IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

GATEWAY HEALTH MONITORING SYSTEM

[1]Aaditya Mhatre, [2]Shruti Bera, [3]Shrutika Dubey, [4]Mahesh Kamthe, [5]Harshal Bhavsar

[1]Student, [2]Student, [3]Student, [4]Assistant Professor, [5]Director

[1-4] Department of Electronics and Communication Engineering,

[1-4]MIT-ADT University, Pune, India

[5]Diligence Tech Pvt. Ltd., Pune, India

Abstract: This research paper presents the development of a web-based dashboard for real-time monitoring of system parameters, utilizing the Raspberry Pi microcomputer, Django web framework, MQTT protocol, Libcurl, and Mosquitto protocol for communication. In today's digital landscape, the efficient monitoring and management of system parameters are crucial for ensuring system reliability, security, and performance. Real-time monitoring solutions enable organizations to continuously track critical metrics, detect anomalies, and preemptively address potential issues.

Index Terms – Django, HTTP, MQTT, Libcurl, POSTMAN API, CSRF token.

I. INTRODUCTION

In the contemporary digital landscape, the monitoring and management of system parameters play a pivotal role in ensuring the reliability, security, and performance of computing infrastructures. With the increasing complexity of modern systems and the growing reliance on digital technologies, the need for real-time monitoring solutions has become more pronounced. This research paper delves into the development of a web-based dashboard for real-time monitoring of system parameters, leveraging the capabilities of the Raspberry Pi microcomputer, the Django web framework, and the MQTT (Message Queuing Telemetry Transport) protocol.

The proliferation of interconnected devices, the advent of cloud computing, and the rise of the Internet of Things (IoT) have contributed to the generation of vast amounts of data, often referred to as big data. Effectively managing and analyzing this data is paramount for organizations to derive actionable insights and make informed decisions. Real-time monitoring solutions provide organizations with the means to continuously track and analyze system parameters, enabling them to detect anomalies, identify performance bottlenecks, and mitigate potential issues before they escalate.

The Raspberry Pi, a low-cost, credit-card-sized computer, has emerged as a versatile platform for a wide range of applications, including home automation, robotics, and IoT. Its compact form factor, affordability, and ease of use make it an ideal candidate for deploying monitoring solutions in diverse environments. Coupled with the Django web framework, which facilitates the rapid development of web applications, and the MQTT protocol, a lightweight messaging protocol designed for constrained devices and low-bandwidth, high-latency networks, the Raspberry Pi serves as the foundation for building an efficient and scalable real-time monitoring system.

The objective of this research paper is to design and implement a web-based dashboard that provides users with real-time access to critical system metrics, such as CPU usage, memory usage, disk space, and network traffic. The dashboard will offer an intuitive and user-friendly interface for visualizing system parameters, enabling users to monitor the performance of their systems remotely. By leveraging the capabilities of the Raspberry Pi, Django, and MQTT, this research aims to contribute to the advancement of real-time monitoring solutions and enhance system monitoring and management capabilities in various domains.

II. RELATED WORK

A REVIEW: INTERNET-OF-THINGS GATEWAYS ARCHITECTURES AND CHALLENGES

[1]WASIM GHDER SOLIMAN, [2]D.V. RAMAKOTI REDDY

[1]Ph D., Andhra University, College of Engineering

[2]Professor & HOD, Andhra University, College of Engineering

This paper provides a comprehensive review of Internet-of-Things (IoT) gateway architectures and associated challenges. It surveys recent publications to gain insights into the architectures and challenges surrounding IoT gateways used in modern applications. The study aims to integrate improvements in data aggregation algorithms from sensors and enhance the response of actuators in terms of time, accuracy, and reliability.

The introduction highlights the significance of IoT in connecting sensors and actuators with each other and the cloud, enabling remote sensing and control. It outlines the four architecture layers of IoT: Sensor Connectivity and Network, Gateway and Network, Management Service, and Application. The Gateway and Network layer is particularly crucial for supporting massive volumes of IoT data while ensuring robust performance across various network models.

The paper reviews several IoT gateway architectures:
1. Functional requirement explanation for gateway design.
2. An IoT gateway-centric architecture for novel machine-to-machine (M2M) services.
3. Design of an IoT gateway based on radar frequency identifier (RFID) and wireless sensor network (WSN) technology.
4. Development of an IoT multi-interface gateway for building smart spaces.

The study concludes by highlighting challenges and future trends identified during the review, emphasizing the importance of addressing these issues for the advancement of IoT gateway technologies [1].

