LAMPIRAN

Lampiran 1. Kode ESP32

```
#include <WiFi.h>
#include <WebServer.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <Adafruit_ADS1X15.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
// Pin I2C Configuration
#define SDA_ADS 21       // Pin
SDA untuk ADS1115 (standar)
#define SCL_ADS 22       // Pin
SCL untuk ADS1115 (standar)
#define SDA_LCD 33       // Pin
SDA untuk LCD
#define SCL_LCD 32       // Pin
SCL untuk LCD
// WiFi dan MQTT Configuration
const char *ssid = "Tselhome-
HAP";
const char *password = "Em-
bul@8888";
const char *mqtt_broker = "bro-
ker.emqx.io";
const int mqtt_port = 1883;
const char *mqtt_username =
"emqx";
const char *mqtt_password =
"public";
// MQTT Topics
const char *topic_ntu = "da-
ta/turbid526/ntu/4456";
// Sensor Configuration
const int16_t ZERO_ADC_LOW =
1560;
// Nilai ADC saat 0 NTU (batas
bawah)
const int16_t ZERO_ADC_HIGH =
1580;
// Nilai ADC saat 0 NTU (batas
atas)
const float ZERO_VOLTAGE =
0.1975;       // Tegangan saat 0
NTU (dihitung dari ZE-
RO_ADC_HIGH)
const float MAX_VOLTAGE =
4.09// Tegangan maksimal sensor
const float maxNTU = 388.0
// Skala maksimal NTU
const float ADC_LSB = 0.125
// 4.096V / 32767 (untuk gain
GAIN_ONE)
const float CAL_VOLTAGE = 0.5;
const float CAL_NTU = 6.4;
// LCD Configuration
#define LCD_ADDRESS 0x27     //
Alamat I2C LCD (bisa 0x27 atau
0x3F)
#define LCD_COLS 16
#define LCD_ROWS 2
// Global Variables
float ntu1 = 0;
float ntu2 = 0;
int16_t currentADC0 = 0;
int16_t currentADC1 = 0;
float currentVoltage0 = 0;
float currentVoltage1 = 0;
unsigned long lastSensorRead =
0;
unsigned long lastPublish = 0;
unsigned long lastLcdUpdate = 0;
unsigned long lastScrollUpdate =
0;
bool wifiConnected = false;
bool mqttConnected = false;
// Objects
WiFiClient espClient;
PubSubClient
mqtt_client(espClient);
Adafruit_ADS1115 ads;  //
ADS1115 menggunakan bus I2C
terpisah
LiquidCrystal_I2C
lcd(LCD_ADDRESS, LCD_COLS,
LCD_ROWS);
WebServer server(80);

void setI2CPins(int sda, int
scl) {
    Wire.end(); // End previous
I2C session
    Wire.begin(sda, scl); //
Start I2C with new pins
}
```

```
void connectToWiFi() {
WiFi.begin(ssid, password);
Serial.print("Connecting to
WiFi");
// Update LCD saat connecting
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Connecting WiFi");
lcd.setCursor(0, 1);
lcd.print("Please wait...");
int attempts = 0;
while (WiFi.status() != WL_CON-
NECTED && attempts < 20) {
delay(1000);
Serial.print(".");
attempts++;
setI2CPins(SDA_LCD, SCL_LCD);
// Update progress di LCD
lcd.setCursor(attempts % 16, 1);
lcd.print(".");
}
if (WiFi.status() == WL_CON-
NECTED) {
wifiConnected = true;
Serial.println("\nConnected to
WiFi");
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
// Update LCD dengan status ber-
hasil
setI2CPins(SDA_LCD, SCL_LCD);
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("WiFi Connected");
lcd.setCursor(0, 1);
lcd.print(WiFi.localIP());
delay(2000);
}else {
wifiConnected = false;
Serial.println("\nFailed to con-
nect to WiFi");
// Update LCD dengan status ga-
gal
setI2CPins(SDA_LCD, SCL_LCD);
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("WiFi Failed");
lcd.setCursor(0, 1);
lcd.print("Check config");
delay(2000);
}
}


void mqttCallback(char *topic,
byte *payload, unsigned int
```

```
LiquidCrystal_I2C
lcd(LCD_ADDRESS,
LCD_COLS, LCD_ROWS);
WebServer server(80);

void setI2CPins(int
sda, int scl) {
    Wire.end(); // End
previous I2C session
    Wire.begin(sda,
scl); // Start I2C
with new pins
}




void connectToMQTT() {
if (!wifiConnected)
return;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Connecting
MQTT");
lcd.setCursor(0, 1);
lcd.print("bro-
ker.emqx.io");
while (!mqtt_cli-
ent.connected())
{String client_id =
"esp32-S2-client-" +
String(WiFi.mac-
Address());
Serial.printf("Con-
necting to MQTT Broker
as %s...\n", cli-
ent_id.c_str());
if (mqtt_client.con-
nect(cli-
ent_id.c_str(),
mqtt_username,
mqtt_password)) {Se-
rial.println("Con-
nected to MQTT bro-
ker");
mqtt_client.sub-
scribe(topic_ntu);
mqttConnected = true;
// Update LCD dengan
status berhasil
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("MQTT Con-
nected");
lcd.setCursor(0, 1);
```

```
length) {Serial.print("Message
received on topic [");
Serial.print(topic);
Serial.print("]: ");
String message = "";
for (int i = 0; i < length; i++)
{message += (char)payload[i];
}
Serial.println(message);
}

void publishSensorDataJSON(const
char *topic, float value) {if
(!mqttConnected) return;
StaticJsonDocument<200> doc;
doc["ntu1"] = String(value,
4);char jsonBuffer[256];
serializeJson(doc,
jsonBuffer);mqtt_client.pub-
lish(topic, jsonBuffer);
Serial.print("Published JSON to
");
Serial.print(topic);
Serial.print(": ");
Serial.println(jsonBuffer);
}

void publishSensorData(const
char *topic, float value) {
if (!mqttConnected) return;
char msg[10];
snprintf(msg, 10, "%.4f",
value);
mqtt_client.publish(topic, msg);
Serial.print("Published to ");
Serial.print(topic);
Serial.print(": ");
Serial.println(msg);
}
void readsensor() {
// Baca nilai ADC
setI2CPins(SDA_ADS, SCL_ADS); //
Set I2C pins for ADS1115
currentADC0 = ads.readADC_Sin-
gleEnded(0);
Serial.print("Raw ADC Value: ");
Serial.println(currentADC0)
// Hitung tegangan aktual
currentVoltage0 = currentADC0 *
ADC_LSB / 1000;  // Konversi

void updateLCD() {
// Ganti display mode setiap 3
detik
lcd.print("Ready to
send");
delay(2000);
}
else {
Serial.print("Failed
to connect to MQTT
broker, rc=");Se-
rial.println(mqtt_cli-
ent.state());
mqttConnected = false;
// Update LCD dengan
status gagal
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("MQTT
Failed");
lcd.setCursor(0, 1);
lcd.print("Retry in
5s");
delay(5000);
break; // Exit loop to
prevent blocking
}
}
}
ke volt
Serial.print("Voltage
1 : ");
Serial.print(cur-
rentVoltage1, 4);
Serial.println(" V");
