penyuplai utama. Rangkaian catu daya terdiri dari berbagai komponen penting, yaitu resistor dengan nilai 100 ohm, 0.1 ohm, dan 220 ohm, LED, transistor TIP2955, dioda bridge, IC 7805, fuse 250V, serta trafo 1A. Setiap komponen memainkan peran krusial dalam memastikan aliran tegangan yang stabil dan aman untuk operasional alat, sehingga mendukung kinerja alat dengan optimal.

3.1.2 Spesifikasi Alat

Dalam pembuatan dan realisasi alat, dibutuhkan perangkat keras untuk merancang dan merealisasikan *hardware*. Spesifikasi perangkat keras dapat dilihat pada Tabel 3.1.

3.1.3 Diagram Blok Alat

Untuk mempermudah pengerjaan Rancang Bangun Sistem *Monitoring Level* Air dalam Galon yang terintegrasi dengan *firebase*, maka dibuatlah diagram blok yang dapat menjelaskan keseluruhan proses kerja sistem. Gambar 3.2 menunjukkan diagram blok sistem *monitoring level* air dalam galon.



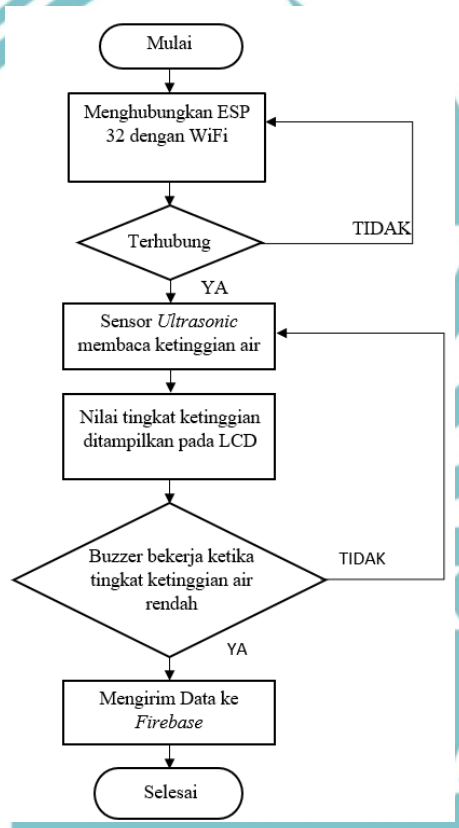
Gambar 3.2 Diagram Blok Sistem *Monitoring Level* Air dalam Galon

Tabel 3.1 Spesifikasi Perangkat Keras

No	Perangkat	Spesifikasi	Keterangan
1	Power Supply	Tegangan *Input*	220V AC
		Tegangan *Output*	5V DC
2	Dispenser Galon Bawah	Daya	5W-400W
		Bahan Bodi	220V~AC/50-60Hz
		Muatan	Plastik PP dan ABS
		Dilengkapi	Galon Bawah
		Dimensi	Lampu Indikator
		Panjang Kabel	30 x 30,5 x 101,5 cm
			1,5 m
3	Pompa Galon Elektrik	Daya	4W
		Voltage	5V-DC
		Kapasitas Baterai	1200mAh
		Material	ABS + Silicon
		Pipa Stainless	304
4	Sensor Ultrasonik JSN-SR04T	Tegangan Kerja	5V-DC
		Jarak Pengukuran	20 cm – 600 cm
		Dimensi	41 mm x 28.5 mm
		Frekuensi Ultrasonik	40 kHz
		Tahan Air	Ya
5	ESP 32	CPU	Dual-core Xtensa® 32-bit LX6
		Kecepatan Clock	240 MHz
		RAM	520Kb Internal
		Flash	16MB
		WiFi	802.11 b/g/n
		Bluetooth	Bluetooth v4.2 BR/EDR dan BLE
6	LCD OLED	Piksel	128 x 64
		Tegangan	5V-DC
7	Buzzer	Tegangan	4V-8V
		Arus	30mA / 5V-DC
		Max Volume	85dB

3.1.4 *Flowchart* Alat

Berikut adalah *flowchart* dari alat *monitoring level* air dalam galon untuk depot air minum berbasis android. *Flowchart* pada Gambar 3.3 menunjukkan alur sistem pengukuran ketinggian air yang dimulai dengan menghubungkan ESP32 ke WiFi. Setelah terhubung, sensor ultrasonik membaca ketinggian air dan menampilkannya di LCD. Jika ketinggian air rendah, buzzer aktif, dan data dikirim ke *firebase*. Proses berakhir setelah semua langkah selesai. Jika koneksi *WiFi* gagal, sistem akan terus mencoba hingga berhasil.



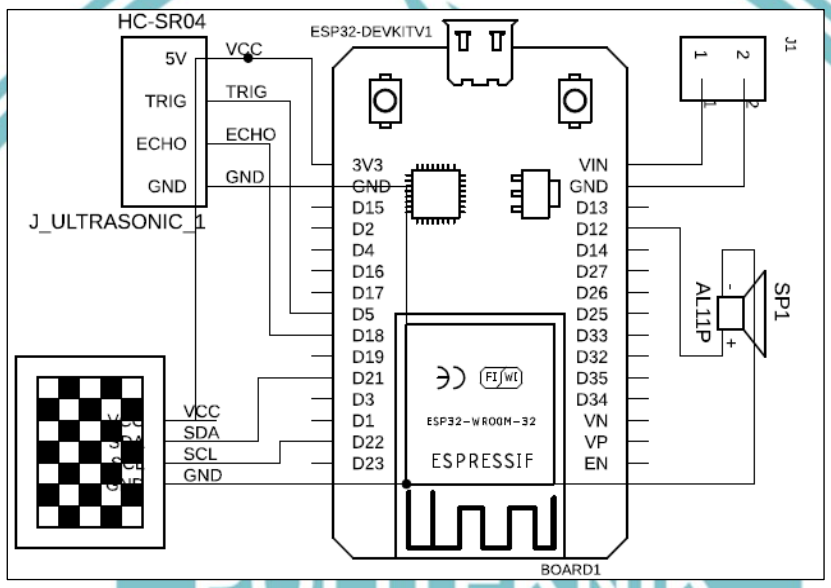
Gambar 3.3 *Flowchart* Alat Sistem *Monitoring Level* Air dalam Galon

3.1.5 Perencanaan Alat

a. Perencanaan Mikrokontroller

Sebelum masuk ke proses realisasi, alat terlebih dahulu perlu dilakukan perencanaan alat dengan merancang terlebih dahulu. Koneksi pin dari beberapa komponen utama yang digunakan dalam perancangan alat, yaitu Sensor Ultrasonik JSN SR04T, LCD OLED, dan *Buzzer*, yang semuanya terhubung ke ESP32. Pin *trig* dari Sensor Ultrasonik JSN SR04T terhubung ke pin 5 ESP32, sementara pin

echo terhubung ke pin 18. Pin SDA dari LCD OLED dihubungkan ke pin 21 ESP32, dan pin SCL ke pin 22. Selain itu, pin VCC dari Buzzer dihubungkan ke pin 12 ESP32. Terminal blok juga ditambahkan untuk mempermudah pemasangan tegangan dari catu daya, yang terhubung ke pin VIN dan GND pada ESP32. Pin-pin ini telah dideklarasikan dalam program dengan nilai const int trigPin = 5; const int echoPin = 18; const int buzzerPin = 12;, untuk memastikan koneksi sesuai dengan gambar yang ditampilkan. Gambar 3.4 merupakan gambar desain skematik alat sistem, dan Tabel 3.2 merupakan komponen yang terhubung pada pin ESP32.



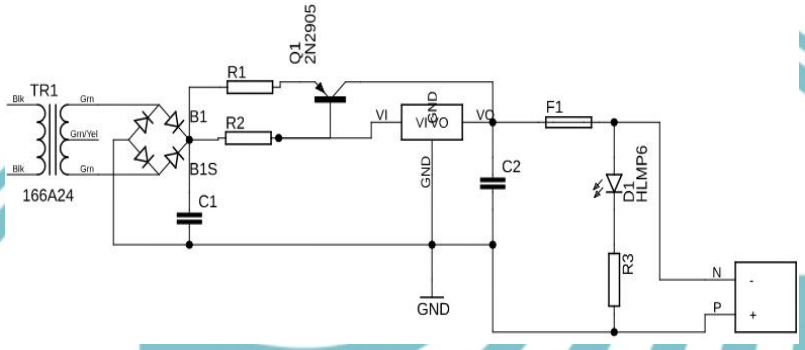
Gambar 3.4 Skematik Rangkaian Alat Sistem *Monitoring Level* Air dalam Galon

Tabel 3.2 Komponen Pada ESP32

No	Komponen	Pin Pada ESP	Keterangan
1	Sensor Ultrasonik JSN-SR04T	5	*Echo*
		18	*Trig*
2	LCD OLED	21	SDA
		22	SCL
3	Buzzer	12	VCC
4	Terminal Blok	VIN	VCC
		GND	GND

b. Perencanaan Catu Daya

Agar alat nantinya mempunyai sumber tegangan sendiri maka diperlukan catu daya sebagai penyuplai tegangan sehingga alat dapat bekerja dengan baik. Gambar 3.5 merupakan gambar rangkaian catu daya yang akan di realisasikan. Komponen-komponen yang digunakan yaitu, resistor 100 ohm, 0.1 ohm, 220ohm, LED, transistor TIP2955, Dioda Bridge, IC7805, Fuse 250V, dan Trafo 1A.



Gambar 3.5 Rangkaian Catu Daya

Power supply ini dirancang untuk mengonversi arus AC menjadi arus DC yang stabil menggunakan berbagai komponen utama. Transformator (trafo 1A) menurunkan arus AC dari sumber listrik, yang kemudian disearahkan oleh dioda bridge menjadi arus DC. IC 7805 digunakan sebagai regulator untuk menghasilkan tegangan 5V DC yang stabil. Transistor TIP2955 berfungsi sebagai penguat daya untuk meningkatkan arus *output* sesuai kebutuhan beban. LED, yang dipasangkan dengan resistor pembatas arus (100 Ohm atau 220 Ohm), berperan sebagai indikator operasional power supply. Fuse 250V melindungi rangkaian dari arus berlebih, sementara resistor 0.1 Ohm digunakan sebagai pengindra arus. Kombinasi komponen ini memastikan *output* tegangan yang andal dan aman untuk berbagai aplikasi elektronik.

3.1.6 Cara Kerja Alat

Sistem *monitoring level* air bekerja dengan menggunakan sensor *Ultrasonic* JSN-SR04T. Kemudian sensor *Ultrasonic* JSN-SR04T akan mengirimkan *input* an data ke *firebase*. Pada saat sensor membaca objek berketinggian < 20 cm akan memberikan data status air yaitu air penuh, saat berketinggian < 30 cm akan memberikan data status air yaitu air setengah, dan saat berketinggian > 30 cm akan

memberikan data status air yaitu air habis. Ketika air habis buzzer akan berbunyi dan akan muncul notifikasi pada aplikasi android.

3.1.7 Deskripsi Aplikasi

Aplikasi android ini memungkinkan pengguna untuk memantau ketinggian air dan status air secara *real-time*. Selain itu, aplikasi ini terintegrasi dengan *firebase.* Penjual dapat dengan mudah melihat lokasi pelanggan dan alamat mereka di peta, meningkatkan efisiensi dalam pengelolaan dan pelayanan.

3.1.8 Spesifikasi Aplikasi

Dalam pembuatan dan realisasi aplikasi, dibutuhkan perangkat lunak. Tabel 3.3 merupakan spesifikasi dari perangkat lunak.

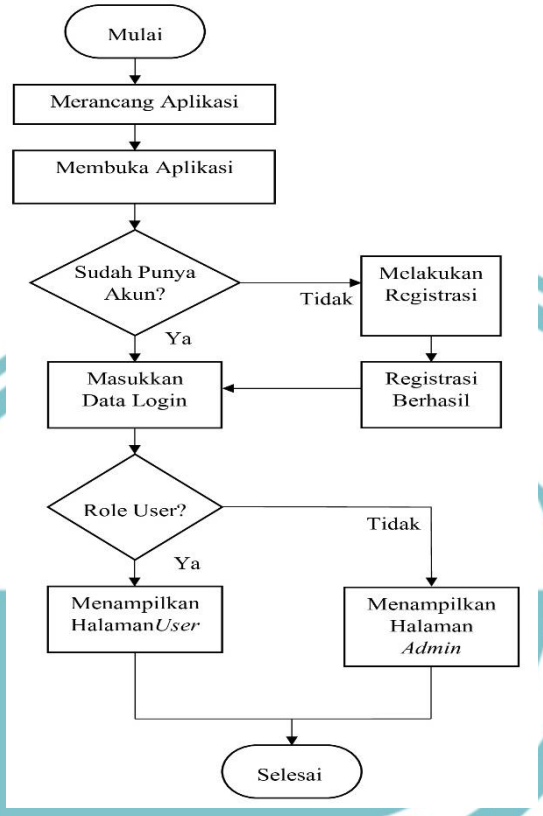
Tabel 3.3 Spesifikasi Perangkat Lunak

No	Perangkat	Spesifikasi	Keterangan
1	Arduino IDE	Versi	V 2.3.2
		Operating System	Windows/Mac/Linux
		Serial Monitor	115200
		Bahasa	C++
2	Android Studio	Android *Software* Development Kit	Android 13 Versi 34
		Android Gradel Plugin Version	Versi 8.5.0 Version 8.2.1
		Gradel Version	Version 33.0.2
		Build Tools Version	

3.1.9 *Flowchart* Aplikasi

Berikut adalah flowchart dari aplikasi android *monitoring level* air dalam galon untuk depot air minum berbasis android. *Flowchart* pada Gambar 3.6 menggambarkan alur proses penggunaan aplikasi mulai dari perancangan hingga akses halaman berdasarkan peran pengguna. Proses dimulai dengan perancangan dan dilanjutkan dengan membuka aplikasi. Sistem kemudian memeriksa apakah pengguna sudah memiliki akun; jika tidak, pengguna diminta untuk registrasi. Setelah login, sistem menentukan peran pengguna (*User atau Admin*) dan menampilkan halaman yang sesuai. Proses ini diakhiri dengan tampilan halaman

yang relevan bagi pengguna, menandakan selesai.



