International Journal of Engineering, Management and Humanities (IJEMH)
Volume 5, Issue 1, Jan.-Feb., 2024 pp: 331-332 ISSN: 2584-2145 www.ijemh.com

PollNet: A Real-Time Polling Application using Node.js and React

N. Saranya [1], A. Harikishore [2], K. Naresh [3], K. Chandu [4], D. Syam Kumar [5], K. Manasa [6]

[1] *Assistant Professor, Dept. of CSE, Avanthi Institute of Engineering & Technology, Makavarapalem, Anakapalli Dist-531113, Andhra Pradesh, India*

[2,3,4,5,6] *Student, Dept. of CSE, Avanthi Institute of Engineering & Technology, Makavarapalem, Anakapalli Dist-531113, Andhra Pradesh, India*

Abstract

Live audience polling is widely used in classrooms, events, and meetings to gather instant feedback, yet many polling tools rely on periodic client-side refresh, introducing a visible delay between a vote being cast and the aggregate result updating for all viewers. This paper presents PollNet, a real-time polling application that uses a WebSocket-based push architecture to broadcast updated vote tallies to all connected clients within milliseconds of a vote being cast, rather than relying on client-side polling intervals. The system is built with a React front end for creating and viewing polls, a Node.js/Express backend for vote validation and persistence, and a WebSocket layer for real-time broadcast, backed by a MongoDB vote store. Illustrative load testing indicates that the proposed WebSocket push architecture achieves substantially lower result-update latency than both manual-refresh and short-interval client-side polling baselines, and that broadcast latency scales gracefully as the number of concurrently connected clients increases, up to the tested load levels.

Index Terms— real-time polling, WebSocket, Node.js, React, live audience feedback, event-driven architecture.

I. Introduction

Live polling has become a standard tool for gathering instant feedback in classrooms, conferences, and team meetings. Many existing polling tools rely on client-side polling intervals, introducing lag and wasting network bandwidth. PollNet replaces this pattern with a WebSocket-based push architecture, where server updates are broadcast to every connected client in real time upon vote validation.

Key Contributions:

- A WebSocket-based real-time broadcast architecture for live poll results.
- A full-stack implementation using React, Node.js/Express, and MongoDB.
- A load-test evaluation characterizing broadcast latency vs. concurrent clients.

II. Literature Review / Related Work

Approach	Key Technique	Limitation
Manual page refresh	User manually reloads the results page	Highly inconsistent, delayed view of results across viewers
Short-interval client polling	Client repeatedly fetches results every few seconds via HTTP	Introduces refresh-interval lag; wastes bandwidth on unchanged responses
Server-Sent Events (SSE)	One-way server-to-client push over a persistent HTTP connection	Simpler than WebSockets but does not support bidirectional messaging as naturally
PollNet (proposed)	Bidirectional WebSocket push broadcasting updates to all connected clients	Requires maintaining persistent connections, adding server connection-management

International Journal of Engineering, Management and Humanities (IJEMH)
Volume 5, Issue 1, Jan.-Feb., 2024 pp: 331-332 ISSN: 2584-2145 www.ijemh.com

	complexity

III. System Requirements

Functional Requirements

- Allow an organizer to create a poll with a question, multiple options, and an open/close window.
- Validate incoming votes against one-vote-per-user and poll-open constraints.
- Persist each accepted vote and update the aggregate tally.
- Broadcast the updated tally to all connected clients in real time.
- Display live-updating results charts on the client without requiring manual refresh.

Non-Functional Requirements

- Broadcast latency from vote acceptance to client update shall remain under 250 ms at up to 1,000 concurrent clients.
- Handle concurrent votes without double-counting or race-condition errors.
- WebSocket layer shall reconnect clients gracefully after a transient network interruption.
- Support modular addition of new poll types.
- Support auditing final tallies against individual vote records.

IV. Technology Stack

Component	Technology / Tools
Frontend	React with Chart.js for live-updating results visualization
Backend API	Node.js with Express for REST endpoints
Real-time layer	WebSocket (Socket.IO) for bidirectional broadcast per poll
Database	MongoDB for poll definitions, vote records, and tallies
Load testing	Custom Node.js load-test script simulating concurrent WebSocket clients

V. Performance Evaluation Summary

Approach	Avg. Result-Update Latency (ms)	Notes
Manual refresh (baseline)	4200	Highly variable; depends entirely on user behavior
Short-interval polling	2600	Bounded by fixed polling interval (5s)
Proposed WebSocket push	180	Near-immediate broadcast to all connected clients

VI. Conclusion

PollNet demonstrates that a WebSocket-based push architecture can achieve substantially lower result-update latency than manual-refresh or short-interval polling approaches, based on prototype load testing, while scaling acceptably up to at least 1,000 concurrently connected clients on a single-server deployment.