DESIGN AND FABRICATION OF **BLUETOOTH** AND **WI-FI** CONTROL SYSTEM OF VARIOUS DOMASTIC APPLIANCES

OBJECTIVE

- Smart Switch Board is a multipurpose home automation device built around the ESP32 microcontroller.
- It controls two sockets via relays and receives input from two manual switches
- The system supports three control modes: Bluetooth, MQTT (via EMQX broker), and a local web server.

HARDWARE AND COMPONENTS

S.NO	NAME AND DISCRIPTION	No/ length	RATE	TOTAL
1	ESP32 : MICROCONTROLLER	1	₹350	₹350
2	RELAY MODULE	2	₹100	₹200
3	PUSH BUTTON	1	₹30	₹30
4	PUSH SWITCH	2	₹50	₹100
5	SOCKET	2	₹40	₹80
6	JUMPER WIRES	20	₹5	₹100
7	5v POWER ADAPTER	1	₹150	₹150
8	WIRES	3m	₹20	₹60
9	OUTER BOADY	1	₹100	₹100

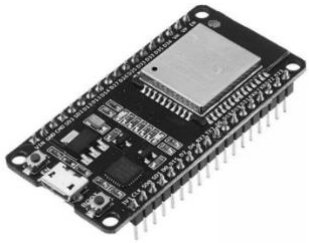
TOTAL = 350+200+30+100+80+100+150+60+100

= ₹1170

DISCRIPTION OF COMPONENTS

ESP32 MICROCONTROLLER

ESP32 is a popular, low-cost, low-power **Wi-Fi and Bluetooth-enabled** microcontroller developed by **Espressif** Systems. It is widely used in Internet of Things (**IoT**) projects due to its versatility, strong processing power, and wireless connectivity.



Key Features of ESP32

1. Dual-Core Processor

- 32-bit Xtensa LX6 dual-core (or single-core in some variants), clocked up to 240 MHz
- More powerful than the older ESP8266

2. Wireless Connectivity

- **Wi-Fi (802.11 b/g/n)** – Supports both Station (client) and Access Point (AP) modes
- **Bluetooth** – Supports Bluetooth Classic (BT) and Bluetooth Low Energy (BLE)

3. Memory

- 520KB SRAM (for data)

- 4MB Flash memory (for program storage; varies by module)

4. Peripherals & Interfaces

- Up to 36 programmable GPIOs
- **Analog Inputs (ADC):** 12-bit resolution, up to 18 channels
- **Digital Interfaces:** SPI, I2C, I2S, UART, CAN, PWM
- **Touch Sensors:** Built-in capacitive touch sensing
- **Hall Effect Sensor:** Detects magnetic fields
- **DAC:** Digital-to-Analog Converter for analog output

5. Low Power Modes

- Supports deep sleep mode (~10µA), ideal for battery-powered projects

---

How to Use ESP32

1. Install **Arduino IDE** (or **PlatformIO**)
2. Add **ESP32 Board Support** via Board Manager
3. Write and Upload Code

Relay Module

A relay module is an electrically operated switch that allows a low-power signal from a microcontroller (e.g., ESP32) to control high-power circuits like lights, motors, or appliances. It provides safe isolation between the control circuit and the load circuit.

How Does a Relay Work?

1. Control Side (Low Voltage)

- A small voltage (e.g., 3.3V or 5V) from the ESP32 triggers an internal electromagnet.
- The electromagnet mechanically opens or closes the circuit.

2. Load Side (High Voltage)

- Switches AC or DC loads (e.g., lights, fans, motors)
- Common ratings: 10A at 250V AC or 10A at 30V DC (varies by module)

Wiring a Relay Module to ESP32

Relay Pin	ESP32 Pin
VCC	5V (or 3.3V)
GND	GND
IN (Signal)	Any GPIO

Safety Tips

- Use a **separate power supply** for high-voltage loads
- **Insulate high-voltage terminals** to prevent electric shocks
- **Check the relay's voltage/current ratings** and don't exceed them

---

PUSH BUTTON/SWITCH

- A push button is a type of electrical switch that makes or breaks a connection only while it is being pressed. As soon as you release the button, the connection is broken again.
- It is a momentary switch, meaning it works only temporarily while being pressed.

PUSH BUTTON.

PUSH SWITCH

TECHNOLOGIES, WE HAVE USED

1. **MQTT** {Message Queuing Telemetry Transport}
2. **AP** {ACCESS POINT}
- **BLE** {BLUETOOTH LOW ENERGY}

MQTT:

MQTT is a **lightweight messaging protocol** designed for **low-bandwidth, high-latency**, or **unreliable networks**. It's widely used in **IoT (Internet of Things)** applications because it minimizes power and data usage.