// Kalibrasi nilai NTU
if (currentADC0 >=
ZERO_ADC_LOW && cur-
rentADC0 <=
ZERO_ADC_HIGH) {ntu1 =
0.0;  // Dianggap 0
NTU dalam range ini
}else if (currentADC0
< ZERO_ADC_LOW) {
// Jika nilai di bawah
range zero
ntu1 = 0.0;
Serial.println("Warn-
ing: ADC value below
zero point!");
}else if (currentVolt-
age0 <= CAL_VOLTAGE ){
```

```
if (millis() - lastLcdUpdate >= 3000) {displayMode = (displayMode + 1) % 4;
lastLcdUpdate = millis();
scrollPosition = 0;
}
// Update setiap 500ms untuk animasi scroll if (millis() - lastScrollUpdate >= 500) {
setI2CPins(SDA_LCD, SCL_LCD); // Set I2C pins for LCD
lcd.clear();
switch (displayMode) {
case 0: // Tampilan utama NTU
lcd.setCursor(0, 0);
lcd.print("Turbidity Level");
lcd.setCursor(0, 1);
lcd.print(ntu1, 2);
lcd.print(" NTU");
if (ntu1 > 200) {
lcd.setCursor(11, 1);
lcd.print("HIGH!");
}
break;
case 1: // Tampilan tegangan dan ADC
lcd.setCursor(0, 0);
lcd.print("ADC: ");
lcd.print(currentADC0);
lcd.setCursor(0, 1);
lcd.print("V: ");
lcd.print(currentVoltage0, 4);
lcd.print("V");
break;
case 2: // Status koneksi
lcd.setCursor(0, 0);
lcd.print("WiFi:");
lcd.print(wifiConnected ? "OK" : "NO");
lcd.print(" MQTT:");
lcd.print(mqttConnected ? "OK" : "NO");
lcd.setCursor(0, 1);
lcd.print("Uptime:");
lcd.print(millis() / 1000);
lcd.print("s");
break;
```

```
void setup() {
Serial.begin(115200);
pinMode(34, INPUT);
// Initialize LCD
```

```
ntu1 = (currentVoltage0 / CAL_VOLTAGE) * CAL_NTU;
}else {
// Hitung tegangan terkoreksi (dikurangi offset)
float correctedVoltage = currentVoltage0 - ZERO_VOLTAGE - CAL_VOLTAGE ;
float maxCorrectedVoltage = MAX_VOLTAGE - ZERO_VOLTAGE - CAL_VOLTAGE ;
// Hitung NTU (proporsional terhadap tegangan)
ntu1 = CAL_NTU + (correctedVoltage / maxCorrectedVoltage) * (maxNTU - CAL_NTU);
// Batasi nilai antara 0-maxNTU
ntu1 = constrain(ntu1, 0.0, maxNTU);
}
Serial.print("Calculated NTU 1: ");Serial.println(ntu1, 2);
setI2CPins(SDA_ADS, SCL_ADS); // Set I2C pins for ADS1115
currentADC1 = ads.readADC_SingleEnded(1);
Serial.print("Raw ADC Value: ");
Serial.println(currentADC1);
currentVoltage1 = currentADC1 * ADC_LSB / 1000;  // Konversi ke volt
Serial.print("Voltage 2: ");
Serial.print(currentVoltage1, 4);
Serial.println(" V");
float ntu2 = map(currentVoltage1, 1.1, 4.09, 50, 0);
Serial.print("Calculated NTU 2: ");Serial.println(ntu2, 2);
```

```
setI2CPins(SDA_LCD, SCL_LCD);
lcd.init();
lcd.backlight();
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Turbidity Monitor");
lcd.setCursor(0, 1);
lcd.print("Starting...");
Serial.println("LCD initialized
on pins SDA:" + String(SDA_LCD)
+ " SCL:" + String(SCL_LCD));
delay(2000);
// Initialize ADS1115
setI2CPins(SDA_ADS, SCL_ADS);
if (!ads.begin()) {Se-
rial.println("Failed to initial-
ize ADS.");
setI2CPins(SDA_LCD, SCL_LCD);
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("ADS1115 Error");
lcd.setCursor(0, 1);
lcd.print("Check wiring");
while (1);
}ads.setGain(GAIN_ONE);
Serial.println("ADS1115 initial-
ized on pins SDA:" +
String(SDA_ADS) + " SCL:" +
String(SCL_ADS));
// Connect to WiFi
connectToWiFi();
 if (wifiConnected) {
// Setup MQTT
mqtt_client.setServer(mqtt_bro-
ker, mqtt_port);
mqtt_client.setCallback(mqtt-
Callback);
connectToMQTT();
// Setup Web Server
server.on("/", handleRoot);
server.on("/data", handleData);
server.onNotFound(handleNot-
Found);
server.begin();
Serial.println("Web server
started");Serial.print("Access
the dashboard at:
http://");Seral.println(WiFi.lo-
calIP());}
```

```
}case 3: // Informasi
jaringan (scroll)
lcd.setCursor(0, 0);
lcd.print("Network
Info");
lcd.setCursor(0, 1);
String ipString = "IP:
" + WiFi.lo-
calIP().toString();
if (ipString.length()
> 16) {
// Scroll teks panjang
int startPos = scroll-
Position % (ip-
String.length() - 15);
lcd.print(ip-
String.sub-
string(startPos,
startPos + 16));
scrollPosition++;
} else {
lcd.print(ipString);
}
break;
}
lastScrollUpdate =
millis();
}
}
// Web Server Handlers
void handleRoot() {
server.send(200,
"text/html",
htmlPage);
}
void handleData() {
StaticJsonDocu-
ment<300> doc;
doc["ntu"] = ntu1;
doc["adc"] = curren-
tADC0;
doc["voltage"] = cur-
rentVoltage0;
doc["wifi_connected"]
= wifiConnected;
doc["mqtt_connected"]
= mqttConnected;
doc["ip_address"] =
WiFi.lo-
calIP().toString();
doc["uptime"] = mil-
lis();
String jsonString;
serializeJson(doc,
jsonString);
```

```
server.send(200, "applica-
tion/json", jsonString);
}
void handleNotFound() {
server.send(404, "text/plain",
"Not Found");
}
// Tampilkan pesan ready di LCD
setI2CPins(SDA_LCD, SCL_LCD);
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("System Ready");
lcd.setCursor(0, 1);
lcd.print("Monitoring...");
delay(2000);
}
void loop() {
// Handle web server
if (wifiConnected)
{server.handleClient();
}// Read sensor every 500ms
if (millis() - lastSensorRead >=
500) {readsensor();
lastSensorRead = millis();
}// Update LCD display
updateLCD();
// Publish to MQTT every 500ms
if (millis() - lastPublish >=
500) {
if (mqttConnected) {
publishSensorDataJSON(topic_ntu,
ntu1);publishSensor-
Data(topic_ntu,
ntu1);}lastPublish = millis();}
// Check WiFi connection
if (WiFi.status() != WL_CON-
NECTED) {wifiConnected =
false;mqttConnected = false;
} else {wifiConnected
= true;
// Try to reconnect
MQTT if disconnectedif
(!mqtt_client.con-
nected()) {
mqttConnected = false;
static unsigned long
lastMqttAttempt = 0;
if (millis() -
lastMqttAttempt >
10000) { // Try every
10 seconds
connectToMQTT();
lastMqttAttempt = mil-
lis();
}} else {
mqttConnected =
true;}}
// Handle MQTT
if (mqttConnected) {
mqtt_client.loop();
}delay(10); // Small
delay for stability
} constTextStyle(font-
Size: 16))) .toList(),
), const
```

Lampiran 2. Kode Android Studio

About.dart

```
import ' package : flutter/
material. dart '; class
AboutPage extends Stateless-
Widget { AboutPage ({ super.
