

TOYOTA

Learning Module

METER COMBI



Capability Center

METER COMBI

DAFTAR ISI

Bagian 1 Pendahuluan	1
1.1 Pengantar	1
1.2 Apa Itu Meter Combi	1
Bagian 2 Komponen Sistem Meter Combi.....	3
2.1 Komponen Sistem	3
2.1.1 Perangkat Keras (<i>Hardware</i>).....	3
2.1.2 Perangkat Lunak (<i>Software</i>)	6
2.2 Instalasi dan Persiapan	8
2.2.1 Instalasi Awal Perangkat Keras (<i>Hardware</i>).....	8
2.2.2 Setup Raspberry Pi and Flutter	9
2.2.3 Instalasi dan Persiapan Arduino IDE.....	13
2.2.4 Instalasi Library yang Digunakan pada ESP32	13
Bagian 3 Pemrograman ESP 32	17
3.1 Memprogram ESP 32 untuk Membaca Sensor	17
3.1.1 Bagian 1: Inklusi <i>Library</i> dan Definisi Pin	17
3.1.2 Bagian 2 : Fungsi <i>setup()</i>	19
3.1.3 Bagian 3: Fungsi <i>loop()</i>	21
Bagian 4 Pemrograman Pertama pada Flutter	22
4.1 Struktur Direktori Proyek Flutter.....	22
4.2 Menjalankan Aplikasi Pertama.....	23
4.3 Widget dan Layout di Flutter	25
4.3.1 Apa itu Widget.....	25
4.3.2 Membuat Widget Sederhana	26
4.3.3 Mengatur Tata Letak (<i>Layout</i>)	29
Bagian 5 Pembuatan Desain Sederhana Meter Combi	31
5.1 Struktur Proyek Flutter	31
5.2 Desain UI Meter Combi	31
5.2.1 Pembuatan User Interface.....	31
Bagian 6 Integrasi Flutter dengan ESP 32 (Arduino).....	37
6.1 Integrasi Sistem	37

METER COMBI

6.1.1	Arsitektur Sistem	37
6.1.2	Tools & Packages Flutter yang Digunakan	37
6.1.3	Mengubah Data dari JSON Menjadi Objek Dart.....	37
6.1.4	Komunikasi Serial dari ESP ke Raspberry Pi.....	38
6.1.5	Kode ESP 32 Meter Combi	38
Bagian 7 Evaluasi & Refleksi		39
7.1	Evaluasi Pemahaman (Kuis Singkat).....	39
7.2	Refleksi Diri.....	39
7.3	Penutup Pelatihan.....	40
Bagian 8 LAMPIRAN		42

METER COMBI

Bagian 1 | Pendahuluan

1.1 Pengantar

Modul Pelatihan "Membangun Simulator Meter Combi Interaktif" ini dirancang untuk memberikan pemahaman komprehensif mengenai integrasi teknologi dalam sistem otomotif modern. Di tengah pesatnya perkembangan digitalisasi, kemampuan untuk memahami dan berinteraksi dengan perangkat keras dan lunak menjadi aset yang sangat berharga. Melalui simulasi meter combi, peserta akan diajak menjelajahi prinsip dasar di balik panel instrumen kendaraan, sebuah komponen vital yang menampilkan berbagai informasi penting bagi pengemudi. Melalui modul ini, peserta akan mendapatkan pemahaman mendasar tentang: Modul ini memiliki beberapa tujuan, yaitu :

- 1. Peningkatan Literasi Teknologi:** Memberikan fondasi pengetahuan yang kokoh mengenai konsep dasar teknologi dan literasi digital yang relevan dengan dinamika industri otomotif.
- 2. Pengenalan Pemrograman:** Memperkenalkan dasar-dasar pemrograman perangkat keras menggunakan ESP32 dan pengembangan antarmuka pengguna (UI) melalui Flutter, dirancang khusus bagi individu tanpa latar belakang teknis sebelumnya.
- 3. Pengalaman Praktis (Hands-on):** Menyediakan kesempatan untuk terlibat langsung dalam perakitan, pemrograman, dan kustomisasi sistem simulasi, sehingga peserta dapat mengobservasi secara langsung implementasi konsep teoretis.
- 4. Persiapan Pengembangan Keterampilan:** Modul ini berfungsi sebagai titik awal yang strategis dalam program *training development* dan peningkatan kompetensi (level-up skill) di TMMIN, mempersiapkan peserta untuk menghadapi tantangan teknologi yang berkembang di masa mendatang.

1.2 Apa Itu Meter Combi



Gambar 1. Meter Combi pada Mobil

METER COMBI

Meter Combi, atau *Instrument Cluster*, merupakan pusat informasi vital di dalam kendaraan, terletak strategis di hadapan pengemudi. Fungsinya esensial dalam menyajikan berbagai data penting mengenai kondisi dan performa kendaraan secara *real-time*. Komponen ini umumnya menampilkan indikator krusial seperti speedometer, takometer, indikator level bahan bakar, dan suhu mesin, serta berbagai lampu peringatan. Melalui tampilan yang komprehensif ini, pengemudi dapat memantau status operasional kendaraan dan mengambil keputusan berkendara yang tepat.

Dalam konteks kendaraan modern, termasuk yang mengadopsi fitur otonom parsial, meter combi telah berevolusi dari sekadar penunjuk analog menjadi sistem digital canggih, kerap berupa layar LCD atau LED yang dapat dikustomisasi. Teknologi ini memungkinkan data dari berbagai sensor di seluruh kendaraan dikumpulkan, diproses, dan kemudian ditampilkan secara dinamis pada meter combi. Memahami teknologi di balik sistem ini sangat penting karena memberikan wawasan tentang bagaimana data mentah dari lingkungan fisik diubah menjadi informasi yang bermakna bagi pengemudi, serta bagaimana sistem elektronik kendaraan saling berkomunikasi. Pengetahuan ini merupakan fondasi vital bagi banyak sistem cerdas yang terintegrasi di mobil masa kini.

Modul ini dirancang khusus untuk memberikan Anda pengalaman *hands-on* dalam memahami sistem tampilan meter combi secara langsung, melalui pembangunan versi simulasinya sendiri. Melalui proses ini, Anda akan secara signifikan meningkatkan pemahaman teknologi dan literasi digital Anda. Modul ini adalah langkah awal yang efektif bagi Anda yang baru memulai pembelajaran pemrograman dan integrasi sensor dari nol. Pada akhirnya, partisipasi dalam modul ini akan sangat mendukung Anda dalam program *training development* dan peningkatan keterampilan (*level-up skill*) di TMMIN, mempersiapkan Anda untuk beradaptasi dengan inovasi teknologi otomotif di masa mendatang.

METER COMBI

Bagian 2 | Komponen Sistem Meter Combi

2.1 Komponen Sistem

2.1.1 Perangkat Keras (*Hardware*)

Modul simulator ini memanfaatkan beberapa perangkat keras utama yang bekerja sama untuk mensimulasikan fungsi meter combi.

1. Raspberry Pi 5

Perangkat ini berperan sebagai "otak" utama sistem simulasi Anda. Raspberry Pi 5 adalah komputer mini serbaguna yang cukup kuat untuk menjalankan sistem operasi dan aplikasi tampilan *dashboard* yang akan kita buat dengan Flutter. Kehadirannya memungkinkan visualisasi data sensor yang interaktif di layar monitor.



Gambar 2. Raspberry Pi 5

2. ESP32 C6

Ini adalah mikrokontroler kecil yang sangat efisien, berfungsi sebagai "jembatan" antara sensor-sensor fisik dan Raspberry Pi. ESP32-C6 dirancang untuk membaca data dari berbagai sensor dan kemudian mengirimkannya ke Raspberry Pi untuk diproses dan ditampilkan.

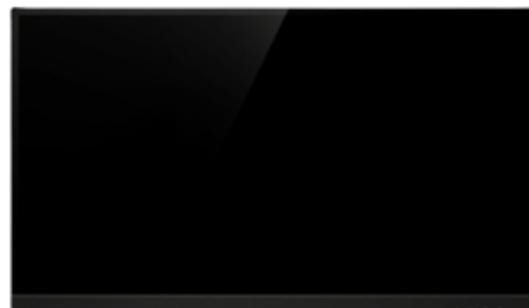


Gambar 3. ESP32-C6

METER COMBI

3. Layar Monitor

Monitor ini adalah antarmuka visual utama Anda. Di sinilah aplikasi *dashboard* Flutter yang kita bangun akan menampilkan semua data sensor secara *real-time*. Ini akan menirukan tampilan panel instrumen mobil sesungguhnya.



Gambar 4. Layar Monitor

4. Sensor

Ini adalah komponen yang mendeteksi perubahan fisik di lingkungan dan mengubahnya menjadi sinyal listrik yang dapat dibaca oleh ESP32.

- a.) **Suhu** : Modul sensor ini akan mengukur suhu, yang kemudian akan ditampilkan sebagai indikator suhu mesin pada meter combi simulasi Anda. Modul ini menggunakan sensor DS18B20.
- b.) **Speed (Kecepatan)**: Sensor ini, kemungkinan berupa potensiometer atau *rotary encoder*, akan Anda gunakan untuk mensimulasikan kecepatan kendaraan atau putaran mesin (RPM). Saat Anda memutar atau menggesernya, nilai kecepatan akan berubah. Modul ini menggunakan potensiometer untuk mensimulasikan perubahan kecepatan.
- c.) **Tegangan Baterai**: Modul sensor tegangan ini akan mengukur level tegangan listrik, yang kemudian dapat Anda gunakan untuk menampilkan indikator kondisi baterai kendaraan pada *dashboard*. Modul ini menggunakan potensiometer untuk mensimulasikan perubahan tegangan baterai.
- d.) **Level**: Sensor jarak, umumnya sensor ultrasonik, akan mengukur seberapa jauh objek berada di depannya. Dalam simulasi ini, kita akan

METER COMBI

menggunakannya untuk menirukan level bahan bakar di tangki. Modul ini menggunakan sensor ultrasonik untuk mensimulasikan perubahan level bensin.

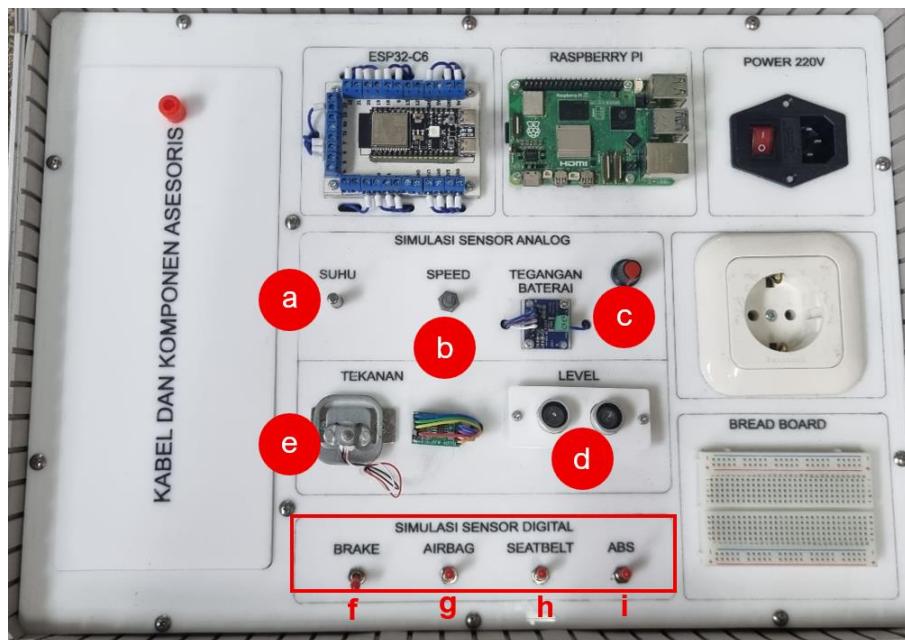
e.) **Tekanan:** Sensor ini dirancang untuk mendeteksi tekanan, seperti tekanan oli atau tekanan ban, dan akan memberikan data yang relevan untuk *dashboard* Anda.

f.) **Brake (Rem):** Ini adalah tombol atau sakelar yang saat ditekan akan mensimulasikan kondisi penggereman dan mengaktifkan lampu indikator rem pada *dashboard*.

g.) **Airbag:** Tombol atau sakelar ini akan digunakan untuk mensimulasikan kondisi sistem *airbag* dan mengaktifkan lampu indikator *airbag*.

h.) **Seatbelt:** Tombol atau sakelar ini akan menirukan status sabuk pengaman, menyala atau mati, pada *dashboard* Anda.

i.) **ABS:** Tombol ini akan mensimulasikan kondisi sistem penggereman anti-lock (ABS) dan mengaktifkan lampu indikator ABS.



