TED UNIVERSITY

CMPE 492

Low Level Design Report v2

- **Dilara Yargıcı** - 20290006896
- **Berna Tanrıverdi** - 63295214002
- **Aslı Dölek** - 67915217862
- **Yaşar Mert Dirican** – 13630073160

1. Introduction

Purpose and Scope

The purpose of this Low-Level Design (LLD) document is to provide a comprehensive technical blueprint for the **PriceLink** system. While the High-Level Design (HLD) established the three-tier IoT architecture, this document defines the granular internal logic, database schemas, and inter-component communication protocols. PriceLink aims to eliminate paper waste and pricing inconsistencies in retail through a synchronized E-Ink labeling ecosystem. This LLD serves as the primary reference for the implementation phase, ensuring that the integration between the Firebase Cloud, IoT Gateway, and Smart Tags is robust, scalable, and power-efficient.

Design Goals

The system is engineered based on three core principles:

- **Persistent Reliability:** Ensuring that shelf prices remain visible even during total power or network failures through bistable display technology and offline-first database synchronization .
- **Operational Transparency:** Providing administrators with a real-time feedback loop (ACK signals) to verify that a digital price change has been physically updated on the shelf.
- **Power Optimization:** Utilizing asynchronous event-driven communication to maintain a multi-year battery life for end-node devices.

1.1 Object Design Trade-offs

In the transition from architectural concepts to detailed object design, the following engineering trade-offs were made:

- **1.1.1 Eventual Consistency vs. Strong Consistency:** PriceLink adopts an Eventual Consistency model. Given the "Deep Sleep" nature of IoT tags, forcing strong consistency would result in unacceptable UI latency and system timeouts.
  Instead, the database utilizes a "Pending" state pattern; a price is committed to the cloud immediately, and the physical state is updated asynchronously when the tag polls the gateway.

- **1.1.2 Document-Oriented (NoSQL) vs. Relational (SQL):** Google Firebase Firestore was selected over traditional RDBMS. While SQL offers superior join operations, Firestore's NoSQL document model provides native real-time listeners and seamless horizontal scaling, critical for pushing updates to thousands of tags simultaneously .
- **1.1.3 Edge Intelligence vs. Cloud Dependency:** A Hybrid Logic approach was implemented. Complete cloud dependency would render the store non-functional during ISP outages. By utilizing the Gateway as a local cache (Edge Layer), the store can continue pairing operations and status monitoring offline, syncing back via the Firebase SDK once the connection is restored.

1.2 Interface Documentation Guidelines

PriceLink adopts a hybrid real-time communication architecture that combines MQTT and WebSocket protocols to ensure efficient, scalable, and low-latency data exchange. Unlike traditional REST-based architectures, the system follows an event-driven communication model, which eliminates continuous polling and reduces unnecessary network overhead.

The communication layer supports three primary interaction paths:

1. Backend ↔ Gateway communication
2. Gateway ↔ Smart Tag communication
3. Backend ↔ Frontend (Dashboard) synchronization

1.2.1 Communication Protocol Selection

Protocol	Role & Implementation	Key Features
MQTT	Backend-to-Gateway / Gateway-to-Tag	Publish/subscribe model; designed for low-power IoT devices .
WebSocket	Backend-to-Frontend Dashboard	Persistent, bidirectional communication for real-time updates and logs .

1.2.2 Message Format Guidelines

All communication messages are structured in JSON format to ensure consistency and interoperability.