key });final List < String >
team Members = [" Septyawan
Bagus W.", "24014012 "];final
List < String > changelog = ["
v2 .0.0 - Bug fixes and new UI
Improvement"," v1 .3.0 - Fix-
ing MQTT Function on UI & Add
local storage MQTT Client","
v1 .2.0 - UI enhancements and
MQTT Client Features", " v1
.1.0 - Added new UI monitoring
feature ",];@ overrideWidget
build ( Build Context context)
{ return Scaffold (app Bar:
App Bar(title : const Text ("
About "), background Color :
Colors. orange ,),body : Cen-
ter( child : Column (main Ax-
isAlignment : Main AxisAlign-
ment . center , children :
[Clip RRect(borderRadius: Bor-
derRadius. circular (20) ,
child :          Image .
asset( ' asset/ images/
logo_bulet. png ',width : 250
,height: 250 ,fit: Box Fit.
cover ,),),const Sized Box (
height: 20), const Text(" De-
veloped by :",style :
TextStyle ( fontSize : 18 ,
fontWeight:  FontWeight.
bold),),const Sized Box ( Col-
umn (children : team Members.
map (( name ) => Text( name ,
style : const TextStyle (
fontSize : 16))). to List
(),),const Sized Box ( height:
20), const Text(" Changelog
:",style : TextStyle ( font-
Size : 18 ,  fontWeight:
FontWeight. bold ),),const
Sized Box ( height: 10), Col-
umn (children : changelog. map
(( log ) => Padding (padding :
const Edge Insets. symmetric(
vertical: 4), child : Text(log
,textAlign :  TextAlign . cen-
ter ,style : const TextStyle (
fontSize : 16),),)). to List
(),),const Sized Box ( height:
30), const Text(" Dibuat Guna
Mendukung Pengembangan Ap-
likasi Tugas Akhir", style :
TextStyle ( fontSize : 16 ,
fontWeight: FontWeight. bold
,color:  Colors. grey
),),],),),);}}
```

main.dart

```
import 'package:flutter/mate-
rial.dart'; import 'pack-
age:shared_prefer-
ences/shared_prefer-
ences.dart'; import 'pack-
age:get_time_ago/get_time_ago.
dart'; import 'package:syncfu-
sion_flut-
ter_charts/charts.dart'; im-
port 'package:flutter_na-
tive_splash/flutter_na-
tive_splash.dart'; import
'package:intl/intl.dart'; im-
port 'dart:async'; import
'dart:convert'; import
'notf_setting.dart'; import
'mqtt_service.dart'; import
'weather.dart'; import
"about.dart"; final
MqttService mqttService =
MqttService(); void main() {
WidgetsBinding widgetsBinding
= WidgetsFlutterBinding.en-
sureInitialized(); FlutterNa-
tiveSplash.preserve(widgets-
Binding: widgetsBinding); run-
App(MyApp()); FlutterNa-
tiveSplash.remove(); } class
ChartData { final DateTime
timestamp; final double pay-
load;
ChartData(this.timestamp,
this.payload); } class MyApp
```

```
extends StatefulWidget { const
MyApp({super.key}); @override
MyAppState createState() =>
MyAppState(); } class My-
AppState extends State<MyApp>
{ int currentIndex = 0; final
List<Widget> children = [
HomeScreen(), Analyt-
icsScreen(), SettingsScreen(),
]; void onTabTapped(int index)
{ setState(() {

currentIndex = index; }); }
@override Widget build(Build-
Context context) { return Ma-
terialApp( debug-
ShowCheckedModeBanner: false,
// Menghilangkan tulisan debug
title: 'Monitoring Sensor',
theme: ThemeData( prima-
rySwatch: Colors.orange,
fontFamily: 'Roboto', ), home:
Scaffold( body: children[cur-
rentIndex], bottomNavigation-
Bar: BottomNavigationBar(
items: const <BottomNaviga-
tionBarItem>[ BottomNaviga-
tionBarItem( icon:
Icon(Icons.home), label:
'Home', ), BottomNavigation-
BarItem( icon: Icon(Icons.ana-
lytics), label: 'Analytics',
), BottomNavigationBarItem(
icon: Icon(Icons.settings),
label: 'Settings', ), ], cur-
rentIndex: currentIndex, se-
lectedItemColor: Colors.or-
ange, unselectedItemColor:
Colors.grey, onTap: onTab-
Tapped, ), ), ); } } class
HomeScreen extends Stateful-
Widget { @override HomeScreen-
State createState() =>
HomeScreenState(); const
HomeScreen({super.key}); }

class HomeScreenState extends
State<HomeScreen> { final
WeatherService weatherService
= WeatherService(); String
topic = ''; String uptime =
'No data'; String dataTime =
''; String turbid = '0';
String temperature = 'Load-
ing...'; String city = 'Sema-
rang'; String turbidvalue =
'''; String lowestTurbid =
'0.0'; String lowestTurbid-
Timestamp = ''; String high-
estTurbid = '0.0'; String
highestTurbidTimestamp = '';
String averageTurbid = '0.0';
Timer? timer; @override void
initState() { super.init-
State(); getTop-
icFromSharedPreferences();
readTopic();
loadWeatherData(); timeAgo();
GetTimeAgo.setDefault-
Locale('id'); startTimer(); }
@override void dispose() {
timer?.cancel(); // Hentikan
timer saat widget dihapus su-
per.dispose(); } void start-
Timer() { timer = Timer.peri-
odic(Duration(seconds: 10),
(Timer timer) { // Refresh se-
tiap 10 detik setState(() {
readTopic(); // Ambil data
terbaru timeAgo(); }); }); }
Future<void> getTop-
icFromSharedPreferences()
async { // Mengambil topik
dari SharedPreferences final
prefs = await SharedPrefer-
ences.getInstance(); topic =
prefs.getString('Topic') ??
'default_topic'; }

Future<void> readTopic() async
{ await getTop-
icFromSharedPreferences(); fi-
nal prefs = await SharedPref-
erences.getInstance(); String
key = 'mqtt_history_$topic';
List<String>? history =
prefs.getStringList(key); if
(history == null) { history =
[]; }

double lowestValue = dou-
ble.infinity; String
lowestTimestamp = ''; String
highestTimestamp = ''; double
highestValue = -double.infin-
ity; double totalValue = 0.0;
// Untuk menyimpan total nilai
int dataCount = his-
tory.length; // Jumlah data
final DateTime now =
DateTime.now(); // Waktu saat
ini final DateTime last24Hours
```

```
= now.subtract(Duration(hours:
24)); // Waktu 24 jam yang
lalu for (String item in his-
tory) { final Map<String, dy-
namic> mqttData =
jsonDecode(item); // Ambil
payload dan timestamp final
double count = double.try-
Parse(mqttData['payload'] ??
