engineering proceedings

MDPI

Proceeding Paper

Design and Implementation of a Mobile Application for IoT-Based Autoclave Management †

Todor Todorov [1,2,*] and Valentin Tonkov [1]

[1] Department of Information Technologies, St. Cyril and St. Methodius University of Veliko Tarnovo, 5000 Veliko Tarnovo, Bulgaria; v.b.tonkov@ts.uni-vt.bg
[2] Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, 1113 Sofia, Bulgaria
* Correspondence: t.todorov@ts.uni-vt.bg
† Presented at the International Conference on Electronics, Engineering Physics and Earth Science (EEPES 2025), Alexandroupolis, Greece, 18–20 June 2025.

Abstract

This paper presents a case study on the integration of embedded IoT hardware with a modern Android application, demonstrated through the development of a compact autoclave system for small-scale food sterilization. The device is controlled by an ESP8266-based module and communicates securely with a Kotlin-based Android app via MQTT using HiveMQ. The app incorporates advanced Android design patterns such as coroutines, LiveData, Navigation UI, and DataStore. Each device is uniquely addressable and fully configurable from the mobile interface. The work highlights Android's role as a powerful interface for managing embedded IoT systems.

Keywords: Internet of Things (IoT); autoclave; MQTT communication; mobile device control; real-time monitoring

check for updates

Academic Editor: Grigor Mihaylov

Published: 28 August 2025

Citation: Todorov, T.; Tonkov, V. Design and Implementation of a Mobile Application for IoT-Based Autoclave Management. *Eng. Proc.* **2025**, *104*, 57. https://doi.org/10.3390/engproc2025104057

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The rapid expansion of Internet of Things (IoT) technologies has enabled the development of affordable, connected devices across a wide range of applications, including industrial automation, environmental monitoring, and smart home systems [1,2]. While the embedded hardware required for such systems is widely available, designing effective user interfaces for configuration and control remains a challenge, particularly in contexts where usability, portability, and reliability are essential.

This paper presents a practical approach for integrating embedded IoT hardware with a modern Android-based user interface. A compact autoclave system is used as a case study to demonstrate the effectiveness of this integration. The aim is to explore how contemporary Android development practices can be leveraged to improve the usability and flexibility of embedded systems in real-world deployments. The proposed solution highlights the potential of combining mobile technologies with IoT to create accessible and scalable control platforms.

2. Motivation and Necessity

Traditional autoclaves designed for small-scale use typically rely on manual controls, offering limited or no feedback mechanisms. This can result in inconsistent sterilization cycles, inefficient energy use, and reduced operational safety. Although IoT technologies have been widely adopted in industrial, environmental, and home automation systems,

their application in autoclave automation remains unexplored. A notable exception is the work of Saragih et al. [3], who proposed a smart autoclave prototype incorporating IoT concepts. However, their implementation lacked features such as secure communication, mobile app integration, and support for managing multiple devices—capabilities that are addressed in the present work. To enable such functionality in a cost-effective and reliable way, a proven IoT hardware platform was chosen. The integration of ESP8266 microcontrollers with the MQTT protocol has been successfully demonstrated in numerous IoT applications, particularly in home automation [4] and related domains [5–7]. These studies validate the feasibility and reliability of using ESP8266 and MQTT for real-time communication, forming the foundation of our approach. A modern user interface is critical for configuration and monitoring in real-world settings. Android devices are widely available, familiar to most users, and offer a flexible platform for building user interfaces to control IoT devices. Despite its popularity, few studies have examined how modern Android development practices—such as coroutines, ViewModel architecture, and LiveData observation—can be effectively applied to IoT scenarios. This project addresses that gap by demonstrating how these tools can be used to create a mobile interface for configuring and monitoring an embedded system.

3. System Design and Implementation

3.1. System Overview

The system is composed of two main components: an autoclave with embedded controller responsible for managing the autoclaving process, and an Android-based mobile application that provides a user-friendly interface for configuration and monitoring. These components communicate through an MQTT protocol [8–10] using a secure, cloud-hosted broker (HiveMQ), enabling real-time data [11] exchange over the Internet (Figure 1). At the hardware level, the controller reads temperature data from a digital sensor, controls the heating process, and manages state transitions throughout the autoclaving cycle. The controller exposes configurable parameters such as temperature and time duration, which are stored persistently and can be modified remotely. The physical autoclave enclosure is custom-fabricated, and each unit is tested with compressed air at 5 atmospheres using an air compressor. On the software side, the Android application is developed in Kotlin using modern development patterns. It uses MQTT to send commands to the controller and receive real-time status updates and configuration. The app employs Android Jetpack components such as ViewModel [12], LiveData [13] and the Navigation Component, along with Kotlin coroutines [14] for asynchronous operations. Device-specific settings, including connection credentials and unique identifiers, are stored using the DataStore API. The development was implemented after researching good practices in software programming processes [15]. Each autoclave unit is uniquely identified and interacts with the app through dedicated MQTT topics. The identifier is derived from the device MAC address. This design ensures that multiple devices can be managed concurrently without cross-interference.

