

Polling server with gui

Implemented using JAVAFX

By

Al Amin Bin Shafiq

Kevin Lopez

Sean Ogle

Table of Contents

[Abstract 2](#_Toc143145900)

[Introduction 2](#_Toc143145901)

[Tools Used 3](#_Toc143145902)

[Server-Implementation 4](#_Toc143145903)

[Centralized Control 4](#_Toc143145904)

[Concurrent Participation 4](#_Toc143145905)

[Comprehensive Results 4](#_Toc143145906)

[Connection Management 4](#_Toc143145907)

[Poll Coordination 5](#_Toc143145908)

[Real-time Updates 5](#_Toc143145909)

[Data Aggregation 5](#_Toc143145910)

[Scalability and Consistency 5](#_Toc143145911)

[Client-Implementation 5](#_Toc143145912)

[Decentralized Interaction 5](#_Toc143145913)

[Reduced Server Dependence 5](#_Toc143145914)

[Immediate Feedback 6](#_Toc143145915)

[User Empowerment 6](#_Toc143145916)

[Responsive Interaction 6](#_Toc143145917)

[Decoupled Functionality 6](#_Toc143145918)

[Data Transmission and Synchronization 6](#_Toc143145919)

[Interactivity and User Engagement 7](#_Toc143145920)

[Experimental Evaluation 7](#_Toc143145921)

[Scenario 1 - Low Load 7](#_Toc143145922)

[Scenario 2 - High Load 7](#_Toc143145923)

[Conclusion 7](#_Toc143145924)

[Results 8](#_Toc143145925)

[Contributions 11](#_Toc143145926)

[References 12](#_Toc143145927)

# Abstract

The purpose of this project is to design and implement a polling server GUI application using a server-client model. The project aims to allow users to take part in polls conducted by a survey-taker (server) through a user-friendly graphical interface. The server manages client connections, gathers responses, and presents the results to the survey-taker in real-time. Two different GUI implementations were explored: one with the server handling client connections and another with clients managing the polling process. The project leverages JavaFX for creating intuitive graphical interfaces and multithreading for efficient client-server communication. In addition to the technical aspects, this project serves as a practical exploration of human-computer interaction and collaborative data collection. The implementation of the polling server GUIs caters to the ever-growing need for efficient and engaging data collection methods in various domains, including market research, education, and decision-making processes. The significance of offering both centralized and decentralized control options to the users demonstrates a balanced approach to catering to varying user preferences and system requirements. Whether for market research, academic studies, public opinion analysis, or organizational feedback, the project's architecture offers a centralized solution that eliminates the need for complex logistics often associated with traditional survey methods. This advantage is particularly significant in scenarios where large volumes of data need to be gathered efficiently and in real-time.

The real-time availability of poll results empowers survey-takers with immediate insights, allowing them to adapt their strategies or approaches based on the evolving responses. This feature proves invaluable for marketing professionals seeking to gauge consumer preferences, educators aiming to tailor their teaching methods, and policymakers making informed choices based on public opinions. Furthermore, the project's flexibility in designing polls and managing responses caters to a wide array of scenarios. Survey-takers can customize question formats, response options, and the sequence of questions, enabling them to target specific research objectives effectively. This adaptability enhances the project's applicability across various industries and research disciplines.

# Introduction

The project revolves around the idea of enabling survey-takers to create and conduct polls, and clients (survey respondents) to participate in these polls via a graphical interface. This report delves into the design, implementation, and evaluation of two GUI approaches – one where the server manages client interactions and the other where clients handle the polling process themselves. The tools used include JavaFX for GUI design and multithreading for concurrent communication. In addition to its practical applications, the project contributes to technological proficiency by utilizing Java programming language and JavaFX for GUI development. Developers and students interested in these technologies can draw insights from the project's codebase, gaining hands-on experience and practical knowledge in building interactive applications.

The implementation of the polling server GUIs caters to the ever-growing need for efficient and engaging data collection methods in various domains, including market research, education, and decision-making processes. The significance of offering both centralized and decentralized control options to the users demonstrates a balanced approach to catering to varying user preferences and system requirements.