Gambar 5. Sensor yang Digunakan

d e b c a

METER COMBI

5. Bread Board:

Papan kecil ini sangat berguna untuk merangkai sirkuit elektronik sementara. Ini memungkinkan Anda menghubungkan komponen-komponen sensor ke ESP32 tanpa perlu menyolder, sehingga proses eksperimen menjadi lebih mudah dan fleksibel.



Gambar 6. Mini Breadboard

6. Kabel dan Power Input 220V:

Berbagai kabel penghubung akan digunakan untuk menyatukan semua komponen, dan *power input* 220V adalah sumber daya listrik utama untuk mengaktifkan seluruh sistem simulasi Anda.

Untuk mengoperasikan perangkat keras di atas dan membuat visualisasi yang diinginkan, kita akan menggunakan beberapa perangkat lunak penting.



Gambar 7. Power Input 220V

2.1.2 Perangkat Lunak (Software)

Untuk mengoperasikan perangkat keras di atas dan membuat visualisasi yang diinginkan, kita akan menggunakan beberapa perangkat lunak penting.

1. Arduino IDE

Ini adalah lingkungan pengembangan terpadu (IDE) yang akan Anda gunakan untuk menulis, mengedit, dan mengunggah kode program ke ESP32. Arduino

METER COMBI

IDE dikenal karena kesederhanaannya, membuatnya ideal bagi pemula untuk memulai pemrograman mikrokontroler.



Gambar 8. Arduino IDE

2. Flutter & Dart SDK

Ini adalah seperangkat alat pengembangan perangkat lunak (SDK) yang memungkinkan Anda membangun aplikasi *user interface* (UI) yang menarik dan responsif. Flutter menggunakan bahasa pemrograman Dart, dan akan menjadi pondasi untuk menciptakan *dashboard* meter combi virtual Anda yang berjalan di Raspberry Pi.



Gambar 9. Flutter



Gambar 10. Dart

3. Sistem Operasi Raspberry Pi

Sama seperti komputer desktop Anda memiliki Windows atau macOS, Raspberry Pi juga memerlukan sistem operasi. Raspberry Pi OS adalah sistem operasi yang disarankan dan akan menyediakan lingkungan yang dibutuhkan untuk menjalankan aplikasi Flutter dan mengelola koneksi dengan ESP32.

4. Visual Studio Code (VS Code)

Ini adalah editor kode yang sangat populer dan serbaguna. Anda akan menggunakannya untuk menulis kode Dart/Flutter untuk aplikasi *dashboard* Anda, serta kode C++ untuk ESP32 jika diperlukan. VS Code menyediakan fitur-fitur yang memudahkan proses pemrograman.

METER COMBI

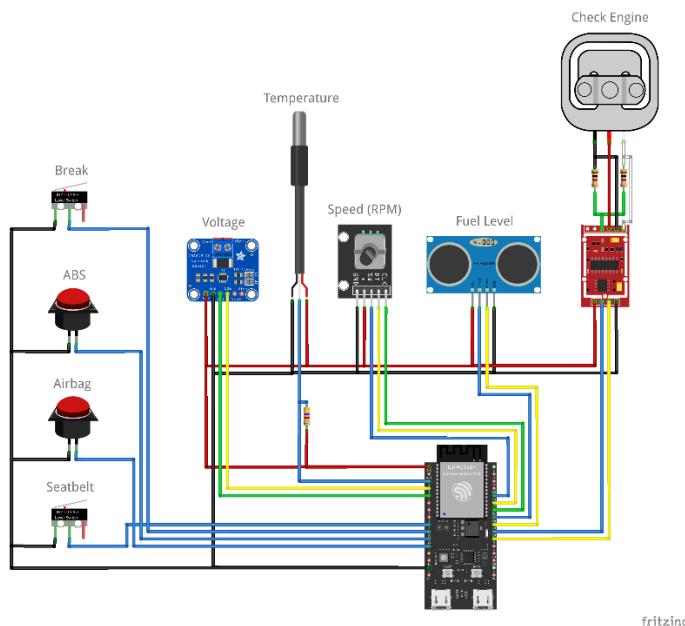


Visual Studio Code

Gambar 11. Visual Studio Code

2.2 Instalasi dan Persiapan

2.2.1 Instalasi Awal Perangkat Keras (*Hardware*)



Gambar 12. Wiring Diagram

Ikuti panduan berikut untuk mempersiapkan *hardware* pada *Training Kit* Anda:

- Menghubungkan Daya Utama (Power 220V):** Pertama, sambungkan kabel power yang tersedia ke sumber listrik eksternal (stop kontak dinding) yang menyediakan daya 220V. Kemudian, colokkan ujung kabel tersebut ke soket power 220V pada *kit* simulator Anda. Setelah terpasang, geser sakelar pada bagian 'Power 220V' ke posisi 'On' untuk mengalirkan daya.
- Menyediakan Daya untuk Raspberry Pi:** Untuk mengaktifkan Raspberry Pi, colokkan kabel USB yang terhubung dengan adaptor daya ke soket daya yang tersedia di *kit* simulator. Selanjutnya, sambungkan ujung kabel dengan konektor Type-C ke *port* daya pada Raspberry Pi. Langkah ini memastikan Raspberry Pi Anda menerima suplai daya yang optimal untuk beroperasi.

METER COMBI

3. **Menyediakan Daya untuk ESP32:** Kemudian, hubungkan kabel USB dari salah satu *port* USB pada Raspberry Pi ke *port type C* pada ESP32. Koneksi ini tidak hanya berfungsi sebagai saluran data, tetapi juga menyediakan daya listrik yang diperlukan agar ESP32 dapat menyala. Penting untuk diperhatikan bahwa *port type C* di sisi kanan ESP32 umumnya digunakan untuk mengunggah kode program, sedangkan *port* di sisi kiri dapat digunakan hanya untuk membaca data sensor.
4. **Menghubungkan Daya ke Layar Monitor:** Untuk menghidupkan layar monitor, colokkan kabel power monitor ke kepala *charger* yang sesuai, lalu sambungkan kepala *charger* tersebut ke stop kontak eksternal. Sangat disarankan untuk tidak mencolokkan daya monitor langsung ke Raspberry Pi, karena daya yang disediakan oleh Raspberry Pi mungkin tidak mencukupi, yang dapat menyebabkan monitor tidak menyala atau berfungsi tidak optimal.
5. **Menghubungkan Tampilan Monitor ke Raspberry Pi:** Gunakan kabel HDMI monitor dan colokkan salah satu ujungnya ke *port* HDMI pada monitor. Untuk koneksi ke Raspberry Pi, Anda memerlukan **dongle converter HDMI ke Micro-HDMI**, kemudian colokkan ujung Micro-HDMI ke *port* Micro-HDMI pada Raspberry Pi. Pastikan semua koneksi kokoh untuk memastikan tampilan dari Raspberry Pi dapat muncul di monitor.
6. **Verifikasi Awal Setelah Instalasi Hardware:** Setelah semua kabel terhubung dengan benar, amati indikator pada perangkat Anda. Raspberry Pi akan menyala dengan cahaya **berwarna kuning** jika sudah mendapatkan daya dan mulai beroperasi. Demikian pula, ESP32 akan menunjukkan indikasi daya dengan lampu **berwarna merah**. Setelah itu, layar monitor seharusnya akan **otomatis menyala** dan menampilkan *homepage* atau antarmuka awal dari sistem operasi Raspberry Pi. Jika semua indikator ini terlihat, berarti instalasi *hardware* dasar Anda telah berhasil dilakukan!

2.2.2 Setup Raspberry Pi and Flutter

Bagian ini akan memandu Anda dalam menyiapkan lingkungan perangkat lunak yang diperlukan pada Raspberry Pi untuk pengembangan aplikasi *dashboard* Anda.

1. Persiapkan Perangkat dan Alat

Peralatan	Keterangan
Raspberry Pi 5	Unit Utama
Micro SD Card (32 GB atau lebih, class 10)	Media penyimpanan sistem

METER COMBI

Komputer Windows/macOS/linux	Untuk download & instal OS ke SD card
SD card reader	Untuk menulis OS ke microSD card
Power Adapter USB-C 5V 5A	Catu daya resmi untuk Pi 5
Kabel HDMI + Monitor	Tampilan layar Raspberry Pi
Keyboard & Mouse (opsional)	Untuk konfigurasi manual
Koneksi Internet	Dibutuhkan saat setup pertama kali

2. Unduh & Install Sistem Operasi (Raspberry Pi OS):

a) Download Raspberry Pi Imager

- Kunjungi: <https://www.raspberrypi.com/software/>
- Unduh & install Raspberry Pi Imager sesuai OS komputermu.

b) Pasang microSD Card ke komputer melalui card reader.

c) Jalankan Raspberry Pi Imager, lalu pilih:

- Choose OS → Raspberry Pi OS (64-bit) → *Recommended*.
- Choose Storage → Pilih drive microSD kamu.
- (Opsional) Klik ikon pengaturan (Advanced Options) untuk:
 - Setting Wi-Fi
 - Enable SSH
 - Set hostname
 - Auto-login

d) Klik WRITE, lalu tunggu proses flashing selesai (5–10 menit).

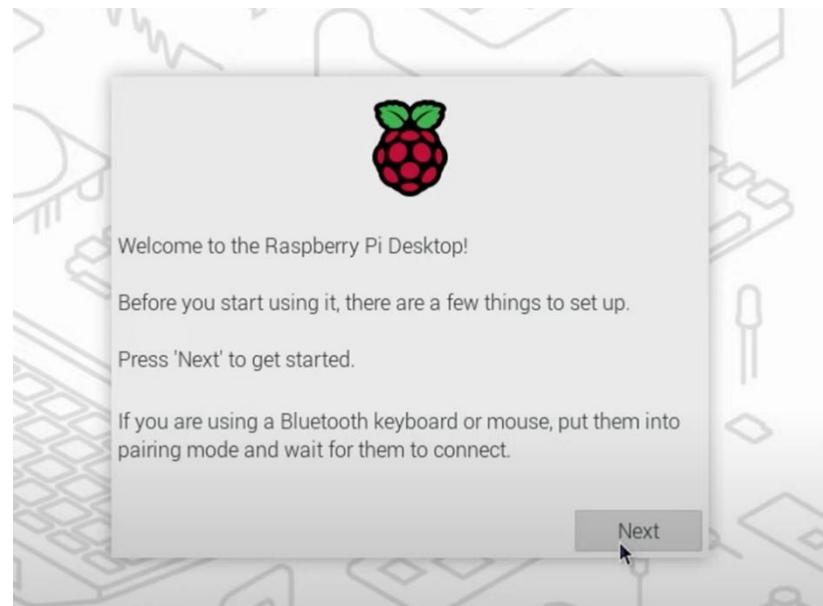
3. Hubungkan dan Nyalakan Raspberry Pi 5

- Masukkan microSD Card ke slot Raspberry Pi 5.
- Hubungkan monitor ke HDMI 0 (port paling dekat ke power).
- (Opsional) Hubungkan keyboard & mouse ke port USB.
- Colokkan power adapter USB-C untuk menyalaikan Raspberry Pi

LED akan menyala dan layar akan menampilkan proses boot pertama.