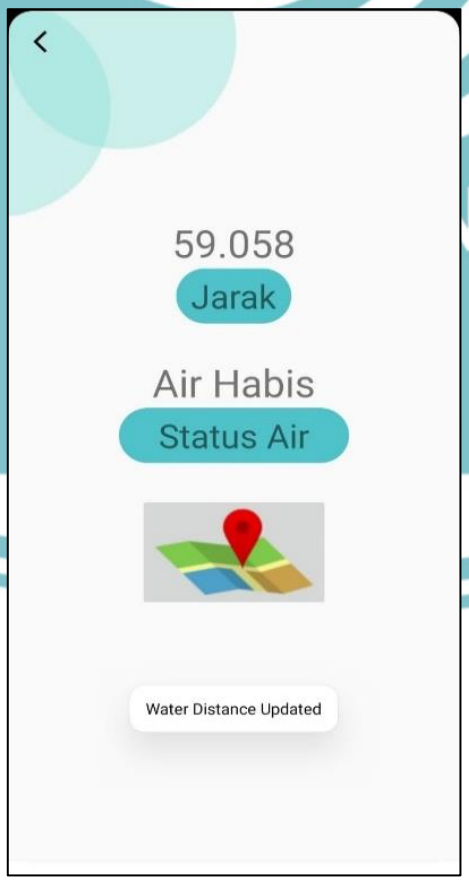
Gambar 3.6 Flowchart Aplikasi Android Sistem *Monitoring Level* Air dalam Galon

3.1.10 Cara Kerja Aplikasi

Ketika aplikasi dibuka, pengguna akan diarahkan ke halaman utama (*main activity*). Jika pengguna sudah memiliki akun, mereka dapat langsung masuk (*login*); namun, jika belum, mereka dapat melakukan registrasi terlebih dahulu. Pengguna yang sudah memiliki akun dan mengklik tombol masuk akan diarahkan ke halaman masuk (*login activity*), sementara pengguna yang memilih registrasi akan diarahkan ke halaman registrasi (*register activity*). Selanjutnya, data diri pengguna akan dibaca dan aplikasi akan menentukan tampilan berikutnya berdasarkan peran pengguna, yaitu *admin* atau *user*. Jika yang masuk adalah *admin*, akan diarahkan ke halaman *admin* (*admin activity*). Di halaman *admin*, terdapat tombol yang jika ditekan akan mengarahkan ke halaman *admin* kedua (*admin activity 2*) dan admin dapat mengamati data seluruh pengguna. Sebaliknya, jika yang masuk adalah *user*, mereka akan diarahkan ke halaman pengguna (*user activity*).

3.1.11 Perencanaan Aplikasi

Aplikasi android ini dirancang dengan tujuan untuk menyediakan berbagai halaman dan fitur yang memudahkan pengguna dalam mengelola dan memantau sistem. Halaman *splash screen* akan menyambut pengguna dengan antarmuka yang menarik saat aplikasi pertama kali dibuka. Selanjutnya, aplikasi akan menyediakan halaman *login* dan halaman *register* yang dirancang untuk memberikan akses yang aman dan terorganisir bagi pengguna baru maupun yang sudah terdaftar. Selain itu, aplikasi ini akan memiliki halaman *admin* yang memungkinkan melihat data pengguna, serta dapat melihat alamat dari *customer*. Halaman user yang memberikan akses kepada pengguna biasa untuk memantau status dan informasi yang relevan. Salah satu fitur utama aplikasi ini adalah notifikasi *pop-up* yang akan muncul ketika air habis, memberikan peringatan langsung kepada pengguna agar dapat segera mengambil tindakan. Gambar 3.7 merupakan ilustrasi halaman aplikasi android sistem *monitoring level* air.



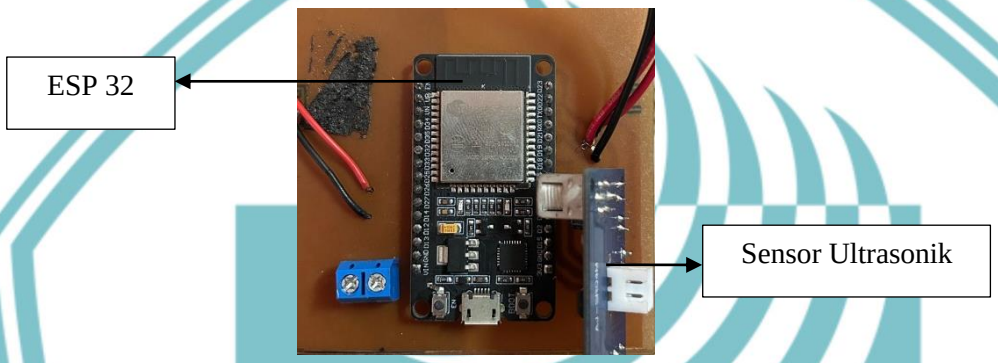
Gambar 3.7 Ilustrasi Aplikasi Android Sistem *Monitoring Level* Air dalam Galon

3.2 Realisasi Alat Sistem *Monitoring Level* Air

Realisasi alat ini adalah menjelaskan bagaimana *hardware* bekerja alat sistem *monitoring level* air dalam galon untuk depot air minum berbasis android.

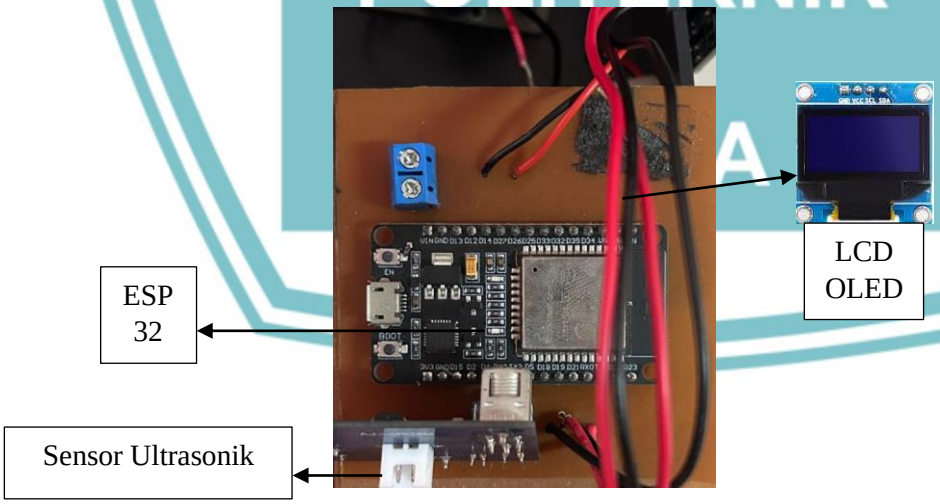
3.2.1 Realisasi Pin ESP32

Realisasi pin ESP32 dilakukan dengan menyolder pada pcb yang telah di fabrikasi. Gambar 3.8 merupakan realisasi pin Sensor Ultrasonik JSN-SR04T dengan ESP 32 yaitu pin *trig* terhubung ke pin 5 dan *echo* terhubung ke pin 18



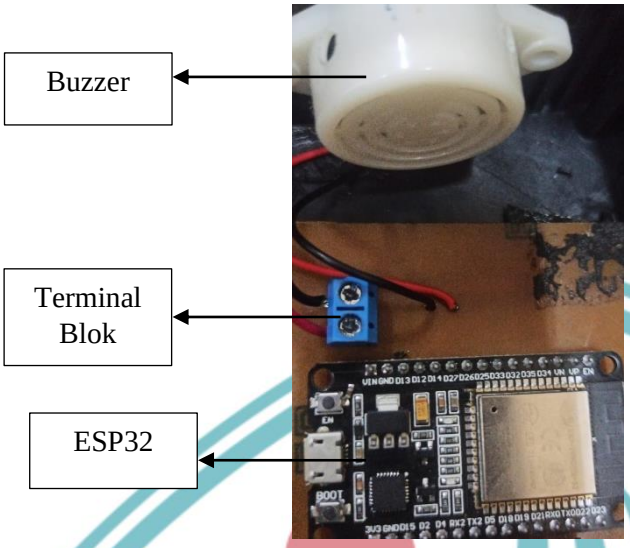
Gambar 3.8 Realisasi Pin Sensor Ultrasonik dengan ESP32

Berikutnya gambar 3.9 dilakukan realisasi pin LCD OLED dengan ESP32 yaitu pin SDA terhubung dengan pin 21 dan pin SCL terhubung dengan pin 22.



Gambar 3.9 Realisasi Pin LCD OLED dengan ESP 32

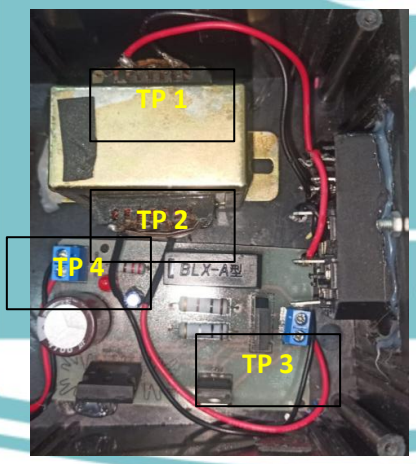
Berikutnya gambar 3.10 dilakukan realisasi pin *Buzzer* dengan ESP32 yaitu pin VCC terhubung dengan pin 12.



Gambar 3.10 Realisasi Pin Buzzer dengan ESP32

3.2.2 Realisasi Perangkat Catu Daya

Untuk menjalankan sistem "Rancang Bangun Sistem *Monitoring Level* Air Dalam Galon Untuk Depot Air Minum berbasis Android", digunakan catu daya dengan tegangan *output* sebesar 5 Volt DC. Tegangan ini diperlukan untuk menyalakan ESP32, dan komponen lainnya.



Gambar 3.11 Realisasi Catu Daya

Catu daya ini mengonversi tegangan AC menjadi DC yang stabil dengan trafo 1A dan dioda bridge. IC 7805 meregulasi tegangan menjadi 5V, sementara transistor TIP2955 meningkatkan arus *output*. LED sebagai indikator dilengkapi dengan resistor pembatas arus, dan fuse 250V melindungi dari arus berlebih. Resistor 0.1 Ohm berfungsi sebagai pengindra arus, menjadikan power supply ini

andal dan aman untuk berbagai aplikasi elektronik.

3.2.3 Realisasi Sistem *Library* pada ESP32

Realisasi sistem *library* kumpulan perintah yang akan digunakan untuk mengatur semua mikrokontoler. Pada penggunaan kurs roda ada beberapa *library* yang dibutuhkan seperti *Wire.h, WiFi.h, Adafruit_GFX.h, Adafruit_SSD1306.h, addons/TokenHelper.h, addons/RTDBHelper.h dan juga FirebaseESP32.h.* Semua library harus di import kedalam *Software* Arduino IDE agar program yang sudah dibuat dapat berjalan dengan baik, jika salah satu tidak terpasang maka semua sistem yang dirancang tidak dapat di jalankan

a. Inisiasi *Library*

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <FirebaseESP32.h>
#include "addons/TokenHelper.h"
#include "addons/RTDBHelper.h"
```

Program di atas adalah daftar *library* yang digunakan pada program yang memiliki fungsi agar mikrokontroler dapat berjalan dengan baik serta optimal. *Library* pada Arduino Ide adalah sebuah modul yang berisikan sekumpulan kode fungsi, *class* dan *variabel* yang berfungsi untuk mempermudah atau menyederhanakan suatu pemrograman.

1) *Wire.h* digunakan untuk mengimpor *library* Wire, yang diperlukan untuk komunikasi *Inter-Integrated Circuit* (I2C). *Library* ini digunakan untuk berkomunikasi dengan perangkat I2C, seperti sensor dan *display*.

2) *Adafruit_GFX.h* digunakan untuk mengimpor *library* Adafruit_GFX, yang berfungsi untuk menggambar grafik pada layar display.

3) *Adafruit_SSD1306.h* digunakan untuk mengimpor *library* Adafruit_SSD1306, yang dirancang khusus untuk mengendalikan layar OLED yang menggunakan driver SSD1306. *Library* ini berfungsi untuk menampilkan informasi pada layar OLED yang terhubung melalui I2C.

4) *FirebaseESP32.h* digunakan untuk mengimpor *library Firebase*ESP32, yang memungkinkan komunikasi dengan *Firebase*, platform backend yang menyediakan

database real-time, penyimpanan, otentikasi, dan lainnya. *Library* ini berfungsi untuk membuat ESP32 untuk terhubung dan berinteraksi dengan *Firebase*.

5) *Addons/TokenHelper.h* digunakan untuk engimpor pustaka TokenHelper, yang digunakan untuk membantu dalam pengelolaan token akses *Firebase*. Token ini diperlukan untuk otentikasi dan memastikan koneksi yang aman dengan *Firebase*.

6) *Addons/RTDBHelper.h* digunakan untuk mengimpor *library* RTDBHelper, yang mempunyai fungsi tambahan untuk berinteraksi dengan *Firebase* Realtime Database. Pustaka ini memudahkan operasi database yaitu membaca dan menulis data di *Firebase*.

b. Program *Library*

```
// Konfigurasi WiFi
#define WIFI_SSID "Telkom C" // Nama SSID WiFi
#define WIFI_PASSWORD "odyganteng" // Password WiFi
```

Bagian kode ini mengatur konfigurasi Wi-Fi untuk proyek. Dengan mendefinisikan dua parameter, yaitu “WIFI_SSID” dan “WIFI_PASSWORD”, mikrokontroler ESP32 dapat diatur untuk terhubung ke jaringan Wi-Fi tertentu. Dalam kode ini, nama SSID yang digunakan adalah "Telkom C", dan kata sandi Wi-Fi yang diperlukan adalah "odyganteng". Setelah parameter ini didefinisikan, pustaka “WiFi.h” dapat digunakan untuk menghubungkan ESP32 ke jaringan Wi-Fi yang sesuai.