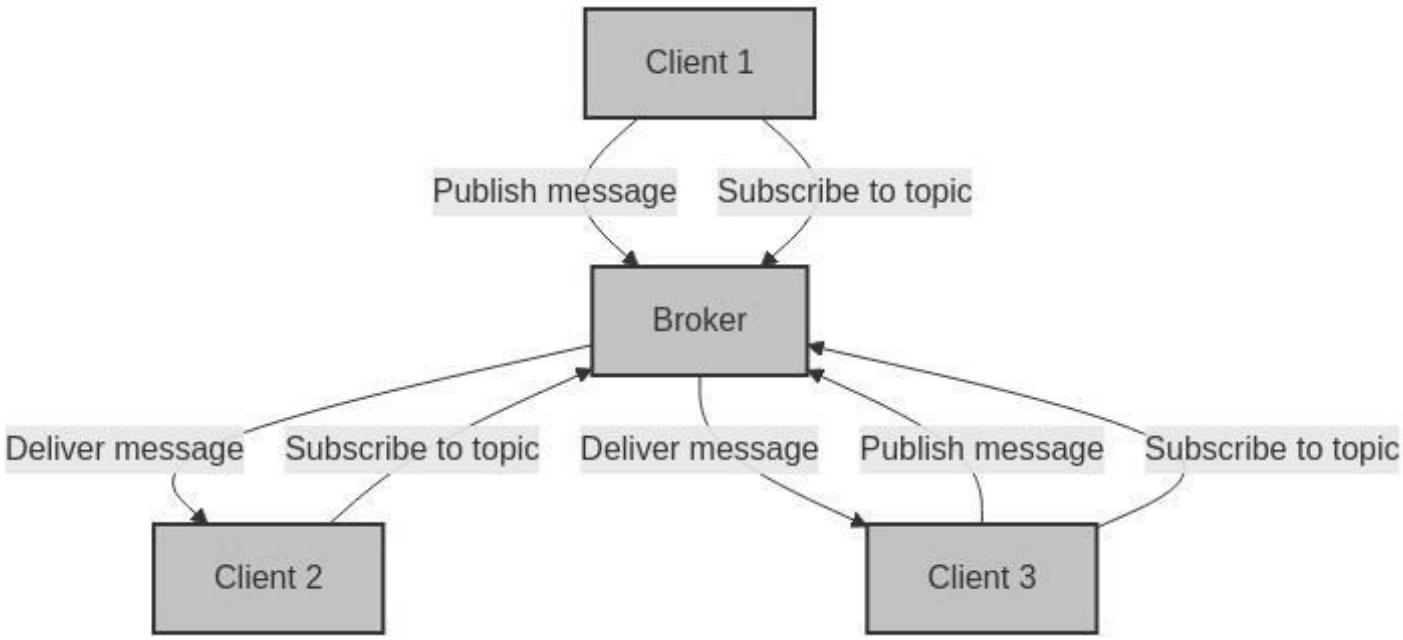
- **Full form:** Message Queuing Telemetry Transport
- **Developed by:** IBM
- **Transport layer:** Works over TCP/IP

Key Concepts in MQTT

Term	Description
Broker	The central server that handles all message routing.
Client	Any device (ESP32, phone, PC) that connects to the broker.
Publisher	A client that sends (publishes) messages to a topic.
Subscriber	A client that receives messages from a topic.
Topic	A label or path (like `home/light1/status`) used to categorize messages.

How MQTT Works

1. **Clients connect to a broker.**
2. **Publishers** send data to a **topic** on the broker.
3. **Subscribers** receive data from topics they subscribed to.
4. The **broker** ensures the messages go from publishers to the right subscribers.



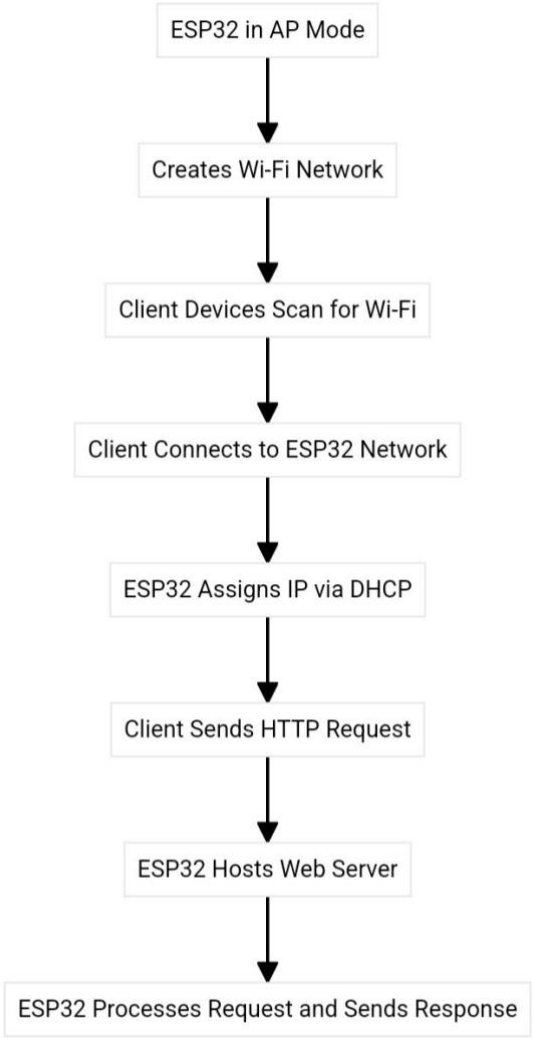
FLOWCHART OF WORKING OF MQTT

AP(ACCESS POINT)

Access Point (AP) Mode is one of the operating modes of Wi-Fi-enabled devices like the **ESP32**, where the device creates its **own Wi-Fi network** that other devices (like phones, laptops, etc.) can connect to directly.

In **AP Mode**:

- The **ESP32** acts like a **Wi-Fi router**, broadcasting its own **SSID** (network name).
- **No internet** connection is required.
- Other devices **connect directly** to the ESP32 by selecting its Wi-Fi network.
- Once connected, these devices can interact with the **ESP32**.



BLE (BLUETOOTH LOW ENERGY)

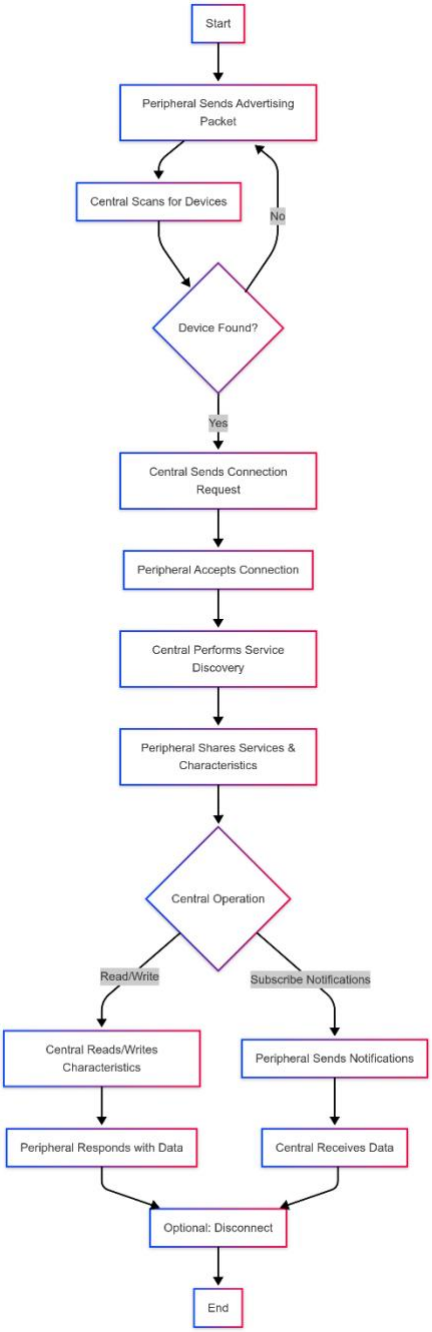
- BLE (Bluetooth Low Energy) is a wireless personal area network technology designed for short-range communication with minimal power consumption. It's commonly used in applications where energy efficiency is critical, such as wearable devices, smart home systems, medical equipment, and fitness trackers.