4. Setup Awal Raspberry Pi OS

METER COMBI

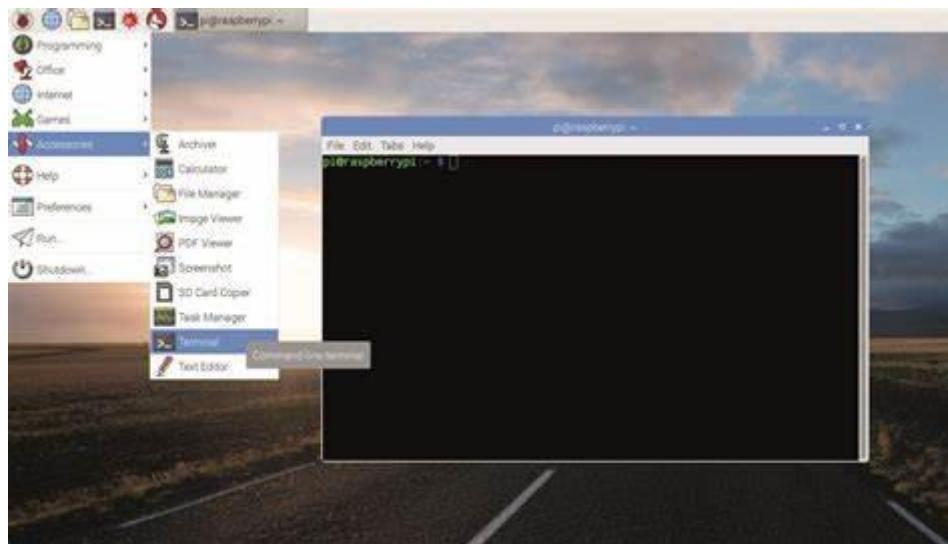


Gambar 13. Tampilan Awal Instalasi Rapberry Pi

Saat boot pertama kali, kamu akan disambut oleh Setup Wizard:

- Pilih bahasa, zona waktu, dan lokasi.
- Buat username & password.
- Sambungkan ke jaringan Wi-Fi jika belum disetting sebelumnya.
- Lakukan update sistem jika tersedia.
- Setelah selesai, Raspberry Pi akan restart dan menampilkan desktop.

5. Install Flutter dan Dependensi



Gambar 14. Tampilan Raspberry Pi Awal

METER COMBI

- a) Buka Terminal → Jalankan perintah berikut:

```
sudo apt update  
sudo apt install git unzip xz-utils curl -y
```

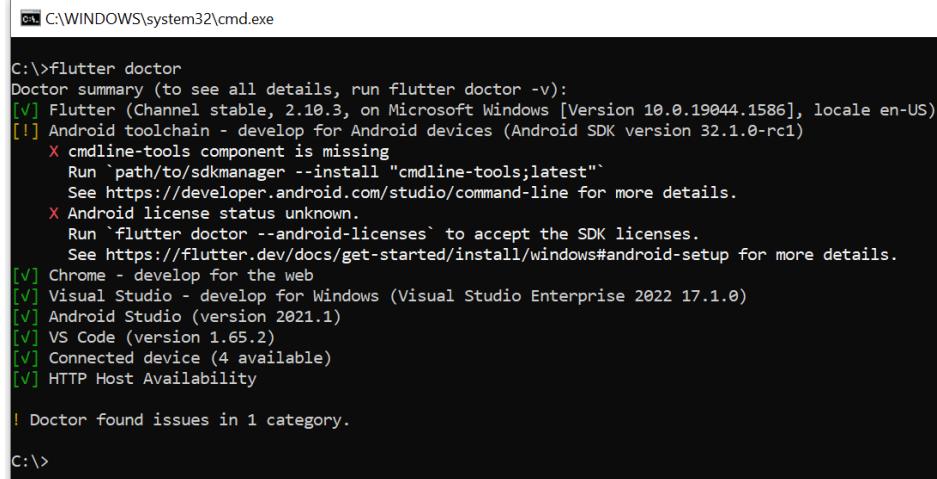
- b) Unduh Flutter SDK ARM64 (Linux):

```
cd ~  
  
curl -O  
https://storage.googleapis.com/flutter_infra_release/releases/stable/linux/flutter_linux_3.19.6-  
stable.tar.xz  
  
tar xf flutter_linux_3.19.6-stable.tar.xz
```

- c) Tambahkan Flutter ke PATH:

```
echo 'export PATH="$PATH:$HOME/flutter/bin"' >> ~/.bashrc  
  
source ~/.bashrc  
  
flutter doctor
```

- d) Install dependensi tambahan jika diminta oleh flutter doctor.



```
C:\>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[!] Android toolchain - develop for Android devices (Android SDK version 32.1.0-rc1)
    X cmdline-tools component is missing
      Run `path/to/sdkmanager --install "cmdline-tools;latest"`
      See https://developer.android.com/studio/command-line for more details.
    X Android license status unknown.
      Run `flutter doctor --android-licenses` to accept the SDK licenses.
      See https://flutter.dev/docs/get-started/install/windows#android-setup for more details.
[!] Chrome - develop for the web
[!] Visual Studio - develop for Windows (Visual Studio Enterprise 2022 17.1.0)
[!] Android Studio (version 2021.1)
[!] VS Code (version 1.65.2)
[!] Connected device (4 available)
[!] HTTP Host Availability

! Doctor found issues in 1 category.

C:\>
```

Gambar 15. Tampilan Flutter Doctor

METER COMBI

2.2.3 Instalasi dan Persiapan Arduino IDE

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

DOWNLOAD OPTIONS

Windows Win 10 and newer, 64 bits

Windows MSI installer

Windows ZIP file

Linux AppImage 64 bits (X86-64)

Linux ZIP file 64 bits (X86-64)

macOS Intel, 10.15: "Catalina" or newer, 64 bits

macOS Apple Silicon, 11: "Big Sur" or newer, 64 bits

[Release Notes](#)

Gambar 16. Instalasi Arduino IDE

1. Arduino IDE sudah tersedia di repository Raspberry Pi OS, jadi kamu bisa install langsung:

```
sudo apt install arduino -y
```

2. (Opsional) Install Versi Terbaru Arduino IDE (2.x) via File AppImage. Berikan izin executable:

```
chmod +x Arduino_IDE_x.y.z_Linux_ARM64.AppImage
```

```
./Arduino_IDE_x.y.z_Linux_ARM64.AppImage
```

3. Cek dan Hubungkan Board Arduino / ESP32

a) Colokkan Arduino/ESP32 ke USB Raspberry Pi

b) Jalankan perintah ini untuk cek koneksi

```
ls /dev/ttyUSB*
```

```
ls /dev/ttyACM*
```

2.2.4 Instalasi Library yang Digunakan pada ESP32

Untuk memungkinkan ESP32 berkomunikasi dengan sensor, Anda perlu menginstal *library* yang tepat di Arduino IDE. *Library* adalah kumpulan kode yang memudahkan Anda untuk menggunakan fitur-fitur tertentu dari perangkat keras tanpa harus menulis semua kode dari awal.

METER COMBI

1. Menambahkan Board ESP32 ke Arduino IDE:

Langkah pertama adalah membuat Arduino IDE "mengenali" ESP32 sebagai *board* yang dapat diprogram.

- a) **Buka Aplikasi Arduino IDE:** Luncurkan aplikasi Arduino IDE di komputer Anda.
- b) **Akses Preferences:** Klik menu **File** di bagian atas, lalu pilih **Preferences**.
- c) **Tambahkan URL Board Manager:** Di jendela Preferences, cari kolom berlabel "Additional boards manager URLs". Salin dan tempel URL berikut ke dalam kolom tersebut:
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json. Jika sudah ada URL lain, Anda bisa menambahkannya dengan tanda koma (,) sebagai pemisah.
- d) **Klik OK:** Setelah menempelkan URL, klik tombol **OK** untuk menyimpan perubahan.
- e) **Buka Board Manager:** Sekarang, klik ikon **Board Manager** di bilah sisi kiri Arduino IDE (terlihat seperti *board* kecil). Ini akan membuka jendela Board Manager.
- f) **Cari dan Instal ESP32:** Di kolom pencarian Board Manager, ketik "**ESP32**". Anda akan melihat opsi "esp32 by Espressif". Klik tombol **Install** di sampingnya.
- g) **Tunggu Proses Instalasi:** Proses ini akan memakan waktu beberapa menit dan memerlukan koneksi internet karena Arduino IDE akan mengunduh data yang diperlukan. Tunggu hingga proses instalasi selesai.
- h) **Pilih Board dan Port:** Setelah instalasi *board* selesai, hubungkan ESP32 Anda ke komputer menggunakan kabel USB. Kemudian, di Arduino IDE, klik pada bagian pemilihan *board* dan *port* (biasanya di bagian bawah jendela atau di menu "Tools" -> "Board" dan "Tools" -> "Port"). Pilih "**ESP32C6 Dev Module**" dan *port* USB yang sesuai dengan ESP32 Anda (misalnya, yang memiliki label "Serial Port (USB)").

2. Langkah-langkah Menginstal Library Sensor:

Setelah ESP32 dikenali, Anda perlu menginstal *library* untuk masing-masing sensor yang akan Anda gunakan. Ini akan sangat menyederhanakan kode pemrograman Anda.

METER COMBI

Metode 1: Menggunakan Library Manager (Direkomendasikan untuk Sebagian Besar Library)

1. **Buka Library Manager:** Di Arduino IDE, klik ikon **Library Manager** di bilah sisi kiri (terlihat seperti rak buku). Ini akan membuka jendela Library Manager.
2. **Cari dan Instal Library:**
 - Di kolom pencarian Library Manager, ketik nama *library* sensor yang Anda perlukan. Misalnya, untuk sensor suhu: ketik "**Dallas Temperature**".
 - Cari *library* yang relevan (misalnya "Dallas Temperature by Miles Burton" atau "Dallas Temperature by Adafruit") dan klik tombol **Install** di sampingnya.
 - **Perhatikan Dependensi:** Beberapa *library* mungkin memiliki "dependensi" (ketergantungan) pada *library* lain. Jika muncul pop-up yang menanyakan apakah Anda ingin menginstal *dependensi* tersebut, pilih "**Install All**" (atau "Install with dependencies"). Ini penting agar *library* utama dapat berfungsi dengan baik.
 - Ulangi langkah ini untuk setiap *library* sensor yang Anda perlukan yang tersedia di Library Manager:
 - AiEsp32RotaryEncoder (untuk *rotary encoder*).
 - DFRobot_HX711 (untuk sensor berat/tekanan).
 - Adafruit_INA219 (untuk sensor tegangan).
 - DHT kxn by Adafruit (untuk sensor suhu/kelembaban DHT11 jika digunakan).
3. **Tunggu hingga Proses Instalasi Selesai:** Pastikan Anda menunggu setiap *library* selesai diinstal sebelum melanjutkan ke yang berikutnya.

Metode 2: Menginstal Library dari File ZIP (untuk Library Eksternal/Tidak Ada di Manager):

Beberapa *library* mungkin tidak tersedia di Library Manager atau Anda ingin menggunakan versi tertentu yang diunduh langsung (misalnya dari GitHub). Untuk kasus ini, Anda dapat menginstalnya secara manual:

1. **Unduh Library dalam Format ZIP:** Kunjungi sumber *library* eksternal (misalnya halaman GitHub proyek *library* tersebut). Cari tombol "Code" atau "Download" dan pilih opsi "Download ZIP". Simpan file ZIP tersebut

METER COMBI

ke komputer Anda. Contoh spesifik yang perlu diinstal dengan cara ini adalah *library OneWire*.

