Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

# A Software-Based Digital Twin Prototype using Virtual Sensor Data, MQTT and Streamlit

**Samrudhi Sunil Balikai**
Department of Computer Science and Engineering
(Artificial Intelligence and Data Science)
Padmabhooshan Vasantraodada Patil Institute of Technology, Budhgaon, Sangli, India

**Pratiksha Mansing Jamdade**
Department of Computer Science and Engineering
(Artificial Intelligence and Data Science)
Padmabhooshan Vasantraodada Patil Institute of Technology, Budhgaon, Sangli, India

**Abstract - This paper describes the development of a small software-based Digital Twin prototype for a smart-room example. The prototype uses Python programs to generate sample temperature, humidity, light and motion values. These values are arranged in JSON messages and are intended to be exchanged through MQTT. The software also includes a virtual gateway, virtual fan and light states, an SQLite database and a Streamlit dashboard. Terminal outputs and dashboard captures are used to describe the behaviour observed during development. The available outputs show the generation of sample readings, some state updates and database-save messages. However, connection failures involving the broker and DNS were also observed. For this reason, the work does not claim that MQTT communication was reliable in every run, or that the prototype measures a real room. It is presented as an educational implementation that brings together the main software stages of a Digital Twin workflow without physical sensors or actuators.**

**Keywords - Digital Twin, Internet of Things, MQTT, virtual sensors, Python, SQLite, Streamlit, software simulation.**

## I. INTRODUCTION

A Digital Twin is a digital representation of an object, process or environment that is updated with information about the system it represents. Depending on its purpose, it may be used to monitor conditions, run simulations, analyse behaviour or support decisions. A static digital model does not necessarily change when the physical system changes; a connected Digital Twin is intended to reflect those changes as new data are received. Digital Twins have been studied in areas such as manufacturing, smart buildings, transportation, healthcare and energy management [1]. Their practical use depends on reliable data, suitable synchronization, interoperability and validation [1]–[3]. Reviews focused on the built environment also point to difficulties with data integration, consistency and cybersecurity [10], [11].

In this work, a smart room is used as an example to demonstrate a software-based workflow. Python modules generate temperature, humidity, light and motion values, which are arranged as JSON messages. MQTT is included as the intended messaging method between the publisher and the Digital Twin processing module. The software model also maintains virtual fan and light states, stores available readings in SQLite and displays information through a Streamlit dashboard. The example is entirely software-based: it does not use physical sensors, a microcontroller or hardware actuators.

## II. PROBLEM STATEMENT AND OBJECTIVES

### *A. Problem Statement*

In many small monitoring applications, information may be produced by separate components and can be difficult to review in one place. A single digital view can make available readings easier to inspect and can show the current state of an example system. This project examines how a basic software workflow can combine sample-data generation, JSON formatting, message exchange, state processing, local storage and dashboard display. Since the readings are simulated, the prototype is not used to measure or validate the behaviour of a real installation.

### *B. Objectives*

- To describe the main components of a software-based Digital Twin workflow.
- To develop separate software modules for generating and processing virtual sensor data.
- To generate sample temperature, humidity, light and motion readings.
- To format the readings as JSON and use MQTT as the intended publish–subscribe communication method.
- To maintain virtual fan and light states in the software model.
- To store available readings in SQLite and present them through a Streamlit dashboard.

## III. LITERATURE REVIEW

Yao et al. [1] reviewed Digital Twin definitions, technologies, architectures, evaluation methods and application areas. Their review explains how data and models are used to represent physical systems and discusses issues that affect practical developmentIt gives a general background of the software workflow discussed in this paper. Kibria and Kittur [2] investigated the role of Digital Twin technologies in computer engineering research. Their review includes the evolution of Digital Twin applications and the issues to be taken into account in the design of such systems. This work is relevant to the present prototype, because even a small software implementation requires clear module boundaries and reliable data handling.



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Gervasi et al. [3] studied systems of multiple Digital Twins and proposed a framework to describe their relations. They said the terminology and interoperability should be tackled when separate twins are to work together. 1 A similar problem occurs in a smaller context when software modules exchange readings and update a shared representation of system state.