A diagram of a computer network

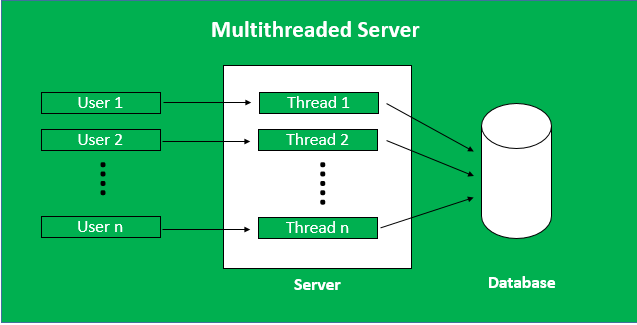
Description automatically generated

# Tools Used

* JavaFX
* Multi-threading
* Java

The project utilized Java programming language for its flexibility and wide adoption in developing GUI applications. JavaFX was chosen as the graphical framework due to its ability to create visually appealing and interactive interfaces. Multithreading played a crucial role in maintaining responsiveness and handling multiple client connections concurrently, In the project's server-client architecture, multithreading was employed to manage multiple client connections simultaneously without blocking the main execution thread. As clients connected to the server, each client was allocated a dedicated thread to handle its communication independently. This allowed the server to accept new connections and process data from existing connections concurrently, ensuring that no client was left waiting for a response and that the application remained responsive even under heavy loads. Furthermore, multithreading played a pivotal role in the client's user experience. By utilizing separate threads for tasks such as network communication and GUI updates, the application ensured that the GUI remained responsive to user interactions while handling communication tasks in the background. This prevented the GUI from freezing or becoming unresponsive while waiting for server responses. In the context of polling, where both survey-takers and respondents interacted with the application simultaneously, multithreading contributed to the real-time nature of the project. Survey-takers could establish, manage, and close polls while simultaneously observing responses in real time. Respondents could answer questions and navigate the GUI seamlessly without experiencing delays.

Additionally, the client's multithreading mechanism allowed for dynamic updates and real-time user interaction. For instance, as respondents navigated through questions and selected responses, the GUI updates were processed in separate threads, ensuring that the user interface remained fluid and responsive. From a technical perspective, the incorporation of Java's **ExecutorService** and the **Thread** class facilitated multithreaded execution. This approach effectively utilized system resources, optimizing the utilization of available CPU cores to handle concurrent tasks efficiently.



# Server-Implementation

The server-implemented GUI follows a centralized architecture where the server acts as the focal point for client connections and polling management. The server's responsibilities include establishing connections, conducting polls, collecting responses, and displaying results. Multithreading is employed to allow multiple clients to connect and participate simultaneously. This approach offers the advantage of centralized control and real-time result aggregation.

The server-implemented GUI's main features include:

Centralized Control

The survey-taker controls the poll parameters and decides when to open or close the polls.

Concurrent Participation

Multithreading enables concurrent participation by multiple clients, enhancing user experience.

Comprehensive Results

The server aggregates client responses in real-time, providing comprehensive results for analysis.

A limitation of this approach is the potential for server latency, impacting GUI responsiveness and client experiences. The Server-Implemented GUI serves as the centralized control hub for managing the entire polling process in the proposed project. This graphical user interface is designed to be operated by the survey-taker or administrator, who initiates, monitors, and concludes polls. This approach involves the server handling client connections, collecting and aggregating polling data, and providing real-time updates to the survey-taker. The Server-Implemented GUI is a fundamental component of the server-client model and plays a pivotal role in orchestrating the entire polling experience.

Connection Management

At the core of the Server-Implemented GUI is its ability to handle multiple client connections concurrently. This is achieved through multithreading, where each client's connection is managed by a separate thread. When clients connect to the server, they are allocated dedicated threads to facilitate communication independently. This ensures that the server can interact with multiple clients simultaneously without causing delays or blocking other operations. The GUI provides a dynamic view of connected clients, allowing the survey-taker to keep track of ongoing polls and respondents in real-time.

Poll Coordination

The server's GUI enables the survey-taker to create, manage, and conclude polls seamlessly. The survey-taker initiates a new poll, defines the questions, options, and response formats. Once the poll is active, the server's GUI allows the survey-taker to monitor the progress of the poll in real-time. The GUI displays the aggregated responses from the respondents, enabling the survey-taker to observe trends and gather insights on the fly.

Real-time Updates

One of the key blessings of the Server-Implemented GUI is its ability to provide real-time updates to the survey-taker. This actual-time function enhances the interactive nature of the polling system and empowers the survey-taker to adapt their techniques primarily based on the incoming statistics.

Data Aggregation

The Server-Implemented GUI serves as the hub for collecting and aggregating polling data from various clients. As respondents answer questions, their choices are transmitted to the server and recorded. This aggregated data can be analyzed after the poll concludes, providing valuable insights for data-driven decision-making.