2. **Jangan Ekstrak File ZIP:** Biarkan file tersebut dalam format .zip.
3. **Buka Arduino IDE:** Pastikan Arduino IDE Anda terbuka.
4. **Pilih "Add .ZIP Library...":** Klik menu **Sketch** di bagian atas, arahkan kursor ke **Include Library**, lalu pilih **Add .ZIP Library...**
5. **Pilih File ZIP:** Sebuah jendela akan muncul. Navigasikan ke lokasi tempat Anda menyimpan file ZIP *library* yang telah diunduh, pilih file tersebut, dan klik "Open".
6. **Verifikasi Instalasi:** Arduino IDE akan memberikan pesan di bagian bawah jendela bahwa *library* telah berhasil ditambahkan. Anda juga dapat memeriksa apakah *library* sudah muncul di daftar "Contributed Libraries" di menu Sketch > Include Library.

Bagian 3 | Pemrograman ESP 32

Pada bagian ini dijelaskan tentang kode pada ESP 32 dan Raspberry Pi yang digunakan pada simulator meter combi.

3.1 Memprogram ESP 32 untuk Membaca Sensor

Kode ESP32 ini dirancang untuk membaca data dari berbagai sensor yang terpasang pada *kit* simulator Anda. Setelah membaca dan mengolah data, ESP32 akan mengirimkannya ke Raspberry Pi melalui komunikasi serial. Data yang dikirim ini nantinya akan ditampilkan pada aplikasi *dashboard* Flutter di layar monitor, mensimulasikan panel instrumen kendaraan. Potensi error dan konflik antar-versi pustaka bisa dihindari. Kode lengkap simulator ini bisa anda lihat pada bagian lampiran.

Berikut adalah penjelasan detail dari setiap bagian kode :

3.1.1 Bagian 1: Inklusi *Library* dan Definisi Pin

1. Penyertaan *Library* (`#include <LibraryName.h>` atau `#include "LibraryName.h"`):

Baris-baris ini merupakan instruksi penting bagi kompilator untuk menyertakan (*include*) kode dari *library* eksternal yang telah Anda instal di Arduino IDE. *Library* ini berisi fungsi-fungsi dan definisi siap pakai yang sangat memudahkan interaksi dengan berbagai komponen perangkat keras tanpa perlu menulis semua kode dasar dari awal.

- `<OneWire.h>`: Diperlukan untuk komunikasi dengan sensor suhu DS18B20.
- `<DallasTemperature.h>`: *Library* khusus untuk sensor suhu DS18B20, yang bergantung pada *library* OneWire.
- `"AiEsp32RotaryEncoder.h"`: Digunakan untuk mengelola pembacaan putaran dari *rotary encoder*, yang berfungsi untuk simulasi kecepatan atau RPM.
- `<Wire.h>`: *Library* ini esensial untuk komunikasi I2C (Integrated Circuit), sebuah protokol yang digunakan oleh sensor INA219.
- `<Adafruit_INA219.h>`: *Library* spesifik untuk sensor tegangan INA219.
- `<DFRobot_HX711.h>`: Digunakan untuk mengoperasikan sensor HX711 (sensor berat/tekanan).

METER COMBI

2. Definisi Pin (#define PIN_NAME PIN_NUMBER):

Perintah #define digunakan untuk memberikan nama yang mudah diingat (seperti TRIG_PIN) pada nomor pin fisik tertentu di ESP32. Pendekatan ini meningkatkan keterbacaan kode secara signifikan, karena Anda dapat merujuk pin dengan namanya daripada nomor yang kurang intuitif.

- *ONE_WIRE_BUS* 4: Menentukan pin GPIO4 pada ESP32 sebagai jalur data untuk sensor suhu DS18B20.
- *TRIG_PIN* 20 dan *ECHO_PIN* 19: Mengatur pin GPIO20 (Transmitter) dan GPIO19 (Receiver) untuk sensor jarak Ultrasonik (HC-SR04), yang digunakan untuk simulasi level bahan bakar.
- *ROTARY_ENCODER_A_PIN* 22, *ROTARY_ENCODER_B_PIN* 21, *ROTARY_ENCODER_BUTTON_PIN* 23: Mendefinisikan pin-pin ESP32 yang terhubung ke *rotary encoder*.
- *BREAK_PIN* 10, *ABS_PIN* 11, *AIRBAG_PIN* 2, *SEATBELT_PIN* 8: Menentukan pin-pin untuk tombol indikator digital pada *kit simulator*, seperti indikator rem, ABS, *airbag*, dan sabuk pengaman.

3. Pembuatan Objek untuk Sensor:

Setelah *library* disertakan, kita membuat "objek" (atau instansi) dari *library* tersebut untuk setiap sensor yang akan digunakan. Setiap objek ini adalah variabel yang memiliki semua fungsi dan data yang diperlukan untuk berinteraksi dengan sensor fisik yang bersangkutan.

- *OneWire oneWire(ONE_WIRE_BUS);* : Membuat objek oneWire untuk mengelola komunikasi melalui pin yang ditentukan untuk OneWire.
- *DallasTemperature tempSensor(&oneWire);* : Membuat objek tempSensor yang khusus untuk sensor suhu, dan mengaitkannya dengan objek oneWire yang sudah dibuat.
- *AiEsp32RotaryEncoder rotaryEncoder = ...;* : Membuat objek rotaryEncoder untuk mengelola *rotary encoder* menggunakan pin-pin yang telah didefinisikan sebelumnya.
- *Adafruit_INA219 ina219;* : Membuat objek ina219 untuk berinteraksi dengan sensor tegangan INA219.
- *DFRobot_HX711 MyScale(9, 18);* : Membuat objek MyScale untuk sensor berat/tekanan HX711, dengan menentukan pin data (9) dan pin clock (18) yang terhubung ke sensor tersebut.

4. Fungsi *Interrupt Service Routine (ISR)* *readEncoderISR()*:

METER COMBI

`void IRAM_ATTR readEncoderISR()` adalah fungsi khusus yang dikenal sebagai *Interrupt Service Routine* (ISR). Fungsi ini akan dieksekusi secara otomatis dan dengan kecepatan tinggi setiap kali *rotary encoder* mendeteksi adanya putaran atau perubahan posisi. Penanda IRAM_ATTR memberikan instruksi kepada ESP32 untuk menempatkan fungsi ini di memori khusus yang dapat diakses dengan sangat cepat, memastikan operasi tanpa gangguan dan respons yang instan terhadap input *encoder*.

3.1.2 Bagian 2 : Fungsi `setup()`

1. *Fungsi void setup(void):*

Bagian ini adalah *initialization block* dari program Anda. Semua pengaturan awal dan inisialisasi perangkat keras serta *library* dilakukan di sini. Kode yang berada di dalam fungsi `setup()` akan dieksekusi **hanya satu kali** saat ESP32 pertama kali dinyalakan atau di-reset. Ini memastikan semua komponen siap sebelum program utama mulai berjalan berulang kali.

2. *Serial.begin(115200); :*

Baris ini memulai komunikasi serial antara ESP32 dan perangkat yang terhubung (dalam kasus ini, Raspberry Pi) pada kecepatan 115200 *baud* (bit per detik). Kecepatan ini harus diatur sama pada kedua perangkat agar komunikasi data dapat terjadi dengan lancar dan benar. Ini merupakan saluran utama untuk pengiriman data sensor.

3. *tempSensor.begin(); :*

Perintah ini menginisialisasi *library* DallasTemperature, memungkinkan ESP32 untuk mulai berkomunikasi dengan sensor suhu DS18B20 yang terhubung.

4. *pinMode(PIN, MODE); :*

Fungsi `pinMode()` digunakan untuk mengkonfigurasi setiap pin digital pada ESP32, menentukan apakah pin tersebut akan berfungsi sebagai input atau output.

- **OUTPUT:** Pin akan digunakan untuk mengirim sinyal listrik, seperti yang dilakukan oleh TRIG_PIN pada sensor ultrasonik untuk memancarkan gelombang suara.
- **INPUT:** Pin akan digunakan untuk menerima sinyal listrik, seperti ECHO_PIN dari sensor ultrasonik yang mendeteksi pantulan

METER COMBI

suara, atau pin-pin yang terhubung ke tombol-tombol indikator digital.

- INPUT_PULLUP: Ini adalah mode INPUT khusus yang mengaktifkan resistor *pull-up* internal di dalam ESP32. Penggunaan ini sangat ideal untuk tombol-tombol, karena memastikan pin memiliki nilai 'HIGH' (tegangan tinggi) secara *default* saat tombol tidak ditekan. Saat tombol ditekan, pin akan beralih ke 'LOW' (tegangan rendah), yang kemudian dapat dideteksi oleh program.

5. **rotaryEncoder.begin();** dan

rotaryEncoder.setup(readEncoderISR); :

Perintah ini menginisialisasi objek *rotary encoder* dan menghubungkannya dengan fungsi *readEncoderISR()* yang telah kita definisikan sebelumnya sebagai *Interrupt Service Routine*. Dengan demikian, ESP32 dapat secara efisien dan akurat mendeteksi setiap putaran atau "langkah" dari *encoder*. Fungsi *setBoundaries* dan *disableAcceleration* selanjutnya mengatur batasan nilai putaran (misalnya 0 hingga 100) dan menonaktifkan fitur akselerasi untuk memastikan pembacaan putaran yang lebih stabil dan presisi.

6. **Wire.begin(5, 6); :**

Baris ini menginisialisasi komunikasi I2C (Inter-Integrated Circuit) pada pin GPIO5 sebagai SDA (*Serial Data Line*) dan GPIO6 sebagai SCL (*Serial Clock Line*). Komunikasi I2C adalah protokol standar yang digunakan oleh banyak sensor, termasuk sensor tegangan INA219, untuk bertukar data dengan mikrokontroler.

7. **if (!ina219.begin()) { ... } :**

Blok kode ini mencoba untuk menginisialisasi sensor tegangan INA219. Jika proses inisialisasi gagal (misalnya, karena sensor tidak terhubung dengan benar atau ada masalah pada *chip*), program akan mencetak pesan error "Failed to find INA219 chip" ke Serial Monitor. Setelah itu, program akan masuk ke dalam *loop* tanpa henti (*while (1) { delay(10); }*), secara efektif menghentikan eksekusi kode selanjutnya hingga masalah diperbaiki atau ESP32 di-reset, memberikan indikasi yang jelas adanya masalah.

8. **ina219.setCalibration_16V_400mA(); :**

Perintah ini mengatur kalibrasi untuk sensor tegangan INA219. Dengan konfigurasi ini, sensor mampu mengukur tegangan hingga

METER COMBI

16 Volt dan arus hingga 400 miliAmpere, yang sesuai untuk kebutuhan simulasi pembacaan tegangan baterai.

3.1.3 Bagian 3: Fungsi loop()

1. *Fungsi void loop(void):*

Ini adalah jantung dari program Anda. Kode yang berada di dalam fungsi loop() akan dieksekusi secara **berulang-ulang** dan terus-menerus setelah fungsi setup() selesai dijalankan. Di sinilah ESP32 secara aktif akan membaca data dari sensor, memprosesnya, dan mengirimkannya.