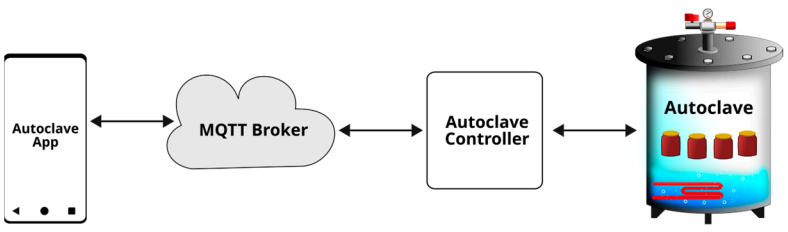


Figure 1. System architecture.

The physical implementation of the system is shown in Figure 2, which includes both the embedded controller and the autoclave enclosure (ET Boyan Tonkov, Karaisen, Bulgaria).



(**a**) (**b**)

Figure 2. Hardware components. (**a**) The embedded controller, mounted in its housing with OLED display and wiring; (**b**) the custom-fabricated autoclave enclosure.

3.2. Embedded Controller Design

The embedded controller is built around an ESP8266 microcontroller (Espressif Systems, Shanghai, China), selected for its built-in Wi-Fi connectivity, and suitability for MQTT communication. It uses the multitasking approach described in article [16].

A digital temperature sensor (Dallas DS18B20, Maxim Integrated, San Jose, CA, USA) is used to measure the internal temperature of the autoclave. The sensor is periodically queried, and its readings are used to determine the current phase of the process. The heater is controlled through a digital output pin, which activates a solid-state relay (SSR) (Fotek Controls Co., Ltd., New Taipei City, Taiwan). Safety and stability are ensured through a simple hysteresis-based control algorithm that maintains the autoclaving temperature within acceptable bounds. The controller operates as a finite state machine with clearly defined stages, including initialization, heating, valve closing, autoclaving, and completion. These states are reported via MQTT messages for remote monitoring. In the absence of an internet connection, the user can still view the current status and the last applied configuration (temperature and time) on a local OLED display (Solomon Systech Limited, Hong Kong, China) mounted on the controller box. However, the display is not interactive—parameter adjustment requires the mobile application. Configuration data such as target temperature, time duration, and Wi-Fi credentials are stored in the internal flash memory using the LittleFS file system. Upon receiving a new configuration via MQTT, the controller updates and persists these values. All communication is secured using TLS to ensure data integrity and prevent unauthorized control.

3.3. Android Application

The Android application serves as the primary interface for configuring, controlling, and monitoring the autoclave device. It is developed in Kotlin and follows modern Android architectural patterns to ensure responsiveness, maintainability, and security. The app communicates with the autoclave controller. The app periodically sends requests to the device and listens for real-time updates, such as temperature readings, process status, and remaining time. To manage background communication and asynchronous operations, the app uses Kotlin coroutines, enabling non-blocking MQTT interactions. The core logic is encapsulated within a ViewModel that holds and exposes relevant data through LiveData, allowing the user interface to automatically reflect current device status. Figure 3 provides

an overview of the application's architecture, highlighting the interplay between Kotlin coroutines, ViewModel, LiveData, and MQTT communication.