Price Update Message Example:

JSON


```
{
 "messageType": "PRICE_UPDATE",
 "productId": "PRD-1042",
 "tagId": "TAG-0098",
 "newPrice": 799.90,
 "currency": "TRY",
 "timestamp": "2026-03-17T15:30:00Z",
 "campaignId": "CMP-2026-04"
}
```


Tag Acknowledgment Message Example:

JSON


```
{
 "messageType": "ACK",
 "tagId": "TAG-0098",
 "productId": "PRD-1042",
 "status": "UPDATE_RECEIVED",
 "batteryLevel": 81,
 "signalStrength": -67,
 "timestamp": "2026-03-17T15:30:02Z"
}
```

Error Reporting Message Example:

JSON


```
{
"messageType": "ERROR_REPORT",
"tagId": "TAG-0098",
 "errorCode": "E_INK_REFRESH_FAIL",
"details": "Hardware timeout during partial refresh",
"batteryLevel": 14,
 "timestamp": "2026-03-27T22:45:00Z"
 }
```


Bulk Campaign Update Example:

JSON


```
{ "messageType": "BULK_UPDATE",
 "campaignId": "SPRING_SALE_2026",
"targetCategory": "Electronics",
"discountPercentage": 15.0,
"executionTime": "2026-04-01T08:00:00Z"
}
```

1.2.3 Topic and Channel Structure

Hierarchical topic structure for MQTT-based communication:

- pricelink/store/{storeId}/tag/{tagId}/price-update
- pricelink/store/{storeId}/tag/{tagId}/ack
- pricelink/store/{storeId}/gateway/{gatewayId}/status
- pricelink/store/{storeId}/campaign/{campaignId}/bulk-update

1.2.4 Delivery and Reliability Rules

- Price update messages require high delivery reliability.
- Each update must be followed by an acknowledgment (**ACK**).
- **Timeout Handling:** If no ACK is received, the system retries the update and marks tag status as **Pending** or **Disconnected** .
- **QoS Levels:** Lower QoS for non-critical status updates; Higher QoS for critical price updates and ACKs .

1.2.5 Error Handling and Retry Mechanism

- Failed updates are automatically retried and recorded in **TransactionLogs** .
- If the connection is lost, the gateway buffers messages locally and retransmits upon restoration .
- Existing prices remain valid until successful update confirmation.

1.2.6 Security Guidelines

- Channels are encrypted using **TLS / WSS**.
- **RBAC:** Role-Based Access Control is enforced.
- Authorized services only; authentication via secure tokens.
- Critical operations are logged for auditing purposes.

1.2.7 Backend Communication Flow

A price update triggered from the management panel is processed by the backend and published via MQTT. The gateway receives the message and forwards it to the smart tag. After the display update, the tag sends an ACK back through the gateway to the backend . The backend then notifies the frontend dashboard via WebSocket to reflect the real-time status .

1.3 Engineering Standards

The low-level communication and backend design follows established engineering standards:

- **1.3.1 UML and Software Design Standards:** Documented using Package, Class, and Sequence Diagrams following the **IEEE 1016 Software Design Description** standard .
- **1.3.2 MQTT Communication Standard:** Adheres to **MQTT version 5.0** for lightweight, scalable publish/subscribe architecture .
- **1.3.3 WebSocket Communication Standard:** Follows the **RFC 6455** standard for persistent, full-duplex communication over TCP .
- **1.3.4 Security and Ethical Standards:** Adheres to **OWASP** best practices and aligns with the **ACM Code of Ethics** .