'0.0') ?? 0.0; final String
gettime = mqtt-
Data['timestamp'] ?? ''; final
timed = DateTime.parse(get-
time); final String timestamp
= DateFormat('yyyy-MM-dd
HH:mm').for-
mat(timed).toString(); if
(timed.isAfter(last24Hours)) {
// fungsi menghitung rata2 da-
lam 24 jam totalValue +=
count; // Count Total untuk
Average dataCount++; double
averageValue = dataCount > 0 ?
totalValue / dataCount : 0.0;
// Rata-rata setState(() { av-
erageTurbid = aver-
ageValue.toStringAsFixed(2);
// Simpan rata-rata dengan 2
digit desimal }); } if (count
< lowestValue) { lowestValue =
count; lowestTimestamp =
timestamp; }else if (count >
highestValue) { highestValue =
count; highestTimestamp =
timestamp; } setState(() {
turbid = count.toString();

if (count > 100) { Notifica-
tionService.showNotification(
title: 'Warning Turbidity
Meningkat', body: 'Trubidity
Naik $turbid NTU - Warning
🟤' ); } lowestTurbid =
lowestValue.toString();
lowestTurbidTimestamp =
lowestTimestamp; highestTurbid
= highestValue.toString();
highestTurbidTimestamp = high-
estTimestamp; }); } } Fu-
ture<void> loadWeatherData()
async { try { Map<String, dy-
namic> weatherData = await
weatherService.getWeather();
double temp =
weatherData['main']['temp'];
setState(() { temperature =
'${temp.toStrin-
gAsFixed(1)}°C'; }); } catch
(e) { setState(() { tempera-
ture = 'Error'; }); } } Fu-
ture<void> timeAgo() async {
await getTopicFromSharedPref-
erences(); final prefs = await
SharedPreferences.get-
Instance(); String key =
'mqtt_history_$topic';
List<String>? history =
prefs.getStringList(key); if
(history != null && his-
tory.isNotEmpty) { for (String
item in history) { final
Map<String, dynamic> mqttData
= jsonDecode(item); // Ambil
timestamp final String
timestamp = mqtt-
Data['timestamp']; DateTime
dataTime =
DateTime.parse(timestamp);
setState(() {//Set State untuk
mengupdate UI var live = Get-
TimeAgo.parse(dataTime); //
Parsing ke Waktu uptime =
live.toString(); //Pass data
ke string agar tampil ke UI
}); }} }

@override Widget build(Build-
Context context) { return
Scaffold( appBar: AppBar(
backgroundColor: Colors.or-
ange, elevation: 0, leading:
Padding( padding: const Edge-
Insets.all(8.0), child: Cir-
cleAvatar( backgroundImage:
AssetImage('asset/images/Per-
son.jpg'), ), ), title:
Text('Monitoring App',
textAlign: TextAlign.center,
style: TextStyle(fontSize: 24,
fontWeight: FontWeight.bold,
fontFamily: 'Roboto'), ), ),
body: SingleChildScrollView(
padding: const EdgeIn-
sets.all(16.0), child: Column(
crossAxisAlignment: CrossAxis-
Alignment.start, children: [
Text( 'Hi Operator', style:
TextStyle(fontSize: 24, font-
Weight: FontWeight.bold), ),
SizedBox(height: 8), Text(
'Welcome to Apps', style:
TextStyle(fontSize: 16, color:
```

```
Colors.grey[600]), ), SizedBox(height: 28), // Menambahkan kartu informasi IntrinsicHeight( child: Row( mainAxisAlignment: MainAxisAlignment.spaceBetween, children: [ Expanded( child: InfoCard( color: Colors.purple, title: temperature, subtitle: city,

status: '',

), ), SizedBox(width: 8), // Menambahkan jarak antara kartu Expanded( child: InfoCard( color: Colors.pink, title: turbid, subtitle: 'NTU', status: 'Normal', ), ), SizedBox(width: 8), // Menambahkan jarak antara kartu Expanded( child: InfoCard( color: Colors.orange, title: uptime, subtitle: 'Last Alive', status: '', ), ), ], ) ), SizedBox(height: 24), // Kartu biasa dengan warna oranye Row( mainAxisAlignment: MainAxisAlignment.spaceBetween, children: [ Expanded( child: SimpleCard( title: 'Data Terendah Terakhir', familyMembers: '$lowestTurbid NTU', devices: 'Pada: $lowestTurbidTimestamp' ), ), ], ), SizedBox(height: 24), // Kartu biasa dengan warna oranye Row( mainAxisAlignment: MainAxisAlignment.spaceBetween, children: [ Expanded(

child: SimpleCard( title: 'Data Tertinggi Terakhir', familyMembers: '$highestTurbid NTU', devices: 'Pada: $highestTurbidTimestamp' ), ), ], ), SizedBox(height: 24), // Kartu biasa dengan warna oranye Row( mainAxisAlignment: MainAxisAlignment.spaceBetween, children: [ Expanded( child: SimpleCard( title: 'Rata - Rata Harian', familyMembers: '$averageTurbid NTU', devices: '', ), ), ], ), ], ), ); } } class AnalyticsScreen extends StatefulWidget { const AnalyticsScreen({super.key}); @override AnalyticsScreenState createState() => AnalyticsScreenState(); } class AnalyticsScreenState extends State<AnalyticsScreen> { Future<List<ChartData>>? chartData; String topic = ''; Timer? timer; Future<void> getTopicFromSharedPreferences() async { // Mengambil topik dari SharedPreferences final prefs = await SharedPreferences.getInstance(); topic = prefs.getString('Topic') ?? 'default_topic'; } Future<List<ChartData>> getChartData() async {

final prefs = await SharedPreferences.getInstance(); String key = 'mqtt_history_$topic'; // Ganti dengan key yang sesuai List<String>? history = prefs.getStringList(key); if (history == null || history.isEmpty) { return []; } List<ChartData> chartData = []; final DateTime now = DateTime.now(); // Waktu saat ini final DateTime last15Minutes = now.subtract(Duration(minutes: 15)); // Waktu 15 menit yang lalu for (String item in history) { final Map<String, dynamic> mqttData = jsonDecode(item); final double payload = double.tryParse(mqttData['payload'] ?? '0.0') ?? 0.0; final String timestamp = mqttData['timestamp'] ?? ''; // Konversi timestamp ke DateTime final DateTime dateTime = DateTime.parse(timestamp); // Cek apakah data masih dalam 15 menit terakhir if (dateTime.isAfter(last15Minutes)) { // Tambahkan ke list chartData chartData.add(ChartData(dateTime, payload)); } } return