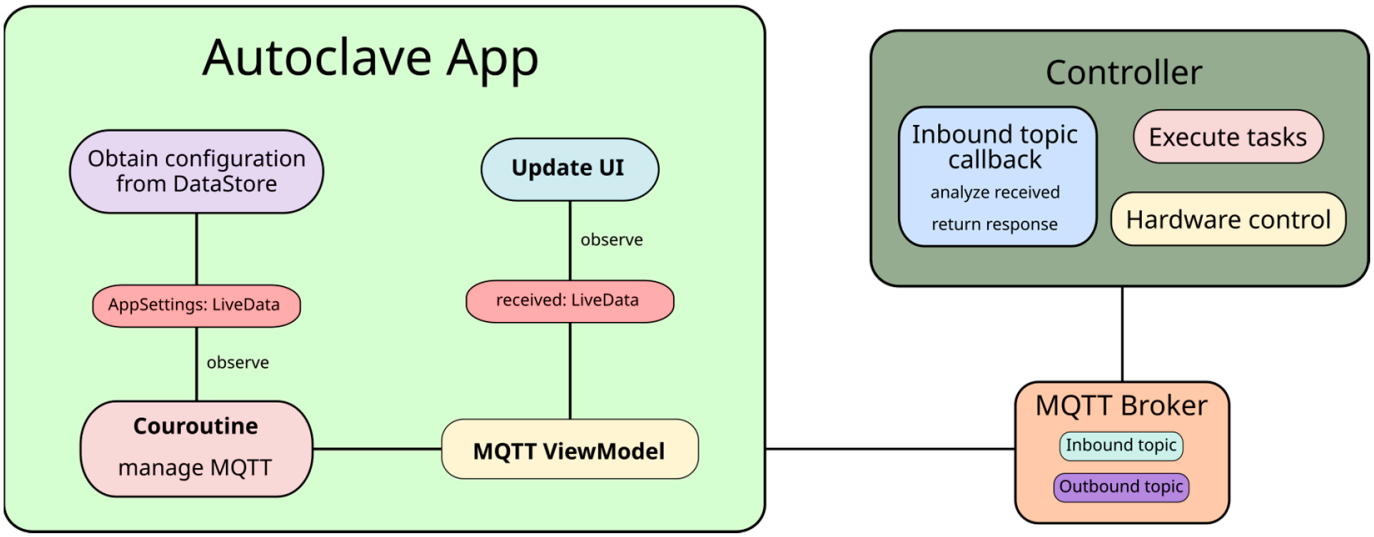


Figure 3. High-level view of communication and architecture components within the autoclave Android app and controller.

Navigation within the app is implemented using Android's Navigation Component, including fragment transitions with Safe Args and a navigation drawer for structured access to settings and status views. Application preferences, including broker credentials and device IDs, are stored persistently using DataStore, a modern alternative to SharedPreferences.

The application supports two languages—Bulgarian and English—which can be selected from the app's settings menu.

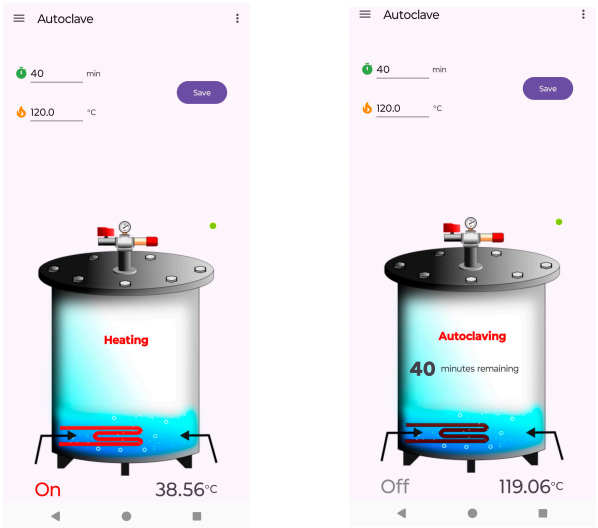
In terms of user experience, the app includes several accessibility features. It adapts its layout, font sizes, and image assets based on the system-wide font scaling settings, and it supports dark mode, automatically adjusting visual elements to maintain clarity and contrast.

The user interface presents key information such as:

- Current temperature and heater status
- Remaining autoclaving time
- Autoclave state (e.g., heating, working, ready)

Configuration parameters (target temperature and time) can be adjusted in the app. When the user presses the Save button, a coroutine is launched within the app's dedicated MQTT ViewModel, which publishes a message containing the updated settings to the controller. If the controller successfully acknowledges the command, the app displays a confirmation via a Toast message, informing the user that the configuration has been applied. The new values are also saved in the device's flash memory for persistent storage.

Screenshots of the main interface is shown in Figure 4, displaying the current temperature, autoclave status, and configuration controls.



(**a**) (**b**)

Figure 4. Main screen of the Android application. (**a**) The device is in the heating phase; (**b**) the autoclaving phase has started, showing the remaining time.

3.4. Communication and Synchronization

The controller subscribes to its designated input topic and listens for JSON-encoded commands from the application. Supported commands include configuration updates (e.g., autoclaving temperature and duration), requests for current settings, and status queries. The controller parses incoming messages using ArduinoJson and responds by publishing a reply to the output topic. The Android app subscribes to the output topic and reacts to incoming messages in real time. Each message is parsed using the JSONObject class, and the user interface is updated via LiveData. Periodic status polling is implemented using Kotlin coroutines, allowing the app to publish a getInfo command every few seconds without blocking the main thread. The response includes the current temperature, autoclave status, remaining time, and heater activity.