2. Packages

2.1 Embedded Libraries (Hardware Layer)

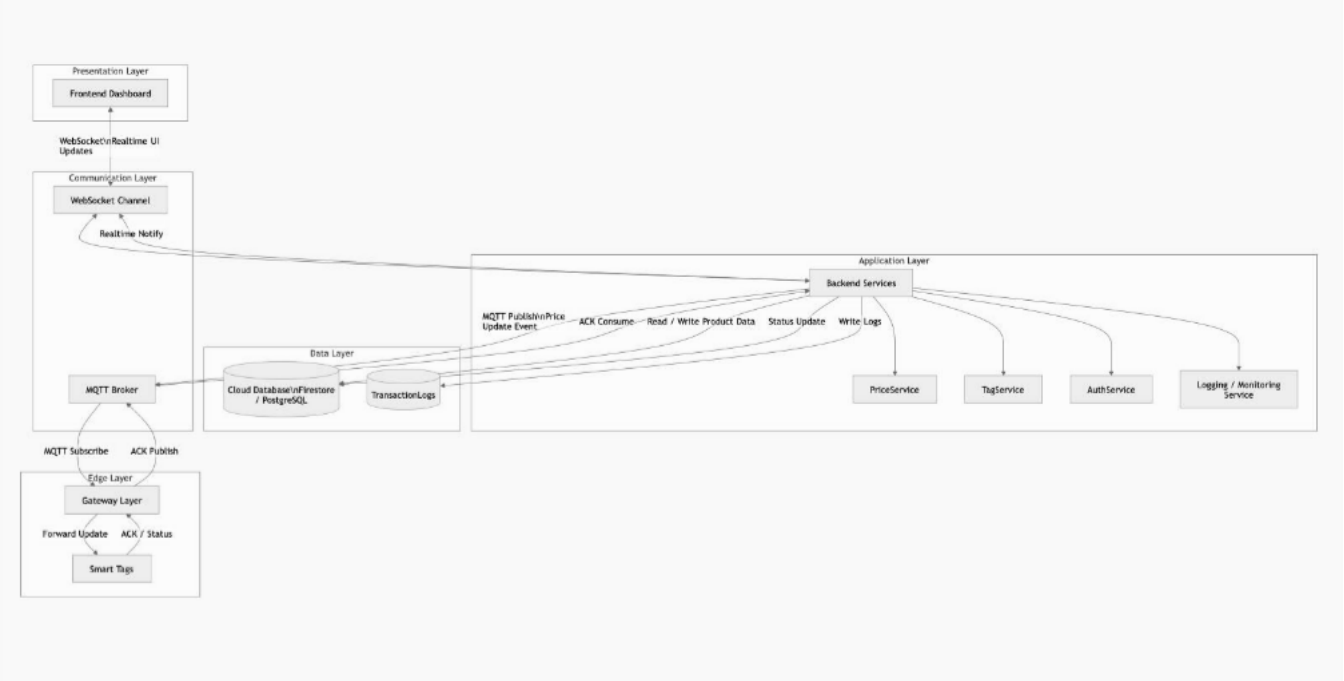
The ESP32 platform uses the following libraries for hardware abstraction and data processing :

Package / Library	Purpose	Key Functionality
GxEPD2	E-ink Display Driver	Low-level control for full and partial refresh cycles to reduce power.
WiFi.h	Network Connectivity	Manages the ESP32 built-in Wi-Fi radio for gateway connection.
HTTPClient	Cloud Communication	Request/response cycle for status reports and updates.
ArduinoJson	Data Serialization	Parsing tag status and price update payloads.

2.2 System Architecture Overview

PriceLink uses a distributed architecture where MQTT handles device-level messaging and WebSocket provides real-time dashboard synchronization .

Figure 2.1 System Architecture (Package Diagram)



3. Class Interfaces

3.1 Hardware Layer (Component 1 - Smart Tag Logic)

The internal logic comprises the TagController and DisplayManager classes.

3.1.1 TagController Class

Central coordinator for network synchronization and power-saving.

Attributes:

- **tagID** (private, String): The device's MAC address/unique identity.
- **batteryStatus** (private, int): Percentage of battery life (0–100).
- **isAwake** (private, boolean): Flag indicating readiness for deep sleep vs. active state.

Method Name	Return Type	Parameters	Description
connectToNetwork()	boolean	String ssid, String pass	Initializes the Wi-Fi module and establishes a secure connection to the local IoT gateway.
deepSleepMode()	void	int duration	Deactivates the CPU and non-essential peripherals to conserve energy, keeping only the RTC active for timed wake-up.
checkPendingUpdates()	void	None	Polls the gateway or Firestore listener to check for the is_pending flag regarding price updates.
reportStatus()	void	None	Transmits a heartbeat signal containing real-time battery level and signal strength (RSSI) telemetry.