Key Characteristics of BLE:

- **Low power consumption**
-
- **Short-range communication (typically ~10 meters)**
-
- **Efficient for periodic data transfers**
- **Supports broadcasting and connection-based communication**
- **Used in IoT, healthcare, fitness, and more**

How BLE Works (Basic Flow)

1. **Advertising**
   The **peripheral** sends out small packets (ads) saying, "Hey, I'm here!"
2. **Scanning**
   The **central** listens for these packets and sees what devices are nearby.
3. **Connection**
   The central connects to a chosen peripheral.
4. **Service Discovery**
   The central asks, "What do you offer?" and the peripheral responds with its services and characteristics.
5. **Data Exchange**
   The central can **read**, **write**, or **subscribe** to characteristics to get updates.
6. **Disconnection**
   The connection ends to save power or when done.



CONNECTION AND CIRCUIT DIAGRAM

In this smart board project, the **ESP32** microcontroller manages **two relays**, **manual switches**, a **mode selection button**, and a **status LED**. Additionally, it controls two **230V** sockets through the relays. The connections are as follows:

Power Supply to ESP32

The **ESP32** is powered using a **5V regulated adapter**:

5V from Adapter → **VIN** pin of ESP32

GND from Adapter → **GND** pin of ESP32

This provides the necessary operating voltage for the ESP32 board.

Relay Module Connections

Two relays are used for switching the 230V AC sockets. The relay module is powered externally:

Relay Module **VCC** → **5V** from Adapter

Relay Module **GND** → **GND** from Adapter

Relay 1 **IN** pin → **GPIO 16** (ESP32)

Relay 2 **IN** pin → **GPIO 17** (ESP32)

The ESP32 controls the relays through GPIO 16 and 17 to switch the connected sockets.

Socket Wiring (230V AC Load Control)

Two AC sockets are connected through the relays to allow switching of high-voltage appliances:

Phase (Live) wire from AC supply → Common (**COM**) terminal of both relays

Normally Open (NO) terminal of Relay 1 → **Socket 1** Phase input

Normally Open (NO) terminal of Relay 2 → **Socket 2** Phase input

Neutral wire from AC supply → **Neutral terminals of both sockets**

This configuration ensures that the sockets receive AC power only when the corresponding relay is activated (closed).

Manual Control Switches

Two push-button switches are used for local manual control:

Switch 1 → **GPIO 26** (ESP32)

Switch 2 → **GPIO 27** (ESP32)

These act as digital inputs, enabling manual toggling of the respective relays.

Mode Selection Button

A dedicated push-button is connected to:

GPIO 4 (ESP32)

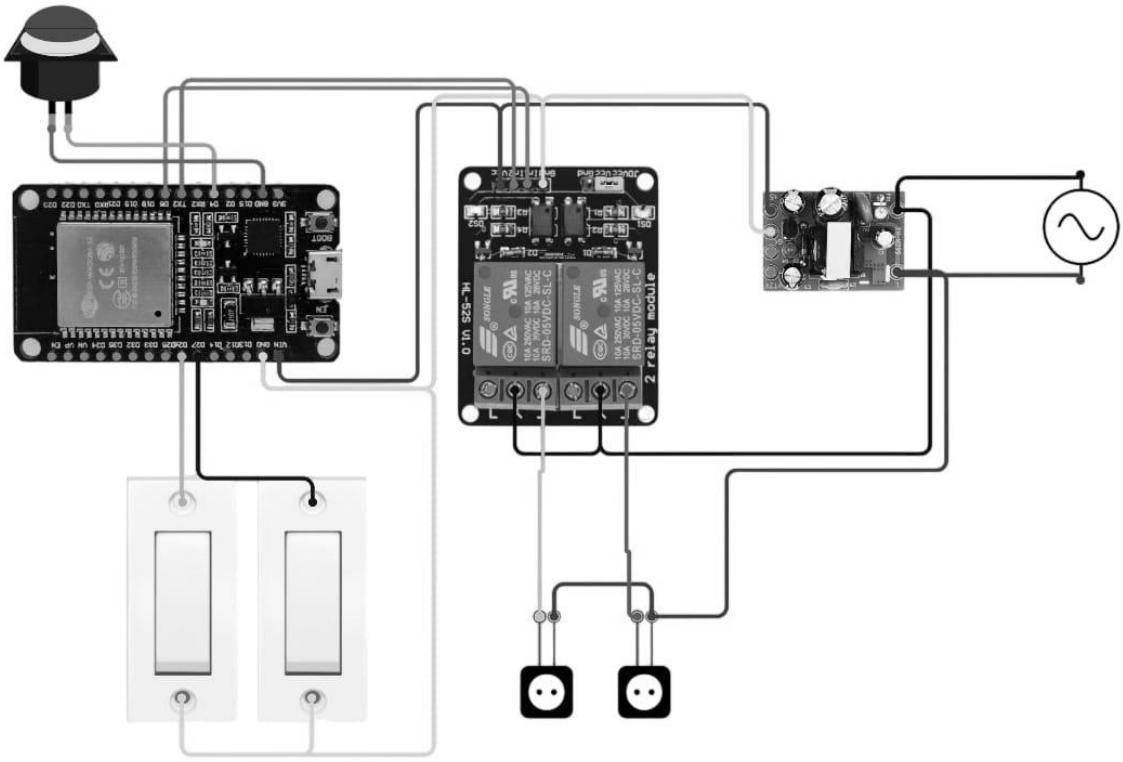
This button allows the user to cycle through different modes of operation (e.g., Bluetooth, MQTT, Web Server).

