**Contents**

1. Title
2. Abstract
3. Review of Literature
4. Objectives
5. Problem Identification and formulation of problem statement
6. Road map, impact and advantages of the proposed work
7. References

**Dispatcher Process Monitoring Dashboard**

**Abstract:**

The Dispatcher Process Monitoring Dashboard is a real-time tracking and monitoring system designed to enhance the efficiency of dispatcher operations. The system utilizes HTML, CSS, and JavaScript for the frontend, while Node.js and Express.js handle the backend operations. Web Sockets enable seamless communication between the frontend and backend, ensuring dynamic updates. The data, available in JSON format from an API link, is displayed in a structured tabular format for ease of analysis. This dashboard aims to automate monitoring, reduce manual intervention, and improve decision-making by providing live updates and performance insights.

**Review of Literature:**

1. **Existing Systems & Their Limitations**

Many existing dispatcher monitoring systems rely on periodic polling methods, which introduce inefficiencies and delays in process tracking. Research highlights the drawbacks of traditional systems, such as increased network overhead and slow response times due to frequent API requests.

1. **Use of Web Sockets for Real-Time Data Updates**

Studies on Web Sockets emphasize their advantages in reducing latency, ensuring persistent connections, and eliminating the need for frequent API polling. Web Sockets are widely used in applications requiring live data streaming, making them an ideal choice for process monitoring dashboards.

1. **API and JSON Data Handling**

The use of RESTful APIs and JSON format is common in modern monitoring systems due to their lightweight structure and easy integration capabilities. Research shows that JSON-based API responses improve data handling, especially in web-based applications where data needs to be dynamically updated.

1. **Frontend Technologies for Visualization**

User experience plays a crucial role in monitoring dashboards. Research on web technologies like HTML, CSS, and JavaScript reveals that a well-structured frontend improves readability and user engagement. The tabular representation of data provides clear insights into dispatcher processes.

**Objectives:**

* Develop a dynamic and real-time monitoring dashboard for dispatcher processes.
* Implement an intuitive and responsive frontend using HTML, CSS, and JavaScript.
* Establish real-time communication between the frontend and backend using Web Sockets.
* Ensure efficient data retrieval from the API in JSON format and display it in a structured tabular view.
* Optimize system performance by minimizing API polling and network overhead.
* Enhance security measures for API communication and user data protection.
* Provide scalability for future enhancements and integration with other systems.

**Problem Identification and Formulation of Problem Statement:**

**Problem Identification:**

Traditional dispatcher process monitoring involves manual tracking or inefficient periodic polling, leading to delays in identifying system failures. These issues cause unnecessary workload, slow decision-making, and poor system performance.

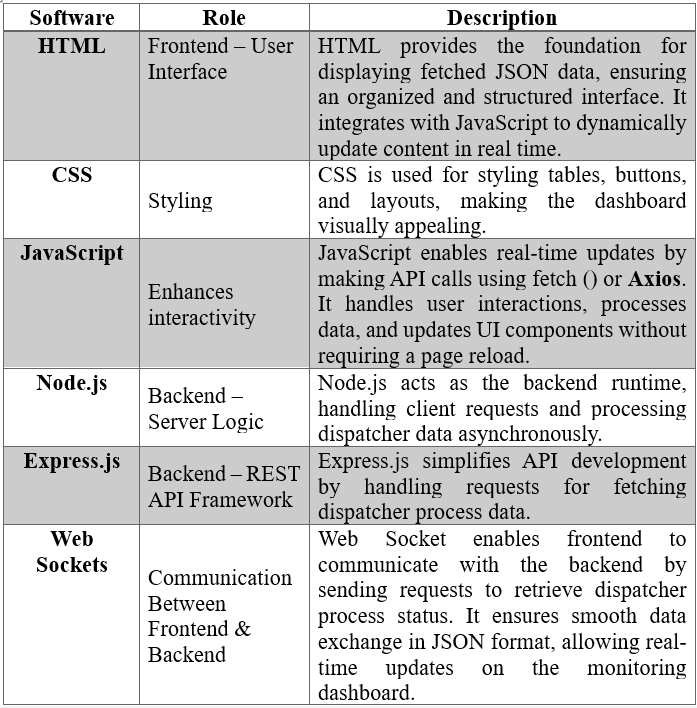
**Problem Statement:**

Develop a **Dispatcher Process Monitoring Dashboard** that leverages real-time data streaming using Web Sockets, efficiently displays API-based JSON data in a tabular format and ensures seamless interaction between frontend and backend technologies.