3.1.2 DisplayManager Class

Abstracts e-ink complexity for energy-efficient visual updates.

Method Name	Return Type	Parameters	Description
updateEInkDisplay()	void	double price	Initiates the physical refresh cycle of the E-ink panel to display the updated product price.
renderStaticImage()	void	None	Leverages bistable technology to ensure the current image remains visible indefinitely without consuming power.
clearGhosting()	void	None	Executes a full-white refresh cycle to eliminate residual image artifacts and maintain high display contrast.
showLowBatteryIcon()	void	None	Renders a visual warning indicator on the screen when the device's battery level falls below the 20% threshold.

3.1.3 Hardware Logic - State Machine Flow

The Smart Tag operates on a cyclical Event-Driven State Machine to achieve a 3-4 year battery life:

1. **Wake-Up:** RTC triggers wake-up from deep sleep.
2. **Synchronization:** Connects to Wi-Fi and checks is_pending flag in Firestore.
3. **Update (Conditional):** If update is pending, refresh screen; otherwise, proceed.
4. **Reporting:** Sends **ACK** signal and battery telemetry to the Cloud.
5. **Sleep:** Calls deepSleepMode() to minimize power draw until the next cycle.

3.2 Data Layer (Component 2 - Persistent Management)

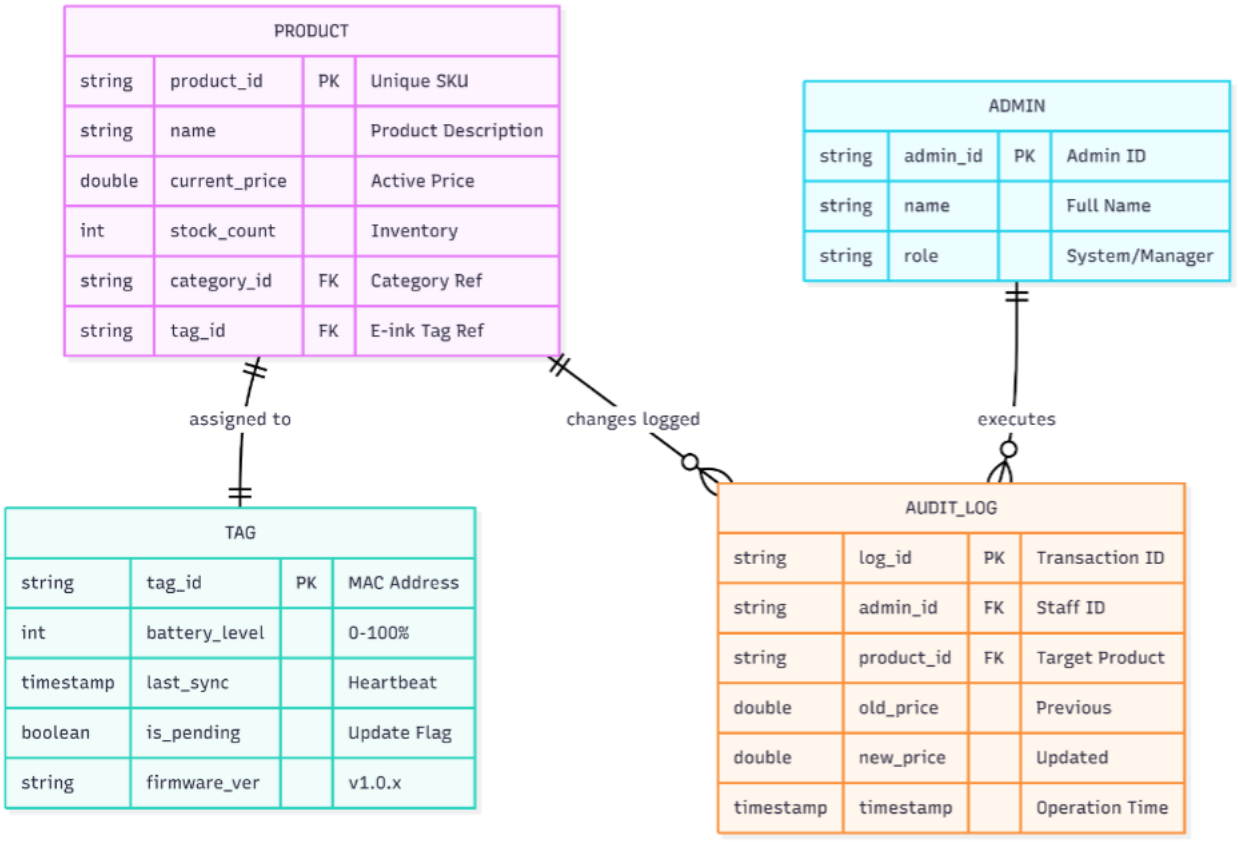
Handles structural data models and Firestore abstraction for seamless data flow.