```
// Masukkan API Key proyek Firebase
#define                                                    API_KEY
"zWdaKSRrZySUuTUAunCQ4Lm4Ha2dviEa2zcsjcQn"
// Masukkan URL RTDB Firebase
#define   DATABASE_URL   "https://water-level-monitoring-
7a5c6-default-rtdb.asia-southeast1.firebasedatabase.app/"
// Mendefinisikan objek Firebase
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
```

Bagian kode ini mengatur konfigurasi untuk menghubungkan mikrokontroler ESP32 dengan layanan *Firebase*. Kode ini dimulai dengan mendefinisikan kunci API proyek *Firebase*, yang diperlukan untuk otentikasi dan komunikasi dengan layanan *Firebase*.

Selain itu, URL dari *Firebase* Realtime Database (RTDB) menunjukkan

lokasi spesifik dari database. Kode ini juga mencakup deklarasi beberapa objek yang digunakan untuk berinteraksi dengan *Firebase*: *FirebaseData* untuk pengambilan dan penyimpanan data, untuk otentikasi *FirebaseAuth* pengguna, *FirebaseConfig* untuk menyimpan konfigurasi koneksi, dan *FirebaseJson* untuk mengubah data dalam format JSON. Pengaturan ini penting agar mikrokontroler ESP32 dapat terhubung dengan benar ke *Firebase* dan berfungsi sesuai dengan konfigurasi yang telah ditetapkan.

```
// Deklarasi pin
const int trigPin = 5;      // Pin trigger untuk sensor
jarak
const int echoPin = 18;     // Pin echo untuk sensor jarak
const int buzzerPin = 12;  // Pin buzzer
// Inisialisasi display OLED
#define SCREEN_WIDTH 128  // Lebar layar OLED
#define SCREEN_HEIGHT 64  // Tinggi layar OLED
#define OLED_RESET    -1  // Pin reset tidak digunakan
Adafruit_SSD1306   display(SCREEN_WIDTH,   SCREEN_HEIGHT,
&Wire, OLED_RESET);
```


```
// Deklarasi pin
const int trigPin = 5;      // Pin trigger untuk sensor
jarak
const int echoPin = 18;     // Pin echo untuk sensor jarak
const int buzzerPin = 12;  // Pin buzzer
// Inisialisasi display OLED
#define SCREEN_WIDTH 128  // Lebar layar OLED
#define SCREEN_HEIGHT 64  // Tinggi layar OLED
#define OLED_RESET    -1  // Pin reset tidak digunakan
Adafruit_SSD1306   display(SCREEN_WIDTH,   SCREEN_HEIGHT,
&Wire, OLED_RESET);
```


Bagian kode diatas berfungsi untuk mengatur pin untuk sensor jarak dan *buzzer*, dengan pin 5 sebagai *trigger*, pin 18 sebagai *echo*, dan pin 12 untuk *buzzer*. Selain itu, kode ini menginisialisasi layar OLED berukuran 128x64 piksel. Objek *Adafruit_SSD1306* diatur untuk mengontrol layar OLED menggunakan komunikasi I2C.

```
void setup() {
  // Inisialisasi pin
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
  // Inisialisasi komunikasi serial
  Serial.begin(115200);
// Menghubungkan ke Wi-Fi
```

Kode fungsi “setup()” diatas memulai proses inisialisasi perangkat keras dan koneksi jaringan. Pertama, konfigurasi pin dilakukan dengan menetapkan pin

trigger dan buzzer sebagai *output*, sedangkan pin echo sebagai *input*. Selanjutnya, komunikasi serial diinisialisasi dengan baud rate 115200 untuk memungkinkan pemantauan melalui serial monitor. Langkah berikutnya adalah menghubungkan mikrokontroler ke jaringan Wi-Fi menggunakan SSID dan kata sandi yang telah ditentukan. Proses koneksi dipantau dengan menampilkan pesan "*Connecting to Wi-Fi*" pada serial monitor dan menambahkan titik-titik setiap 300 milidetik hingga koneksi berhasil terjalin. Setelah koneksi Wi-Fi berhasil, alamat IP yang diperoleh akan ditampilkan pada serial monitor. Fungsi ini memastikan bahwa perangkat keras dan jaringan diatur dengan benar sebelum melanjutkan ke tahap berikutnya dalam operasi program.

```
Serial.println();
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  Serial.println();

  // Inisialisasi tampilan OLED
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Stop jika alokasi gagal
  }
  display.display();
  delay(2000); // Tunggu 2 detik
  display.clearDisplay();
```

Setelah koneksi Wi-Fi berhasil, alamat IP yang diperoleh ditampilkan pada serial monitor untuk memastikan bahwa perangkat telah terhubung ke jaringan. Selanjutnya, inisialisasi layar OLED dilakukan. Jika inisialisasi gagal, pesan "SSD1306 allocation failed" akan ditampilkan pada serial monitor dan program akan berhenti berfungsi. Jika inisialisasi berhasil, layar OLED akan menampilkan informasi selama 2 detik sebelum dibersihkan. Langkah-langkah ini memastikan bahwa perangkat telah terhubung ke jaringan Wi-Fi dengan benar dan bahwa layar OLED siap untuk menampilkan informasi lebih lanjut selama operasi program.

```
 // Konfigurasi Firebase
                                          config.api_key
=  "AIzaSyBWXBB2ZBNWMeMsjBBrpGxzlvfNy82vkKs";
  config.database_url = "https://water-level-monitoring-
7a5c6-default-rtdb.asia-
southeast1.firebasedatabase.app/";

   auth.user.email = "ferdinandardhi0907@gmail.com"; //
Ganti dengan email Firebase Anda
  auth.user.password = "admin123"; // Ganti dengan password
```

```
   // Menetapkan fungsi callback untuk pembuatan token
jangka panjang
  config.token_status_callback = tokenStatusCallback; //
lihat addons/TokenHelper.h

  // Inisialisasi Firebase
  Firebase.begin(&config, &auth); // Menggunakan pointer
ke config dan auth
  Firebase.reconnectWiFi(true);
}
```

Konfigurasi Firebase dimulai dengan menetapkan kunci API dan URL *database* yang sesuai untuk proyek. Kunci API digunakan untuk otentikasi, sedangkan URL database menunjuk ke lokasi spesifik di *Firebase Realtime Database*. Setelah itu, informasi otentikasi pengguna diatur dengan menetapkan email dan kata sandi yang diperlukan untuk akses ke Firebase. Fungsi *callback* untuk pembuatan token jangka panjang juga ditetapkan menggunakan "config.token_status_callback", yang merujuk pada file "TokenHelper.h". Langkah terakhir adalah inisialisasi Firebase menggunakan konfigurasi dan otentikasi yang telah ditetapkan dengan "Firebase.begin(&config, &auth)", serta memastikan koneksi Wi-Fi tetap aktif dengan "Firebase.reconnectWiFi(true)". Proses ini memastikan bahwa perangkat dapat berkomunikasi dengan Firebase secara aman dan berkelanjutan.

```
void loop() {
  // Mengukur jarak
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
long duration = pulseIn(echoPin, HIGH);
   float distance = duration * 0.034 / 2; // Menghitung
jarak dalam cm
String waterStatus;

  Serial.print("Jarak: ");
  Serial.print(distance);
  Serial.println(" cm");

  // Menampilkan jarak pada OLED
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print("Jarak: ");
```

```
    display.print(distance);
  display.println(" cm");
```

Fungsi "*loop()*" berfungsi untuk mengukur ketinggian dan menampilkan hasilnya pada layar OLED. Proses dimulai dengan mengirimkan sinyal trigger dari pin sensor jarak; pin trigger diatur ke level rendah selama 2 mikrodetik, kemudian ke level tinggi selama 10 mikrodetik, dan kembali ke level rendah untuk memulai pengukuran. Durasi pulsa yang diterima pada pin *echo* dihitung menggunakan fungsi "*pulseIn()"*, dan ketinggian dihitung berdasarkan durasi tersebut dengan rumus "distance = duration x 0.034 / 2", menghasilkan jarak dalam sentimeter.

Hasil pengukuran ketinggian kemudian ditampilkan di serial monitor dengan format "Ketinggian: [nilai] cm". Selain itu, layar OLED diperbarui untuk menampilkan informasi tersebut. Layar OLED dibersihkan terlebih dahulu dengan "*display.clearDisplay()*", kemudian teks diatur dengan ukuran 1 dan warna putih. Posisi kursor diatur di sudut kiri atas layar, dan informasi ketinggian dicetak dengan format yang sama seperti di serial monitor.

```
if (distance <= 20) {
    waterStatus = "Air Penuh";
  } else if (distance <= 30) {
    waterStatus = "Air Setengah";
  } else {
    waterStatus = "Air Habis";

  }

  display.setCursor(0, 10);
  display.println("Status: " + waterStatus);

  display.display();

  if (distance >= 45) {
    digitalWrite(buzzerPin, HIGH);
  } else {
    digitalWrite(buzzerPin, LOW);
  }

  // Mengirim data ke Firebase
  sendDataToFirebase(distance, waterStatus);

  // Delay sebelum pembacaan berikutnya
  delay(1000);
}
```

Kode ini melanjutkan fungsi "loop()" dengan menentukan status air berdasarkan ketinggian yang diukur oleh sensor. Jika ketinggian yang terdeteksi

kurang dari atau sama dengan 20 cm, maka status air ditetapkan sebagai "Air Penuh". Jika ketinggian berada dalam rentang 21 hingga 30 cm, statusnya adalah "Air Setengah". Untuk ketinggian lebih dari 30 cm, status air akan menjadi "Air Habis". Informasi mengenai status air ini kemudian ditampilkan pada layar OLED tepat di bawah pengukuran ketinggian.

Fungsi ini juga mengatur *buzzer*, di mana *buzzer* akan aktif jika ketinggian mencapai 30 cm atau lebih, dengan menyalakan pin *buzzer*. Sebaliknya, jika ketinggian kurang dari 30 cm, *buzzer* akan dimatikan. Selain itu, data ketinggian dan status air dikirim ke *Firebase* menggunakan fungsi "*sendDataToFirebase()*". Sebagai bagian dari siklus pembacaan, program memberikan jeda selama 1 detik sebelum memulai pembacaan ketinggian berikutnya.

```
void sendDataToFirebase(float distance, String
waterStatus) {
  // Membuat objek JSON untuk menyimpan data
  FirebaseJson json;
  json.set("distance", distance);
  json.set("waterStatus", waterStatus);
// Mengirim data ke Firebase
    if (Firebase.setJSON(fbdo, "/Customer1SensorData",
json)) {
    Serial.println("Data terkirim ke Firebase");
  } else {
    Serial.print("Gagal mengirim data: ");
    Serial.println(fbdo.errorReason());
  }
}
```

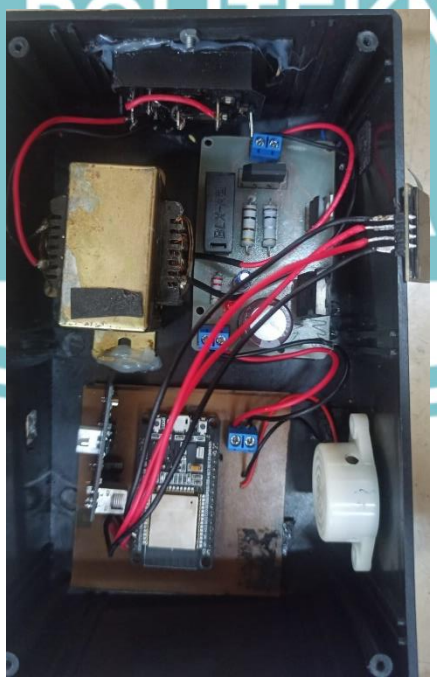
Fungsi "*sendDataToFirebase()*" bertanggung jawab untuk mengirimkan data ketinggian dan status air ke *Firebase*. Proses dimulai dengan membuat objek JSON menggunakan "*FirebaseJson*" untuk menyimpan informasi yang akan dikirim. Data yang disertakan dalam objek JSON mencakup nilai ketinggian dan status air, yang diambil dari parameter fungsi.

Setelah objek JSON siap, data tersebut dikirim ke *Firebase* menggunakan fungsi "*Firebase.setJSON()*", dengan path "/Customer1SensorData" sebagai lokasi penyimpanan dalam database *Firebase*.

Jika pengiriman data berhasil, pesan "Data terkirim ke *Firebase*" akan ditampilkan pada serial monitor. Namun, jika pengiriman data gagal, akan muncul pesan kesalahan yang disertai dengan alasan kegagalan dari objek "*FirebaseData*".

3.2.4 Pengintegrasian Mikrokontroller dengan Catu Daya

Pada tahap ini, alat sistem dan catu daya telah selesai direalisasikan, dan langkah berikutnya adalah menyambungkannya secara keseluruhan. Proses ini dimulai dengan catu daya menerima tegangan AC dari sumber listrik utama. Tegangan AC ini kemudian diolah oleh catu daya, yang berfungsi mengubah tegangan AC menjadi tegangan DC yang stabil dan sesuai dengan kebutuhan komponen elektronik yang akan digunakan. Keluaran tegangan DC dari catu daya ini selanjutnya akan disalurkan ke ESP32, yang merupakan pusat kendali dari seluruh sistem. Setelah ESP32 menerima tegangan DC yang sesuai, ia akan mendistribusikan tegangan tersebut ke semua komponen lain yang terhubung, seperti sensor, modul komunikasi, dan komponen *output* lainnya, memastikan bahwa setiap bagian sistem mendapatkan suplai daya yang cukup untuk beroperasi dengan baik. Dengan distribusi tegangan yang tepat ini, seluruh sistem siap untuk dioperasikan dan melaksanakan fungsi yang telah dirancang sebelumnya.



Gambar 3.12 Rangkaian Mikrokontroler
Terintegrasi dengan Catu Daya

3.3 Realisasi Aplikasi

Program aplikasi ini dirancang dengan menggunakan Android Studio. Aplikasi yang dibuat diberi nama "Water Level Monitoring". Aplikasi tersebut digunakan untuk menampilkan hasil pembacaan sensor ultrasonik, water status, dan google maps untuk menunjukkan alamat pelanggan. Smartphone yang mempunyai aplikasi "Water Level Monitoring" harus terhubung dengan internet agar dapat mendapatkan informasi ter-*úpdate*.

3.3.1 Realisasi *Layout* Aplikasi Android

1) Membuat Tampilan *Splash Screen*

Splash Screen adalah tampilan awal yang ditampilkan saat aplikasi pertama kali dibuka. Bagian ini hanya menampilkan gambar dan ditampilkan selama 3 detik.

Gambar 3.13 Desain *Splash Screen* Aplikasi Android Sistem *Monitoring Level* Air dalam Galon

Untuk membuat *splash screen,* terlebih dahulu menentukan desain yang akan ditampilkan.