2. Pengiriman Data Melalui Serial (ke Raspberry Pi)

Bagian ini bertanggung jawab untuk mengirimkan data sensor yang telah diolah dari ESP32 ke Raspberry Pi. Anda bisa menyesuaikan nama variabel yang akan dikirimkan melalui serial.



```
Serial.printf("Speed:%.0f\n", rpm);
Serial.printf("Temperature:%.1f\n", tempC);
Serial.printf("Level:%.0f\n", distance_cm);
Serial.printf("Pressure:%.1f\n", weight);
Serial.printf("Voltage:%.2f\n", loadvoltage);
Serial.printf("Brake:%d\n", breakVal);
Serial.printf("ABS:%d\n", absVal);
Serial.printf("Airbag:%d\n", airbagVal);
Serial.printf("Seatbelt:%d\n", seatbeltVal);
```

Gambar 17. Kode untuk Mengirimkan Data Secara Serial

Bagian 4 | Pemrograman Pertama pada Flutter

4.1 Struktur Direktori Proyek Flutter

Struktur direktori proyek Flutter biasanya terdiri dari beberapa folder dan file yang berfungsi untuk mengorganisir kode, sumber daya, dan konfigurasi proyek. Berikut adalah penjelasan singkat tentang struktur direktori proyek Flutter yang umum:

1. **android**: Folder ini berisi proyek Android native untuk proyek Flutter Anda. Ini termasuk file-file seperti proyek Gradle, sumber kode Java/ Kotlin, dan konfigurasi Android lainnya.
2. **ios**: Mirip dengan folder “android”, folder ini berisi proyek iOS native untuk proyek Flutter Anda. Ini termasuk file-file seperti proyek Xcode, sumber kode Swift/Objective-C, dan konfigurasi iOS lainnya.
3. **lib**: Folder ini adalah tempat utama untuk kode Dart aplikasi Anda. Ini adalah tempat dimana Anda akan menulis kode Flutter Anda, termasuk file-file seperti widget, utilitas, layanan, dan logika bisnis aplikasi.
4. **test**: Folder ini berisi unit test dan widget test untuk proyek Anda. Ini memungkinkan Anda untuk menulis dan menjalankan tes otomatis untuk memastikan bahwa aplikasi Anda berperilaku sesuai yang diharapkan.
5. **assets**: Jika Anda memiliki file sumber daya statis seperti gambar, font, atau file konfigurasi yang digunakan dalam aplikasi Anda, Anda dapat meletakkannya di sini. Mereka akan diakses oleh aplikasi Anda melalui Flutter menggunakan path relatif.
6. **pubspec.yaml**: File ini adalah file konfigurasi proyek Flutter Anda. Di sini Anda mendefinisikan paket-paket Dart yang digunakan dalam proyek Anda, sumber daya aplikasi seperti font atau gambar, serta konfigurasi proyek lainnya.
7. **.metadata**: File ini adalah file metadata yang digunakan oleh Flutter untuk melacak dependensi proyek Anda.

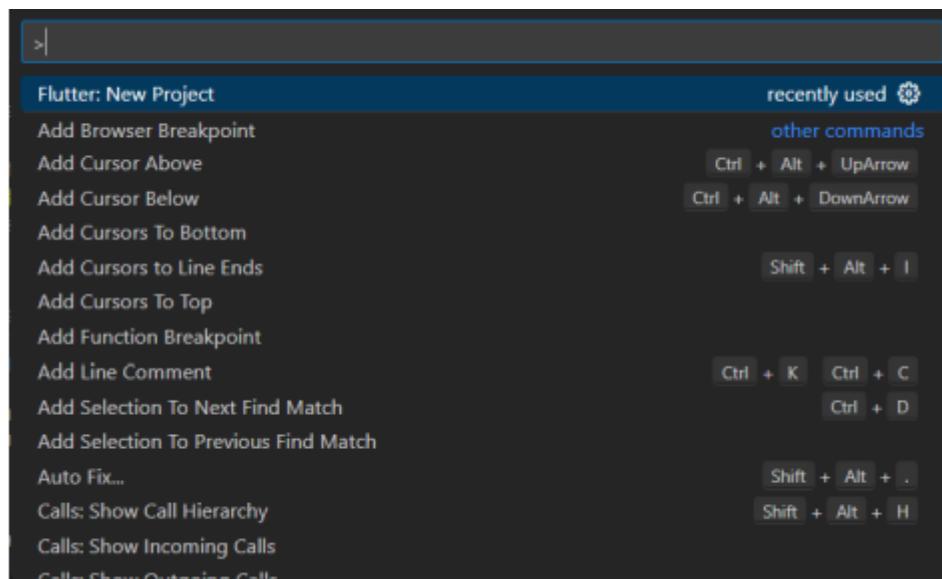
Selain itu, Anda juga mungkin akan menemukan file-file konfigurasi lainnya yang tergantung pada alat pengembangan atau editor yang Anda gunakan, seperti `.gitignore` untuk mengabaikan file saat Anda menggunakan Git, atau file konfigurasi spesifik editor seperti `.vscode` untuk Visual Studio Code.

METER COMBI

Struktur ini membantu dalam menjaga kode proyek Anda terorganisir dan mudah dikelola, serta memisahkan kode Flutter Anda dari kode platform spesifik seperti Android dan iOS.

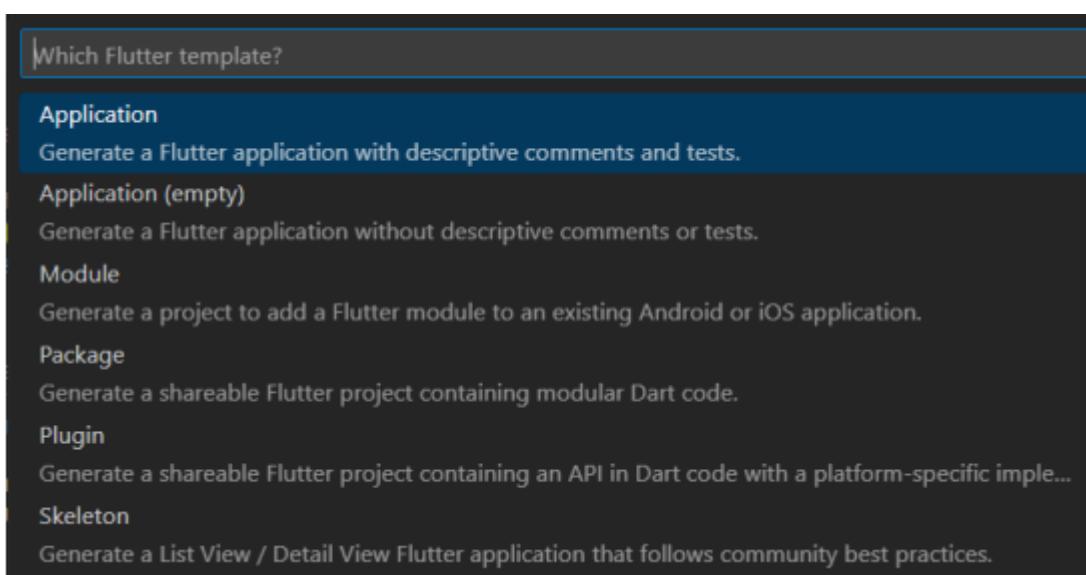
4.2 Menjalankan Aplikasi Pertama

Buat project baru menggunakan Command Pallette. Pilih menu View dan klik Command Pallette sehingga muncul tampilan berikut:



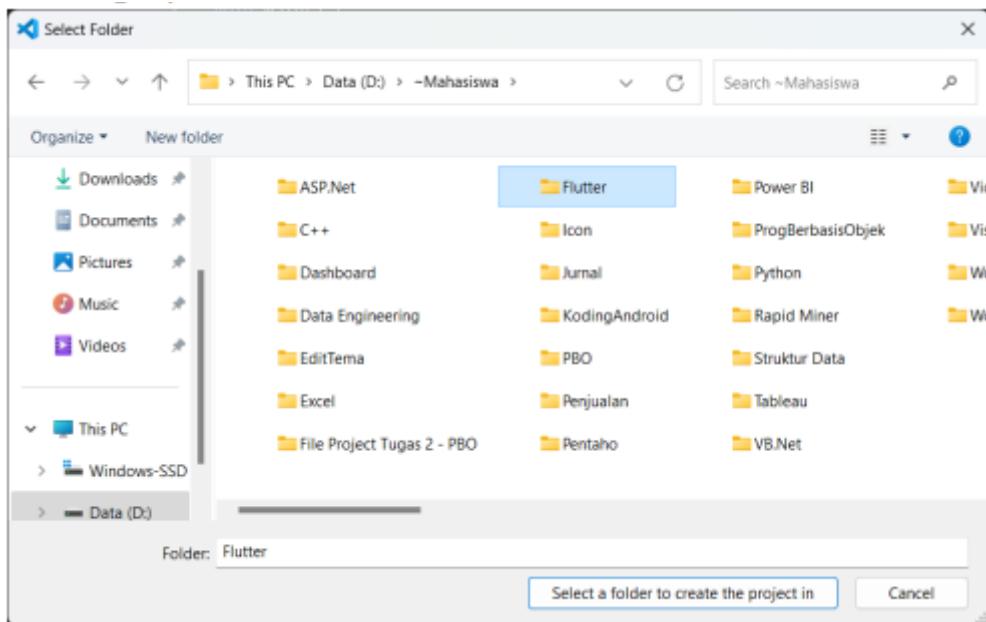
Gambar 18. Tampilan VSCode Ketika Membuat Project Flutter

Kemudian ketikkan flutter, pilih Flutter New Project, maka akan muncul tampilan berikut



Gambar 19. Tampilan Ketika Akan Membuat New Project

METER COMBI



Gambar 20. Tampilan File Explorer

Pilih menu Application. Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**

Beri nama projectnya, dengan mengetik aplikasi_pertama kemudian tekan Enter. Saat membuat project baru, kode program awal yang akan didapatkan pada **main.dart** akan terbagi menjadi tiga bagian :

```
1 import 'package:flutter/material.dart';
2 import 'package:flutter/services.dart';
3
4 void main() {
5   runApp(MyApp());
6 }
7
8 class MyApp extends StatelessWidget {
9   const MyApp({Key key}) : super(key: key);
10
11   // This widget is the root of your application.
12   @override
13   Widget build(BuildContext context) {
14     return MaterialApp(
15       title: 'Flutter Demo',
16       theme: ThemeData(
17         primarySwatch: Colors.blue,
18       ),
19       home: MyHomePage(title: 'Flutter Demo Home Page'),
20     );
21   }
22 }
23
24 class MyHomePage extends StatefulWidget {
25   const MyHomePage({Key key, this.title}) : super(key: key);
26
27   final String title;
28
29   @override
30   _MyHomePageState createState() => _MyHomePageState();
31 }
32
33 class _MyHomePageState extends State<MyHomePage> {
34 }
```

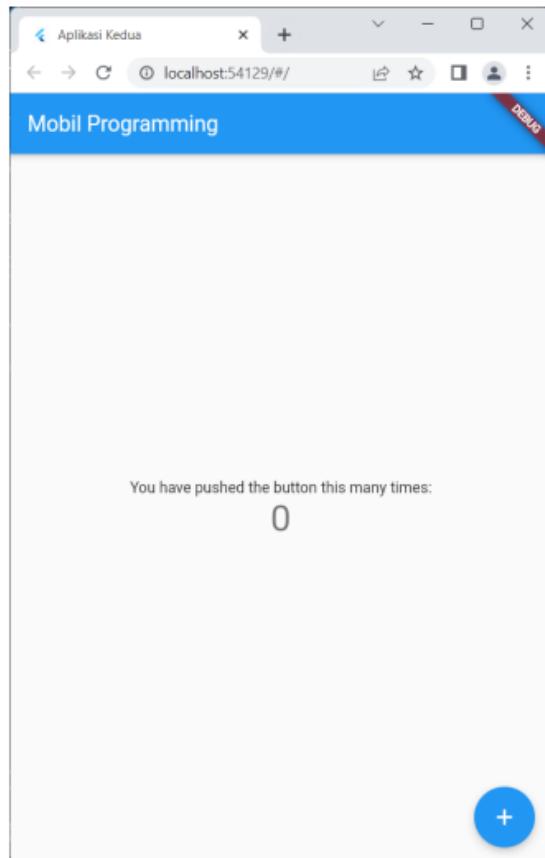
Gambar 21. Tampilan VSCode Project Flutter

1. Bagian **import** : bagian import adalah tempat kita mendeklarasikan atau mengimpor library yang dibutuhkan pada aplikasi.