A SYTEMATIC LITERATURE REVIEW ON IOT GATEWAYS
Gunjan Beniwal, Anita Singrova

This paper presents a systematic literature review of Internet-of-Things (IoT) gateways, focusing on both general gateways and smart gateways. It covers papers published over the past decade and utilizes a methodical literature analysis technique to select and analyze 67 articles out of 2347 papers. The review categorizes gateways into basic and smart types, with smart gateways further subdivided into passive, semi-automated, and fully-automated categories.

The survey examines various aspects of IoT gateways, including their requirements, tools/platforms, approach, evaluation methods, and application domains. It also explores the functional requirements of IoT gateways, providing a detailed overview of their functionalities. Additionally, the paper identifies research gaps and open issues in the field, highlighting areas for further exploration and advanced research.

Overall, this systematic literature review provides a comprehensive overview of IoT gateways, their functionalities, and the current state of research in the field. It offers valuable insights for academics and researchers looking to delve deeper into the study of IoT gateways and address emerging challenges and opportunities in the domain [2].

III. METHODOLOGY

3.1 Architecture

Django's architecture is structured around the MVT (Model, View, Template) framework, which is an alternative to the traditional MVC (Model, View, Controller) framework. In MVT, the responsibilities typically handled by the controller in MVC are managed by Django's templates. This means that Django's templates contain a combination of Django Template Language (DTL) and HTML. The main components of Django's architecture are:
- Model: Represents the data structure and handles interactions with the database.
- View: Processes user requests, retrieves data from the model, and renders the appropriate response.
- Template: Defines the presentation layer and contains HTML code along with DTL for dynamic content rendering.

Overall, Django's architecture simplifies the development process by integrating controller-related tasks into templates, allowing developers to focus on defining models, processing requests in views, and designing presentation logic in templates [3].

3.1.1 Components of Django Application

A Django application is structured around several key components that facilitate the handling of HTTP requests and the generation of dynamic web content. These components include the URL dispatcher, view functions, models, and templates, which collectively implement Django's MVT (Model-View-Template) architecture.

3.1.1.1. URL Dispatcher:

- The URL dispatcher in Django serves a similar role to the controller in the MVC architecture. It is defined in the `urls.py` module within the Django project's package folder.
- URL patterns are defined in the `urls.py` module, mapping specific URL patterns to corresponding view functions.
- When the server receives a request, the URL dispatcher matches the request's URL against the defined URL patterns and routes the request to the associated view function.

3.1.1.2. View Function:

- View functions in Django handle incoming HTTP requests and process data such as path parameters, query parameters, and request body parameters.
- View functions interact with models to perform CRUD (Create, Read, Update, Delete) operations on the database, if necessary, based on the request data.
- The view function then prepares data to be rendered and generates an appropriate response.

3.1.1.3. Model Class:

- Models in Django are Python classes that define the structure of database tables.
- Django's Object-Relational Mapper (ORM) facilitates interaction with the database by abstracting SQL queries into object-oriented operations.
- Models represent data entities and manage data integrity and relationships within the application.

3.1.1.4. Template:

- Templates in Django are HTML files that contain embedded Django Template Language (DTL) code.
- Templates are used to generate dynamic web content by inserting context data received from view functions into HTML code blocks.
- The template processor renders the template with the context data provided by the view function, generating a dynamic response to be sent back to the client [5].