Status LED

A single LED is connected to:

GPIO 2 (ESP32)

This LED indicates system status or operating mode based on the firmware logic.



CIRCUIT DIAGRAM

WORKING

The Smart Board is an intelligent control system built around the ESP32 microcontroller, designed to provide flexible and reliable control of electrical devices through multiple interfaces including web, Bluetooth, and manual switches.

System Overview

The system consists of:

- ESP32 Microcontroller (Main Controller)

- Two Relay Outputs

- Two Manual Switches

- One Mode Selection Button

- One Status LED

- Multiple Communication Interfaces (WiFi, BLE, MQTT)

Operating Modes

1. **Access Point** (AP) Mode

- Creates its own WiFi network

- Web interface accessible through:

  - SSID: **SmartBoard_AP**

  - Password: **12345678**

- Direct control without internet

- Configuration available

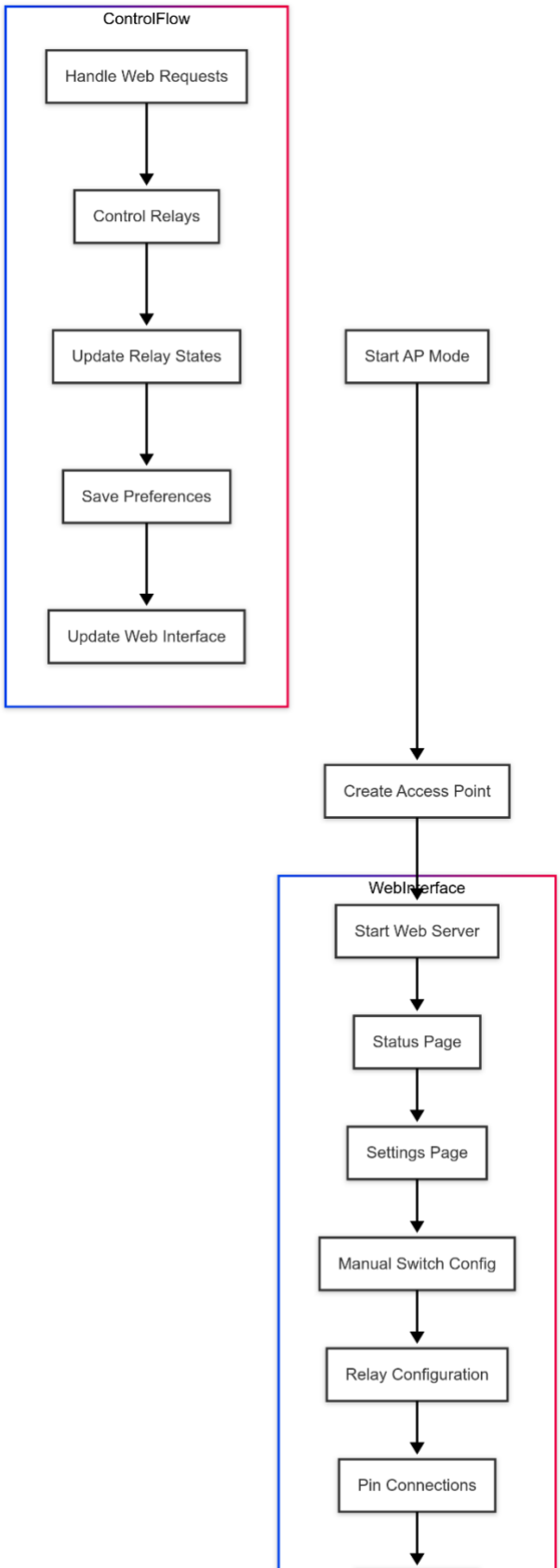
2. **Wi-Fi + MQTT Mode**

- Connects to existing WiFi network

- MQTT broker connection

- Remote control capability

- Status updates through MQTT



Control Methods

1. **Web Interface**

- Accessible in AP mode

- Dashboard for status monitoring

- Control panel for relays

- Configuration settings

- Manual switch setup

- Pin configuration

2. **MQTT Control**

- Topic: smartboard/relays

- Commands:

  - ON: {"relay0": true,” relay1”: true}

  - OFF: {"relay0": false,” relay1”: false}

- Status updates published

- Real-time monitoring

3. **Bluetooth (BLE) Control**

- Service UUID: **4fafc201-1fb5-459e-8fcc-c5c9c331914b**

- Characteristic UUID: **beb5483e-36e1-4688-b7f5-ea07361b26a8**

- Commands same as MQTT

- Direct device connection

4. **Manual Switch Control**

- Direct hardware control

- Independent of communication modes

- Emergency override capability

- Immediate response

Working Flow

1. System Initialization

1. Power on

2. Load preferences from memory
3. Initialize hardware pins
4. Setup communication interfaces {BLE + AP}
5. Check mode button status

2. Mode Selection

- If mode button pressed:
    - Toggle between AP and Wi-Fi + MQTT modes
    - Save new mode to preferences
    - Reboot system

3. Operation

- Continuous monitoring of:
    - Mode button
    - Manual switches
    - Communication interfaces
    - Relay states
- Status updates through all interfaces
- Safety checks and error handling

Safety Features

1. Hardware Safety

- Internal pull-up resistors
- Proper power supply isolation
- Relay protection circuits
- Emergency manual override

2. Software Safety

- Input validation
- State monitoring
- Error handling
- Automatic recovery
- Preference saving

3. Communication Safety

- Secure Wi-Fi connection
- MQTT authentication
- BLE security
- Web interface protection