```

```
chartData; } void startTimer()
{ timer = Timer.periodic(Dura-
tion(seconds: 7), (Timer
timer) { // Refresh setiap 5
detik setState(() { chartData
= getChartData(); // Ambil
data terbaru }); }); } @over-
ride void initState() { su-
per.initState(); getTop-
icFromSharedPreferences();
chartData = getChartData(); //
Ambil data saat inisialisasi
startTimer(); } @override void
dispose() { timer?.cancel();
// Hentikan timer saat widget
dihapus super.dispose(); }

@override Widget build(Build-
Context context) { return
Scaffold( appBar: AppBar(
backgroundColor: Colors.or-
ange, elevation: 0, title:
Text('Chart Analisa TF 15 Men-
it'), ), body: Future-
Builder<List<ChartData>>( fu-
ture: chartData, builder:
(context, snapshot) { if
(snapshot.connectionState ==
ConnectionState.waiting) { re-
turn Center(child: Circu-
larProgressIndicator()); }
else if (snapshot.hasError) {
return Center(child: Text('Er-
ror: ${snapshot.error}')); }
else if (!snapshot.hasData ||
snapshot.data!.isEmpty) { re-
turn Center(child: Text('No
data available')); } else {
return Padding( padding: const
EdgeInsets.all(16.0), child:
SfCartesianChart( prima-
ryXAxis: DateTimeAxis( title:
AxisTitle(text: 'Waktu'),
dateFormat: DateFormat('dd/MM
HH:mm'), // Format tanggal dan
waktu ), primaryYAxis: Numeri-
cAxis( title: AxisTitle(text:
'Turbidity'), ), series: <Car-
tesianSeries<ChartData,
DateTime>>[
LineSeries<ChartData,
DateTime>( // Ganti StepLi-
neSeries dengan LineSeries
dataSource: snapshot.data!,
xValueMapper: (ChartData data,
_) => data.timestamp,
yValueMapper: (ChartData data,
_) => data.payload, color:
Colors.orange, width: 2, mark-
erSettings: MarkerSettings(is-
Visible: true), // Tampilkan
marker ), ], ), ); } }, )

); } } class SettingsScreen
extends StatelessWidget {
const SettingsScreen({su-
per.key}); @override Widget
build(BuildContext context) {
return Scaffold( appBar: Ap-
pBar( backgroundColor: Col-
ors.orange, elevation: 0, ti-
tle: Text('Settings'), ),
body: ListView( children: [
ListTile( leading:
Icon(Icons.settings), title:
Text('MQTT Settings'), onTap:
() { Navigator.push( context,
MaterialPageRoute(builder:
(context) => MqttSet-
tingsScreen()), ); }, ), List-
Tile( leading:
Icon(Icons.info), title:
Text('About'), onTap: () {
Navigator.push( context, Mate-
rialPageRoute(builder: (con-
text) => AboutPage()), ); },
), ], ), ); } } class InfoCard
extends StatelessWidget { fi-
nal Color color; final String
title;

final String subtitle; final
String status; const Info-
Card({required this.color, re-
quired this.title, required
this.subtitle,required
this.status, super.key});
@override Widget build(Build-
Context context) { return
Card( color: color, child:
Padding( padding: const Edge-
Insets.all(16.0), child: Col-
umn( children: [ Text( title,
style: TextStyle(fontSize: 24,
fontWeight: FontWeight.bold,
color: Colors.white),
textAlign: TextAlign.justify,
), SizedBox(height: 8), Text(
subtitle, style:
TextStyle(fontSize: 16, color:
Colors.white), textAlign:
TextAlign.center, ),
```

```
SizedBox(height: 8), Text(sta-
tus, style: TextStyle(font-
Size: 16, color: status ==
'Warning' ? Colors.red : Col-
ors.white), textAlign:
TextAlign.justify), ], ), ),
); } } class SimpleCard ex-
tends StatelessWidget { final
String title; final String
familyMembers; final String
devices; const SimpleCard({re-
quired this.title, required
this.familyMembers, required
this.devices, super.key});
@override Widget build(Build-
Context context) { return Ges-
tureDetector( onTap: () {

// Navigasi ke halaman detail
atau pengaturan Naviga-
tor.push( context, Materi-
alPageRoute(builder: (context)
=> MqttPage()), ); }, child:
Card( color: Colors.orange,
child: Padding( padding: const
EdgeInsets.all(16.0), child:
Column( crossAxisAlignment:
CrossAxisAlignment.start,
children: [ Text( title,
style: TextStyle(fontSize: 20,
fontWeight: FontWeight.bold),
), SizedBox(height: 4),
Text(familyMembers, style:
TextStyle(fontSize: 16)),
SizedBox(height: 4), Text(de-
vices, style: TextStyle(font-
Size: 16, color: devices ==
'Warning' ? Colors.red : Col-
ors.black)), ], ), ), ), ); }
} class MqttPage extends
StatelessWidget { const
MqttPage({super.key}); @over-
ride Widget build(BuildContext
context) { return Scaffold(
appBar: AppBar( title:
Text('MQTT Page'), ), body:
Center( child: Text('MQTT Page
Content'), ), ); } }

class MqttSettingsScreen ex-
tends StatefulWidget { const
MqttSettingsScreen({su-
per.key}); @override MqttSet-
tingsScreenState createState()
=> MqttSettingsScreenState();
} class
```

```
MqttSettingsScreenState ex-
tends State<MqttSet-
tingsScreen> { final _formKey
= GlobalKey<FormState>();
String _brokerAddress = '';
String _port = ''; String
_clientId = ''; String _topic
= ''; bool _isSaved = false;
bool _isConnected = false;
bool _isSubscribed = false;
@override void initState() {
super.initState(); _loadSet-
tings(); _checkMqttConnec-
tion(); Notification-
Service.initialize(); if (_is-
Connected) { _isConnected =
mqttService.isConnected; } }
Future<void> _loadSettings()
async { SharedPreferences
prefs = await SharedPrefer-
ences.getInstance();
setState(() { _brokerAddress =
prefs.getString('broker-
Address') ?? ''; _port =
prefs.getString('port') ?? '';
_clientId =
prefs.getString('clientId') ??
''; _topic =
prefs.getString('Topic') ??