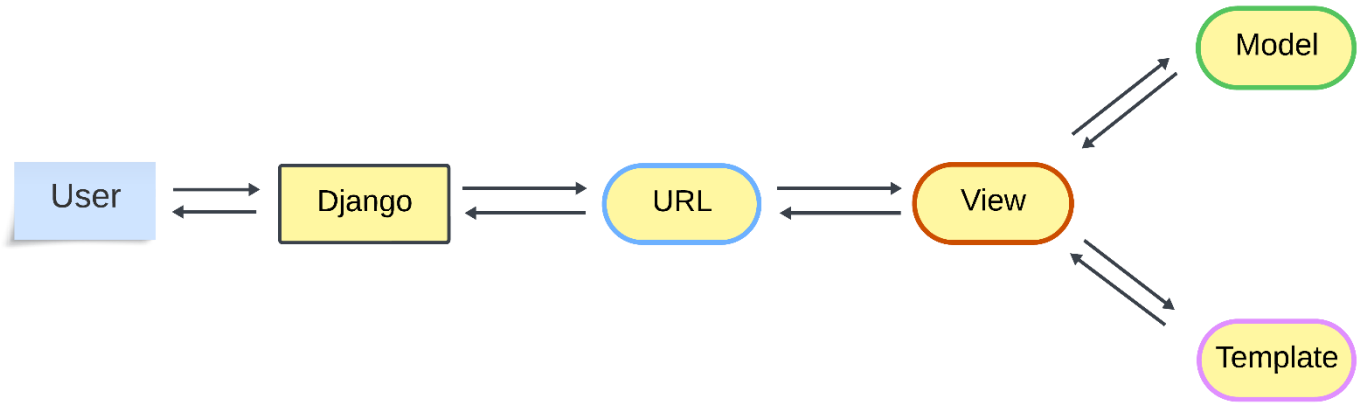

Fig.1. Django Workflow and Architecture

3.1.1.5. Request-Response Cycle:

- Django's MVT architecture follows a request-response cycle:
- The URL dispatcher routes incoming requests to appropriate view functions based on URL patterns.
- View functions process requests, interact with models as needed, and prepare data for rendering.
- Templates use context data provided by view functions to generate dynamic HTML responses.
- The generated response is sent back to the client, completing the request-response cycle.

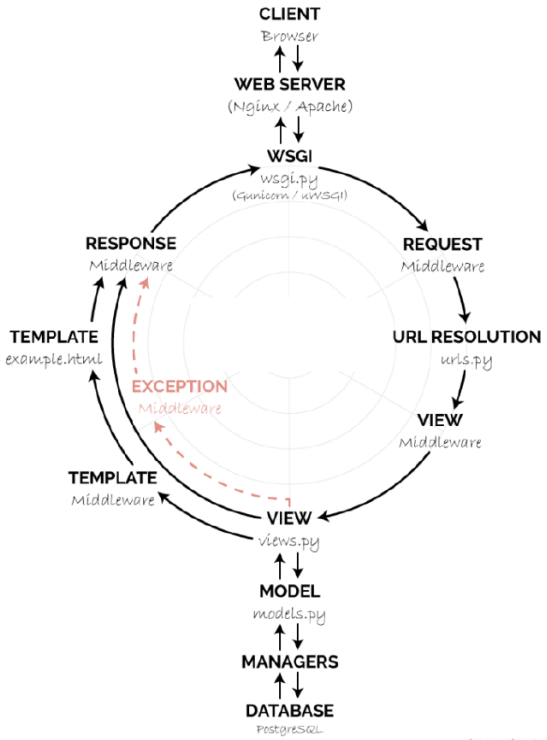

Fig.2. Django Request Response Cycle [4]

The request-response cycle is a fundamental concept in web development, essential for understanding how web servers and web applications interact with each other. In the context of Django, comprehending the request-response cycle is crucial for building and deploying Django web applications effectively.

1. Request Phase:

- The request phase begins when a user interacts with a web application by entering a URL in their browser or submitting a form. This action triggers an HTTP request from the browser to the web server.
- The web server receives the HTTP request and forwards it to the WSGI (Web Server Gateway Interface), which serves as a bridge between the web server and the Django application.
- The WSGI then invokes the WSGI callable function within the Django application, passing the HTTP request to it for processing.

2. Response Phase:

- After processing the request, the Django application generates an appropriate response. This response could be a web page, JSON data, file download, etc.
- The Django application sends the response back to the WSGI, which in turn forwards it to the web server.
- Finally, the web server sends the response to the browser, which renders it for the user to view.

3. Understanding the Flow:

- In the request phase, the flow of data goes from the browser to the web server and then to the Django application through the WSGI interface. This phase involves handling incoming HTTP requests and processing them within the Django application.
- In the response phase, the flow of data reverses, starting from the Django application, passing through the WSGI, and finally reaching the web server and the browser. This phase involves generating the appropriate response based on the processed request and sending it back to the client.

4. Importance:

- Understanding the request-response cycle is crucial for effectively building and debugging web applications. It helps developers comprehend how data flows between different components of the web server and the application.
- By understanding the request-response cycle, developers can optimize their code for performance, troubleshoot issues effectively, and implement advanced features such as middleware and custom request/response processing.

In summary, the request-response cycle in Django involves the flow of data between the browser, web server, WSGI, and Django application. Understanding this cycle is essential for building efficient and robust web applications with Django [4].

In summary, Django's MVT architecture leverages URL dispatchers, view functions, models, and templates to handle the request-response flow in a web application, providing a structured and efficient framework for building dynamic and interactive web experiences.