3.2.1 Data Models (Attributes)

Collection	Attribute	Description	Data Type
Products	product_id	Document ID / Unique SKU	String
	name	Human-readable product name	String
	current_price	Active price to be displayed	double
	stock_count	Real-time inventory level	int
	category_id	Reference to Product Category	String
	tag_id	Reference to the physical E-ink device	String
Tags	tag_id	Hardware MAC address / Unique ID	String
	battery_level	Current energy percentage (0-100)	int
	is_pending	Boolean flag for un-synced price changes	boolean
	last_sync	Timestamp of the last heartbeat	Timestamp
	firmware_ver	Current software version (ESP32)	String
Audit Logs	log_id	Unique ID for the transaction	String
	admin_id	Reference to the staff member	String
	product_id	Reference to the affected product	String
	old_price	Price before modification	double
	new_price	Price after modification	double
	timestamp	Exact time of the change	Timestamp

3.2.2 Logical Data Schema (ER Diagram)

Illustrates logical connections (1:1 and 1:N) critical for referential integrity and audit trails .



3.2.3 Database Service Class (FirebaseConnector)

Standardized interface for communication and logic modules.

Method Name	Return Type	Parameters	Description
fetchProductData()	ProductObj	String productId	Retrieves details for UI rendering.
setPendingFlag()	void	String tagId, bool state	Toggles is_pending for refresh cycle.
recordPriceLog()	boolean	LogObj logData	Writes entry into Audit_Log upon price change.
updateTagTelemetry()	void	String tagId, int batt	Updates battery/sync from heartbeats.

3.3 Application Layer (Component 3 - Backend & Communication Flow)

Ensures efficient device communication and real-time user feedback.

3.3.1 Message Format Guidelines

JSON-structured for interoperability.

- **Price Update Message:** Contains productId, tagId, newPrice, and campaignId.
- **Tag Acknowledgment Message:** Contains tagId, status (UPDATE_RECEIVED), and batteryLevel.

3.3.2 Logic Rules and Relationships

The backend enforces specific cardinality rules:

- **Product to Tag (1:1):** One product is linked to exactly one smart tag at a time.
- **Product to Audit_Log (1:N):** One product can generate many historical price records.
- **Admin to Audit_Log (1:N):** One administrator can perform many updates for accountability.