```
<ImageView
      android:id="@+id/splash_image"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerInParent="true"
      android:src="@drawable/logo"/>
```

Kode diatas merupakan kode xml di android studio untuk menampilkan *layout splash screen.*

2) Membuat Tampilan *Main Activity*

Main Activity adalah tampilan yang muncul setelah tampilan *splash screen*. Halaman *Main Activity* akan menampilkan tombol untuk *login, register, dan* halaman *about me*.

```
<Button
    android:layout_width="wrap_content"
    android:layout_marginTop="25dp"
    android:layout_height="wrap_content"
    android:layout_below="@id/txt_2"
    android:width="300dp"
    android:layout_centerHorizontal="true"
    android:text="Login"
    android:background="@drawable/buttonshape"
    android:textColor="@color/white"
    android:textSize="14sp"
    android:fontFamily="@font/poppinsbold"
    android:id="@+id/btn_1"
    />
<Button
    android:layout_width="wrap_content"
    android:layout_marginTop="10dp"
    android:layout_height="wrap_content"
    android:layout_below="@id/btn_1"
    android:width="300dp"
    android:layout_centerHorizontal="true"
    android:text="Register"
    android:background="@drawable/buttonshape"
    android:textColor="@color/white"
    android:textSize="14sp"
    android:fontFamily="@font/poppinsbold"
    android:id="@+id/btn_2"
    />
<Button
    android:layout_width="wrap_content"
    android:layout_marginTop="10dp"
    android:layout_height="wrap_content"
    android:layout_below="@id/btn_2"
    android:width="300dp"
    android:layout_centerHorizontal="true"
    android:text="About Me"
    android:background="@drawable/buttonshape"
    android:textColor="@color/white"
    android:textSize="14sp"
    android:fontFamily="@font/poppinsbold"
    android:id="@+id/btn_3"
    />
```

Kode diatas merupakan kode xml pada android studio untuk mendesain *layout main activity* yang menampilkan *button login, register,* dan *about me* serta tampilan pendukung lainnya.

3) Membuat Tampilan *Login Activity*

Login Activity adalah tampilan ketika *button login* ditekan. Halaman ini membantu admin dan *user* untuk masuk ke akun mereka masing masing.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:background="@drawable/buttonshape"
    android:layout_marginTop="10dp"
    android:hint="Enter Your Email"
    android:paddingLeft="10dp"
    android:backgroundTint="@color/white"
    android:fontFamily="@font/poppinsregular"
    android:id="@+id/edt_email"
    />
<EditText
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:background="@drawable/buttonshape"
    android:layout_marginTop="10dp"
    android:hint="Confirm Password"
    android:paddingLeft="10dp"
    android:backgroundTint="@color/white"
    android:fontFamily="@font/poppinsregular"
    android:id="@+id/edt_conf_password"
    android:inputType="textPassword"
```

Kode diatas merupakan kode xml di android studio untuk mendesain *login activity* yang menampilkan kolom untuk memasukkan *email* dan *password*.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Login"
    android:layout_below="@id/ly_1"
    android:layout_centerHorizontal="true"
    android:background="@drawable/buttonshape"
    android:width="300dp"
    android:textColor="@color/white"
    android:textSize="18sp"
    android:id="@+id/L_btn_1"
    android:fontFamily="@font/poppinsbold"
    />
```

Kode diatas merupakan lanjutan kode xml di android studio untuk mendesain *login activity* yang menampilkan *button login* dibawah kolom untuk memasukkan *email* dan *password.*

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Not have an account?"
    android:fontFamily="@font/poppinsregular"
    android:textSize="12sp"
    />

<!-- Teks dengan link untuk registrasi, di sebelah kanan
teks sebelumnya -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Register"
    android:fontFamily="@font/poppinsbold"
    android:textSize="12sp"
    android:textColor="@color/secondarycolor"
    android:id="@+id/txt_register"
    />
```

Kode diatas merupakan lanjutan kode xml di android studio untuk mendesain *login activity* yang menampilkan *button* untuk mengalihkan ke halaman registrasi jika pengguna belum memiliki akun.

4) Membuat tampilan *Register Activity*

Register Activity adalah tampilan ketika button register ditekan. Halaman ini membantu untuk membuat akun.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:background="@drawable/buttonshape"
    android:layout_marginTop="10dp"
    android:hint="Enter Your Email"
    android:paddingLeft="10dp"
    android:backgroundTint="@color/white"
    android:fontFamily="@font/poppinsregular"
    android:id="@+id/edt_email"/>
<EditText
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:background="@drawable/buttonshape"
    android:layout_marginTop="10dp"
    android:hint="Create Password"
    android:paddingLeft="10dp"
    android:backgroundTint="@color/white"
    android:fontFamily="@font/poppinsregular"
    android:id="@+id/edt_password"
    android:inputType="textPassword"/>
<EditText
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:background="@drawable/buttonshape"
```

```
android:layout_marginTop="10dp"
android:backgroundTint="@color/white"
android:paddingLeft="10dp"
android:fontFamily="@font/poppinsregular"
android:hint="Role (User)"
android:id="@+id/edt_role"/>
```

Kode diatas merupakan kode xml di android studio untuk mendesain *register activity* yang menampilkan kolom untuk memasukkan *email, password* dan *role* yang akan didaftarkan.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Register"
    android:layout_below="@id/ly_1"
    android:layout_centerHorizontal="true"
    android:background="@drawable/buttonshape"
    android:width="300dp"
    android:textColor="@color/white"
    android:textSize="18sp"
    android:id="@+id/R_btn_1"
    android:fontFamily="@font/poppinsbold"/>
```

Kode diatas merupakan lanjutan kode xml di android studio untuk mendesain *register activity* yang menampilkan *button* untuk menyelesaikan registrasi.

5) Membuat Tampilan Admin Activity

Admin Activity adalah tampilan ketika yang *login* adalah *role admin*. Halaman ini akan menampilkan tampilan data *level* dan status air milik *customer*.

```
<TextView
    android:id="@+id/datajarakCustomer1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="0"
    android:layout_marginTop="50dp"
    android:layout_centerHorizontal="true"
    android:textSize="25dp"
    android:fontFamily="@font/poppinsregular"
    android:textColor="@color/black" />
```

Kode diatas merupakan kode xml di android studio untuk mendesain *admin activity* yang menampilkan ketinggian air dari *customer 1*.

```
<TextView
    android:id="@+id/dataStatusCustomer1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="0"
    android:layout_below="@id/textJarakCustomer1"
    android:layout_marginTop="10dp"
    android:layout_centerHorizontal="true"
    android:textSize="25dp"
    android:fontFamily="@font/poppinsregular"
    android:textColor="@color/black" />
```

Kode diatas merupakan lanjutan kode xml di android studio untuk mendesain *admin activity* yang menampilkan status air dari *customer 1*.

```
<RadioButton
    android:id="@+id/textNavigateMe"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Navigate Me!"
    android:layout_below="@id/textAlamatCustomer1"
    android:layout_marginTop="5dp"
    android:layout_centerHorizontal="true"
    android:textSize="25dp"
    android:fontFamily="@font/poppinsbold" />
```

Kode diatas merupakan lanjutan kode xml di android studio untuk mendesain *admin activity* yang menampilkan *button* yang berfungsi untuk navigasi ke alamat *customer 1* melalui *google maps*.

6) Membuat Tampilan User Activity

User Activity adalah tampilan ketika *button login* ditekan. Halaman ini akan menampilkan tampilan jika *role* yang *login* adalah *role user*.

```
<TextView
    android:id="@+id/textUser1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="User 1"
    android:layout_marginTop="50dp"
    android:layout_centerHorizontal="true"
    android:textColor="@color/black"
    android:textSize="30dp"
    android:fontFamily="@font/poppinsextra" />
```

Kode diatas merupakan kode xml di android studio untuk mendesain *user activity* yang menampilkan tulisan sebagai penanda akun user berapa yang sedang dibuka.

```
<TextView
    android:id="@+id/datajarakAirKamu1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="30dp"
    android:fontFamily="@font/poppinsregular"
    android:text="0"
    android:textColor="@color/black"
    android:textSize="25dp"
    android:layout_below="@id/textUser1" />
```

Kode diatas merupakan lanjutan kode xml di android studio untuk mendesain *user activity* yang menampilkan ketinggian air dari *user 1*.

```
<TextView
    android:id="@+id/dataStatusAirKamu1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="0"
    android:layout_below="@id/textJarakAirKamu1"
    android:layout_marginTop="30dp"
    android:layout_centerHorizontal="true"
    android:textSize="25dp"
    android:fontFamily="@font/poppinsregular"
    android:textColor="@color/black" />
```

Kode diatas merupakan lanjutan kode xml di android studio untuk mendesain *user activity* yang menampilkan status air dari *user 1*.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Antar Sekarang"
    android:layout_below="@id/textStatusAirKamu1"
    android:layout_marginTop="100dp"
    android:background="@drawable/buttonshape"
    android:width="300dp"
    android:textSize="18sp"
    android:fontFamily="@font/poppinsbold"
    android:textColor="@color/white"
    android:layout_centerHorizontal="true"
    android:id="@+id/buttonAntarSekarangUser1"/>
```

Kode diatas merupakan lanjutan kode xml di android studio untuk mendesain *user activity* yang menampilkan *button* pada halaman *user 1*. *Button* tersebut berfungsi untuk mengirimkan pesan berupa notfikiasi ke *admin* jika *user* ingin melakukan pengantaran air.

3.3.2 Realisasi Fungsi dari *Layout* Aplikasi

Setelah desain aplikasi per halaman telah dibuat, masing masing activity akan di coding agar dapat berfungsi sesuai dengan yang diinginkan.

1) *Binding Splash Screen*

Berikut merupakan *coding* agar *Splash Screen* dapat ditampilkan diawal aplikasi dibuka.

```
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding =
ActivitySplashScreenBinding.inflate(layoutInflater)
        setContentView(binding.root)
        supportActionBar?.hide() // Menyembunyikan
ActionBar jika ada

        // Mengatur delay untuk splash screen (misalnya 3
detik)
        Handler(Looper.getMainLooper()).postDelayed({
            // Memulai MainActivity setelah delay
            startActivity(Intent(this,
MainActivity::class.java))
            finish() // Menutup SplashScreenActivity
        }, 3000) // 3000 milidetik = 3 detik
    }
}
```

Pada metode `onCreate` di *SplashScreenActivity*, tampilan layar splash diinisialisasi dengan `ActivitySplashScreenBinding` dan *ActionBar* disembunyikan dengan `supportActionBar?.hide()`. Sebuah `Handler` digunakan untuk menunda selama 3 detik sebelum memulai *MainActivity* menggunakan `startActivity(Intent(this, MainActivity::class.java))`, dan kemudian menutup `SplashScreenActivity dengan finish()`. Ini memastikan pengguna langsung dialihkan ke *MainActivity* setelah layar *splash* selesai.

2) Fungsi *Binding Login Button* di *Main Activity* ke *Login Activity, Register Button* ke *Register Activity,* dan *About Me Button* ke *About Me Activity.*

```
// Metode onCreate dipanggil saat aktivitas dibuat
override fun onCreate(savedInstanceState: Bundle?) {
    // Inisialisasi binding dengan layout inflater
    binding = ActivityMainBinding.inflate(layoutInflater)
    super.onCreate(savedInstanceState)
    // Mengatur tampilan konten ke root dari binding
    setContentView(binding.root)
```

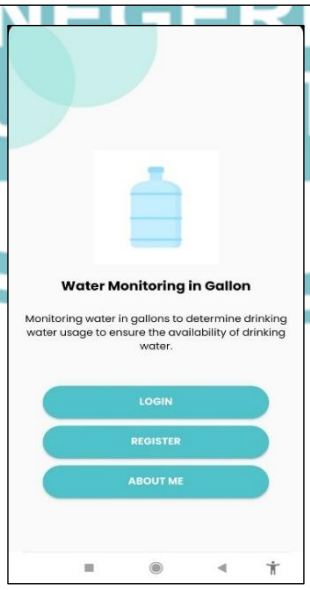
Metode `onCreate` diaktifkan saat aktivitas dibuat, menginisialisasi binding dengan `ActivityMainBinding.inflate(layoutInflater)`. Setelah memanggil `super.onCreate(savedInstanceState)`, tampilan konten diatur ke root dari binding menggunakan `setContentView(binding.root)`, menampilkan layout yang telah dihubungkan.