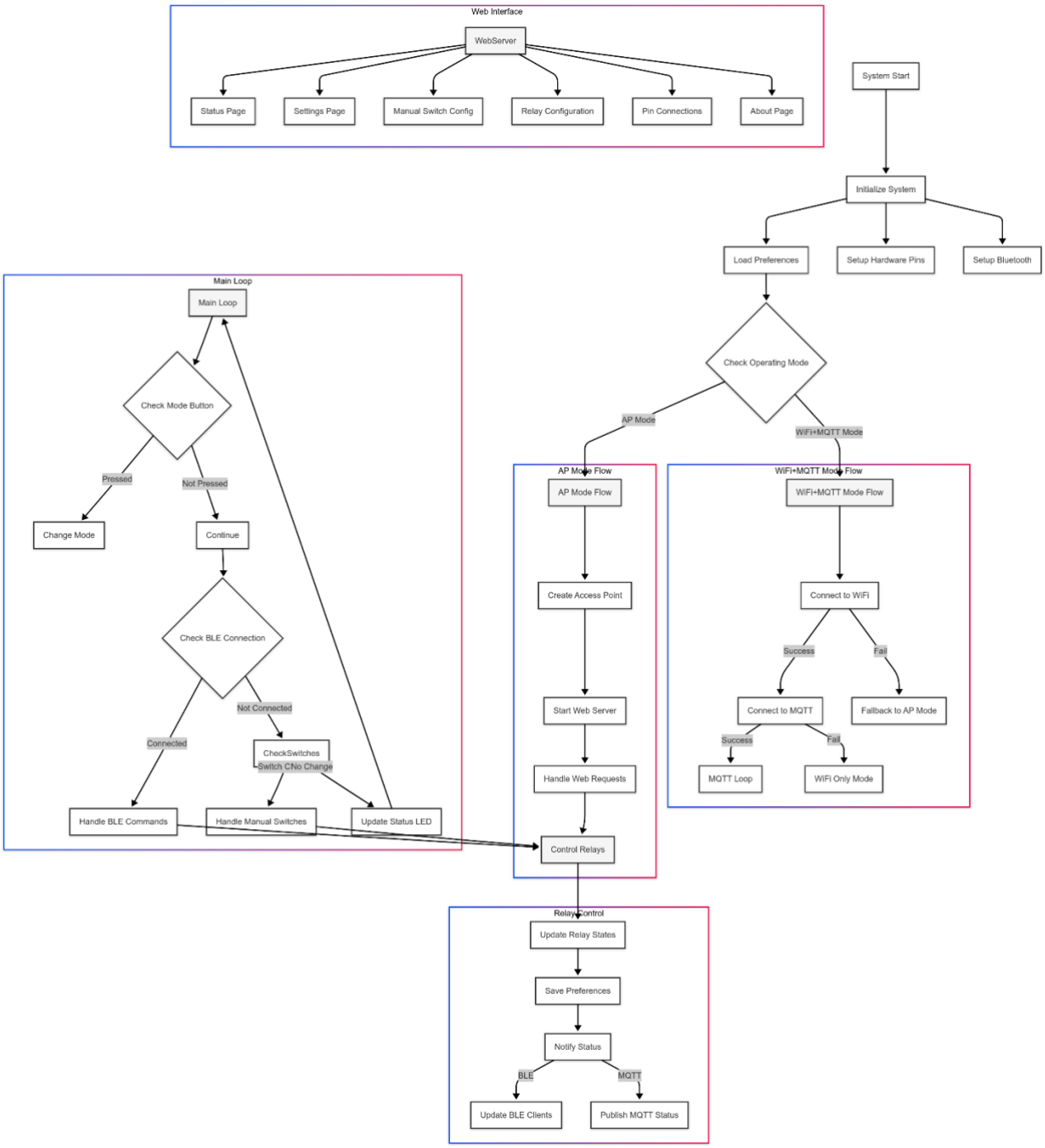
Troubleshooting

Common Issues

1. **Wi-Fi Connection Problems**
    a. Check credentials
    b. Verify network availability
    c. Restart system
2. **Relay Not Responding**
    a. Check power supply
    b. Verify GPIO connections
    c. Test manual switch
3. **Mode Switching Issues**
    a. Hold button for >3 seconds
    b. Check status LED
    c. Verify preferences

Status LED Patterns

- **Fast Blink**: AP Mode
- **Solid On**: Connected With MQTT
- **Off**: Power Off



FLOW DIAGRAM OF WORKING OF WHOLE PROJECT

OVERVIEW OF WEB PAGE

As we have already mentioned, we can control the relays in three modes: **BLE**, **MQTT**, and **AP**. In this section, we will look at the overview of the **web page used in AP mode**.

Accessing the Web Interface

1. Connect to the Smart Board's WiFi network:
   - SSID: `SmartBoard_AP`
   - Password: `12345678`
2. Open a web browser
3. Enter the default IP address: `192.168.4.1`

Main Dashboard

URL: /

Features

- Real-time relay status display
- Individual relay control buttons
- Status indicators for each relay
- Quick access navigation menu
- System status overview

Layout

```
<div class="container">
  <div class="header">
    <h1>SmartBoard Control</h1>
  </div>
  <div class="nav">
    <a href="/">Status</a>
    <a href="/settings">Settings</a>
    <a href="/manual-switches">Manual Switches</a>
    <a href="/relay-config">Relay Config</a>
```

```
    <a href="/pin-connections">Pin Connections</a>
    <a href="/about">About</a>
  </div>
  <div class="relay-grid">
    <!-- Relay Control Cards -->
  </div>
</div>
```

Settings Page

URL: `/settings`

Sections

1. **WiFi Settings**
    a. SSID configuration
    b. Password management
    c. Connection preferences
2. **MQTT Settings**
    a. Server address
    b. Port configuration
    c. Username/Password
    d. Topic settings
3. **System Settings**
    a. Device name
    b. AP mode password
    c. System preferences

Features

- Form-based configuration
- Real-time validation
- Save/Reset options
- Connection testing

Manual Switches Page

URL: `/manual-switches`

Configuration Options

1. **Switch 1 Settings**
    a. Relay assignment
    b. Active state (HIGH/LOW)
    c. Debounce settings
2. **Switch 2 Settings**
    a. Relay assignment
    b. Active state (HIGH/LOW)
    c. Debounce settings

Features

- Dropdown selection for relay assignment
- Active state configuration
- Save/Reset functionality
- Status display