3.3.3 Sequence Diagram Flow

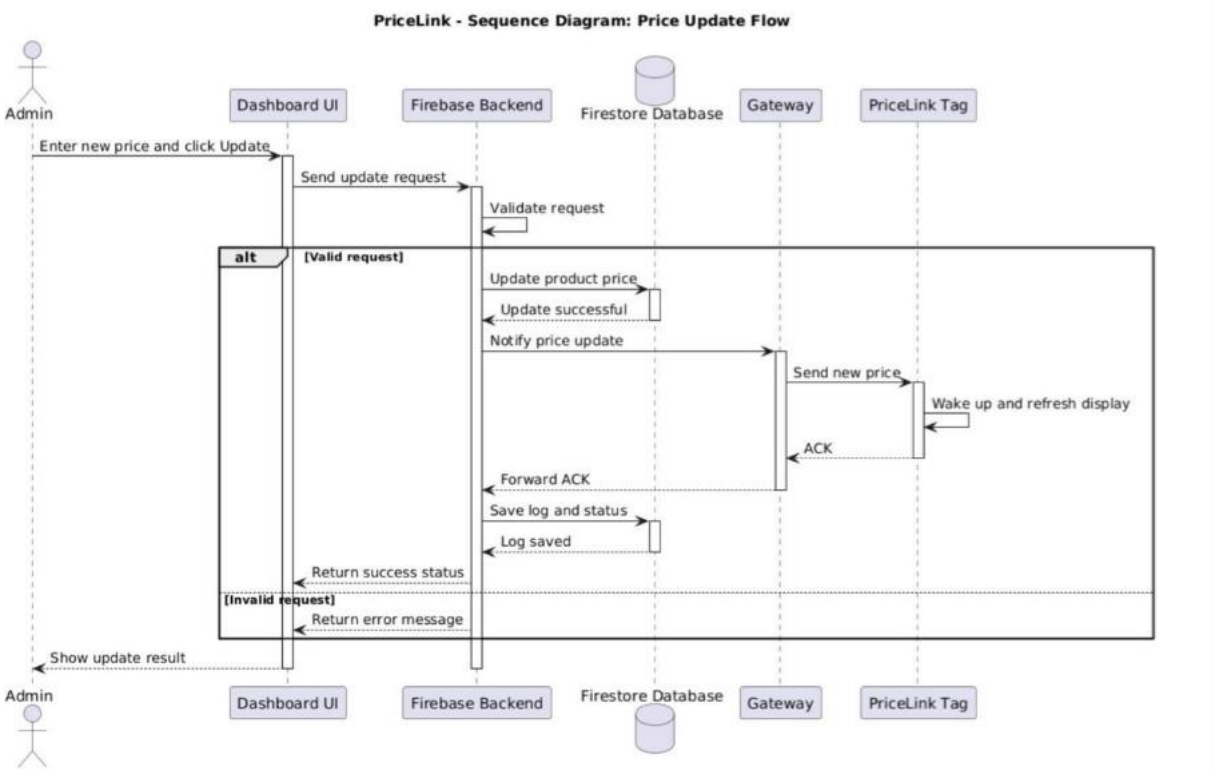
This sequence diagram represents the complete flow of a price update operation initiated from the PriceLink Management Panel and propagated to the physical E-Ink tags through the system architecture. The process begins when the administrator updates a product price using the web-based dashboard interface. The frontend sends the updated price request to the Firebase backend via a secure API call. The backend validates the request, updates the Firestore database, and triggers real-time listeners.

Once the database is updated, the Gateway device receives the change through a persistent connection (WebSocket/MQTT-like communication) and translates this update into a signal. The signal is then transmitted to the corresponding PriceLink Tag. The tag wakes up from sleep mode, refreshes its E-Ink display with the new price, and sends an acknowledgment

(ACK) back through the Gateway to the backend. Finally, the frontend dashboard reflects the updated status (e.g., “Update Received”). This sequence ensures real-time synchronization between the digital system and physical store environment while maintaining system reliability and feedback mechanisms.

Step-by-Step Sequence Flow:

1. **Admin → Dashboard:** enters new price
2. **Dashboard → Backend:** sends update request (API call)
3. **Backend → Firestore:** validates & updates product price
4. **Firestore → Gateway:** triggers real-time update event
5. **Gateway → Tag:** sends wireless update signal
6. **Tag → Display:** refreshes E-Ink screen
7. **Tag → Gateway:** sends ACK
8. **Gateway → Backend:** forwards ACK
9. **Backend → Dashboard:** updates status (Success)



3.4 Frontend Components

The **DashboardUI** represents the frontend interface layer of the PriceLink system. It allows administrators to interact with the system by performing operations such as product management, price updates, tag monitoring, and campaign execution. It communicates with the backend through API calls and displays real-time data using Firebase listeners.