Cho and Noh [4] developed a Digital Twin of a factory which is based on the MQTT to synchronize information between the physical and digital systems. Their work shows the use of a light-weight messaging protocol in an industrial setting. The current prototype also implements MQTT, but with a different scope: the readings are software generated and reliable end-to-end delivery has not been established yet..

Bouchabou et al. [5] proposed a smart home Digital Twin simulator for recognition of activities of daily living. They show how a simulated home environment can be used to develop and study data-driven applications. It is related to this work because both use a home-based setting to represent activities or conditions without requiring every input to come directly from a physical installation.

Cotti et al. [6] explored a Digital Twin supported by machine learning for energy management in smart homes. Their work considers how a Digital Twin can be used for more than displaying readings, including the analysis of energy use. The present prototype does not include a machine-learning model, but its data and storage modules could be extended for analysis after suitable data have been collected.

Nuevo-Gallardo et al. [7] investigated building Digital Twins for energy optimization. Their approach uses simulation, Functional Mock-up Units, container-based execution and surrogate models. This study illustrates the use of simulation and model-based techniques in building applications and provides context for the smart-room example used here.

AlBalkhy et al. [10] reviewed Digital Twins in the built environment, covering their definitions, applications and implementation challenges. They discuss the need to combine information from connected systems and to address interoperability and validation. These issues make it important to explain how data move between the modules in a software prototype.

Mousavi et al. [11] reviewed Digital Twin applications and capabilities in the built environment. Their discussion includes real-time data integration and analytics, as well as barriers such as data consistency, system integration and cybersecurity. These topics are relevant to this prototype because its operation depends on messages being exchanged with consistent data fields.

Palley et al. [12] reviewed the use of machine learning together with Digital Twins for smart-building operation and energy management. They describe opportunities to combine sensor information, real-time data and analytical models. No machine-learning method is implemented in the present work. However, the data-generation and storage stages could support later analysis if a suitable dataset is collected and evaluated.

Zhang et al. [13] reviewed frameworks and enabling technologies for building Digital Twins. They discuss issues related to different data sources, interoperability, integration and automation. Their review supports the use of clearly separated modules and consistent data exchange. The current prototype illustrates these ideas in a smaller scale through its sensor, messaging, state-processing, storage and dashboard modules.

### *A. Research Gap and Motivation*

I notice that the studies discussed above cover Digital Twin architectures, factory synchronization, smart-home simulation and building energy applications. Many of these applications use equipment, domain-specific models or dedicated integration frameworks. I also see that the reviews report continuing difficulties, with data, interoperability and reliable integration. Digital Twin architectures also appear in the discussion of factory synchronization. Factory synchronization faces challenges. Smart-home simulation also shows issues. Building energy applications also suffer from these problems. Heterogeneous data, interoperability and reliable integration remain problematic. This project uses a limited and educational way of doing things. It includes data generation, JSON formatting, MQTT-based communication, software state updates, SQLite storage and dashboard display all, in one clear and documented process. The work is therefore an example to practice with; it does not present a new Digital Twin model or say that the system is validated in an industrial setting.

## IV. SYSTEM ARCHITECTURE

The prototype consists of software modules that move data from one step, to the next. First the virtual sensor module makes sample readings. The publisher then adds a timestamp and a virtual gateway identifier. Then it readies a JSON message. MQTT is supposed to carry this message from the publisher to a client that has subscribed.The gateway module is the part of the system that forwards software. The Digital Twin module takes the information that it receives and updates the virtual system state. The gateway module can store the readings locally in SQLite. The Streamlit dashboard shows the readings and the states of the actuators.