Relay Configuration Page

URL: `/relay-config`

Configuration Options

1. **Relay 1 Settings**
    a. Active state (HIGH/LOW)
    b. Pin configuration
    c. Initial state
2. **Relay 2 Settings**
    a. Active state (HIGH/LOW)
    b. Pin configuration
    c. Initial state
3. **System Pins**
    a. Mode button pin selection
    b. Status LED pin selection

Features

- Pin mapping visualization
- Configuration validation
- Save/Reset options

- Status display

Pin Connections Page

URL: `/pin-connections`

Display Sections

1. **Relay Connections**
    a. Relay 1: GPIO 16
    b. Relay 2: GPIO 17
2. **Manual Switch Connections**
    a. Switch 1: GPIO 26
    b. Switch 2: GPIO 27
3. **System Connections**
    a. Mode Button: GPIO 4
    b. Status LED: GPIO 2

Features

- Visual pin layout
- Connection status
- Pin function display
- Configuration reference

About Page

URL: `/about`

Sections

1. **Features**
    a. 2 Relay Control
    b. Manual Switch Support
    c. Web Interface
    d. MQTT Integration
    e. Bluetooth Control
    f. Configurable Active States
    g. Persistent Settings
2. **Technical Details**

    a. ESP32 Based
    b. WiFi and Bluetooth Support
    c. Web Server for Remote Control
    d. MQTT for IoT Integration
    e. Manual Switch Support
3. **Development Team**
    a. Project Lead: **MOHIT**
    b. Team Members:
        i. ATUL
        ii. ABHAY
        iii. SAMIKSHA
        iv. NIKHIL
        v. AKHILESH

Features

- Team information
- Project details
- Technical specifications
- Contact information

Common Features Across All Pages

Navigation

- Consistent top navigation bar
- Quick access to all sections
- Breadcrumb navigation
- Status indicators

Styling

- Responsive design
- Mobile-friendly layout
- Dark/Light theme support
- Modern UI elements

User Experience

- Real-time updates
- Interactive controls

- Status notifications

Smart App Documentation

Overview and Architecture

Overview

The **Smart App** is a Flutter-based mobile application developed to control and monitor smart devices using both **Bluetooth Low Energy (BLE)** and **Message Queuing Telemetry Transport (MQTT)** protocols. It delivers a unified and intuitive interface for managing smart relays and other IoT-enabled devices.

Core Features

- **Dual-protocol support**: Seamless communication with devices via BLE and MQTT.
- **Real-time status monitoring**: Instant updates on device states and connectivity.
- **Remote device control**: Operate smart relays and connected devices from anywhere.
- **Secure connection management**: Ensures reliable and encrypted communication.
- **Persistent settings storage**: Saves user preferences and device settings locally.

Technical Architecture

- **Frontend**: Developed using Flutter and Dart.
- **State Management**: Utilizes the Provider pattern for efficient state handling.
- **Data Storage**: Implements `SharedPreferences` for local persistence.
- **Communication Protocols**: Supports both BLE and MQTT for device communication.
- **UI Framework**: Based on Material Design principles for a consistent and modern user interface.

Modular Design

The application architecture promotes clean separation of concerns and maintainability through the following layers:

- **Services Layer**: Handles BLE and MQTT protocol communications.

- **Providers**: Manage application and device state across the app.
- **Screens**: Comprise the user interface and interactions.
- **Models**: Define the structure for data objects used throughout the app.

BLE Implementation

Overview

The **Bluetooth Low Energy (BLE)** implementation enables direct, low-latency control and monitoring of smart devices over local connections. It is designed to ensure reliable communication, efficient reconnection, and real-time status tracking.

BLE Service: `ble_service.dart`

This core service handles all BLE-related operations in the app, including:

- **Device Discovery**
  - Scans for nearby BLE-enabled devices
  - Filters results using predefined service UUIDs
  - Extracts and displays key device information
- **Connection Management**
  - Establishes and maintains secure BLE connections
  - Monitors connection state and notifies the app of changes
  - Automatically attempts reconnection if a connection is lost
- **Data Communication**
  - Performs read and write operations on BLE characteristics
  - Sends control commands to connected devices
  - Implements periodic status polling for consistent updates
- **Status Monitoring**
  - Provides real-time updates on relay states and device status
  - Displays connection indicators in the UI
  - Handles connection failures and includes recovery mechanisms

Key Features Summary

Feature	Description
Device Discovery	Scans and lists BLE devices based on specific service UUIDs
Secure Connection	Establishes and maintains BLE sessions with auto-reconnect capabilities
Data Handling	Reads from and writes to device characteristics, including control commands
Real-time Monitoring	Tracks device state changes and reports errors or disconnections

---

MQTT Implementation

Overview

The **MQTT** implementation enables cloud-based control and monitoring of smart devices, facilitating remote interaction through lightweight, real-time messaging. It is ideal for managing devices outside of local BLE range, providing persistent connectivity via an MQTT broker.

MQTT Service: `mqtt_service.dart`

This service encapsulates all MQTT communication logic, including:

- **Connection Management**
  - Establishes secure connections with MQTT brokers using authentication credentials
  - Monitors the connection state and triggers appropriate callbacks
  - Implements automatic reconnection strategies to maintain persistent connectivity
- **Message Handling**
  - Parses JSON-formatted messages received from subscribed topics

    - Triggers real-time state updates across the app
    - Includes robust error detection and fallback mechanisms
- **Topic Management**
    - Dynamically subscribes to topics based on device ID or type
    - Filters messages to ensure relevance to the connected device
    - Supports configuration of Quality of Service (QoS) levels for message delivery guarantees
- **State Synchronization**
    - Maintains relay state consistency between the cloud and app
    - Updates the UI in response to MQTT events
    - Broadcasts device status for multi-client visibility

Key Features Summary

Feature	Description
Secure Connectivity	Authenticated connections with MQTT brokers, with auto-reconnect logic
Message Handling	Processes JSON messages for real-time device control and feedback
Topic Management	Dynamically manages topic subscriptions with QoS control and message filtering
State Sync	Synchronizes cloud-reported device states with app UI and logic

---

User Interface and Settings

Overview

The Smart App features a clean, intuitive user interface designed with **Material Design** principles. It offers streamlined navigation and flexible settings management to accommodate both local (BLE) and remote (MQTT) device control.