3.4.1 Main Classes and Methods

1. **AuthUI:** Manages secure user authentication (login.html) and session control.
    a. Methods: login(), logout(), validateCredentials(), createSession().
2. **DashboardUI:** Main overview screen (dashboard.html) responsible for system status and summary cards.
    a. Methods: renderDashboard(), fetchSummaryData(), refreshStatus(), showAlerts().
3. **ProductManagementUI:** Handles all product-related operations (products.html).
    a. Methods: listProducts(), addProduct(), editProduct(), deleteProduct(), updatePrice(), searchProduct().
4. **TagMonitoringUI:** Monitors health and status (tag-monitoring.html) of PriceLink tags.
    a. Methods: showTagStatus(), displayBatteryLevel(), showConnectionStatus(), showLowBatteryWarning(), showDisconnectedTags().
5. **CampaignUI:** Manages bulk campaign operations (bulk-campaign-update.html).
    a. Methods: createCampaign(), applyDiscount(), applyFixedPrice(), previewChanges(), confirmCampaign(), retryFailedUpdates().
6. **ProductTagPairingUI:** Manages associations via QR pairing (product-tag-pairing.html).
    a. Methods: scanTagQRCode(), loadCurrentPairing(), selectNewProduct(), compareOldAndNewProduct(), confirmReassignment().
7. **TransactionLogsUI:** Displays and filters activity logs (transaction-logs.html).
    a. Methods: fetchLogs(), filterLogs(), displayLogDetails(), searchByUsername(), searchByActionType().

3.4.2 Dashboard Attributes

Attribute Name	Type	Description
userID	String	Unique ID of the logged-in administrator
username	String	Username of the active user
sessionToken	String	Authentication token used for secure session control
productList	Array	List of products displayed in the UI
selectedProduct	Object	Product currently selected for editing or updates
selectedTag	Object	Tag currently selected in monitoring or pairing
tagStatusList	Array	List of all tag statuses shown in the monitoring screen
batteryLevel	Number	Battery percentage value of a selected tag
connectionStatus	String	Current connection state (Connected / Disconnected)
updateStatus	String	Result of the latest operation (Success, Failed, Pending)
campaignData	Object	Stores parameters (discount type, value, dates)
previewList	Array	Calculated prices shown before campaign confirmation
retryQueue	Array	Stores failed or pending updates for retries
pairingData	Object	Stores current product–tag assignment data
qrCodeValue	String	QR code content scanned during tag pairing
logList	Array	Collection of transaction logs for display
actionType	String	Type of action recorded (update, delete, pairing)
timestamp	DateTime	Time of the performed action

3.4.3 UI Design Principles

- **Real-time feedback:** Instant update status using Firebase listeners.
- **Usability:** Simple and intuitive interface for store staff.
- **Error handling:** Clear error and retry messages.
- **Responsiveness:** Works on tablets and desktops.
- **Scalability:** Supports multi-store operations.

4. Glossary

- **Bistable Display (E-Ink):** Retains an image indefinitely without power .
- **MQTT:** Lightweight, publish-subscribe protocol for Cloud-to-Gateway communication.
- **Deep Sleep:** Low-power mode; CPU off, RTC active for periodic wake-ups.
- **ACK (Acknowledgment):** Digital signal verifying successful screen refresh.
- **RBAC:** Role-Based Access Control

5. References

- **IEEE Std 1016-2009:** Software Design Descriptions.
- **OASIS MQTT version 5.0 Standard:** Organization for the Advancement of Structured Information Standards.
- **IETF RFC 6455:** The WebSocket Protocol.
- **OWASP Security Best Practices & ACM Code of Ethics**.
- **Google Firebase & PostgreSQL Documentation** .