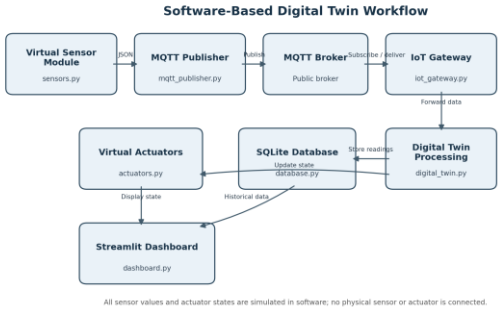



*Fig. 1. Software architecture of the proposed Digital Twin prototype.*

### *A. Description of Main Components*

The sensors.py module generates sample temperature, humidity, light and motion values; these values are simulated and are not physical measurements. The mqtt_publisher.py module adds a timestamp and virtual gateway identifier, formats the data as JSON and publishes it to the configured topic. The iot_gateway.py module represents the gateway stage in software. The identifier ESP32_VIRTUAL_001 is only a label; it does not mean that an ESP32 board is connected. MQTT is expected to deliver a published message to subscribed clients when the broker address, port and topic settings match. The digital_twin.py module processes messages that are received and updates the software state. The actuators.py



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

module maintains virtual fan and light ON/OFF states. The database.py module handles local SQLite storage, and dashboard.py presents the readings and system information available to it.

### *B. Implementation Workflow*

- Generate sample readings using the virtual sensor module.
- Attach a timestamp and virtual gateway identifier and encode the readings as JSON.
- Publish the JSON message to the configured MQTT topic.
- Receive the message through the subscribed Digital Twin client when broker communication is available.
- Process the received values and update the virtual system state.
- Store available readings in the SQLite database.
- Display available sensor values and virtual actuator states on the Streamlit dashboard.

## V. USE-CASE VIEW

The use-case diagram shows the main activities available in the demonstration. An operator can start the virtual sensor process, view the readings that are available, inspect the virtual fan and light states, and review stored information. The MQTT broker is used as the message-exchange service, between the publisher and the subscriber. The dashboard shows information that comes from the software modules.. The sensors, gateway and actuators in this diagram are software representations, not connected physical devices.

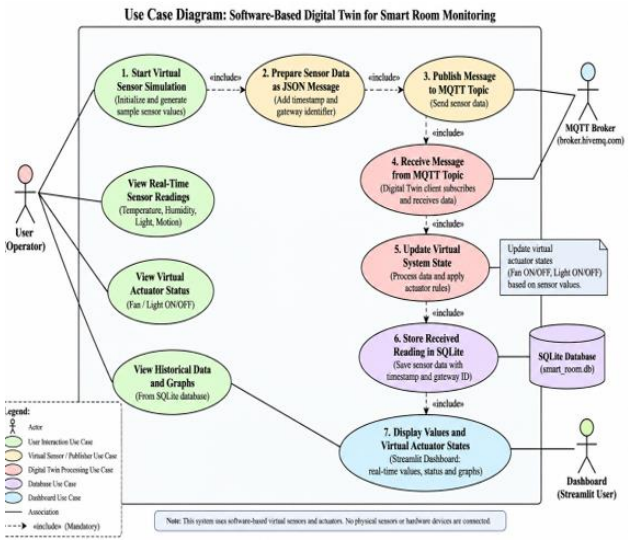



*Fig. 2. Use-case diagram of the software-based smart-room Digital Twin.*

## VI. TOOLS, TECHNOLOGIES AND DATA

The prototype uses a set of software tools for generating, exchanging, processing, storing and displaying data. Python is used to write the application modules, and Visual Studio Code is the development environment used to edit and run the files. MQTT provides the publish–subscribe messaging approach described in the OASIS specification [8]. A public broker is used for the demonstration, so message exchange depends on Internet access and on the broker being available. Streamlit is used to build the browser-based dashboard. SQLite provides local storage through a self-contained database engine [9]. JSON is used to arrange the readings in a format suitable for message exchange. Since no physical sensors are connected, the virtual sensor module supplies sample values. The Python and Streamlit documentation provide details of the software frameworks used [14], [15].

### *A. Example Data Fields*

| Field | Meaning | Example unit / format |
|---|---|---|
| gateway_id | Identifier of the virtual gateway | ESP32_VIRTUAL_001 |
| temperature | Simulated temperature reading | °C |
| humidity | Simulated relative humidity | % |
| light | Simulated light level | lux |
| motion | Simulated motion value or state | Numeric value / state |
| timestamp | Time when the reading is generated | Date and time |