Main Screens

1. BLE Dashboard

Enables users to discover, connect to, and control local devices via Bluetooth.

- **Device Scanning Interface**

  Displays a list of nearby BLE devices with filtering by service UUID.
- **Connection Status Display**

  Real-time feedback on the connection state with each discovered device.
- **Relay Control Panel**

  Toggle switches or buttons for interacting with relays and smart devices.
- **Status Indicators**

  Visual cues for device activity, relay states, and connection health.

2. MQTT Dashboard

Facilitates cloud-based control and monitoring of IoT devices via MQTT.

- **Broker Connection Form**

  Allows entry of broker URL, port, username, and password.
- **Topic Management**

  Input fields for subscribing to and publishing on specific MQTT topics.
- **Relay Control Interface**

  Controls similar to the BLE dashboard, adapted for MQTT messaging.
- **Connection Status**

  Visual indicators for broker connectivity and message flow.

3. Settings Screen

Central hub for managing app preferences and connection configurations.

- **Protocol Selection**

  Choose between BLE and MQTT as the primary communication method.
- **Connection Parameters**

  Configure BLE UUIDs or MQTT broker details (host, port, credentials).

- **Authentication Details**
  Securely store login or token-based access credentials for MQTT brokers.
- **Persistent Storage**
  Saves all configuration settings locally using `SharedPreferences`, ensuring they persist across app restarts.

Settings Management

Overview

The **Settings Management** module provides flexible control over application behavior, connection preferences, and user interface personalization. All settings are stored persistently using `SharedPreferences`, ensuring consistency across sessions.

Categories of Settings

1. Connection Settings

Used to configure and maintain communication with BLE devices or MQTT brokers.

- **Server Addresses**
  - MQTT broker URLs or IP addresses
  - BLE service UUIDs for device filtering
- **Port Configurations**
  - MQTT port numbers (default and custom)
- **Authentication Credentials**
  - Secure storage of usernames, passwords, and tokens for MQTT
- **Topic Names**
  - User-defined MQTT topics for publishing and subscribing

2. Device Settings

Define how the app interacts with connected devices.

- **Relay Configurations**

    - Custom naming and control setup for individual relays
- **Control Parameters**
    - Toggle settings like default states, debounce times, or multi-relay sequences
- **Status Preferences**
    - Options for how status updates are displayed (e.g., icons, colors, logs)
- **Notification Settings**
    - Enable or disable push/local notifications for specific device events

3. App Preferences

User interface and experience customization.

- **Theme Selection**
    - Light, dark, or system-default mode
- **Language Options**
    - Localization settings for multi-language support
- **Notification Preferences**
    - General toggle for app notifications and behavior (sound, vibration, banners)
- **Data Retention**
    - Options to manage cached data, logs, or auto-reset intervals

---

Security Features

Overview

Security is a core component of the Smart App's design. Whether communicating with local BLE devices or cloud-connected MQTT services, the app ensures that all user data, credentials, and communication channels are protected through modern security best practices.

Authentication

- **Secure Credential Storage**

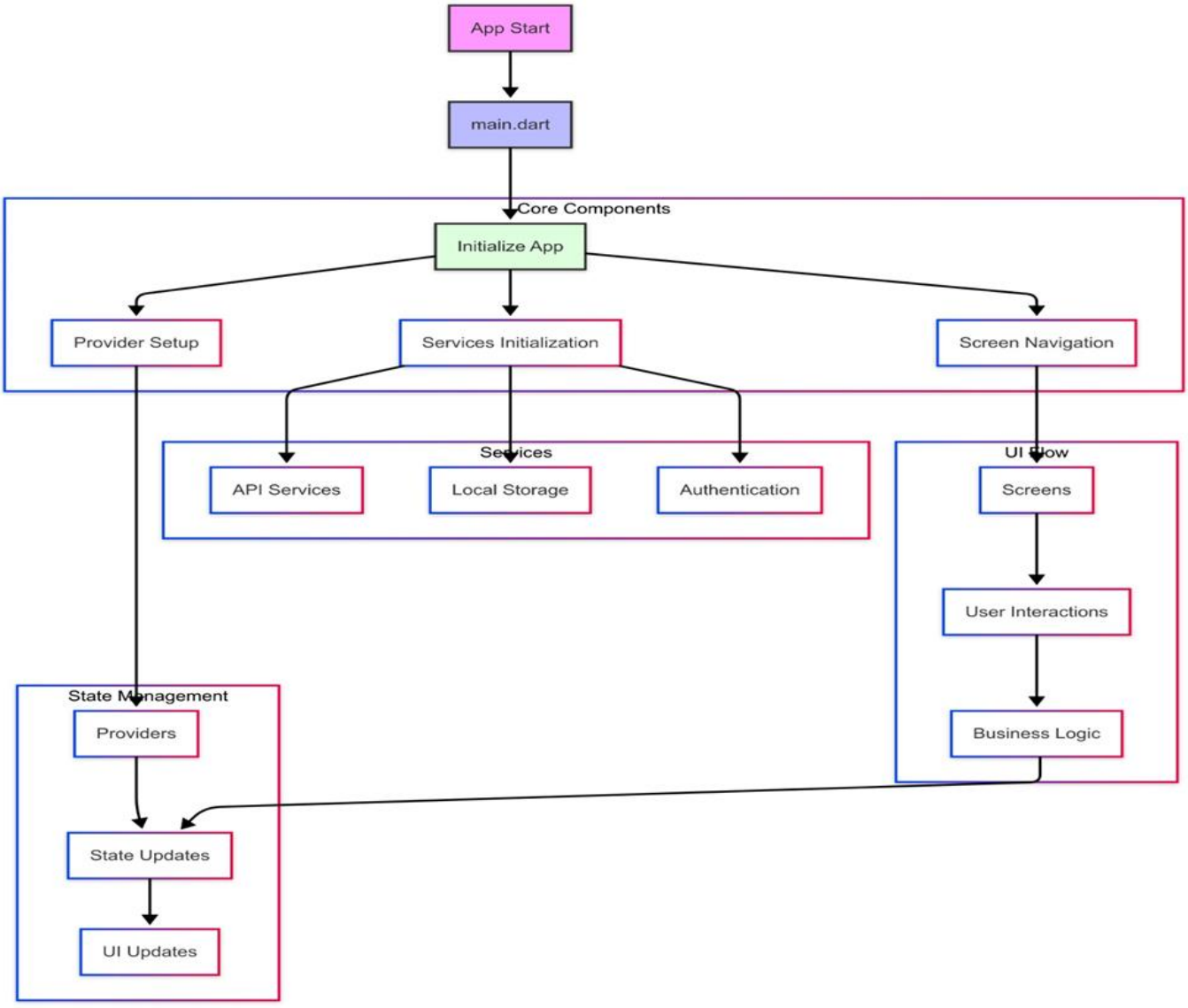
    - User credentials (e.g., MQTT usernames and passwords) are stored using secure, encrypted mechanisms compatible with platform-specific best practices.
- **Encrypted Communication**
    - MQTT connections support TLS/SSL to ensure encrypted data transmission between the app and broker.
    - BLE communication is handled securely with authentication and, where supported, encryption at the protocol level.
- **Session Management**
    - Maintains authenticated sessions while automatically refreshing or invalidating credentials as needed.
    - Prevents unauthorized reconnection attempts.