**Methodology:**

**Technology Stack:**

* **Frontend:** HTML, CSS, JavaScript (for UI and data visualization)
* **Backend:** Node.js, Express.js (for API handling and WebSocket server)
* **Data Communication:** Web Sockets (for real-time updates)
* **Data Format:** JSON (fetched from an API and displayed in a table)

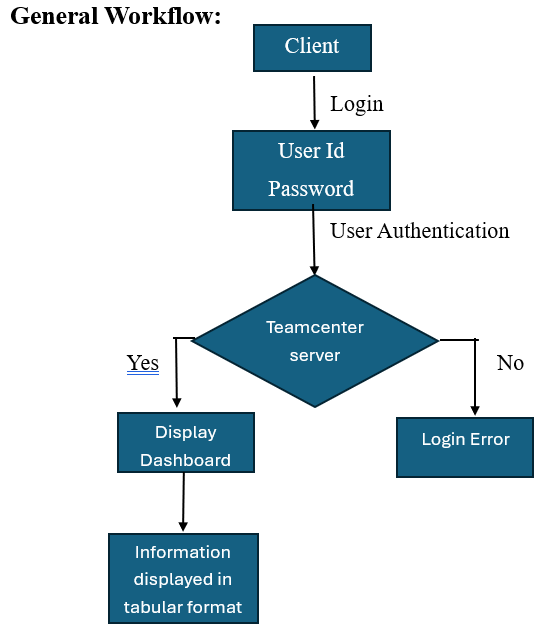


**System Development Phases:**

1. **Requirement Analysis:** Define the essential data points to be displayed in the dashboard.
2. **Frontend Development:** Design a responsive user interface using HTML, CSS, and JavaScript.
3. **Backend Development:** Implement Node.js and Express.js for API handling and WebSocket integration.
4. **WebSocket Implementation:** Establish real-time communication between frontend and backend.
5. **Data Processing:** Fetch JSON data from the API and update it dynamically.
6. **Testing & Optimization:** Debug, optimize, and ensure efficient API calls.
7. **Deployment:** Deploy the system for real-world use and monitor performance.

**System Architecture:**

* **Frontend Layer:** Handles UI components and displays real-time data in a tabular format.
* **Backend Layer:** Manages API requests and WebSocket communication.
* **Data Layer:** Fetches JSON-formatted data from an external API and processes it for display.
* **Communication Layer:** Uses Web Sockets to ensure live updates without excessive API polling.



**Road Map:**

* **Phase 1:** Requirement gathering and feasibility analysis.
* **Phase 2:** UI/UX design for an intuitive dashboard.
* **Phase 3:** Backend development with WebSocket integration.
* **Phase 4:** API integration and JSON data parsing.
* **Phase 5:** Testing and performance optimization.
* **Phase 6:** Deployment and scalability planning.

**Impact of the Proposed Work:**

* **Increased Efficiency:** Automates dispatcher process tracking and reduces workload.
* **Improved Decision-Making:** Provides real-time insights to enhance operational performance.
* **Scalability & Adaptability:** Can be expanded to monitor additional processes or integrate with other systems.

**Advantages:**

1. **Real-time Updates:** Eliminates delays by using Web Sockets for live data streaming.
2. **Optimized Performance:** Reduces unnecessary API requests, improving responsiveness.
3. **User-Friendly Interface:** Ensures clear data representation and easy navigation.
4. **Security & Reliability:** Implements secure API communication and robust error handling.
5. **Scalable & Flexible:** Allows future enhancements for additional process monitoring.