METER COMBI

2. Bagian **main** : bagian main adalah fungsi utama dari aplikasi yang akan menjadi entri point. Fungsi ini akan dieksekusi pertama kali saat aplikasi dibuka.
3. Bagian **widget** : Bagian widget adalah tempat kita membuat widget. Aplikasi Flutter sebenarnya terdiri dari susunan widget. Widget bisa kita bilang elemen-elemen seperti Tombol, Teks, Layaout, Image, dan sebagainya.

Untuk menjalankannya yaitu menggunakan New Terminal dan ketikkan **flutter run**. Tampilan yang akan muncul :



Gambar 22. Tampilan UI Program Awal Flutter

4.3 Widget dan Layout di Flutter

4.3.1 Apa itu Widget

Dalam konteks Flutter, “widget” merujuk pada konsep dasar untuk membangun antarmuka pengguna (UI). Widget adalah bagian-bagian kecil dari antarmuka pengguna yang dapat berupa elemen visual seperti tombol, teks, gambar, atau bahkan layout yang lebih kompleks seperti daftar atau kartu.

METER COMBI

Di Flutter, hampir semua yang Anda lihat di layar adalah widget. Bahkan, aplikasi Flutter sendiri adalah widget. Widget dalam Flutter dapat dibagi menjadi dua jenis utama:

StatelessWidget: Widget yang tidak memiliki keadaan (state) internal. Artinya, mereka tidak dapat berubah setelah dibuat. Stateless widget bergantung pada input dan konfigurasi yang diberikan untuk menampilkan antarmuka pengguna. Contohnya adalah widget Text yang menampilkan teks statis atau widget Icon yang menampilkan ikon tertentu.

StatefulWidget: Widget yang memiliki keadaan (state) internal yang dapat berubah sepanjang waktu.

Stateful widget dapat merespons perubahan input, interaksi pengguna, atau perubahan kondisi aplikasi lainnya dengan memperbarui UI sesuai dengan keadaan (state) yang berubah. Contohnya adalah widget TextField yang memungkinkan pengguna untuk memasukkan teks dan merespons perubahan input.

Widget dalam Flutter dibangun dengan cara yang berlapis-lapis. Anda dapat menggabungkan widget-widget ke dalam struktur yang lebih kompleks, yang kemudian dapat digunakan kembali sebagai widget yang lebih besar. Konsep ini dikenal sebagai komposisi widget, dan memungkinkan pengembangan UI yang bersih, terorganisir, dan mudah dipelihara.

Selain itu, Flutter memiliki konsep widget “Stateless” dan “Stateful” yang merupakan bagian dari paradigma pembangunan UI deklaratif. Ini berarti UI didefinisikan dalam kode sebagai representasi dari keadaan (state) aplikasi saat ini, dan Flutter akan secara otomatis mengelola perubahan UI sesuai dengan perubahan keadaan aplikasi. Ini adalah salah satu kekuatan utama Flutter yang membuat pengembangan antarmuka pengguna menjadi lebih cepat, mudah, dan konsisten di seluruh platform.

4.3.2 Membuat Widget Sederhana

AppBar mungkin bisa disinonimkan dengan tag pada HTML, sebab AppBar merupakan fungsi untuk membuat head dari sebuah aplikasi, dimana didalamnya terdapat title yang dapat digunakan untuk menampilkan brand atau page apa yang sedang dibuka dari aplikasi tersebut. Buat project baru menggunakan Command Pallete.

Pilih menu Application. Pilih folder yang diinginkan, kemudian klik tombol Select a folder to create the project in, Beri nama projectnya, dengan mengetik widget_sederhana kemudian tekan Enter. Buka file lib/main.dart, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

METER COMBI

```
import 'package:flutter/material.dart';

void main() {
  runApp(HomePage());
}

class HomePage extends StatelessWidget {
  build(context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.red[800],
          leading: Icon(Icons.home),
          title: Text('Flutter Widget Sederhana')
        ),
      )
    )
  }
}
```

Gambar 23. Kode UI Sederhana

Penjelasan:

1. Line-1, import package yang dibutuhkan, dalam hal ini material.dart. main() adalah fungsi yang pertama kali diajalankan ketika aplikasi sedang di-load, maka apapun yang di-apit didalam main() maka code tersebut akan di-eksekusi.
2. Line-4 kita menggunakan runApp untuk me-render code kedalam screen aplikasi, dalam hal ini terdapat class HomePage() yang akan di-eksekusi.
3. Line-7, kita mendefinisikan sebuah class yang bernama HomePage.
4. Line-9, memberikan nilai balik yang berisi MaterialApp dari package yang telah di-import pada awal code.
5. MaterialApp memiliki property home yang berisi Scaffold widget. Perlu diketahui bahwa Scaffold widget inilah yang memiliki property appBar untuk membuat bar dari sebuah aplikasi, selain appBar, widget ini juga memiliki property lainnya, yakni: BottomAppBar, FloatingActionButton, dan lain sebagainya.
6. Property appBar dari Scaffold berisi AppBar widget.
7. AppBar widget juga memiliki banyak property, diantaranya: title, leading, actions, dan lain sebagainya. Tapi dalam case kali ini kita akan menggunakan property title yang berisi Text widget untuk menampilkan teks yang di-inginkan, leading untuk menampilkan icon home tepat sebelum teks dari title ditampilkan dan backgroundColor untuk memberikan warna pada AppBar.

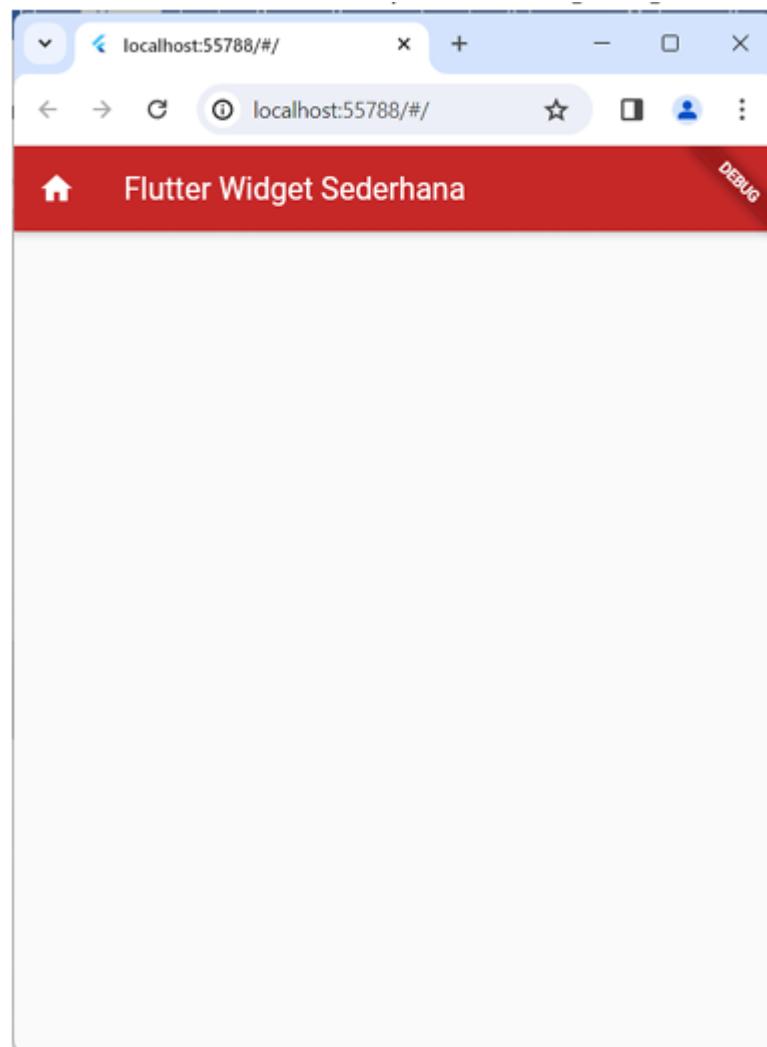
Untuk menjalankan koding, menggunakan terminal. Klik New Terminal. Kemudian ketikkan flutter run, tekan Enter. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

METER COMBI

```
PS D:\Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64     • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)       • chrome   • web-javascript • Google Chrome 111.0.5563.147
Edge (web)         • edge     • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on chrome...
```

Gambar 24. Tampilan Ketika akan Running Kode

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut



Gambar 25. Tampilan UI Widget Sederhana

METER COMBI

4.3.3 Mengatur Tata Letak (*Layout*)

Untuk mengatur tata letak widget di Flutter, Anda dapat menggunakan berbagai widget-layout yang disediakan oleh Flutter, seperti Row, Column, Stack, Container, dan lainnya. Di bawah ini adalah panduan umum untuk mengatur tata letak widget di Flutter

1. Menggunakan Widget-Layout Dasar:

- **Row**: Digunakan untuk menempatkan widget secara horizontal. Anda dapat menambahkan widget ke dalam Row dan mengatur tata letaknya menggunakan properti seperti mainAxisAlignment, crossAxisAlignment, dan mainAxisSize.
- **Column**: Digunakan untuk menempatkan widget secara vertikal. Anda dapat menambahkan widget ke dalam Column dan mengatur tata letaknya menggunakan properti yang sama seperti Row.
- **Stack**: Digunakan untuk menempatkan widget di atas satu sama lain. Anda dapat menambahkan widget ke dalam Stack dan mengatur posisi relatifnya menggunakan widget Positioned.

2. Menggunakan Widget Container:

- **Container**: Digunakan sebagai wadah umum untuk widget lainnya. Anda dapat mengatur tata letak dan penampilan widget di dalam Container menggunakan properti seperti margin, padding, alignment, color, decoration, dan lainnya.

3. Menggunakan Widget ListView:

- **ListView**: Digunakan untuk menampilkan daftar scrollable dari widget. Anda dapat menambahkan widget ke dalam ListView dan mengontrol perilaku scroll dan penampilannya menggunakan properti seperti scrollDirection, shrinkWrap, dan physics.

4. Menggunakan Widget Expanded dan Flexible:

- **Expanded**: Digunakan untuk memperluas widget dalam Row atau Column untuk mengisi ruang yang tersedia tambahan.
- **Flexible**: Digunakan untuk memberikan fleksibilitas dalam menyesuaikan ukuran widget dalam Row atau Column sesuai kebutuhan.

5. Menggunakan Widget SizedBox dan Spacer:

- **SizedBox**: Digunakan untuk menambahkan ruang kosong dengan lebar dan tinggi tertentu di antara widget lainnya.

METER COMBI

- **Spacer:** Digunakan untuk menambahkan ruang kosong yang dapat mengisi ruang yang tersedia dalam Row atau Column.

6. Menggunakan Properti Padding:

- Properti padding dapat digunakan pada widget tertentu untuk menambahkan ruang di sekeliling widget tersebut.

Dengan menggabungkan dan mengatur widget menggunakan widget layout dan properti yang sesuai, Anda dapat membuat tata letak yang sesuai dengan kebutuhan aplikasi Anda dalam Flutter. Penting untuk memahami konsep tata letak dan memilih widget dan properti yang tepat untuk mencapai tata letak yang diinginkan.