''; _isSaved = _broker-
Address.isNotEmpty ||
_port.isNotEmpty || _clien-
tId.isNotEmpty; }); } void
_checkMqttConnection() {
setState(() { _isConnected =
mqttService.isConnected; }); }
Future<void> _saveSettings()
async { if (_formKey.current-
State!.validate()) {
_formKey.currentState!.save();

SharedPreferences prefs =
await SharedPreferences.get-
Instance(); await
prefs.setString('broker-
Address', _brokerAddress);
await prefs.setString('port',
_port); await
prefs.setString('clientId',
_clientId); await
prefs.setString('Topic',
_topic); setState(() { _is-
Saved = true; }); } } Fu-
ture<void> _toggleConnection()
async { int port = int.try-
Parse(_port) ?? 1883; if
```

```
(_isConnected) { await
mqttService.disconnect();
setState(() { _isConnected =
false; _isSubscribed = false;
}); NotificationService.showN-
otification( title: 'MQTT Dis-
connected', body: 'Discon-
nected from MQTT broker - Dis-
connected 🔴' ); } else {
bool connected = await
mqttService.connect( broker:
_brokerAddress, port: port,
clientId: _clientId, topic:
_topic, ); setState(() { _is-
Connected = connected; }); if
(connected) { Notification-
Service.showNotification( ti-
tle: 'MQTT Connected', body:
'Successfully connected to
MQTT broker - Connected ✅',
); } else { Notification-
Service.showNotification( ti-
tle: 'MQTT Connection Failed',
body: 'Failed to connect to
MQTT broker - Failed 🔴', );
}

} } Future<void> _toggleSub-
scribe() async { if (_isSub-
scribed) { // Unsubscribe dari
semua topic bool unsubscribed
= await mqttService.unsub-
scribeAllTopics(); setState(()
{ _isSubscribed = !unsub-
scribed; }); Notification-
Service.showNotification( ti-
tle: 'MQTT Successfully Unsub-
scribed', body: 'Successfully
Not Subscribed ❌', ); } else
{ // Subscribe ke semua topic
bool subscribed = await
mqttService.subscribeToAllTop-
ics(); setState(() { _isSub-
scribed = subscribed; }); No-
tificationService.showNotifi-
cation( title: 'MQTT Success-
fully subscribed', body: 'Suc-
cessfully All Topics Sub-
scribed ☑️',

} } Widget _buildText-
Field(String label, String
value, Function(String) on-
Saved) { return Padding( pad-
ding: const
EdgeInsets.only(bottom: 10),
child: TextFormField( ini-
tialValue: value, decoration:
InputDecoration( labelText:
label, border: OutlineIn-
putBorder(), ), validator:
(value) => value == null ||
value.isEmpty ? 'Please enter
$label' : null, onSaved:
(value) => onSaved(value!), ),
); } Widget _build-
LastSavedSettings() { return
Column( crossAxisAlignment:
CrossAxisAlignment.start,

children: [ Text( 'Last Saved
Settings:', style:
TextStyle(fontWeight: Font-
Weight.bold, fontSize: 16), ),
SizedBox(height: 5),
Text('Broker Address: $_bro-
kerAddress'), Text('Port:
$_port'), Text('Client ID:
$_clientId'), Text('Topic :
$_topic'), ], ); } @override
Widget build(BuildContext con-
text) { return Scaffold( ap-
pBar: AppBar( title:
Text('MQTT Settings'), back-
groundColor: Colors.orange, ),
body: SingleChildScrollView(
child: Padding( padding: const
EdgeInsets.all(16.0), child:
Column( children: [ Form( key:
_formKey, child: Column( chil-
dren: [ _buildTextField('Bro-
ker Address', _brokerAddress,
(value) => _brokerAddress =
value), _buildText-
Field('Port', _port, (value)
=> _port = value), _buildText-
Field( 'Client ID', _clientId,
(value) => _clientId = value),
_buildTextField('Topic',
_topic, (value) => _topic =
value), SizedBox(height: 20),
Row( mainAxisAlignment:
MainAxisAlignment.spaceBe-
tween, children: [ Expanded(
child: ElevatedButton( on-
Pressed: _saveSettings, style:
ElevatedButton.styleFrom(

backgroundColor: Colors.blue),
child: Text('Save Settings'),
), ), SizedBox(width: 10),
```

```
Expanded( child: ElevatedBut-
ton( onPressed: _toggleConnec-
tion, style: ElevatedBut-
ton.styleFrom( background-
Color: _isConnected ? Col-
ors.red : Colors.green, ),
child: Text(_isConnected ?
'Disconnect' : 'Connect'), ),
), SizedBox(width: 10), Ex-
panded( child: ElevatedButton(
onPressed: _isConnected ?
_toggleSubscribe : null,
style: ElevatedButton.style-
From( backgroundColor: _isSub-
scribed ? Colors.red : Col-
ors.green, ), child: Text(_is-
Subscribed ? 'Unsubscribe All'
: 'Subscribe All'), ), ), ],
), SizedBox(height: 20), _is-
Saved ? _buildLastSavedSet-
tings() : Text('No settings
saved yet.', style:
TextStyle(color: Col-
ors.grey)), SizedBox(height:
20), Text( _isConnected ?
'MQTT Status: Connected 🕷' :
'MQTT Status: Disconnected
🔘', style: TextStyle( font-
Weight: FontWeight.bold,

fontSize: 16, color: _isCon-
nected ? Colors.green : Col-
ors.red, ), ),
SizedBox(height: 10), Text(
_isSubscribed ? 'Subscription
Status: All Topics Subscribed
☑' : 'Subscription Status:
Not Subscribed ❌', style:
TextStyle( fontWeight: Font-
Weight.bold, fontSize: 16,
color: _isSubscribed ? Col-
ors.green : Colors.red, ), ),
], ), ), ], ), ), ), ); } }
```

weather.dart

```
import 'dart:convert'; import
'package:http/http.dart' as
http; class WeatherService {
final String apiKey =
'8ede7f8b710ac896e472a47f83700
ac9'; // Ganti dengan API key
Anda final String city = 'Se-
marang'; // Ganti dengan kota
yang diinginkan

Future<Map<String, dynamic>>
getWeather() async { final url
= Uri.parse(
'https://api.openweather-
map.org/data/2.5/weather?q=$ci
ty&appid=$apiKey&units=met-
ric', ); try { final response
= await http.get(url); if (re-
sponse.statusCode == 200) {
return jsonDecode(re-
sponse.body); } else

throw Exception('Failed to
load weather data'); } } catch
(e) { print('Error fetching
weather data: $e'); throw Ex-
ception('Failed to load
weather data'); } } }
```

notf_setting.dart

```
import 'package:flutter_lo-
cal_notifications/flutter_lo-
cal_notifications.dart'; class
NotificationService { static
final FlutterLocalNotifica-
tionsPlugin _flutterLocalNoti-
ficationsPlugin = FlutterLo-
calNotificationsPlugin();
static Future<void> initial-
ize() async { const AndroidI-
nitializationSettings initial-
izationSettingsAndroid = An-
droidInitializationSet-
tings('@mipmap/ic_launcher');
final InitializationSettings
initializationSettings = Ini-
tializationSettings( android:
initializationSettingsAndroid,
); await _flutterLocalNotifi-
cationsPlugin.initialize(ini-
tializationSettings); } static
Future<void> showNotification(
{required String title, re-
quired String body}) async {
const AndroidNotificationDe-
tails androidPlatformChannel-
Specifics = AndroidNotifica-
tionDetails( 'your_chan-
nel_id', 'your_channel_name',
channelDescription:
'your_channel_description',
importance: Importance.max,
priority: Priority.high, show-
When: false, ); const Notifi-
cationDetails
```

```
platformChannelSpecifics = No-
tificationDetails(android: an-
droidPlatformChannelSpecif-
ics); await _flutterLocalNoti-
ficationsPlugin.show( 0, ti-
tle, body, platformChannelSpe-
cifics, p

); } }
```

mqtt_service.dart

```
// mqtt_service.dart import
'dart:async'; import
'dart:convert'; import
'dart:io'; import 'pack-
age:flutter/services.dart';
import 'package:mqtt_cli-
ent/mqtt_client.dart'; import
'package:mqtt_cli-
ent/mqtt_server_client.dart';
import 'package:shared_prefer-
ences/shared_prefer-
ences.dart'; enum MqttConnec-
tionState { disconnected, con-
necting, connected, reconnect-
ing, error } class MqttService
{ static final MqttService
_instance = MqttService._in-
ternal(); factory
MqttService() => _instance;
MqttService._internal(); late
MqttServerClient client;
MqttConnectionState _connec-
tionState = MqttConnection-
State.disconnected; MqttCon-
nectionState get connection-
State => _connectionState;
bool get isConnected => _con-
nectionState == MqttConnec-
tionState.connected; String?
topic; // Connection parame-
ters String? _broker; int?