To ensure reliability, the app monitors whether a response is received after each command. If a message is not acknowledged within a given time frame, the system marks the device as disconnected. This feedback is reflected visually in the UI, allowing the user to detect connection issues or power loss. The logic of this message acknowledgment mechanism is shown in Figure 5.

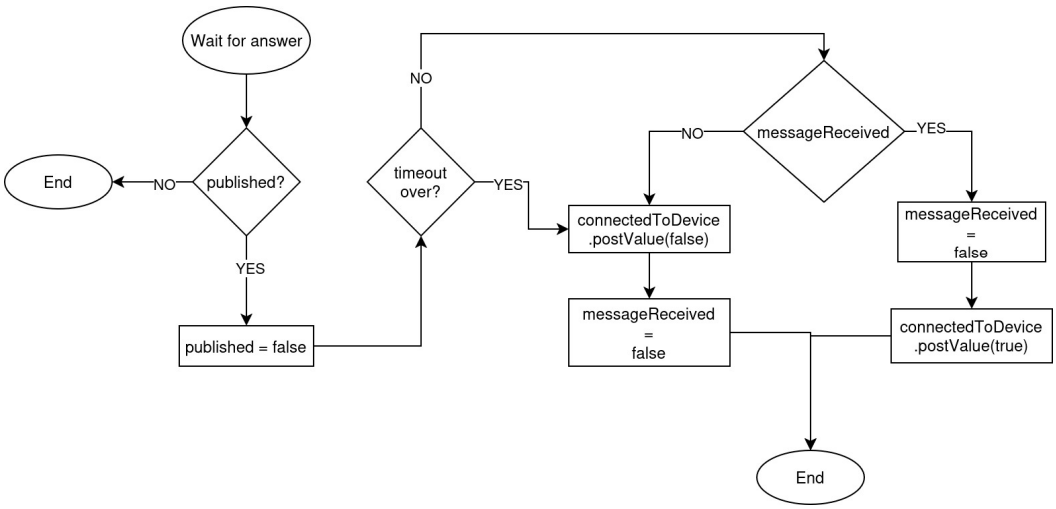


Figure 5. Logic for MQTT response monitoring and synchronization in the Android app.

Connected state management is implemented using the connectedToDevice variable, which is a MutableLiveData within the ViewModel and is exposed to the UI layer as LiveData. Each time the application publishes a message to the autoclave device, the mechanism illustrated in Figure 5 is triggered. Based on whether a response is received within the specified timeout, the variable is updated accordingly, allowing the user interface to reflect the current connectivity status in real time.

4. Results

A total of 24 autoclave units were assembled, configured, and deployed in real-world conditions as part of the product development and commercialization process. Alongside their practical use and testing, the deployment provided an opportunity to assess the consistency of device behavior, the responsiveness of the user interface, and the robustness of the communication between the Android application and the embedded controller.

The devices were distributed among individual users and small food producers for tasks involving food preservation through thermal sterilization. Users reported that the mobile interface simplified configuration and monitoring compared to traditional hardware or manual controls. The ability to adjust autoclaving parameters (temperature and time) and monitor the device status remotely was particularly well received.

In testing, the MQTT communication proved stable and responsive. The system recovered successfully from temporary network interruptions, automatically re-establishing MQTT connections and resynchronizing device state without requiring user intervention.

The Android application's adaptive interface—supporting both Bulgarian and English, dark mode, and font scaling—proved valuable in accommodating diverse user preferences and accessibility needs. No critical bugs were reported during testing, and users were able to operate the device intuitively.

Overall, the deployment demonstrated that the system architecture is suitable for real-world IoT applications that require reliable, user-friendly interaction between embedded hardware and modern Android software.

5. Conclusions

This paper presented the design and implementation of a compact autoclave system that combines embedded IoT hardware with a modern Android application to deliver a user-friendly, configurable, and affordable solution for small-scale food sterilization. The integration of the ESP8266-based controller with secure MQTT communication and a Kotlin-based Android app demonstrates the potential of combining low-cost embedded systems with advanced mobile development frameworks.

The system architecture enables secure, real-time communication, intuitive user interaction, and scalable deployment. A total of 24 units have been successfully produced and tested in real-world conditions, validating the reliability of both the hardware and the mobile interface.