METER COMBI

Bagian 5 | Pembuatan Desain Sederhana Meter Combi

5.1 Struktur Proyek Flutter

lib/

```
|── main.dart          # Entry point aplikasi  
|── models/  
|   |── sensordata.dart    # Penerimaan data dari sensor  
|── screens/  
|   |── dashboard_screen.dart  # Tampilan utama  
|── services/  
|   |── data_services.dart    # Pembacaan data dari sensor  
|── widgets/  
|   |── main_gauge.dart  # Tampilan speedometer  
|── assets  
|   |── images/ # Kumpulan images yang digunakan
```

5.2 Desain UI Meter Combi

5.2.1 Pembuatan User Interface

main.dart

```
import 'package:flutter/material.dart';  
import 'simple_meter_app.dart'; // <<< Import aplikasi meter sederhana  
  
void main() {  
    runApp(const SimpleMeterApp()); // <<< Jalankan aplikasi meter sederhana  
}
```

METER COMBI

simple_meter_app.dart

```
import 'package:flutter/material.dart';
import 'dart:math' as math; // Diperlukan untuk fungsi matematika (pi, cos, sin)

// --- APLIKASI UTAMA (StatelessWidget) ---
class SimpleMeterApp extends StatelessWidget {
  const SimpleMeterApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Combi Meter',
      debugShowCheckedModeBanner: false, // Sembunyikan banner
      "DEBUG"
      theme: ThemeData(
        primarySwatch: Colors.blue, // Tema warna utama
        brightness: Brightness.dark, // Tema gelap
        scaffoldBackgroundColor: const Color(0xFF222222), // Warna latar belakang
      ),
      home: const SimpleMeterHomePage(), // Halaman awal aplikasi
    );
  }
}

// --- HALAMAN UTAMA ( StatefulWidget) ---
class SimpleMeterHomePage extends StatefulWidget {
  const SimpleMeterHomePage({super.key});

  @override
  State<SimpleMeterHomePage> createState() =>
  _SimpleMeterHomePageState();
}
```

METER COMBI

```
class _SimpleMeterHomePageState extends  
State<SimpleMeterHomePage> {  
    double _currentValue = 0.0; // Nilai meteran saat ini  
    final double _maxValue = 100.0; // Nilai maksimum meteran  
  
    // Fungsi untuk memperbarui nilai meteran (digunakan untuk  
    menambah/mengurangi)  
    void _updateValue(double delta) {  
        setState(() {  
            _currentValue = (_currentValue + delta).clamp(0.0, _maxValue); //  
            Tambah/kurang nilai, batas 0 - 100  
        });  
    }  
  
    @override  
    Widget build(BuildContext context) {  
        return Scaffold(  
            appBar: AppBar(  
                title: const Text('Combi Meter'),  
                centerTitle: true,  
            ),  
            body: Center(  
                child: Column(  
                    mainAxisAlignment: MainAxisAlignment.center, // Pusatkan secara  
                    vertikal  
                    children: [  
                        SizedBox(  
                            width: 200, // Lebar area gambar meteran  
                            height: 150, // Tinggi area gambar meteran  
                            child: CustomPaint(  
                                painter: _InlineMeterPainter( // Menggunakan CustomPainter  
                                sebagai kelas inline  
                                currentValue: _currentValue,  
                                maxValue: _maxValue,  
                            ),  
                            ),  
                            ),  
                            const SizedBox(height: 20), // Jarak  
                        ),  
                    ],  
                ),  
            ),  
        );  
    }  
}
```

METER COMBI

```
// Teks untuk menampilkan nilai saat ini
Text(
    'Nilai: ${_currentValue.toStringAsFixed(0)}', // Tampilkan nilai tanpa
desimal
    style: const TextStyle(fontSize: 24, color: Colors.white),
),
],
),
),
),
// Tombol mengambang untuk menambah nilai
floatingActionButton: FloatingActionButton.extended(
 onPressed: () => _updateValue(10.0), // Panggil _updateValue untuk
menambah 10
label: const Text('Tambah Nilai'),
icon: const Icon(Icons.add),
backgroundColor: Colors.green,
),
);
}
}

// --- CUSTOM PAINTER SEBAGAI KELAS INLINE ---
class _InlineMeterPainter extends CustomPainter {
final double currentValue;
final double maxValue;

_INLINE_MeterPainter({required this.currentValue, required this.maxValue});

@Override
void paint(Canvas canvas, Size size) {
// Pusat busur di bawah tengah area gambar
final center = Offset(size.width / 2, size.height);
final radius = size.width / 2;
const strokeWidth = 8.0; // Ketebalan busur

// Sudut busur (setengah lingkaran di bagian bawah)
final startAngle = math.pi; // 180 derajat (kiri)
final sweepAngle = math.pi; // 180 derajat (setengah lingkaran penuh)
```

METER COMBI

```
// Paint untuk busur latar belakang (tidak aktif)
final bgPaint = Paint()
..strokeWidth = strokeWidth
..style = PaintingStyle.stroke
..color = Colors.grey.withOpacity(0.3); // Warna abu-abu transparan

// Menggambar busur latar belakang
canvas.drawArc(
Rect.fromCircle(center: center, radius: radius - strokeWidth / 2),
startAngle,
sweepAngle,
false,
bgPaint,
);

// Paint untuk busur nilai (aktif)
final valuePaint = Paint()
..strokeWidth = strokeWidth
..style = PaintingStyle.stroke
..color = Colors.blueAccent; // Warna biru terang

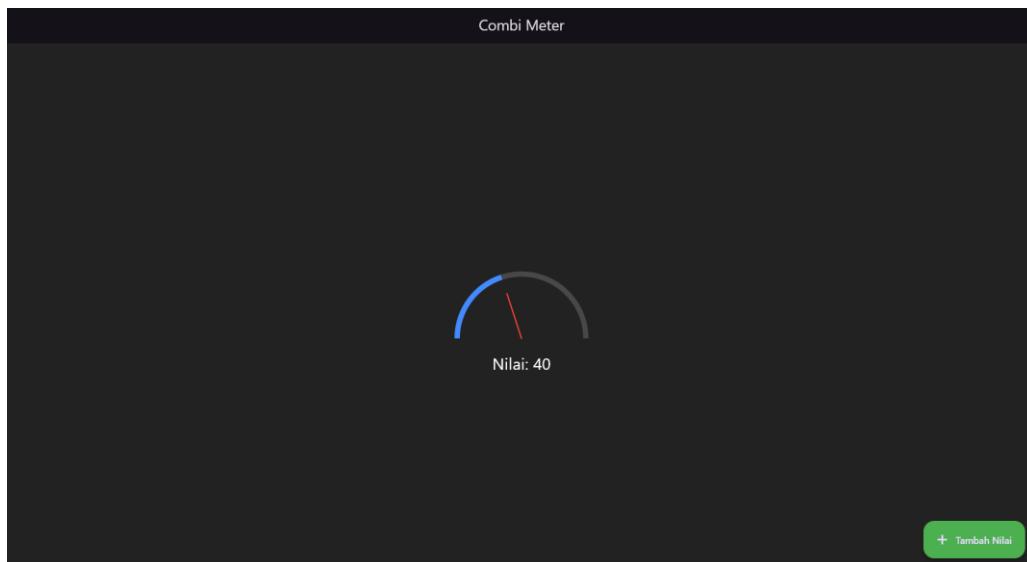
// Menghitung sudut sapuan untuk nilai saat ini
final currentSweep = sweepAngle * (currentValue / maxValue).clamp(0.0,
1.0);

// Menggambar busur nilai aktif
canvas.drawArc(
Rect.fromCircle(center: center, radius: radius - strokeWidth / 2),
startAngle,
currentSweep,
false,
valuePaint,
);

// Menggambar jarum sederhana
final needleAngle = startAngle + currentSweep;
final needleLength = radius * 0.7;
final needlePaint = Paint()
..color = Colors.red
..strokeWidth = 2.0
..strokeCap = StrokeCap.round;
```

METER COMBI

```
final needleTip = center + Offset(needleLength *  
math.cos(needleAngle), needleLength * math.sin(needleAngle));  
    canvas.drawLine(center, needleTip, needlePaint);  
}  
  
@override  
bool shouldRepaint(covariant _InlineMeterPainter oldDelegate) {  
    return oldDelegate.currentValue != currentValue; // Hanya repaint jika  
nilai berubah  
}  
}
```



Gambar 26. Tampilan UI Meter Combi Sederhana

Pada bagian ini, kita telah mempelajari bagaimana membuat tampilan sederhana meter combi yang terdiri atas jarum, bentuk setengah lingkaran, keterangan nilai, dan tombol untuk menambah nilai pada meter combi.

Poin-poin penting yang telah dipelajari :

1. Konsep Sederhana Desain Sederhana Meter Combi

Menggabungkan data visual (teks, ikon, warna, animasi) dalam layout yang mudah dipahami pengguna

2. Interaktivitas

Tombol untuk menambah nilai pada meter combi

3. Penggunaan Flutter untuk User Interface

Flutter memungkinkan kita untuk membuat UI yang dinamis dan modern.

Bagian 6 | Integrasi Flutter dengan ESP 32 (Arduino)

6.1 Integrasi Sistem

Integrasi ini bertujuan agar data yang dikirim dari ESP32 seperti suhu, status pintu, dan tombol starter dapat ditampilkan langsung di antarmuka Flutter secara real-time, baik dalam bentuk teks, grafik, maupun indikator visual lainnya.

6.1.1 Arsitektur Sistem

```
[Sensor & Switches] --> [ESP32] --> (Serial / HTTP) --> [Flutter App] --> [UI Display]
```

ESP32 membaca data dari sensor (mis. DHT11, switch pintu, dll), lalu mengirimkan data ke aplikasi Flutter melalui komunikasi:

- Serial (via USB/OTG/COM port)
- HTTP (via WiFi POST request)

6.1.2 Tools & Packages Flutter yang Digunakan

Package	Fungsi
http	Untuk menerima data dari ESP32 via REST API
flutter_serial_port / usb_serial (opsional)	Untuk komunikasi USB Serial (jika menggunakan kabel)
provider atau riverpod	Untuk mengelola state data sensor secara reaktif
fl_chart	Untuk menampilkan grafik real-time (misalnya suhu)
gauge_indicator	Untuk menampilkan meteran speedometer

6.1.3 Mengubah Data dari JSON Menjadi Objek Dart

Contoh penerapan mekanisme koneksi serial ESP dengan Raspberry Pi

METER COMBI

```
● ● ●

factory SensorData.fromJson(Map<String, dynamic> json) {
    return SensorData(
        speed: (json['speed'] as num?)?.toDouble() ?? 0.0,
        temperature: (json['temperature'] as num?)?.toDouble() ?? 0.0,
        level: (json['level'] as num?)?.toDouble() ?? 0.0,
        pressure: (json['pressure'] as num?)?.toDouble() ?? 0.0,
        voltage: (json['voltage'] as num?)?.toDouble() ?? 0.0,
        brake: (json['brake'] is bool) ? json['brake'] : (json['brake'] as int? ?? 0) == 1,
        abs: (json['abs'] is bool) ? json['abs'] : (json['abs'] as int? ?? 0) == 1,
        airbag: (json['airbag'] is bool) ? json['airbag'] : (json['airbag'] as int? ?? 0) == 1,
        seatbelt: (json['seatbelt'] is bool) ? json['seatbelt'] : (json['seatbelt'] as int? ?? 0) == 1,
    );
}
```

Gambar 27. Kode Penerima Data Serial

6.1.4 Komunikasi Serial dari ESP ke Raspberry Pi

Anda bisa menyesuaikan Port yang digunakan pada device anda. Anda bisa mengecek dengan cara mencari "device manager" pada *search windows*. Kemudian, cari "Ports (COM & LPT)". Port yang digunakan bisa anda lihat disitu.

```
● ● ●

class DataService {
    final SerialPort _port;
    late SerialPortReader _reader;
    DataService({String portName = 'COM30'}) {
        _port = SerialPort(portName);
    }
}
```

Gambar 28. Kode untuk Mengubah Port

6.1.5 Kode ESP 32 Meter Combi

Kode lengkap ESP 32 untuk Meter Combi bisa Anda cek pada bagian 8 : Lampiran

Bagian 7 | Evaluasi & Refleksi

7.1 Evaluasi Pemahaman (Kuis Singkat)

Berikut adalah beberapa pertanyaan yang bisa digunakan oleh instruktur untuk menguji pemahaman dasar peserta:

No.	Pertanyaan
1	Apa fungsi utama dari Meter Combi dalam sebuah kendaraan modern?
2	Sebutkan dua jenis perangkat keras utama yang berperan sebagai "otak" dan "jembatan" dalam sistem simulasi meter combi ini
3	Jelaskan mengapa "Add Python to PATH" penting saat instalasi Python di Windows untuk proyek ini
4	Mengapa Anda perlu menginstal board support ESP32 di Arduino IDE, dan bagaimana cara melakukannya?
5	Apa perbedaan utama antara StatelessWidget dan StatefulWidget dalam Flutter? Berikan contoh masing-masing.
6	Jelaskan fungsi setup() dan loop() dalam pemrograman ESP32. Mengapa keduanya penting?
7	Dalam konteks komunikasi serial ESP32 ke aplikasi Flutter, mengapa karakter \n (newline) penting di akhir setiap pengiriman data KEY:VALUE?
8	Sebutkan dua paket Flutter yang digunakan untuk visualisasi UI dan satu paket untuk komunikasi serial dengan ESP32 dalam proyek ini.
9	Mengapa disarankan untuk tidak mencolokkan daya monitor langsung ke Raspberry Pi?
10	Bagaimana modul ini meningkatkan literasi teknologi dan mempersiapkan peserta untuk menghadapi tantangan teknologi di masa depan?