_port; String? _clientId;
String? _topic; String?
_username; String? _password;
// Auto reconnect properties
Timer? _reconnectTimer; Timer?
_keepAliveTimer; Timer? _con-
nectionTimeoutTimer; bool
_shouldAutoReconnect = true;
int _reconnectAttempts = 0;

int _maxReconnectAttempts =
10; Duration _reconnectDelay =
const Duration(seconds: 5);
Duration _connectionTimeout =
const Duration(seconds: 30);
// Stream controllers final
StreamController<String> _mes-
sageController = StreamCon-
troller<String>.broadcast();
final StreamControl-
ler<MqttConnectionState> _con-
nectionStateController =
StreamController<MqttConnec-
tionState>.broadcast();
Stream<String> get mes-
sageStream => _messageControl-
ler.stream; Stream<MqttConnec-
tionState> get connection-
StateStream => _connection-
StateController.stream; //
Connection methods Fu-
ture<bool> connect({ required
String broker, required int
port, required String clien-
tId, required String topic,
String? username, String?
password, bool autoReconnect =
true, int maxReconnectAttempts
= 10, Duration reconnectDelay
= const Duration(seconds: 5),
}) async { // Store connection
parameters _broker = broker;
_port = port; _clientId = cli-
entId; _topic = topic;
_username = username; _pass-
word = password; _shouldAuto-
Reconnect = autoReconnect;
_maxReconnectAttempts = maxRe-
connectAttempts; _reconnectDe-
lay = reconnectDelay; return
await _attemptConnection(); }
Future<bool> _attemptConnec-
tion() async { if (_broker ==
null || _port == null || _cli-
entId == null) { print('❌
Connection parameters not
set'); _updateConnection-
State(MqttConnectionState.er-
ror); return false; } if
(_connectionState == MqttCon-
nectionState.connecting ||
_connectionState == MqttCon-
nectionState.reconnecting) {

print('⏳ Connection already
in progress'); return false; }
try { _updateConnection-
State(_reconnectAttempts > 0 ?
```

```
MqttConnectionState.reconnecting : MqttConnectionState.connecting); print('🔄 ${_reconnectAttempts > 0 ? "Reconnecting" : "Connecting"} to MQTT broker: $_broker:$_port'); // Cleanup previous client await _cleanupClient(); // Create new client client = MqttServerClient(_broker!, _clientId!); client.port = _port!; client.logging(on: false); client.keepAlivePeriod = 20; client.connectTimeoutPeriod = 30000; // 30 seconds client.autoReconnect = false; // We handle reconnection manually // Set callbacks client.onDisconnected = _onDisconnected; client.onConnected = _onConnected; client.onSubscribed = _onSubscribed; client.onAutoReconnect = _onAutoReconnect; client.onAutoReconnected = _onAutoReconnected; // Setup connection message final connMessage = MqttConnectMessage() .withClientIdentifier(_clientId!) .withWillQos(MqttQos.atMostOnce) .startClean(); if (_username != null && _password != null) { connMessage.authenticateAs(_username!, _password!); } client.connectionMessage = connMessage; // Set connection timeout _startConnectionTimeout(); // Attempt connection await client.connect(); _cancelConnectionTimeout(); final isConnected = client.connectionStatus!.state == MqttConnectionState.connected; if (isConnected) { _updateConnectionState(MqttConnectionState.connected); _reconnectAttempts = 0; await _subscribeToTopics(); _startKeepAliveMonitoring();

print('☑ MQTT Connected successfully!'); } else { _updateConnectionState(MqttConnectionState.error);

print('❌ MQTT Connection failed'); } return isConnected; } catch (e) { _cancelConnectionTimeout(); _updateConnectionState(MqttConnectionState.error);

print('❌ Connection error: $e'); if (_shouldAutoReconnect && _reconnectAttempts < _maxReconnectAttempts) { _scheduleReconnect(); } return false; } } void _startConnectionTimeout() { _connectionTimeoutTimer?.cancel(); _connectionTimeoutTimer = Timer(_connectionTimeout, () { print('⏰ Connection timeout'); _updateConnectionState(MqttConnectionState.error); client.disconnect(); if (_shouldAutoReconnect && _reconnectAttempts < _maxReconnectAttempts) { _scheduleReconnect(); } }); } void _cancelConnectionTimeout() { _connectionTimeoutTimer?.cancel(); _connectionTimeoutTimer = null; } void _onConnected() { print('☑ MQTT Connected callback triggered'); _updateConnectionState(MqttConnectionState.connected); _reconnectAttempts = 0; } void _onDisconnected() { print('🔌 MQTT Disconnected callback triggered'); _updateConnectionState(MqttConnectionState.disconnected); _stopKeepAliveMonitoring(); if (_shouldAutoReconnect && _reconnectAttempts < _maxReconnectAttempts) { _scheduleReconnect(); } else if (_reconnectAttempts >= _maxReconnectAttempts) {

print('🔘 Max reconnection attempts ($_maxReconnectAttempts) reached'); _updateConnectionState(MqttConnectionState.error); } } void _onSubscribed(String topic) { print('📡 Subscribed to:
```

```
$topic'); } void _onAutoRecon-
nect() { print('🔄 Auto re-
connect triggered by mqtt_cli-
ent'); } void _onAutoRecon-
nected() { print('☑ Auto re-
connected by mqtt_client');
_updateConnection-
State(MqttConnectionState.con-
nected); } void _scheduleRe-
connect() { if (!_shouldAuto-
Reconnect) return; _recon-
nectTimer?.cancel(); _recon-
nectAttempts++; // Exponential
backoff with jitter final
backoffDelay = Duration( sec-
onds: (_reconnectDelay.inSec-
onds * (_reconnectAttempts *
0.5)).round().clamp(5, 60) );
print('🔄 Scheduling recon-
nect in ${backoffDelay.inSec-
onds}s (Attempt $_reconnec-
tAttempts/$_maxReconnec-
tAttempts)'); _reconnectTimer
= Timer(backoffDelay, () async
{ if (_shouldAutoReconnect &&
!isConnected) { await _at-
temptConnection(); } }); }
void _startKeepAliveMonitor-
ing() { _stopKeepAliveMonitor-
ing(); _keepAliveTimer =
Timer.periodic(const Dura-
tion(seconds: 30), (timer) {
if (!isConnected) { timer.can-
cel(); return; } // Check if
connection is still alive if
(client.connectionSta-
tus?.state != MqttConnection-
State.connected) { print('💔
Connection lost detected by
keep-alive monitor'); _onDis-
connected(); } else { // Send
ping to test connection

try { client.publishMes-
sage('ping/test', MqttQos.at-
MostOnce, MqttClientPay-
loadBuilder().addString('ping'
).payload!); } catch (e) {
print('💔 Keep-alive ping
failed: $e'); _onDiscon-
nected(); } } }); } void
_stopKeepAliveMonitoring() {
_keepAliveTimer?.cancel();
_keepAliveTimer = null; } void
_updateConnection-
State(MqttConnectionState
newState) { if (_connection-
State != newState) { _connec-
tionState = newState; _connec-
tionStateControl-
ler.add(newState); print('📊
Connection state changed to:
${newState.toString().split('.