The proposed approach serves as a model for other embedded systems that require safe, remote configuration and monitoring through mobile devices. By leveraging Android's modern development tools and IoT communication protocols, the system bridges the gap between low-level embedded control and high-level user experience.

Future development will focus on several key areas. A new version of the controller is planned to use the ESP32 platform, which will support Bluetooth communication. The system will remain configurable to support both Bluetooth and MQTT over Wi-Fi, offering greater flexibility. On the software side, the Android application will be migrated to Jetpack Compose, embracing a declarative UI model that simplifies development, improves performance, and allows for a more dynamic user interface experience. Additionally, an

iOS version of the mobile application is planned to ensure compatibility with a broader range of devices and users.

Author Contributions: Conceptualization, T.T. and V.T.; methodology, T.T. and V.T.; software, T.T. and V.T.; validation, T.T. and V.T.; formal analysis, T.T. and V.T.; investigation, T.T. and V.T.; resources, T.T. and V.T.; data curation, T.T. and V.T.; writing—original draft preparation, T.T. and V.T.; writing—review and editing, T.T. and V.T.; visualization, T.T. and V.T.; supervision, T.T. and V.T.; project administration, T.T. and V.T.; funding acquisition, T.T. and V.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Stefanov, T.; Varbanova, S.; Stefanova, M. Mobile Application for Managing an Automated Irrigation System. *TEM J.* **2024**, *13*, 897–908. [CrossRef]
2. Rose, K.; Eldridge, S.; Chapin, L. *The Internet of Things: An Overview*; The Internet Society (ISOC): Singapore, 2015.
3. Safitri, M.; Pranaditya, A.; Handoko, B.; Anggoro, S. Design and implementation of automatic autoclave temperature and pressure data recording system. *IOP Conf. Ser. Mater. Sci. Eng.* **2021**, *1088*, 012081.
4. Kodali, R.K.; Soratkal, S. MQTT-Based Home Automation System Using ESP8266. In Proceedings of the 2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC), Agra, India, 21–23 December 2016; pp. 1–5.
5. Kashyap, M.; Sharma, V.; Gupta, N. Taking MQTT and NodeMCU to IoT: Communication in Internet of Things. *Procedia Comput. Sci.* **2018**, *132*, 1611–1618. [CrossRef]
6. Kodali, R.K.; Mahesh, K.S. A Low-Cost Implementation of MQTT Using ESP8266. In Proceedings of the 2016 2nd Conference on Contemporary Computing and Informatics (IC3I), Noida, India, 14–17 December 2016; pp. 404–408.
7. Korzukhin, S.V.; Khaydarova, R.R.; Shmatkov, V.N. Configurable IoT Devices Based on ESP8266 SoC System and MQTT Protocol. *J. Sci. Tech. Inf. Technol. Mech. Opt.* **2020**, *20*, 722–728. [CrossRef]
8. Soni, D.; Makwana, A. A Survey on MQTT: A Protocol of Internet of Things (IoT). In Proceedings of the International Conference on Telecommunication, Power Analysis and Computing Techniques (ICTPACT-2017), Chennai, India, 6–8 April 2017; pp. 173–177.
9. Hillar, G.C. *MQTT Essentials—A Lightweight IoT Protocol*; Packt Publishing Ltd.: Birmingham, UK, 2017.
10. Mishra, B.; Kertész, A. The Use of MQTT in M2M and IoT Systems: A Survey. *IEEE Access* **2020**, *8*, 201071–201086. [CrossRef]
11. Atmoko, R.A.; Riantini, R.; Hasin, M.K. IoT Real-Time Data Acquisition Using MQTT Protocol. *J. Phys. Conf. Ser.* **2017**, *853*, 012003. [CrossRef]
12. ViewModel Overview. Available online: https://developer.android.com/ (accessed on 23 April 2025).
13. LiveData Overview. Available online: https://developer.android.com/topic/libraries/architecture (accessed on 23 April 2025).
14. Elizarov, R.; Safronov, I.; Chistyakov, A.; Smelkov, D.; Ismagilov, T. Kotlin Coroutines: Design and Implementation. In Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!), Virtual Event, 20–22 October 2021; pp. 68–84.
15. Damyanov, D.; Hristov, A.; Varbanov, Z. Design patterns over solid and grasp principles in real projects. *Math. Educ. Math.* **2024**, *53*, 76–84. [CrossRef]
16. Todorov, T.; Tonkov, V. Water Flow Management System with Equipment Protection. *Eng. Proc.* **2024**, *70*, 23. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.