```
        binding.btn1.setOnClickListener {

            val intent = Intent(this,
LoginActivity::class.java)

            startActivity(intent)
        }
        binding.btn2.setOnClickListener {
            // Membuat intent untuk membuka
RegisterActivity
            val intent = Intent(this,
RegisterActivity::class.java)

            startActivity(intent)
        }
        binding.btn3.setOnClickListener {
            // Membuat intent untuk membuka
AboutMeActivity
            startActivity(Intent(this,
AboutMeActivity::class.java))
            // Mengakhiri MainActivity
            finish()
        }
    }
}
```



Gambar 3.14 Halaman *Main Activity* Aplikasi Android Sistem *Monitoring Level* Air dalam Galon

Kode ini mengatur tindakan klik untuk tiga tombol. `btn1` membuka *LoginActivity*, `btn2` membuka *RegisterActivity*, dan `btn3` membuka *AboutMeActivity* serta menutup *MainActivity*.

3) Fungsi *Binding Button di Login Activity*.

Berikut merupakan *coding* agar pengguna aplikasi dapat *login* kedalam aplikasi dan diarahkan ke halaman sesuai dengan role nya.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
        binding =
ActivityLoginBinding.inflate(layoutInflater)
    setContentView(binding.root)
    supportActionBar?.hide(
```

Pada metode `onCreate` di *LoginActivity*, layout XML dihubungkan dengan kode Kotlin menggunakan `ActivityLoginBinding.inflate(layoutInflater)` dan ditetapkan sebagai tampilan utama dengan `setContentView(binding.root)`. *ActionBar* disembunyikan dengan `supportActionBar?.hide()` untuk memberikan tampilan layar login yang lebih bersih.

```
// Menginisialisasi Firebase Authentication dan Firestore
auth = FirebaseAuth.getInstance()
db = FirebaseFirestore.getInstance()
```

Kode diatas menginisialisasi Firebase Authentication dan Firestore. `FirebaseAuth.getInstance()` digunakan untuk autentikasi pengguna, sedangkan `FirebaseFirestore.getInstance()` menyediakan akses ke *database cloud Firestore*.

```
    binding.edtConfPassword.inputType =
        InputType.TYPE_CLASS_TEXT or
InputType.TYPE_TEXT_VARIATION_PASSWORD
    binding.txtRegister.setOnClickListener {
        // Membuka Activity Register
        startActivity(Intent(this,
RegisterActivity::class.java))
        finish() // Menutup LoginActivity
    }
    binding.btnBack.setOnClickListener {
        // Membuka MainActivity
        startActivity(Intent(this,
MainActivity::class.java))
        finish() // Menutup LoginActivity
    }
```

```
    binding.LBtn1.setOnClickListener {
        // Mengambil email dan password dari input
pengguna
        val email = binding.edtEmail.text.toString()
        val password =
binding.edtConfPassword.text.toString()
        // Memanggil fungsi untuk login pengguna
        loginUser(email, password)
```

Kode diatas mengatur `edtConfPassword` sebagai *input* teks sandi. Saat `txtRegister` diklik, aplikasi beralih ke *RegisterActivity* dan menutup aktivitas saat ini. Ketika `LBtn1` diklik, email dan password diambil dari `edtEmail` dan `edtConfPassword`, lalu fungsi `loginUser(email, password)` dipanggil untuk autentikasi pengguna.

```
private fun loginUser(email: String, password: String) {
    if (email.isNotEmpty() && password.isNotEmpty()) {
        auth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
                                        checkUserRole()
                } else {
                    Toast.makeText(
                        this,
                        "Login gagal:
${task.exception?.message}",
                        Toast.LENGTH_SHORT
                    ).show()
                }
            }
    } else {
        Toast.makeText(this, "Harap isi semua kolom",
Toast.LENGTH_SHORT).show()
    }
}
```

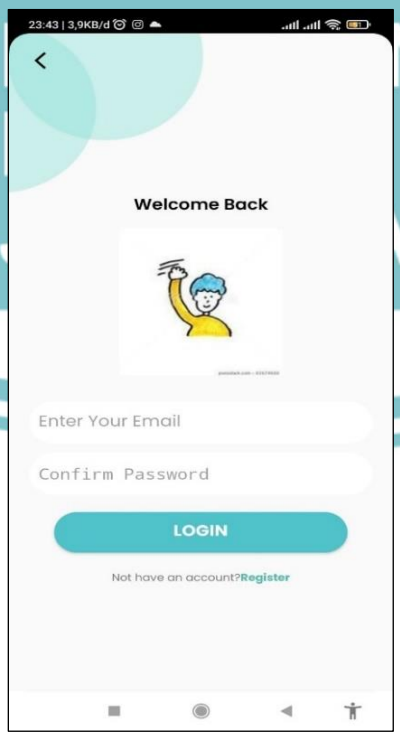
Fungsi `loginUser` memeriksa apakah email dan password tidak kosong sebelum mencoba *login*. Jika *login* berhasil, fungsi `checkUserRole()` dipanggil untuk menentukan peran pengguna. Jika *login* gagal atau jika kolom kosong, pesan kesalahan ditampilkan dengan `Toast`.

```
    private fun checkUserRole() {
        val userId = auth.currentUser?.uid
        if (userId != null) {
            berdasarkan userId
            db.collection("users").document(userId).get()
                .addOnSuccessListener { document ->
                    if (document != null) {
                                                val role
= document.getString("role")
                        if (role != null) {
```

```
                                if (role == "Admin") {
startActivity(Intent(this, AdminActivity::class.java))
                                    finish()
} else if (role == "User1") {  startActivity(Intent(this,
UserActivity::class.java))
                                    finish()
}else if (role == "User2") {

startActivity(Intent(this, UserActivity2::class.java))
                                    finish()
} else {

Toast.makeText(this, "Role tidak dikenal",
Toast.LENGTH_SHORT)
                                        .show()
                                }
                            }
                        } else {
                            Toast.makeText(this, "Dokumen
pengguna tidak ditemukan", Toast.LENGTH_SHORT)
                                .show()
                        }
                    }
                    .addOnFailureListener { exception ->
                        Toast.makeText(this, "Error:
${exception.message}", Toast.LENGTH_SHORT).show()
                    }
            }
        }
    }
```



Gambar 3.15 Tampilan *Login Activity* Aplikasi Android Sistem Monitoring Level Air dalam Galon

Fungsi `checkUserRole` memeriksa peran pengguna berdasarkan `userId`. Jika pengguna ada di Firestore, fungsi ini akan membuka aktivitas yang sesuai (*AdminActivity*, *UserActivity*, atau *UserActivity2*) tergantung pada peran pengguna. Jika peran tidak dikenal atau terjadi kesalahan, aplikasi menampilkan pesan kesalahan.

4) Fungsi *Binding Button di Register Activity.*

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

    auth = FirebaseAuth.getInstance()
    firestore = FirebaseFirestore.getInstance()
    supportActionBar?.hide()
    binding.edtPassword.inputType =
InputType.TYPE_CLASS_TEXT or
InputType.TYPE_TEXT_VARIATION_PASSWORD
    binding.txtLogin.setOnClickListener {
        startActivity(Intent(this,
LoginActivity::class.java))
        finish()
    }
```

Pada metode `onCreate`, aktivitas *RegisterActivity* diinisialisasi dengan menghubungkan layout XML menggunakan `ActivityRegisterBinding.inflate(layoutInflater)` dan mengatur tampilan utama dengan `setContentView(binding.root)`. `FirebaseAuth` dan `FirebaseFirestore` diinisialisasi untuk menangani autentikasi pengguna dan akses *database*. *ActionBar* disembunyikan dengan `supportActionBar?.hide()` untuk tampilan yang lebih bersih. `edtPassword` diatur sebagai *input* teks sandi dengan `InputType.TYPE_CLASS_TEXT`. Selain itu, ketika `txtLogin` diklik, aplikasi berpindah ke *LoginActivity* dan *RegisterActivity* ditutup dengan `finish()` untuk mengakhiri aktivitas saat ini.

```
binding.RBtn1.setOnClickListener {
        val email = binding.edtEmail.text.toString()
val password = binding.edtPassword.text.toString()
val role = binding.edtRole.text.toString())
        registerUser(email, password, role)
    }
```

Kode diatas memberikan perintah etika `RBtn1` diklik, aplikasi mengambil nilai *email*, *password*, dan peran dari `edtEmail`, `edtPassword`, dan `edtRole`, lalu memanggil `registerUser(email, password, role)` untuk mendaftarkan pengguna.

```
private fun registerUser(email: String, password: String,
role: String) {
    if (email.isNotEmpty() && password.isNotEmpty() &&
role.isNotEmpty()) {
        auth.createUserWithEmailAndPassword(email,
password)
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
val user = auth.                    val userId =
user?.uid
                    val userData = hashMapOf(
                        "email" to email,
                        "role" to role
)
```

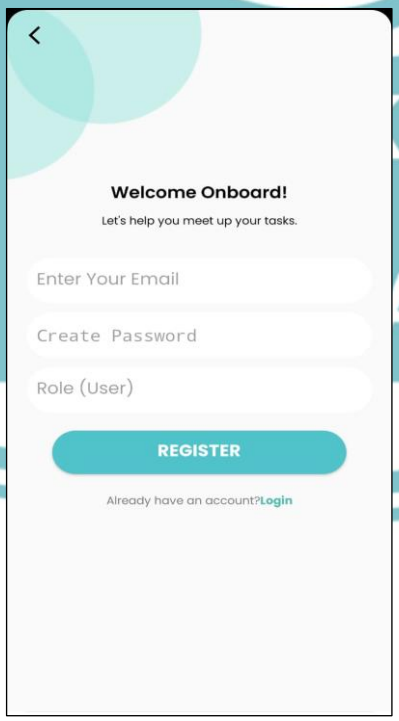
Fungsi `registerUser` pada kode diatas yaitu mendaftarkan pengguna jika *email*, *password*, dan role terisi. Jika registrasi berhasil, fungsi mengambil ID pengguna dan menyimpan data pengguna (*email dan role*) dalam `userData` menggunakan `hashMapOf`.

```
userId?.let {
    firestore.collection("users").document(it) // Akses
koleksi "users" di Firestore
        .set(userData) // Simpan data pengguna
        .addOnSuccessListener {
            Toast.makeText(this, "Registration
Successful!", Toast.LENGTH_SHORT).show() // Tampilkan
pesan sukses
            startActivity(Intent(this,
LoginActivity::class.java)) // Pindah ke aktivitas login
            finish() // Hentikan aktivitas ini setelah
berpindah
        }
        .addOnFailureListener { e ->
            Toast.makeText(this, "Failed to save user
data: ${e.message}", Toast.LENGTH_SHORT).show() //
Tampilkan pesan gagal
        }
}
```

Setelah registrasi berhasil, kode ini menyimpan data pengguna ke koleksi "*users*" di *Firestore*. Jika berhasil, aplikasi menampilkan pesan sukses, berpindah ke *LoginActivity*, dan menutup aktivitas saat ini. Jika gagal, aplikasi menampilkan pesan kesalahan.

```
} else {
                        Toast.makeText(this,
"Registration Failed: ${task.exception?.message}",
Toast.LENGTH_SHORT).show() // Tampilkan pesan gagal
                    }
                }
            } else {
                Toast.makeText(this, "Please fill in all
fields", Toast.LENGTH_SHORT).show() // Tampilkan pesan
jika ada field yang kosong
            }
        }
}
```

Jika pendaftaran gagal, aplikasi menampilkan pesan kesalahan dengan `Toast`. Jika ada field yang kosong, aplikasi menampilkan pesan peringatan "*Please fill in all fields*" dengan `Toast`.



Gambar 3.16 Tampilan *Register Activity*
Aplikasi Android Sistem *Monitoring*
Level Air dalam Galon

5) Fungsi *Binding Button* di *Admin Activity*.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding =
ActivityAdmin2Binding.inflate(layoutInflater)
    setContentView(binding.root)
    supportActionBar?.hide()
    val firebaseDatabase =
FirebaseDatabase.getInstance("https://water-level-
monitoring-7a5c6-default-rtdb.asia-
southeast1.firebasedatabase.app/")
    database = firebaseDatabase.reference

    binding.buttonBack.setOnClickListener {
        startActivity(Intent(this,
AdminActivity::class.java))
        finish()
    }
```

Pada metode `onCreate`, *Admin2Activity* diatur dengan `ActivityAdmin2Bindinxg`, *ActionBar* disembunyikan, dan koneksi ke *Firebase Realtime Database diinisialisasi*. Ketika `buttonBack` diklik, aplikasi berpindah ke *AdminActivity* dan *Admin2Activity* ditutup.

```
    readData()
    readStatus()

    val buttonMaps1: RadioButton =
findViewById(R.id.textNavigateMe)
    buttonMaps1.setOnClickListener {
        val url =
"https://maps.app.goo.gl/kccvtXPaD8EC1AT99?g_st=ac
val intent = Intent(Intent.ACTION_VIEW, Uri.parse(url))

        if (intent.resolveActivity(packageManager) !=
null) {
            startActivity(intent)
        } else {
            Toast.makeText(this, "Google Maps tidak
terinstal", Toast.LENGTH_SHORT).show()
        }
    }

    val buttonMaps2: RadioButton =
findViewById(R.id.textNavigateMe2)
    buttonMaps2.setOnClickListener {
        val url =
"https://maps.app.goo.gl/8Aoa9gjJMsXFqgG59"         val
intent = Intent(Intent.ACTION_VIEW, Uri.parse(url))

        if (intent.resolveActivity(packageManager) !=
null) {
            startActivity(intent)
        } else {
            Toast.makeText(this, "Google Maps tidak
```

Pada kode diatas, fungsi `readData()` dan `readStatus()` dipanggil untuk memproses data yang diperlukan. Dua `RadioButton` diinisialisasi dan diberi tindakan klik. Ketika `buttonMaps1` diklik, aplikasi membuat `Intent` untuk membuka URL *Google Maps* tertentu. Jika aplikasi yang dapat menangani `Intent` tersedia, maka *startActivity* dipanggil untuk membuka URL tersebut. Jika tidak, pesan kesalahan "*Google Maps* tidak terinstal" ditampilkan menggunakan Toast. Proses serupa dilakukan untuk `buttonMaps2` dengan URL *Google Maps* yang berbeda.

```
private fun readData() {
    val distanceRef1 =
database.child("Customer1SensorData").child("distance")
    val distanceRef2 =
database.child("Customer2SensorData").child("distance")

    distanceRef1.addValueEventListener(object :
ValueEventListener {
        override fun onDataChange(dataSnapshot:
DataSnapshot) {
            if (dataSnapshot.exists()) {
                val jarakAir: Float =
dataSnapshot.getValue(Float::class.java) ?: 0.0f
                binding.datajarakCustomer1.text =
jarakAir.toString()
                Toast.makeText(
                    this@AdminActivity2,
                    "Water Distance Updated",
                    Toast.LENGTH_SHORT
                ).show()
            } else {
                Toast.makeText(
                    this@AdminActivity2,
                    "Water Distance Not Found",
                    Toast.LENGTH_SHORT
                ).show()
            }
        }
```

Fungsi dari `readData` yaitu mengakses data ketinggian dari dua referensi di *Firebase Realtime Database*. Ketika data pada `distanceRef1` berubah, fungsi ini memperbarui tampilan dengan nilai ketinggian dan menampilkan pesan "*Water Distance Updated*" jika data ada, atau "*Water Distance Not Found*" jika tidak ada.

```
distanceRef2.addValueEventListener(object :
ValueEventListener {
    override fun onDataChange(dataSnapshot: DataSnapshot)
{
        if (dataSnapshot.exists()) {
            val jarakAir: Float =
dataSnapshot.getValue(Float::class.java) ?: 0.0f
            binding.datajarakCustomer2.text =
jarakAir.toString()
            Toast.makeText(
                this@AdminActivity2,
                "Water Distance Updated",
                Toast.LENGTH_SHORT
            ).show()
        } else {
            Toast.makeText(
                this@AdminActivity2,
                "Water Distance Not Found",
                Toast.LENGTH_SHORT
            ).show()
        }
    }
```

Kode ini menambahkan `ValueEventListener` pada `distanceRef2` untuk memantau perubahan data ketinggian. Jika data ditemukan, tampilan `dataketinggianCustomer2` diperbarui dan pesan "*Water Distance Updated*" ditampilkan. Jika data tidak ditemukan, pesan "*Water Distance Not Found*" ditampilkan.