## VII. IMPLEMENTATION

### *A. Software Modules*

| File / module | Purpose |
|---|---|
| sensors.py | Generates simulated sensor readings. |
| mqtt_publisher.py | Prepares and publishes generated readings to the configured MQTT topic. |
| iot_gateway.py | Represents the software gateway/data-forwarding stage. |
| digital_twin.py | Processes available messages and updates the Digital Twin state. |
| database.py | Creates and manages local SQLite storage. |
| actuators.py | Represents virtual fan and light ON/OFF states. |
| dashboard.py | Displays available readings, connection information and virtual system state. |
| app.py | Entry-point or supporting application file, depending on the project setup. |

### *B. Execution Procedure*

To run the project, open its folder in Visual Studio Code and check that the Python files, database file and required packages are present. Start the MQTT broker configured for the project. Run mqtt_publisher.py in one terminal and start digital_twin.py in another terminal. The dashboard can then be launched with the command py -m streamlit run dashboard.py. The publisher and subscriber need to use the same broker settings and topic name. Use Ctrl+C to stop the running processes. The Python package versions and operating system details are not recorded in the project information, for this paper. The missing Python package versions and operating system details should be added after checking the development computer. **VIII. RESULTS AND DISCUSSION**

The development evidence available for this paper includes terminal outputs and dashboard captures. The terminal outputs



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

show sample JSON messages, changes to the Digital Twin state, virtual fan and light status, and database-save messages. The dashboard captures show sensor readings and virtual actuator information. These outputs indicate that individual parts of the application produced the displayed information during the captured runs. Other development captures show broker/DNS connection failures and connection resets. As a result, the evidence does not yet confirm that messages were delivered successfully from publisher to subscriber in every run.

| Observed item | Evidence described in the project | Interpretation |
|---|---|---|
| Virtual sensor generation | Terminal output showing generated readings | Generation of simulated values is demonstrated. |
| JSON preparation | Terminal output showing JSON sensor messages | Message formatting is demonstrated. |
| Digital Twin state | Terminal output showing state updates and virtual fan/light status | Software state processing is demonstrated in the captured run. |
| SQLite storage | Terminal output containing database-save messages | Save operations are reported; stored records should be independently inspected for verification. |
| Dashboard | Dashboard captures with sensor and actuator information | Dashboard presentation is demonstrated for available data. |
| MQTT connection | Separate captures showing broker/DNS failures and connection resets | Reliable end-to-end communication remains unverified. |

The project information available for this paper does not include measured message-delivery rate, latency, throughput, data loss or reliability over repeated runs. No performance comparison is therefore reported. Such an evaluation would require recording the number of messages sent and received, the publication and reception times, the database insert results and the outcome of repeated tests with a documented broker configuration.

## IX. ADVANTAGES AND LIMITATIONS

### *A. Advantages*

- The dashboard brings the available system readings together in one view.
- When database storage is working, readings can be retained for later inspection.
- The modules illustrate how MQTT publish–subscribe messaging can fit into a Digital Twin workflow.
- The software can be demonstrated without buying physical sensor hardware.
- The separate Python modules make individual parts easier to inspect and modify.

### *B. Limitations*

- Simulated readings do not necessarily reflect real room conditions.
- No physical sensor, microcontroller or actuator is connected to the prototype.
- The public MQTT broker requires Internet access and may not always be available.
- End-to-end message delivery and database persistence still need repeated verification.
- Prediction or optimization would require a suitable dataset, a validated model and measured results.
- The current software demonstration should not be treated as an industrial Digital Twin deployment.

## X. CONCLUSION

The work described in this paper is a software prototype that uses virtual sensor readings, MQTT-related messaging modules, Python, SQLite and Streamlit to demonstrate a basic Digital Twin workflow. A smart-room example was used to connect the stages of sample-data generation, JSON formatting, software state updates, local storage and dashboard display. The terminal and dashboard captures show outputs from several modules, including generated readings, virtual actuator states and database-save messages. At the same time, connection failures were recorded during development. The prototype should therefore be viewed as an educational demonstration rather than a fully tested, real-time Digital Twin connected to physical equipment. More work must be done to run the communication tests again look at the database records that are saved and check how well messages are delivered before we can say the system is more reliable. More work must be done to run the communication tests again look at the database records that are saved and check how well messages are delivered before we can say the system is more reliable.