').last}'); } } // Subscrip-
tion methods Future<bool>
_subscribeToTopics() async {
if (!isConnected) { print('❌
Cannot subscribe, MQTT is not
connected'); return false; }
try { final prefs = await
SharedPreferences.get-
Instance(); topic =
prefs.getString('Topic') ??
_topic; if (topic != null) {
client.subscribe(topic!,
MqttQos.exactlyOnce); _start-
MessageListener(); print('📡
Subscribed to: $topic'); re-
turn true; } else { print('⚠
No topic found to subscribe');
return false; } } catch (e) {
print('❌ Subscription error:
$e'); return false;


}


} void _startMessageListener()
{ client.updates?.listen(
(List<MqttReceivedMes-
sage<MqttMessage?>>? messages)
{ if (messages != null) { for
(var message in messages) {
final receivedTopic = mes-
sage.topic; final payload =
MqttPublishPay-
load.bytesToStringAsString(
(message.payload as
MqttPublishMessage).pay-
load.message); _handleRe-
ceivedMessage(receivedTopic,
payload); } } }, onError: (er-
ror) { print('❌ Error re-
ceiving messages: $error'); if
(_shouldAutoReconnect) {
_onDisconnected(); } }, can-
celOnError: false, ); } void
_handleReceivedMessage(String
receivedTopic, String payload)
```

```
{ print('📨 Received: "$payload" from: "$receivedTopic"'); // Save message to history _saveMqttData(receivedTopic, payload); // Send to stream if (receivedTopic == topic) { if (!_messageController.isClosed) { _messageController.add(payload); } } } // Publishing methods Future<bool> publishMessage(String pubTopic, String message) async { if (!isConnected) { print('❌ Cannot publish, MQTT not connected. Attempting reconnect...'); if (await _attemptConnection()) { return await publishMessage(pubTopic, message); } return false; } try {

final builder = MqttClientPayloadBuilder(); builder.addString(message); client.publishMessage(pubTopic, MqttQos.atLeastOnce, builder.payload!); print('📤 Published: "$message" to "$pubTopic"'); return true; } catch (e) { print('❌ Publish error: $e'); return false; } } // Data persistence methods Future<void> _saveMqttData(String saveTopic, String payload) async { try { final prefs = await SharedPreferences.getInstance(); final historyKey = 'mqtt_history_$saveTopic'; List<String> history = prefs.getStringList(historyKey) ?? []; final mqttData = { 'topic': saveTopic, 'payload': payload, 'timestamp': DateTime.now().toIso8601String(), }; history.add(jsonEncode(mqttData)); // Keep only last 1000 messages to prevent storage bloat if (history.length > 1000) { history = history.sublist(history.length - 1000); } await prefs.setStringList(historyKey, history); print('💾 Data saved: $saveTopic'); } catch (e) { print('❌ Error saving data: $e'); } } Future<List<Map<String, dynamic>>> getSavedHistory(String historyTopic) async { try { final prefs = await SharedPreferences.getInstance(); final history = prefs.getStringList('mqtt_history_$historyTopic') ?? []; return history.map((item) => jsonDecode(item) as Map<String, dynamic>).toList(); } catch (e) { print('❌ Error getting history: $e'); return []; } } Future<void> clearHistory(String historyTopic) async {

try { final prefs = await SharedPreferences.getInstance(); await prefs.remove('mqtt_history_$historyTopic'); print('🗑️ History cleared for: $historyTopic'); } catch (e) { print('❌ Error clearing history: $e'); } } // Control methods Future<bool> reconnect() async { print('🔄 Manual reconnect requested'); _reconnectAttempts = 0; return await _attemptConnection(); } void setAutoReconnect(bool enabled) { _shouldAutoReconnect = enabled; print('🔧 Auto reconnect ${enabled ? "enabled" : "disabled"}'); if (!enabled) { _reconnectTimer?.cancel(); } } void setMaxReconnectAttempts(int attempts) { _maxReconnectAttempts = attempts; print('🔧 Max reconnect attempts set to: $attempts'); } void setReconnectDelay(Duration delay) { _reconnectDelay = delay; print('🔧 Reconnect
```

```
delay set to:
${delay.inSeconds}s'); }
Future<bool>
unsubscribeFromTopic(String
unsubTopic) async { if
(!isConnected) { print('❌
Cannot unsubscribe, MQTT not
connected'); return false; }
try {
client.unsubscribe(unsubTopic)
; print('🔕 Unsubscribed
from: $unsubTopic'); return
true; } catch (e) { print('❌
Unsubscribe error: $e');
return false; }

} // Cleanup methods Fu-
ture<void> _cleanupClient()
async { try { if (client.con-
nectionStatus?.state ==
MqttConnectionState.connected)
{ client.disconnect(); } }
catch (e) { print('⚠️ Error
during client cleanup: $e'); }
} Future<void> disconnect()
async { print('🔌 Disconnect-
ing MQTT...'); _shouldAutoRe-
connect = false; _recon-
nectTimer?.cancel();
_keepAliveTimer?.cancel();
_connectionTimeoutTimer?.can-
cel(); await _cleanupClient();
_updateConnection-
State(MqttConnectionState.dis-
connected); print('☑️ MQTT
Disconnected'); } void dis-
pose() { print('🗑️ Disposing
MQTT Service...'); _shoul-
dAutoReconnect = false; _re-
connectTimer?.cancel();
_keepAliveTimer?.cancel();
_connectionTimeoutTimer?.can-
cel(); if (isConnected) { cli-
ent.disconnect(); } if (!_mes-
sageController.isClosed) {
_messageController.close(); }
if (!_connectionStateControl-
ler.isClosed) { _connection-
StateController.close(); }
print('☑️ MQTT Service dis-
posed'); } // Utility methods
String getConnectionSta-
tusString() { switch
(_connectionState) { case
MqttConnectionState.connected:
r

qttConnectionState.connecting:
return 'Connecting...'; case
MqttConnectionState.reconnect-
ing: return 'Reconnecting...
($_reconnectAttempts/$_maxRe-
connectAttempts)'; case
MqttConnectionState.discon-
nected: return 'Disconnected';
case MqttConnectionState.er-
ror: return 'Connection Er-
ror'; } } Map<String, dynamic>
getConnectionInfo() { return {
'broker': _broker, 'port':
_port, 'clientId': _clientId,
'topic': topic, 'state':
getConnectionStatusString(),
'autoReconnect': _shouldAuto-
Reconnect, 'reconnec-
tAttempts': _reconnec-
tAttempts, 'maxReconnec-
tAttempts': _maxReconnec-
tAttempts, }; } }
```