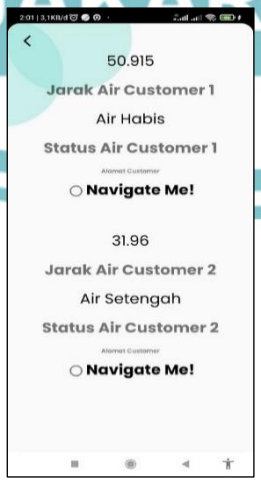
```
private fun readStatus() {
    val statusRef1 =
database.child("Customer1SensorData").child("waterStatus"
)
    val statusRef2 =
database.child("Customer2SensorData").child("waterStatus"
)

    statusRef1.addValueEventListener(object :
ValueEventListener {
        override fun onDataChange(dataSnapshot:
DataSnapshot) {
            if (dataSnapshot.exists()) {
                val waterStatus: String? =
dataSnapshot.getValue(String::class.java)
                binding.dataStatusCustomer1.text =
waterStatus ?: "Unknown"
                Toast.makeText(
                    this@AdminActivity2,
                    "Water Status Updated",
                    Toast.LENGTH_SHORT
                ).show()
            } else {
                Toast.makeText(
                    this@AdminActivity2,
                    "Water Status Not Found",
                    Toast.LENGTH_SHORT
```

Fungsi `readStatus` memantau perubahan status dari `statusRef1`. Jika data status ditemukan, tampilan `dataStatusCustomer1` diperbarui dan pesan "*Water Status Updated*" ditampilkan. Jika tidak ditemukan, pesan "*Water Status Not Found*" ditampilkan.

```
statusRef2.addValueEventListener(object :
ValueEventListener {
    override fun onDataChange(dataSnapshot: DataSnapshot)
{
        if (dataSnapshot.exists()) {
            val waterStatus: String? =
dataSnapshot.getValue(String::class.java)
            binding.dataStatusCustomer2.text =
waterStatus ?: "Unknown"
            Toast.makeText(
                this@AdminActivity2,
                "Water Status Updated",
                Toast.LENGTH_SHORT
            ).show()
        } else {
            Toast.makeText(
                this@AdminActivity2,
                "Water Status Not Found",
                Toast.LENGTH_SHORT
            ).show()
        }
    }
```

Kode ini menambahkan `ValueEventListener` pada `statusRef2` untuk memantau perubahan status. Jika data ditemukan, tampilan `dataStatusCustomer2` diperbarui dan pesan "*Water Status Updated*" ditampilkan. Jika tidak ditemukan, pesan "*Water Status Not Found*" ditampilkan.



Gambar 3.17 Tampilan *Admin Activity* Aplikasi Android Sistem *Monitoring Level* Air dalam Galon

6) Fungsi *Binding Button* di *User Activity*.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityUserBinding.inflate(layoutInflater)
    setContentView(binding.root)
    supportActionBar?.hide()
    val firebaseDatabase =
FirebaseDatabase.getInstance("https://water-level-
monitoring-7a5c6-default-rtdb.asia-
southeast1.firebasedatabase.app/")
    database = firebaseDatabase.reference
Customer 1
    readData()
    readStatus()
    binding.buttonAntarSekarangUser1.setOnClickListener {
        sendMessageToAdmin()
    }
}
```

Dalam metode `onCreate`, *ActivityUser* diinisialisasi dengan `ActivityUserBinding`, menyembunyikan *ActionBar*, dan menghubungkan ke Firebase Database. Fungsi `readData()` dan `readStatus()` dipanggil untuk membaca data dan status dari Firebase untuk *Customer* 1. *Listener* ditambahkan pada `buttonAntarSekarangUser1` untuk memanggil `sendMessageToAdmin()` saat tombol diklik.

```
private fun readStatus() {
    val statusRef1 =
database.child("Customer1SensorData").child("waterStatus"
)

    statusRef1.addValueEventListener(object :
ValueEventListener {
        override fun onDataChange(dataSnapshot:
DataSnapshot) {
            if (dataSnapshot.exists()) {
                val waterStatus: String? =
dataSnapshot.getValue(String::class.java)
                binding.dataStatusAirKamu1.text =
waterStatus ?: "Unknown"
                Toast.makeText(
                    this@UserActivity,
                    "Water Status Updated",
                    Toast.LENGTH_SHORT
                ).show()
            } else {
                Toast.makeText(
                    this@UserActivity,
                    "Water Status Not Found",
                    Toast.LENGTH_SHORT
                ).show()
            }
        }
```

Fungsi `readStatus` memantau perubahan status air dari `statusRef1` di *Firebase*. Jika data ada, `dataStatusAirKamu1` diperbarui dan pesan "Water Status Updated" ditampilkan. Jika tidak ada, pesan "*Water Status Not Found*" ditampilkan.

```
private fun readData() {
    val distanceRef1 =
database.child("Customer1SensorData").child("distance")
    distanceRef1.addValueEventListener(object :
ValueEventListener {
        override fun onDataChange(dataSnapshot:
DataSnapshot) {
            if (dataSnapshot.exists()) {
                val jarakAir: Float =
dataSnapshot.getValue(Float::class.java) ?: 0.0f
                binding.datajarakAirKamu1.text =
jarakAir.toString()
                Toast.makeText(
                    this@UserActivity,
                    "Water Distance Updated",
                    Toast.LENGTH_SHORT
                ).show()
            } else {
                Toast.makeText(
                    this@UserActivity,
                    "Water Distance Not Found",
                    Toast.LENGTH_SHORT
                ).show()
            }
        }
```

Fungsi `readData` memantau perubahan data ketinggian air dari `distanceRef1` di *Firebase*. Jika data ada, `dataketinggianAirKamu1` diperbarui dengan nilai ketinggian air dan pesan "*Water Distance Updated*" ditampilkan. Jika data tidak ditemukan, pesan "*Water Distance Not Found*" ditampilkan.

```
  private fun sendMessageToAdmin() {
        // Mengirim pesan ke Firebase
        val messageRef =
database.child("AdminMessages").push()
        messageRef.setValue("ANTAR SEKARANG by CUSTOMER
1").addOnCompleteListener { task ->
            if (task.isSuccessful) {
                Toast.makeText(this, "Pesan berhasil
dikirim ke Admin", Toast.LENGTH_SHORT).show()
            } else {
                Toast.makeText(this, "Gagal mengirim
pesan ke Admin", Toast.LENGTH_SHORT).show()
            }
        }
    }
```

Fungsi `sendMessageToAdmin` mengirim pesan ke *Firebase*. Pesan "ANTAR SEKARANG by CUSTOMER 1" disimpan di referensi `AdminMessages`. Jika pengiriman berhasil, pesan "Pesan berhasil dikirim ke Admin" ditampilkan. Jika gagal, pesan "Gagal mengirim pesan ke Admin" ditampilkan.

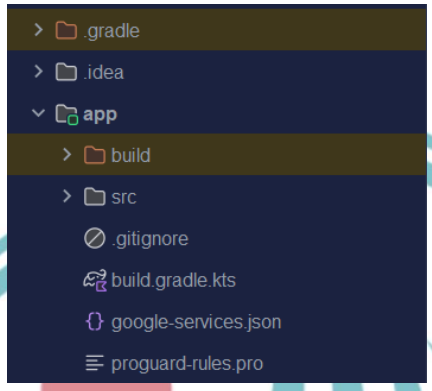
Gambar 3.18Tampilan *User Activity* Aplikasi Android Sistem *Monitoring Level* Air dalam Galon

3.3.3 Pengintegrasian Database ke Aplikasi

1) Menyambungkan *Firebase* ke Aplikasi Android

Pada tahap awal integrasi *Firebase*, pilih opsi Android untuk mendaftarkan aplikasi yang sudah dibuat dan menyambungkannya dengan *database Firebase*. Saat melakukan pendaftaran aplikasi, anda akan diminta mengisi informasi tentang aplikasi Android yang akan didaftarkan. *Firebase* memerlukan identitas unik aplikasi untuk memastikan bahwa data yang dikirim dan diterima berasal dari aplikasi yang benar.Isi `Android package name` dengan nama paket yang sudah dibuat di Kodular pada bagian `publishing` di menu `project setting`. Nama paket harus unik dan dibuat secara manual. `App nickname` adalah nama aplikasi yang Anda buat, sementara `debug signing certificate SHA-1` diisi dengan

Keystore yang terdapat di Kodular pada bagian Preferences. Setelah pendaftaran selesai, unduh file `google-services.json` dan unggah file tersebut ke Android Studio pada direktori `app` di dalam folder `gradle`.



Gambar 3.19 Build Gradle Android Studio

2) Menyambungkan Aplikasi Android ke Realtime Database

Berikut langkah-langkah untuk menyambungkan aplikasi android ke *Firebase Realtime* Database:

a. Membuat Proyek di Firebase Console:

- Kunjungi Firebase Console dan buat proyek baru.
- Setelah proyek dibuat, klik "Add App" dan pilih platform Android.

b. Konfigurasi di Android Studio:

- Buka file `build.gradle` di Android Studio dan tambahkan dependensi untuk *Firebase Realtime Database*:

```
implementation
'com.google.firebase:firebase-
database:20.3.0'
```

- Pastikan juga untuk menambahkan *plugin Google Services* di bagian bawah file `build.gradle (Project)`:

```
apply plugin: 'com.google.gms.google-
services'
```

c. Membuat Instance Firebase di Proyek:

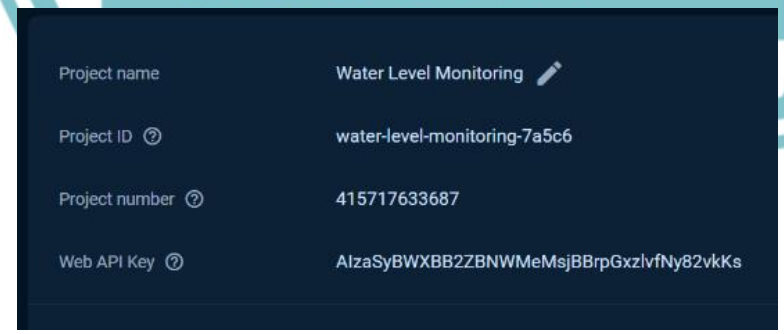
- Di dalam proyek, buat *instance* dari `FirebaseDatabase` dan `DatabaseReference`