7.2 Refleksi Diri

Gunakan pertanyaan ini untuk mendorong peserta mengevaluasi pengalamannya secara pribadi selama pelatihan:

1. Apa hal paling menarik yang kamu pelajari dari pelatihan membangun simulator meter combi ini?
2. Bagaimana menurutmu pemahaman tentang integrasi sensor dan UI seperti ini bisa diaplikasikan atau membantu dalam pekerjaanmu sehari-hari di TMMIN? Berikan contoh spesifik.

3. Apa tantangan terbesar yang kamu hadapi selama memahami materi, baik dari segi perangkat keras, pemrograman, atau integrasi? Bagaimana kamu mengatasinya?
4. Setelah menyelesaikan modul ini, langkah apa yang akan kamu ambil selanjutnya untuk memperdalam pemahaman atau mengembangkan keterampilan terkait teknologi ini? Apakah ada fitur meter combi lain yang ingin kamu coba implementasikan?

7.3 Penutup Pelatihan

Pelatihan ini merupakan fondasi awal yang dirancang untuk memperkenalkan Anda pada konsep dan praktik nyata integrasi teknologi dalam sistem otomotif modern. Di tengah era digitalisasi yang pesat, kemampuan untuk memahami dan berinteraksi dengan perangkat keras dan perangkat lunak menjadi aset yang sangat berharga. Modul ini dirancang untuk memberikan pemahaman komprehensif mengenai integrasi teknologi dalam sistem otomotif modern, mempersiapkan Anda untuk menghadapi tantangan teknologi di masa mendatang.

Melalui pendekatan berbasis praktik, Anda telah diajak memahami alur kerja sebuah sistem cerdas: dari mengumpulkan data fisik, memprosesnya, hingga mengubahnya menjadi informasi visual yang bermakna. Anda telah secara langsung berinteraksi dengan perangkat keras utama seperti mikrokontroler ESP32-C6 sebagai "jembatan" antara sensor dan sistem utama, serta Raspberry Pi 5 sebagai "otak" utama yang menjalankan aplikasi tampilan dashboard. Pengalaman ini memberikan wawasan tentang bagaimana data mentah dari lingkungan fisik diubah menjadi informasi yang bermakna bagi pengemudi.

Secara spesifik dalam konteks simulator meter combi ini, Anda telah terlibat langsung dalam pembangunan versi simulasinya sendiri, yang secara signifikan meningkatkan pemahaman teknologi dan literasi digital Anda. Anda memulai dengan memahami dasar-dasar pemrograman perangkat keras menggunakan ESP32 untuk membaca berbagai sensor seperti suhu, kecepatan (rotary encoder), level (ultrasonik), tekanan, tegangan, serta status digital seperti rem, ABS, airbag, dan sabuk pengaman. Anda juga berhasil mengonfigurasi ESP32 untuk mengirimkan data ini secara serial.

Kemudian, Anda melangkah ke pengembangan antarmuka pengguna (UI) menggunakan Flutter, membangun aplikasi dashboard yang dinamis dan responsif. Anda telah menguasai konsep dasar widget dan tata letak, mampu menciptakan elemen visual seperti speedometer kustom, bar indikator, dan tampilan status lainnya. Puncaknya, Anda berhasil mengintegrasikan data serial langsung dari ESP32 ke aplikasi Flutter Anda, membuat tampilan meter combi berfungsi secara real-time seperti panel instrumen kendaraan sesungguhnya.

METER COMBI

Pelatihan ini juga bertujuan menanamkan pola pikir baru bahwa teknologi adalah alat bantu, bukan pengganti. Karyawan yang mampu memahami dan bekerja bersama teknologi seperti ini akan menjadi bagian penting dari transformasi digital perusahaan, termasuk di TMMIN. Dengan pemahaman dasar yang kuat tentang bagaimana komponen kendaraan berinteraksi dengan sensor dan sistem elektronik, Anda diharapkan mampu mengidentifikasi peluang untuk inovasi dan peningkatan efisiensi di area kerja Anda.

Sebagai penutup, pelatihan ini bukanlah akhir, melainkan awal dari proses pembelajaran yang berkelanjutan. Dunia teknologi otomotif dan integrasi sistem terus berkembang. Semoga pelatihan membangun simulator meter combi ini menjadi pijakan awal yang kuat, membangun rasa ingin tahu, dan mendorong Anda untuk terus menggali potensi diri dalam menyongsong masa depan industri yang semakin cerdas dan otomatis.

METER COMBI

Bagian 8 | LAMPIRAN

Kode ESP 32 :

```
// Include the libraries we need

#include <OneWire.h>

#include <DallasTemperature.h>

#include "AiEsp32RotaryEncoder.h"

#include <Wire.h>

#include <Adafruit_INA219.h>

#include <DFRobot_HX711.h>

#define ONE_WIRE_BUS 4

OneWire oneWire(ONE_WIRE_BUS);

DallasTemperature tempSensor(&oneWire);

#define TRIG_PIN 20 // ESP32 pin GPIO20 connected to Ultrasonic Sensor's TRIG pin

#define ECHO_PIN 19 // ESP32 pin GPIO19 connected to Ultrasonic Sensor's ECHO pin

#define ROTARY_ENCODER_A_PIN 22

#define ROTARY_ENCODER_B_PIN 21

#define ROTARY_ENCODER_BUTTON_PIN 23

#define ROTARY_ENCODER_VCC_PIN -1

#define ROTARY_ENCODER_STEPS 4

#define MAX_ROTARY_VALUE 100

AiEsp32RotaryEncoder          rotaryEncoder      =
AiEsp32RotaryEncoder(ROTARY_ENCODER_A_PIN,    ROTARY_ENCODER_B_PIN,
ROTARY_ENCODER_BUTTON_PIN, -1, ROTARY_ENCODER_STEPS);

void IRAM_ATTR readEncoderISR()
```

METER COMBI

```
{  
    rotaryEncoder.readEncoder_ISR();  
}  
  
unsigned long rpmPeriod = 0;  
  
Adafruit_INA219 ina219;  
  
DFRobot_HX711 MyScale(9, 18);  
  
#define BREAK_PIN 10  
#define ABS_PIN 11  
#define AIRBAG_PIN 2  
#define SEATBELT_PIN 8  
  
void setup(void)  
{  
    // start serial port  
    Serial.begin(115200);  
    // Start up the library  
    tempSensor.begin();  
  
    pinMode(TRIG_PIN, OUTPUT);  
    pinMode(ECHO_PIN, INPUT);  
  
    rotaryEncoder.begin();  
    rotaryEncoder.setup(readEncoderISR);  
    rotaryEncoder.setBoundaries(0, MAX_ROTARY_VALUE, true);  
    rotaryEncoder.disableAcceleration();
```

METER COMBI

```
Wire.begin(5, 6);
if (! ina219.begin()) {
    Serial.println("Failed to find INA219 chip");
    while (1) { delay(10); }
}

ina219.setCalibration_16V_400mA();

pinMode(BREAK_PIN, INPUT_PULLUP);
pinMode(ABS_PIN, INPUT_PULLUP);
pinMode(AIRBAG_PIN, INPUT_PULLUP);
pinMode(SEATBELT_PIN, INPUT_PULLUP);

}

unsigned long lastMillis = 0;
unsigned long period = 1000;
void loop(void)
{
    if(millis() - lastMillis >= period){

        lastMillis = millis();
        tempSensor.requestTemperatures(); // Send the command to get temperatures
        float tempC = tempSensor.getTempCByIndex(0);

        digitalWrite(TRIG_PIN, HIGH);
        delayMicroseconds(10);
        digitalWrite(TRIG_PIN, LOW);
        float duration_us = pulseIn(ECHO_PIN, HIGH);
        float distance_cm = 0.017 * duration_us;
        distance_cm = constrain(distance_cm, 0, 100);
    }
}
```

METER COMBI

```
distance_cm = map(distance_cm, 0, 100, 100, 0);

float shuntvoltage = ina219.getShuntVoltage_mV();
float busvoltage = ina219.getBusVoltage_V();
float current_mA = ina219.getCurrent_mA();
float power_mW = ina219.getPower_mW();
float loadvoltage = busvoltage + shuntvoltage;
loadvoltage = constrain(loadvoltage, 0.00, 4.75);
loadvoltage = (loadvoltage - 0.00)/(4.75 - 0.00) * 100;

float weight = MyScale.readWeight();
weight = constrain(weight, 0, 100);

bool breakVal = !digitalRead(BREAK_PIN);
bool absVal = !digitalRead(ABS_PIN);
bool airbagVal = !digitalRead(AIRBAG_PIN);
bool seatbeltVal = !digitalRead(SEATBELT_PIN);

float step = rotaryEncoder.readEncoder();
float rpm = (step * 60)/MAX_ROTARY_VALUE;
rotaryEncoder.setEncoderValue(0);

//Serial.printf("{\"speed\":"+%.2f,"temperature\":"+%.2f,"level\":"+%.2f,"pressure\":"+%.2f
//"voltage\":"+%.2f,"break\":"+%d,"abs\":"+%d,"airbag\":"+%d,"seatbelt\":"+%d}\n", rpm, tempC,
distance_cm, weight, loadvoltage, breakVal, absVal, airbagVal, seatbeltVal);

Serial.printf("Speed:%.0f\n", rpm); // Mengirim "Speed:nilai_rpm"
Serial.printf("Temperature:%.1f\n", tempC); // Key "Temperature"
Serial.printf("Level:%.0f\n", distance_cm); // Key "Level"
Serial.printf("Pressure:%.1f\n", weight); // Key "Pressure"
```

METER COMBI

```
Serial.printf("Voltage:%.2f\n", loadvoltage); // Key "Voltage" (menggunakan  
busvoltage)  
  
Serial.printf("Brake:%d\n", breakVal); // Key "Brake" (0/1)  
  
Serial.printf("ABS:%d\n", absVal); // Key "ABS"  
  
Serial.printf("Airbag:%d\n", airbagVal); // Key "Airbag"  
  
Serial.printf("Seatbelt:%d\n", seatbeltVal); // Key "Seatbelt"  
  
}  
  
}
```