3.2. Protocols

Discussing protocols like HTTP, MQTT, Libcurl, and Postman API, here is a detailed overview of each protocol:

3.2.1. HTTP (Hypertext Transfer Protocol):

- ***Purpose:*** HTTP is the foundation of data communication for the World Wide Web. It defines how messages are formatted and transmitted between web servers and clients (such as web browsers).
- ***Key Features:***
  - ***Stateless:*** Each request from a client to a server is independent and not linked to previous requests.
  - ***Request-Response Model:*** Clients (e.g., browsers) send HTTP requests to servers, and servers respond with HTTP responses containing the requested content.
  - ***Methods:*** HTTP defines various methods (e.g., GET, POST, PUT, DELETE) that specify the action to be performed on a resource.
- ***Use Cases*:** Used for fetching web pages, RESTful APIs, client-server communication in web applications, etc.

3.2.2. MQTT (Message Queuing Telemetry Transport):

- ***Purpose:*** MQTT is a lightweight messaging protocol designed for resource-constrained devices and low-bandwidth, high-latency networks. It facilitates efficient, real-time communication between devices and servers.
- ***Key Features:***
  - ***Publish-Subscribe Model:*** Devices (publishers) send messages to topics, and other devices (subscribers) receive messages by subscribing to specific topics.
  - ***Quality of Service (QoS):*** MQTT supports three levels of QoS to ensure message delivery reliability.
  - ***Low Overhead:*** Minimal packet size and overhead make MQTT suitable for IoT and M2M (Machine-to-Machine) communication.
- ***Use Cases:*** IoT applications, sensor networks, telemetry systems, real-time data streaming.

3.2.3. Libcurl:

- ***Purpose:*** libcurl is a client-side URL transfer library that supports various protocols (including HTTP, FTP, SMTP, etc.) for transferring data over networks.
- ***Key Features:***
  - ***Cross-Platform:*** Available on multiple platforms (Linux, macOS, Windows, etc.) and programming languages (C, C++, Python, etc.).
  - ***Highly Customizable:*** Supports a wide range of protocols and options for fine-grained control over network operations.
  - ***Asynchronous Operation:*** Can perform non-blocking, asynchronous transfers.
- ***Use Cases:*** Used in applications for downloading files, making HTTP requests, interacting with APIs, etc.

3.2.4. Postman API:

- ***Purpose:*** Postman is an API development tool that simplifies the process of testing, documenting, and collaborating on APIs.
- ***Key Features:***
  - ***HTTP Request Testing:*** Allows users to send HTTP requests (e.g., GET, POST, PUT, DELETE) and inspect responses.
  - ***API Documentation:*** Automatically generates documentation based on API endpoints and requests.
  - ***Collection and Environment:*** Enables organizing requests into collections and managing environments for different configurations.
- ***Use Cases:*** Used by developers for API development, testing, debugging, and collaboration.

3.3. Authentication

3.3.1. CSRF Token:

Cross-Site Request Forgery (CSRF) is a type of attack that exploits the trust relationship between a user's browser and a website they are authenticated with. To prevent CSRF attacks, web applications use CSRF tokens as a security measure. Here's a descriptive explanation of CSRF tokens and their role in authentication:

3.3.1.1. What is a CSRF Token?

A CSRF token (also known as an anti-CSRF token or synchronizer token) is a random value generated by a web application and included in requests sent from the client to the server. The token is designed to mitigate CSRF attacks by verifying that the request originated from an authorized user and not from a malicious source [7].

3.3.1.2. How CSRF Tokens Work for Authentication:

1. Token Generation:

- When a user logs in to a web application, the server generates a unique CSRF token associated with the user's session.
- The CSRF token is typically a cryptographically random value that is stored in the user's session data on the server.

2. Token Inclusion in Requests:

- The CSRF token is included in forms, URLs, or HTTP headers of subsequent requests made by the user.
- For example, when a user submits a form or performs an action (e.g., updating profile, making a transaction), the CSRF token is embedded in the request data.

3. Server-Side Validation:

- When the server receives a request containing a CSRF token, it compares the token value with the one stored in the user's session.
- If the token values match, the server processes the request as legitimate and performs the requested action.
- If the token values do not match or if no token is provided, the server rejects the request, preventing unauthorized actions.