**References:**

[1] Pandey, B., & Sharma, R. (2024). Enhancing Real-time Web Applications with Web Sockets: Performance and Responsiveness. *International Journal of Novel Research and Development*, 9(6), 810-820.

[ijnrd.org](https://www.ijnrd.org/papers/IJNRD2406082.pdf?utm_source=chatgpt.com)

[2] Naik, A. A., & Khare, M. R. (2020). Study of WebSocket Protocol for Real-Time Data Transfer. *International Research Journal of Engineering and Technology*, 7(6), 7020-7023.

[irjet.net](https://www.irjet.net/archives/V7/i6/IRJET-V7I61302.pdf?utm_source=chatgpt.com)

[3] Murley, P., Ma, Z., Mason, J., Bailey, M., & Doupé, A. (2021). WebSocket Adoption and the Landscape of the Real-Time Web. *Proceedings of the Web Conference 2021*, 1639-1651.

[4] Dahal, A. (2023). Real-Time Tracking App: A Dynamic Web Application for Real-Time Position Tracking. *GitHub Repository*.

[github.com](https://github.com/AnwitDahal/Real-Time-Tracking-App?utm_source=chatgpt.com)

[5] Vu, A. (2021). Real-time Backend Architecture Using Node.js, Express, and Google Cloud Platform. *Theseus*.

[theseus.fi](https://www.theseus.fi/bitstream/handle/10024/429165/Anh_Vu.pdf?sequence=2&utm_source=chatgpt.com)

[6] Log Rocket. (2021). Top 6 Tools for Node.js Monitoring. *Log Rocket Blog*.

[blog.logrocket.com](https://blog.logrocket.com/top-tools-node-js-monitoring/?utm_source=chatgpt.com)

[7] Wilinski, R. (2023). Express Status Monitor: Realtime Monitoring Solution for Express-based Node Servers. *GitHub Repository*.

[github.com](https://github.com/RafalWilinski/express-status-monitor?utm_source=chatgpt.com)

[8] Aratus. (2023). Express Performance Monitoring Using Node.js APM. *Aratus Observability Documentation*.

[atatus.com](https://www.atatus.com/observability/express?utm_source=chatgpt.com)

[9] Poe spas, M. (2024). Building a Real-Time Monitoring System with Prometheus, Grafana, and Node.js. *Personal Blog*.

[blog.poespas.me](https://blog.poespas.me/posts/2024/04/26-building-a-real-time-monitoring-system-with-prometheus-grafana-and-nodejs/?utm_source=chatgpt.com)

[10] SSRN. (2023). Real Time Communication Using Web Sockets. *SSRN Electronic Journal*.

[papers.ssrn.com](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4912638&utm_source=chatgpt.com)

[11] IEEE. (2014). Performance Evaluation of WebSocket Protocol for Implementation of Full-Duplex Web Streams. *IEEE Conference Publication*.

[ieeexplore.ieee.org](https://ieeexplore.ieee.org/document/6859715?utm_source=chatgpt.com)

[12] SSRN. (2021). An Analysis of the Performance of WebSockets in Various Programming Languages. *SSRN Electronic Journal*.

[papers.ssrn.com](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3778525&utm_source=chatgpt.com)

[13] IRJET. (2023). Enhancing Real Time Communication and Efficiency with WebSocket. *International Research Journal of Engineering and Technology*, 10(8), 147-150.

[irjet.net](https://www.irjet.net/archives/V10/i8/IRJET-V10I8147.pdf?utm_source=chatgpt.com)

[14] IEEE. (2018). Performance Comparison and Evaluation of WebSocket Frameworks: Netty, Undertow, Vert.x, Grizzly. *IEEE Conference Publication*.

[ieeexplore.ieee.org](https://ieeexplore.ieee.org/document/8605989?utm_source=chatgpt.com)

[15] Codemax. (2023). Building Real-Time Monitoring and Alerting Systems with Express.js. *Codemax Snippets*.

[codemax.app](https://codemax.app/snippet/building-real-time-monitoring-and-alerting-systems-with-express-js/?utm_source=chatgpt.com)