## XI. FUTURE SCOPE

- I will improve MQTT connection. Add reconnection handling and broker-status reporting.
- I will repeat the tests and record MQTT connection message delivery database storage, latency and data loss.
- I will check the SQLite records directly. Improve dashboard historical graphs and report options.
- I will add alerts for simulated readings and later for readings, from physical sensors.
- If prediction is required I will collect a dataset and train and evaluate a machine-learning model.Add authentication, access control and secure message exchange.
- In a version connect appropriate physical sensors and a microcontroller to replace the simulated inputs

## REFERENCES

[1] J.-F. Yao, Y. Yang, X.-C. Wang, and X.-P. Zhang, “Systematic review of digital twin technology and applications,” Visual Computing for Industry, Biomedicine, and Art, vol. 6, art. 10, 2023, doi: 10.1186/s42492-023-00137-4.

[2] N. R. Kibria and J. Kittur, “The emerging role of digital twin technologies in computer engineering research: A systematic literature review,” Frontiers in the Internet of Things, vol. 5, 2026, doi: 10.3389/friot.2026.1870191.

[3] O. Gervasi et al., “Systems of Digital Twins: A systematic literature review and a formal framework,” Digital Engineering, vol. 9, art. 100100, 2026, doi: 10.1016/j.dte.2026.100100.

[4] Y. Cho and S. D. Noh, “Design and implementation of digital twin factory synchronized in real-time using MQTT,” Machines, vol. 12, no. 11, art. 759, 2024, doi: 10.3390/machines12110759.



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

[5] D. Bouchabou, J. Grosset, S. M. Nguyen, C. Lohr, and X. Puig, "A smart home digital twin to support the recognition of activities of daily living," Sensors, vol. 23, no. 17, art. 7586, 2023, doi: 10.3390/s23177586.

[6] L. Cotti, D. Guizzardi, B. R. Barricelli, and D. Fogli, "Enabling end-user development in smart homes: A machine learning-powered digital twin for energy efficient management," Future Internet, vol. 16, no. 6, art. 208, 2024, doi: 10.3390/fi16060208.

[7] C. Nuevo-Gallardo, I. Landa del Barrio, M. Flores Iglesias, J. B. Echeverría Trueba, and C. Fernández Bandera, "Real-time digital twins for building energy optimization through blind control: Functional mock-up units, Docker container-based simulation, and surrogate models," Applied Sciences, vol. 15, no. 24, art. 12888, 2025, doi: 10.3390/app152412888.

[8] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, "MQTT Version 5.0," OASIS Standard, 7 March 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[9] SQLite Consortium, "SQLite Documentation." [Online]. Available: https://www.sqlite.org/docs.html.

[10] W. AlBalkhy, D. Karmaoui, L. Ducoulombier, Z. Lafhaj, and T. Linner, "Digital twins in the built environment: Definition, applications, and challenges," Automation in Construction, vol. 162, art. 105368, 2024, doi: 10.1016/j.autcon.2024.105368.

[11] Y. Mousavi, Z. Gharineiat, A. A. Karimi, K. McDougall, A. Rossi, and S. Gonizzi Barsanti, "Digital twin technology in built environment: A review of applications, capabilities and challenges," Smart Cities, vol. 7, pp. 2594–2615, 2024, doi: 10.3390/smartcities7050101.

[12] B. Palley, J. Poças Martins, H. Bernardo, and R. Rossetti, "Integrating machine learning and digital twins for enhanced smart building operation and energy management: A systematic review," Urban Science, vol. 9, no. 6, art. 202, 2025, doi: 10.3390/urbansci9060202.

[13] X. Zhang, J. Zheng, P. Li, Y. Yang, Y. Ye, F. Causone, and X. Shi, "A review of building digital twins: Framework and enabling technologies," Journal of Building Engineering, vol. 111, art. 113117, 2025, doi: 10.1016/j.jobe.2025.113117.

[14] Python Software Foundation, "Python 3 Documentation." [Online]. Available: https://docs.python.org/3/.

[15] Streamlit, "Streamlit Documentation." [Online]. Available: https://docs.streamlit.io/



(This work is licensed under a Creative Commons Attribution 4.0 International License.)