```
val database: DatabaseReference =
FirebaseDatabase.getInstance().reference
database.child("sensor").child("distance")
.setValue(value)
```

Firebase URL dapat dilihat pada tampilan rel-time database dan *Firebase* token bisa dilihat pada Web API Key. Pada gambar 3.14 menunjukkan tampilan *realtime database* pada *Firebase*.

Gambar 3.20 *Firebase* URL

Pada Gambar 3.14 menunjukkan tampilan *Realtime Database pada Firebase*. Pada kotak berwarna merah merupakan *Firebase* URL. URL akan diisi pada *Activity* sebagai alamat untuk menghubungkan Firebase dengan aplikasi di Android Studio. Pada gambar 3.15 menunjukkan tampilan pada project setting pada Firebase. Firebase token diisi `pada google-service.json` di Android Studio. Untuk memunculkan Web API Key diperlukan untuk Membuat *Authentication* terlebih dahulu Jika Firebase URL dan Firebase token sudah diisi pada properties Firebase di Android Studio maka Firebase sudah tersambung ke Android Studio.



Gambar 3.21 Tampilan Project Setting pada *Firebase*

BAB IV
PEMBAHASAN

Setelah melakukan perancangan dan realisasi sistem proses selanjutnya adalah melakukan pengujian alat. Proses ini merupakan tahap terakhir yang dilakukan dalam pembuatan tugas akhir ini. Hal ini dilakukan agar dapat mengetahui apakah sistem dapat berfungsi dengan baik atau tidak. Pengujian dilaksanakan berdasarkan lokasi dan waktu berikut:

Lokasi : Lab Telekomunikasi, Gd. G, Politeknik Negeri Jakarta
Waktu : 14 Agustus 2024
Pelaksana : Ferdinand Ardhiandos Maruli Tua Sitinjak
Pembimbing : Dr. Yenniwarti Rafsyam, SST., M.T.

4.1 Pengujian Catu Daya

Pengujian hasil tegangan keluaran *power supply* bertujuan untuk mengetahui jumlah tegangan keluaran pada *power supply* telah sesuai dengan kebutuhan alat yang digunakan

4.1.1 Deskripsi Pengujian

Pengujian rangkaian catu daya merupakan pengujian yang dilakukan untuk mengetahui karakteristik catu daya agar sesuai dengan spesifikasi yang diinginkan. Hasil akhir pengujian ini yaitu didapatkan hasil keluaran tegangan yang sesuai dengan kebutuhan sistem yaitu 5 Volt.

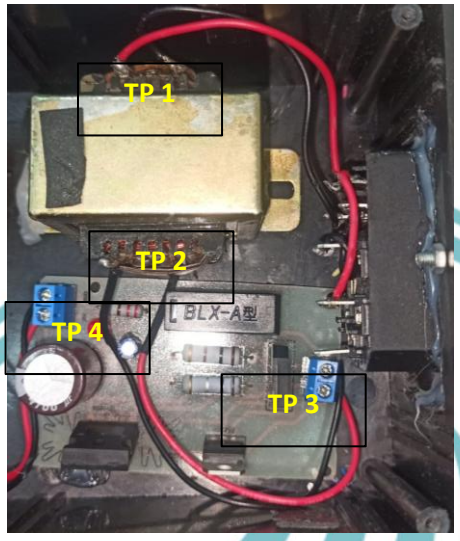
4.1.2 Alat dan Bahan

a. Multimeter digital.
b. *Power supply.*
c. Kabel multimeter.
d. Osiloskop
e. Kabel BNC to *Crocodile*

4.1.3 *Setup* Pengujian Catu Daya

Pengujian dilakukan untuk mengetahui masukan dan keluaran pada tiap titik pengukuran (TP1 – TP4). TP 1 pengukuran dilakukan pada *input* trafo. TP 2

dilakukan pada *output* trafo. TP 3 dilakukan pada dioda bridge. TP 4 dilakukan pada *output* catu daya. Gambar 4 adalah catu daya yang akan diuji.



Gambar 4.1 Pengujian Catu Daya

4.1.4 Prosedur Pengujian

Prosedur pengujian tegangan keluaran catu daya dilakukan dengan cermat dan berurutan untuk memastikan hasil yang akurat. Setiap tahapan pengujian dirancang untuk mengevaluasi performa dan stabilitas tegangan yang dihasilkan. Tahapan pengujian tegangan keluaran catu daya adalah sebagai berikut:

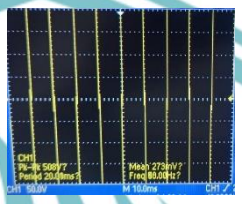
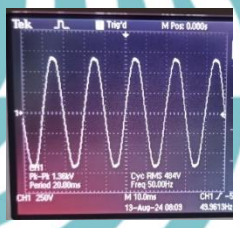
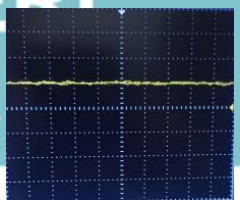
a. Merangkai catu daya seperti Gambar 4.1
b. Pengujian terhadap *input* dari catu daya sebesar 220V AC dari listrik PLN.
c. Pengujian terhadap *output* dari trafo sebesar 9V.
d. Pengujian terhadap dioda *bridge*.
e. Pengujian terhadap *output* dari catu daya sebesar 5V.
f. Mengamati nilai hasil pembacaan pada multimeter dan memasukkan ke dalam Tabel 4.1
g. Mengamati bentuk gelombang pada tiap pengukuran menggunakan osiloskop.

4.1.5 Analisis Hasil Pengujian

Hasil pengujian catu daya telah direkam dan ditampilkan dalam Tabel 4.1. Tabel tersebut berisi data hasil pengukuran yang dilakukan menggunakan multimeter digital serta bentuk gelombang yang diukur dengan osiloskop. Data

tersebut memberikan gambaran menyeluruh mengenai performa catu daya,termasuk tegangan input, *output* trafo, dan *output* setelah melalui dioda bridge, serta analisis gelombang yang diperoleh pada setiap tahap pengujian

Tabel 4. 1 Hasil Pengujian Catu Daya Dengan Multimeter dan Osiloskop

No	TP	Hasil Pada Mulimeter	Bentuk Gelombang	Keterangan
1	TP1	226,28V		AC
2	TP2	9,697V		AC
3	TP3	11,614V		DC
4	TP4	5,03V		DC

Hasil pengujian tegangan rangkaian catu daya diukur menggunakan multimeter digital. Tegangan input trafo tercatat sebesar 226V AC, sedikit lebih tinggi dari nilai nominal 220V, tetapi masih dalam batas yang dapat diterima. Tegangan *output* trafo terukur sebesar 9.6V AC, yang sedikit di bawah nilai nominal 9V, namun masih dalam batas toleransi. Pengujian tegangan input pada trafo dilakukan dengan osiloskop untuk menganalisis bentuk gelombang AC.

Gelombang AC yang terbaca menunjukkan bentuk yang sesuai dengan karakteristik trafo. Tegangan *output* trafo juga diuji dengan osiloskop, dan gelombang AC yang diperoleh sesuai dengan ekspektasi, meskipun terdapat sedikit penurunan tegangan karena *stepdown*.

Setelah melewati dioda *bridge*, tegangan naik menjadi 11.6V DC, yang sesuai dengan perhitungan nilai puncak dari tegangan AC rms. Dengan menggunakan rumus $V_p = V_{rms} \times \sqrt{2}$. Untuk tegangan rms 9V, didapatkan $V_p = 9V \times 1.414 = 12.7V$. Setelah melewati dioda bridge, terdapat penurunan tegangan sekitar 1.4V akibat penurunan tegangan pada dua dioda aktif dalam setiap siklus, sehingga tegangan *output* setelah dioda bridge adalah sekitar 11V. Pengujian tegangan *output* dioda bridge menggunakan osiloskop menunjukkan bentuk gelombang DC yang sesuai dengan catu daya gelombang penuh. (N, 2017)

Output catu daya yang terhubung ke board ESP32 terukur sebesar 5.03V DC, yang sesuai dengan kebutuhan sistem yang memerlukan tegangan 5V. Tegangan yang keluar sebesar 5V disebabkan oleh IC7805 yang memastikan besaran keluaran tegangan yaitu sebesar 5V . Pengukuran ini memastikan bahwa catu daya bekerja sesuai dengan spesifikasi yang dibutuhkan, menyediakan tegangan yang stabil dan andal untuk sistem yang akan digunakan.

4.2 Pengujian Alat Sistem

4.2.1 Deskripsi Pengujian

Pengujian sensor ultrasonik dilakukan untuk mengetahui sensor ultrasonik dapat bekerja dalam mendeteksi tinggi air di dalam galon. Alat-alat yang digunakan dalam melakukan pengujian antara lain:

a. ESP 32
b. Sensor Ultrasonik JSN-SR04T
c. LCD OLED

4.2.2 Prosedur Pengujian

Prosedur pengujian yang dilakukan pada sensor ultrasonik dengan menggunakan ESP32 dan LCD adalah sebagai berikut:

a. Menyiapkan sensor ultrasonik JSN-SR04T, ESP32, dan LCD untuk menampilkan data ketinggian air secara *real-time*.

b. Menghubungkan sensor ultrasonik dengan ESP32 dan menghubungkan LCD OLED dengan ESP32 sesuai dengan diagram pin.

c. Melakukan pengujian sensor ultrasonik dengan meletakkan sensor pada mulut galon, lalu memeriksa ketinggian air yang ditampilkan pada LCD OLED.

4.2.3 *Setup* ESP32

Pengujian ESP32 merupakan pengujian yang dilakukan untuk menghubungkan ESP32 dengan jaringan internet, pengujian dilakukan untuk memastikan agar mikrokontroler dapat mengirimkan data informasi ke *firebase* serta menerima informasi dari *database*. *Setup* dinyatakan berhasil apabila *serial monitor* pada Arduino IDE menampilkan bahwa ESP telah terkoneksi ke IP *WiFi* yang digunakan dan meminta *token firebase* seperti pada Gambar 4.2.

```
Token info: type = id token (GITKit token), status = on request
Token info: type = id token (GITKit token), status = ready
Jarak: 0.00 cm
Data terkirim ke Firebase
Jarak: 0.00 cm
Data terkirim ke Firebase
Jarak: 0.00 cm
Data terkirim ke Firebase
Jarak: 0.00 cm
Data terkirim ke Firebase
Jarak: 0.00 cm
Data terkirim ke Firebase
Jarak: 0.00 cm
```

Gambar 4.2 Setup ESP32 Pada Sistem *Monitoring Level* Air Dalam Galon

4.2.4 *Setup* Sensor Ultrasonik JSN SR04T

Setup Sensor Ultrasonik dilakukan dengan memastikan bahwa pin yang akan digunakan terhubung ke pin yang benar. *Setup* dinyatakan berhasil apabila *board* sensor ultrasonik menampilkan kedip lampu dan *serial monitor* pada Arduino IDE menampilkan data ketinggian dan mengirimkannya ke *firebase* seperti pada Gambar 4.3.

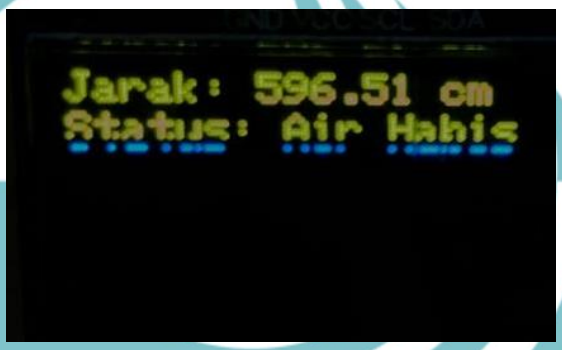
```
....
Connected with IP: 192.168.208.235