Data Protection

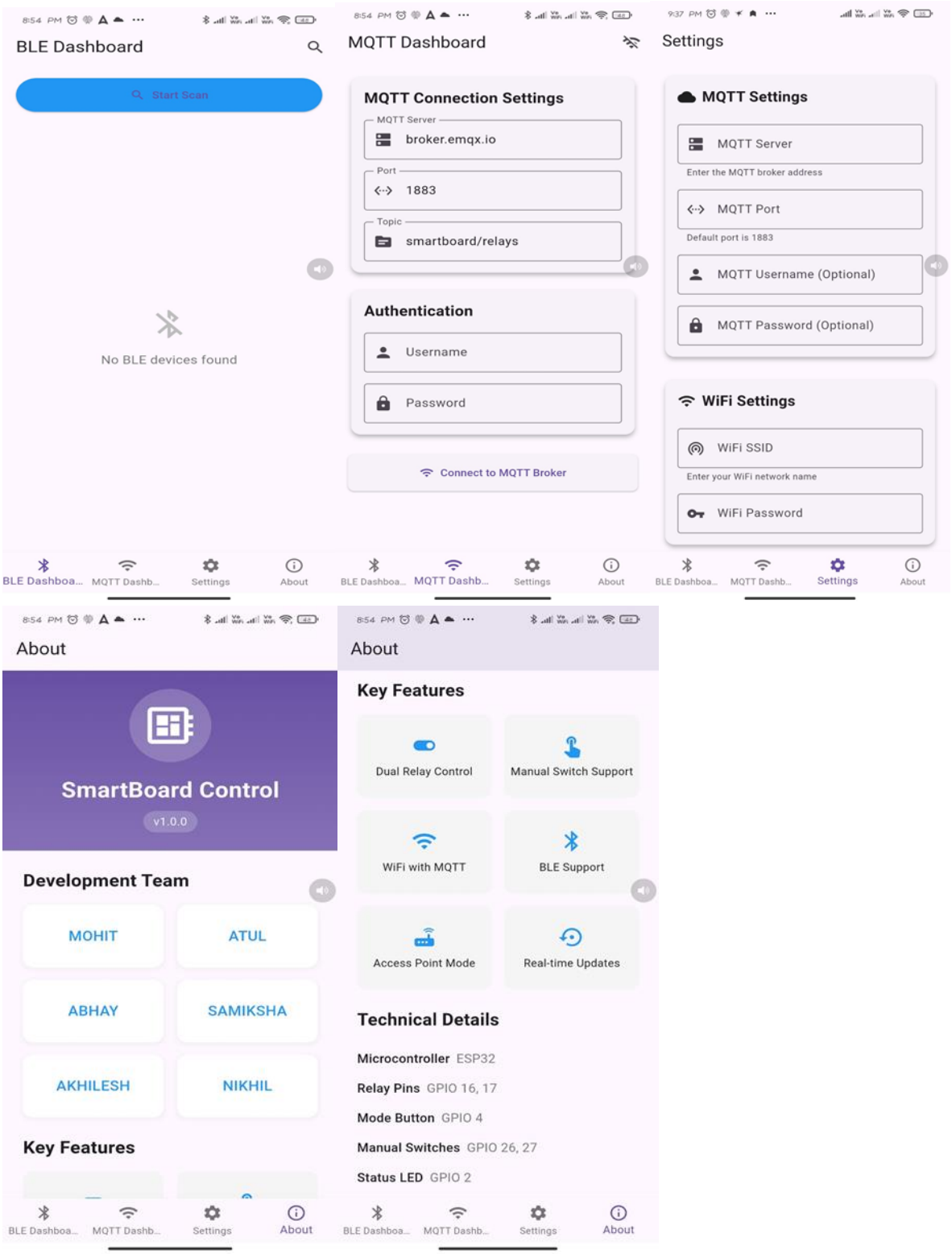
- **Secure Storage**
    - Sensitive data such as authentication tokens, relay configurations, and preferences are stored using `SharedPreferences` or platform-specific secure storage (e.g., iOS Keychain, Android Keystore when applicable).
- **Data Encryption**
    - Local data and credentials can be encrypted to prevent unauthorized access on rooted/jailbroken devices.
- **Access Control**
    - Role-based or feature-based access restrictions can be enforced within the UI to ensure only authorized users can modify critical settings.

Final Notes

The Smart App delivers a **robust, secure, and user-friendly interface** for managing smart devices through both local (BLE) and cloud-based (MQTT) connections. Its **modular architecture** not only promotes easy scalability and maintenance but also supports a **comprehensive and reliable feature set** for real-time control and monitoring.



Flow Diagram of Working Model of app



Different Interface of app