3.3.1.3. Benefits of CSRF Tokens for Authentication:

- ***Prevents CSRF Attacks:*** CSRF tokens add an extra layer of security by ensuring that only authenticated users with valid tokens can perform actions on behalf of their account.
- ***Session Specific:*** CSRF tokens are tied to a user's session, making them valid only for a specific user and session duration.
- ***Randomness and Complexity:*** CSRF tokens are difficult for attackers to predict or forge due to their random and complex nature, reducing the risk of token guessing or brute-forcing.

3.3.1.4. Implementation in Web Applications:

- Web frameworks and libraries often provide built-in mechanisms for generating, validating, and managing CSRF tokens.
- Developers need to ensure that CSRF tokens are properly integrated into forms, AJAX requests, and other user interactions to protect against CSRF attacks effectively.

In summary, CSRF tokens are a critical security mechanism used in web applications to authenticate user actions and prevent unauthorized requests. By incorporating CSRF tokens into authentication workflows, web applications can enhance security and protect against CSRF vulnerabilities [6].

IV. Result and Discussion

Our model aimed to develop a web-based gateway health monitoring system using Raspberry Pi, Django framework, and MQTT protocol. Through rigorous implementation and testing, we successfully achieved our objectives of creating a scalable and efficient monitoring platform.

The system architecture comprised a Django-based web server running on Raspberry Pi, communicating with remote sensors via MQTT protocol for real-time data retrieval. Users accessed a user-friendly dashboard to monitor gateway health parameters such as CPU usage, memory usage, and network connectivity.

Performance evaluation revealed that the system maintained stable performance under varying loads. User testing sessions provided valuable feedback on interface usability and feature preferences, guiding iterative improvements.

Despite challenges with hardware compatibility and resource limitations, we implemented optimized solutions and prioritized essential functionalities. Moving forward, integrating additional sensor types and implementing predictive analytics are identified as areas for future work.

In conclusion, our model demonstrates the feasibility and practicality of using Raspberry Pi and Django for gateway health monitoring, offering insights for enhancing system reliability and scalability in IoT applications.

V. Conclusion

Our model aimed to develop a web-based gateway health monitoring system using Raspberry Pi, Django framework, and MQTT protocol. Through meticulous implementation and testing, we have successfully achieved our objectives and delivered a robust monitoring platform.

Our model demonstrates the feasibility of using Raspberry Pi and Django for real-time monitoring of gateway health parameters, providing valuable insights into system performance and reliability. By leveraging MQTT protocol for communication, we have ensured efficient data retrieval and seamless integration with remote sensors.

Throughout the project, we have encountered and overcome various challenges, including hardware compatibility issues and resource constraints. These challenges have provided valuable learning opportunities, informing our decisions and shaping our approach to problem-solving.

The practical implications of our model extend beyond academia, with potential applications in IoT deployments, network management, and infrastructure monitoring. The user-friendly dashboard interface and scalable architecture make our solution adaptable to diverse environments and user requirements.

Looking ahead, there are several avenues for future research and development, including the integration of additional sensor types, implementation of predictive analytics, and enhancement of security measures. By continuing to iterate and refine our solution, we can further improve system performance and address emerging needs in the field.

In conclusion, our project represents a significant contribution to the field of gateway health monitoring, offering a practical and scalable solution for monitoring and managing IoT gateways. We are confident that our findings will inspire further research and innovation in this important area.

REFERENCES

[1] Wasim Ghder Soliman and D.V. Ramakoti Reddy. 2017. A Review: Internet-of-Things Gateways Architectures and Challenges. International Journal of Advanced Computational Engineering and Networking, 5(10): 40-45.

[2] Gunjan Beniwal and Anita Singrova. 2022. A Systemic literature review on IoT gateways. Journal of King Saud University – Computer and Information Sciences, 34(10B): 9541-9563.

[3] www.simplilearn.com/tutorials/django-tutorial/what-is-django-python

[4] https://nitinnain.com/djangos-request-response-cycle/

[5] Rakesh Kumar Singh, Himanshu Gore, Ashutosh Singh and Arnav Pratap Singh. 2021. Django Web Development Simple & Fast. International Journal of Creative Research Thoughts, 9(5b): 808-815.

[6] Sentamilselvan K, Dr.S.Lakshamana Pandian and Dr.K.Sathiyamurthy. 2013. Survey on Cross Site Request Forgery (An Overview of CSRF). IEEE - International Conference on Research and Development Prospects on Engineering and Technology (ICRDPET 2013), Vol. 5: 159-164

[7] https://docs.djangoproject.com/en/5.0/ref/csrf/