Token info: type = id token (GITKit token), status = on request
```

Gambar 4.3 *Setup* Sensor Ultrasonik Pada Sistem *Monitoring Level* Air Dalam Galon

4.2.5 *Setup* LCD OLED

Proses *setup* LCD OLED dilakukan untuk memastikan bahwa semua pin yang terhubung sudah benar dan sesuai dengan konfigurasi yang diperlukan. Keberhasilan setup ini dapat ditandai dengan booting awal pada LCD, di mana layar akan menyala dan menampilkan informasi yang diharapkan. Setelah *booting*, LCD OLED akan secara otomatis menampilkan parameter ketinggian air dan status air, seperti yang ditunjukkan pada Gambar 4.4. Tampilan ini berfungsi sebagai indikasi bahwa sistem beroperasi dengan baik dan siap untuk digunakan dalam pemantauan level air



Gambar 4.4 *Setup* LCD OLED

4.2.6 Hasil Pengujian

Hasil pengujian antara sensor ultrasonik dengan ESP32 dan LCD dapat dilihat langsung pada saat program sistem dijalankan, dimana nilai ketinggian air dan status air secara *real-time* pada LCD. Pengujian terbagi dalam 3 status yaitu pada saat air penuh, air setengah, dan air habis. Tabel 4.2 merupakan pengujian ketika status air penuh. Hasil yang didapatkan dalam 5 kali pengujian pada status air penuh sesuai dengan keadaan sebenarnya sehingga dapat disimpulkan persentase akurasi pada pembacaan air penuh sebesar 100%.

Tabel 4.2 Data Hasil Pengujian Sensor Ultrasonik Saat Air Penuh

No	Jam	Ketinggian Air	Status Air	Keadaan Sebenarnya
1	13:00	19,99 cm	Air Penuh	Air Penuh
2	13:30	20 cm	Air Penuh	Air Penuh
3	14:00	19,99 cm	Air Penuh	Air Penuh

No	Jam	Ketinggian Air	Status Air	Keadaan Sebenarnya
4	14:30	19,99 cm	Air Penuh	Air Penuh
5	15:00	20,03 cm	Air Penuh	Air Penuh

Berikutnya dilakukan pengujian ketika status air setengah. Hasil pengujiannya dapat dilihat pada Tabel 4.3. Hasil yang didapatkan dalam 5 kali pengujian pada status air setengah sesuai dengan keadaan sebenarnya sehingga dapat disimpulkan persentase akurasi pada pembacaan air setengah sebesar 100%.

Tabel 4.3 Pengujian Sensor Ultrasonik Saat Air Setengah

No	Jam	Ketinggian Air	Status Air	Keadaan Sebenarnya
1	15:30	20,11 cm	Air Setengah	Air Setengah
2	16:00	20,50 cm	Air Setengah	Air Setengah
3	16:30	20,55 cm	Air Setengah	Air Setengah
4	17:00	20,50 cm	Air Setengah	Air Setengah
5	17:30	20,60 cm	Air Setengah	Air Setengah

Berikutnya dilakukan pengujian ketika status air habis. Hasil pengujiannya dapat dilihat pada tabel 4.4. Hasil yang didapatkan dalam 5 kali pengujian pada status air habis 3 kali sesuai dengan keadaan sebenarnya, dan 2 kali tidak. Sehingga dapat disimpulkan persentase akurasi pada pembacaan air penuh sebesar 60%. Kesalahan pembacaan sebanyak dua kali dapat disebabkan oleh gelombang sensor ultrasonik terbiaskan atau mendeteksi bagian bawah galon yang tidak rata.

Tabel 4. 4 Pengujian Sensor Ultrasonik Saat Air Habis

No	Jam	Ketinggian Air	Status Air	Keadaan Sebenarnya
1	18:00	29,60	Air Setengah	Air Habis
2	18:30	30,20	Air Habis	Air Habis
3	19:00	30,21	Air Habis	Air Habis

No	Jam	Ketinggian Air	Status Air	Keadaan Sebenarnya
4	19:30	29,90	Air Setengah	Air Habis
5	20:00	30,10	Air Habis	Air Habis

4.2.7 Analisis Hasil Pengujian

Hasil analisis data menunjukkan bahwa terdapat kesesuaian antara pembacaan sensor dengan kondisi air pada beberapa interval waktu, namun juga terdapat inkonsistensi yang perlu diperbaiki. Pada jam 13:00 hingga 15:00, sistem secara konsisten mendeteksi status "Air Penuh" dengan ketinggian air yang berkisar antara 19,99 cm hingga 20,03 cm, yang sesuai dengan kondisi sebenarnya. Namun, mulai jam 15:30 hingga 17:30, ketinggian dari mulut galon ke permukaan air yang terukur meningkat sedikit menjadi 20,11 cm hingga 20,60 cm, dan sistem mencatat status "Air Setengah", yang juga sesuai dengan kondisi sebenarnya. Pada jam 18:00, di mana sensor menunjukkan ketinggian menjadi 29,60 cm, namun status air yang terdeteksi masih "Air Setengah", meskipun kondisi sebenarnya sudah "Air Habis". Namun pada jam 18:00 – 18:30 pembacaan status air benar dengan keadaan sebenarnya “Air Habis”. Pada jam 19.30 pembacaan sempat kembali berubah ke status “Air Setengah”, dan kembali benar lagi pada jam 20:00. Kesalahan pembacaan sebanyak dua kali dapat disebabkan oleh gelombang sensor ultrasonik terbiaskan atau mendeteksi bagian bawah galon yang tidak rata.

4.3 Pengujian Aplikasi

Pengujian aplikasi android dilakukan untuk membuktikan bahwa aplikasi yang telah dibuat dapat terhubung dengan *database* dan dapat menampilkan data sensor dari alat dan ditampilkan pada aplikasi.

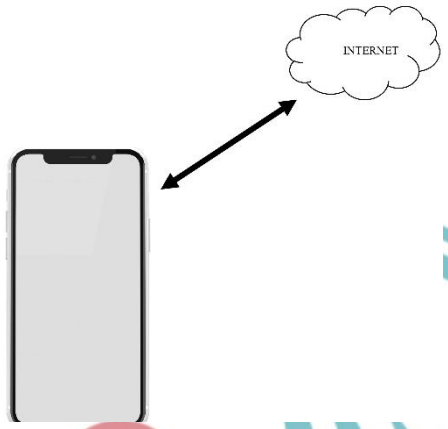
4.3.1 Deskripsi Pengujian

Untuk melakukan pengujian ini aplikasi “Water Level Monitoring” sudah ter*install* dalam *smartphone* yang digunakan sehingga pengujian dapat dilakukan dengan membuka aplikasi lalu melakukan *login* sehingga data dapat diambil dari *firebase* dan dibaca oleh aplikasi android.

4.3.2 Prosedur Pengujian

1. Menghubungkan *smartphone* ke internet untuk memulai pengujian seperti

Gambar 4.5.



Gambar 4.5 Menghubungkan Smartphone ke Internet

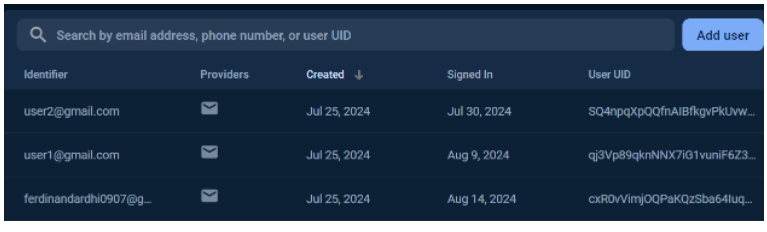
2. Mengoperasikan aplikasi *monitoring level* air pada *smartphone*.
3. Melakukan proses *login* dan mengakses aplikasi seperti pada Gambar 4.7
4. Mengakses halaman aplikasi untuk membaca data dari *firebase* seperti pada Gambar 4.8
5. Memverifikasi data dari f*irebase* ditampilkan dengan benar di aplikasi.

4.3.3 Hasil Pengujian

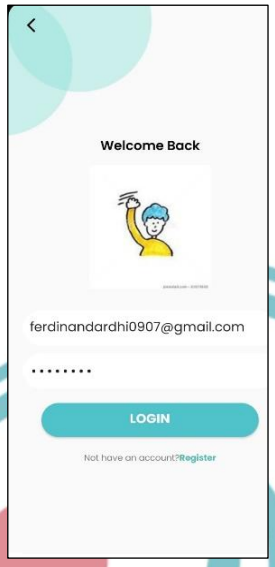
Dari hasil pengujian telah dilakukan menggunakan *smartphone* yang sudah terinstal aplikasi "Water Level Monitoring", maka didapatkan hasil sebagai berikut:

1. Pengujian fitur *authentication* pada proses *login.*

Pada Gambar 4.6 dapat dilihat bahwa daftar akun yang didaftarkan sudah masuk dalam *firebase authentication*. Lalu, pada saat *login* dan memasukkan akun yang sudah terdaftar pada *authentication*, maka proses *login* berhasil. Gambar 4.7 menunjukkan halaman *login.*



Gambar 4.6 *Authentication* Pada *Firebase*



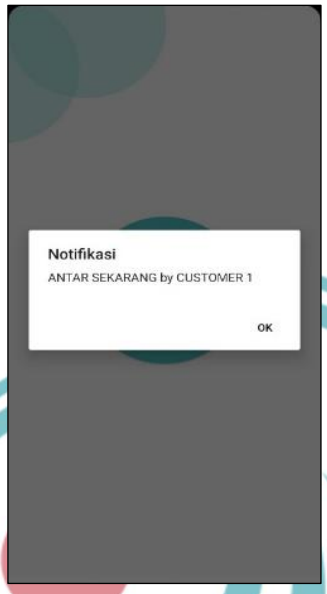
Gambar 4.7 Pengujian Login Activity

2. Pengujian menampilkan data dari sensor pada *Admin Activity*.

Dapat dilihat pada gambar 4.8 merupakan data pada halaman admin. Data yang tertampil pada halaman tersebut adalah, *customer* 1 dan *customer* 2 status airnya yaitu airnya telah habis.

Gambar 4.8 Pengujian *Admin Activity*

Pada gambar 4.9 dapat dilihat terdapat notifikasi *pop-up* yang menampilkan pesan “antar sekarang”, yang pesan tersebut berasal dari *customer* 1.



Gambar 4.9 Pengujian Pengiriman Pesan Oleh Customer 1

3. Pengujian *User Activity*

Halaman *User Activity* dibuat untuk pelanggan untuk mengamati masing-masing data air milik pribadi. Gambar 4.10 berisi merupakan halaman yang terhubung langsung dengan *Realtime Database*. Halaman ini juga memiliki fitur untuk mengirim pesan ke Admin.

Gambar 4.10 Pengujian *User Activity*

4.3.4 Analisa Pengujian *Software*

Pada pengujian fitur *authentication* dalam proses *login*, akun yang didaftarkan telah berhasil masuk ke dalam Firebase *Authentication*, sebagaimana ditunjukkan pada Gambar 4.9. Ketika akun tersebut digunakan untuk *login*, prosesnya berjalan dengan sukses, membuktikan bahwa fitur *authentication* berfungsi dengan baik. Selanjutnya, pengujian menampilkan data dari sensor pada *Admin Activity* menunjukkan bahwa halaman Admin mampu menampilkan data level air dari pelanggan dengan akurat. Pada Gambar 4.8 ditampilkan bahwa air pada pelanggan 1 dan pelanggan 2 telah habis, dan di Gambar 4.9 terlihat adanya notifikasi *pop-up* dengan pesan "antar sekarang" yang berasal dari pelanggan 1. Hal ini mengindikasikan bahwa fitur notifikasi berfungsi sesuai dengan yang diharapkan. Pengujian pada halaman *User Activity* juga menunjukkan hasil yang positif. Halaman ini memungkinkan pelanggan untuk mengamati data level air mereka sendiri yang terhubung langsung dengan *Realtime Database*, sehingga informasi yang ditampilkan selalu terbarui. Selain itu, halaman ini dilengkapi dengan fitur untuk mengirim pesan ke Admin, yang memudahkan interaksi antara pengguna dan administrator sistem.

4.4 Pengujian *Quality of Service* (QoS)

Pengujian QoS memiliki 4 parameter di dalamnya yang harus diamati yaitu *delay, packet loss, throughput,* dan *jitter*.

4.4.1 Deskripsi Pengujian

Pengujian QoS dilakukan dengan mengamati kualitas transmisi data yang terjadi antara alat dengan aplikasi menggunakan *software wireshark*. Setelah itu, hasil yang didapatkan dari *wireshark* digunakan untuk menghitung masing- masing parameter QoS. Agar dapat mengetahui performansi jaringan dari alat yang telah dibuat.

4.4.2 Prosedur Pengujian

1. Menghubungkan laptop dengan *hotspot tethering*.
2. Membuka aplikasi *wireshark*.
3. Memilih koneksi WiFi yang terhubung dari *Hotspot Tethering*.
4. Menjalankan proses pengambilan data selama 1 menit.

5. Meng*capture* data.
6. Mengamati parameter pada menu bar *statistics*.
7. Meng*capture file properties*

4.4.3 Hasil Pengujian

Pada gambar 4.11 merupakan hasil dari *wireshark* yang sudah dijalankan. Untuk menghitung kemampuan nya baik itu througput, delay dan packet loss yang perlu kita ketahui dahulu besaran byte dan packet yang terkirim yaitu;

Bytes :8134184

Packet : 8457

TimeSpan : 303,920s

Interfaces

Interface	Dropped packets	Capture filter	Link type	Packet size limit (snaplen)
Wi-Fi	Unknown	none	Ethernet	262144 bytes

Statistics

Measurement	Captured	Displayed	Marked
Packets	8457	—	—
Time span, s	303.920	—	—
Average pps	27.8	—	—
Average packet size, B	962	—	—
Bytes	8134183	0	0
Average bytes/s	26 k	—	—
Average bits/s	214 k	—	—

Gambar 4.11 Hasil Performa Jaringan Pada *Wireshark*

a. Menghitung *Packet Loss*

Packet loss adalah paket yang hilang dibandingkan dengan jaringan yang diuji di rumah. Paket yang dikirimkan diukur dalam satuan persen. Untuk mengetahui nilai packet loss, digunakan persamaan berikut.

$$Packet\ Loss = \frac{(Paket\ yang\ dikirim - Paket\ yang\ diterima)}{Paket\ yang\ dikirim} \times 100\%$$

$$Packet\ Loss = \frac{(8457 - 8457)}{8457} \times 100\% = 0\%$$

Dari hasil perhitungan, nilai packet loss adalah 0. Ini menunjukkan bahwa tidak ada paket yang hilang dan semua paket yang dikirim berhasil diterima dengan persentase 100%.

b. Menghitung *Throughput*

Throughput adalah ukuran jumlah data yang dapat diproses atau dikirimkan melalui sistem atau jaringan dalam periode waktu tertentu. Ini mengukur kinerja sistem, biasanya dalam bit per detik (bps) atau byte per detik (Bps), menunjukkan seberapa cepat data dapat ditransfer atau diproses.

$$Throughput = \frac{Jumlah\ data\ yang\ dikirim}{Waktu\ pengirim\ data}$$

$$Throughput = \frac{8134183}{303.930} = 26{,}764 \times 8 = 214{,}11$$

Dari perhitungan yang dilakukan , diperoleh nilai throughput sebesar 214,11 *kilobytes.*

c. Menghitung *Delay*

Delay merupakan waktu pengiriman data untuk menempuh jarak dari asal ketujuan. Untuk mengetahui nilai *delay* menggunakan persamaan berikut ini ;

$$Delay = \frac{Waktu\ pengiriman}{Paket\ diterima}$$

$$Delay = \frac{303{,}920}{8457} = 0{,}035s$$

Delay yang didapatkan dengan menggunakan jaringan WiFi di rumah yaitu sebesar 0,035 s.

d. Menghitung *Jitter*

Jitter adalah variasi dari kedatangan paket dalam panjang antrian, waktu pengolahan data dan waktu penghimpunan ulang paket di akhir *jitter*. Untuk mengetahui nilai *jitter* menggunakan persamaan berikut ini:

$$\frac{Total\ variasi\ delay\ (-1)}{Total\ paket\ diterima} = \frac{0{,}035\ (-1)}{8457} = 0{,}0000041$$

4.4.4 Analisa Data *Quality of Service*

Menurut analisis data jaringan, jaringan yang diuji beroperasi dengan sangat baik. Nilai kehilangan paket sebesar 0% menunjukkan bahwa tidak ada paket data yang hilang selama transmisi, yang menunjukkan bahwa jaringan sangat andal. Dengan throughput 214,11 kilobytes per detik, jaringan memiliki kemampuan untuk mengirimkan data dengan cepat dan efisien. Aplikasi *real-time* seperti panggilan suara atau video konferensi membutuhkan pengiriman data yang cepat dengan latensi 0,035 detik. Selain itu, konsistensi waktu kedatangan paket, yang ditunjukkan oleh jitter yang sangat rendah sebesar 0,0000041 detik, sangat penting untuk menjaga kualitas layanan pada aplikasi yang sensitif terhadap waktu seperti VoIP dan streaming media.

4.5 Analisa Sistem

Sistem monitoring level air yang dikembangkan memanfaatkan mikrokontroler ESP32, sensor ultrasonik JSN-SR04T, LCD OLED, dan buzzer, dirancang untuk mengirimkan data sensor ke Firebase dan kemudian diproses dalam aplikasi. Mikrokontroler membaca data dari sensor, mengirimkannya melalui WiFi, serta menampilkan ketinggian dan status air pada LCD OLED dan buzzer. Analisis menunjukkan bahwa sistem ini dapat mendeteksi ketinggian air dengan akurat pada beberapa interval waktu, meskipun terdapat beberapa ketidaksesuaian. Contohnya, pada jam 13:00 hingga 15:00, sistem berhasil mendeteksi status "Air Penuh" dengan ketinggian air yang benar, yaitu sekitar 19,99 cm hingga 20,03 cm. Namun, pada jam 18:00, meskipun sensor menunjukkan ketinggian air 29,60 cm, status yang terdeteksi masih "Air Setengah", padahal kondisi sebenarnya sudah "Air Habis". Namun demikian, pada jam 18:00 hingga 18:30, pembacaan kembali sesuai dengan kondisi aktual.

Pengujian fitur autentikasi saat login menunjukkan bahwa integrasi dengan *Firebase Authentication* berhasil, dan proses *login* berjalan tanpa hambatan. Selain itu, pengujian pada *Admin Activity* menunjukkan bahwa data *level* air pelanggan ditampilkan dengan akurat, termasuk notifikasi *pop-up* yang berfungsi sebagaimana mestinya. Halaman *User Activity* juga menunjukkan hasil yang memuaskan, dengan kemampuan pelanggan untuk memantau data level air yang

terhubung langsung ke *Realtime Database* serta fitur pesan untuk berkomunikasi dengan *administrator*.

Analisis jaringan menunjukkan bahwa sistem ini sangat andal, dengan kehilangan paket sebesar 0%, *throughput* 214,11 *kilobytes* per detik, dan *latensi* 0,035 detik. Jitter yang sangat rendah, yaitu 0,0000041 detik, juga menandakan bahwa jaringan mampu mendukung aplikasi real-time seperti VoIP dan *streaming* media dengan baik. Secara keseluruhan, sistem monitoring air ini bekerja dengan baik, meskipun ada beberapa aspek yang masih perlu ditingkatkan.