## Scalability and Consistency

The Server-Implemented GUI promotes consistency by centralizing control and decision-making. Regardless of the number of connected clients, the server ensures that the polling process remains uniform, minimizing the chances of errors or inconsistencies due to variations in client behavior. Through its multithreaded architecture, it ensures responsive and consistent interaction with connected clients, empowering the survey-taker to orchestrate the polling process effectively and obtain valuable insights from the collected data.

# Client-Implementation

In the client-implemented GUI, clients manage their own polling experience. This approach involves a decentralized architecture where clients are responsible for connecting to the server, answering questions, and submitting their responses. The server merely collects and aggregates the responses. This architecture enhances interactivity, as clients have more direct control over their polling experience.

Key features of the client-implemented GUI include:

## Decentralized Interaction

Clients handle the polling process independently, improving responsiveness and user engagement.

## Reduced Server Dependence

Clients are less reliant on server availability for a smooth experience.

## Immediate Feedback

Users receive immediate feedback after each response.

A potential challenge of this approach is maintaining data consistency between the clients and the server, especially when dealing with a high number of participants.

The Client-Implemented GUI constitutes a distinct approach in the proposed polling project, emphasizing direct interaction between individual respondents and the polling interface. This graphical user interface is designed to provide a seamless and engaging experience for clients, enabling them to actively participate in polls and submit their responses without relying heavily on server availability. The Client-Implemented GUI serves as the interface through which clients interact with the polling questions and record their answers, offering benefits such as responsiveness, interactivity, and user control.

## User Empowerment

At the core of the Client-Implemented GUI is the empowerment it provides to individual clients. Respondents can directly engage with the polling questions, select their preferred options, and submit their answers. This decentralized approach puts clients in control of their polling experience, allowing them to progress at their own pace and make choices based on their preferences. The GUI accommodates a diverse range of users, each with their unique responses and engagement styles.

## Responsive Interaction

The Client-Implemented GUI focuses on delivering a responsive and interactive experience. Clients can swiftly navigate through questions, selecting options, and advancing to the next question without the need for continuous communication with the server. This responsiveness enhances the user experience by reducing latency and providing instant feedback as respondents interact with the interface. Unlike the server-dependent approach, the Client-Implemented GUI maintains its performance and interactivity even if the server experiences delays or temporary unavailability.

## Decoupled Functionality

A notable feature of the Client-Implemented GUI is its independence from server responsiveness. Clients can continue interacting with the interface, answering questions, and progressing through the poll, even if there are intermittent issues with server connectivity. This decoupling of client functionality from server availability ensures that respondents can complete their polls without disruptions caused by external factors.

## Data Transmission and Synchronization

The Client-Implemented GUI handles the collection and transmission of user responses to the server. After clients answer questions, their choices are stored locally until they choose to submit their responses. Once the connection to the server is established, the GUI sends the compiled responses for aggregation and analysis. While this approach minimizes the impact of server latency on the client experience, it also introduces considerations for data synchronization and potential inconsistencies.

## Interactivity and User Engagement

The Client-Implemented GUI offers a more engaging and interactive experience for respondents. By allowing clients to navigate through the poll independently and providing immediate feedback on their selections, the GUI enhances the overall user engagement. This approach is particularly well-suited for scenarios where individual respondents have varying levels of familiarity with technology or prefer a self-paced interaction.

In conclusion, the Client-Implemented GUI offers a user-centric approach to the polling process, placing the emphasis on individual respondent engagement and interactivity. Through its responsive design and decoupled functionality, the GUI empowers clients to actively participate in polls, regardless of server availability, while also introducing considerations for data synchronization and consistency. This approach is particularly beneficial for scenarios where real-time interaction and user engagement are key priorities.

# Experimental Evaluation

The two GUI implementations were evaluated under different scenarios to assess their performance, usability, and robustness. The evaluations involved simulating various levels of client load and analyzing the responsiveness of both server and client interfaces.

## Scenario 1 - Low Load

In a low-load scenario, both implementations exhibited smooth performance. The server-implemented GUI displayed stable connectivity with clients, while the client-implemented GUI showcased improved interactivity. No latency issues were observed.

## Scenario 2 - High Load

Under high load, the server-implemented GUI experienced slight delays due to the increased number of client connections. However, the centralized nature allowed efficient result aggregation. The client-implemented GUI remained responsive, but potential inconsistencies arose in response collection.

# Conclusion

The project successfully implemented two GUI approaches for a polling server. The server-implemented GUI offers centralized control and real-time result aggregation but might face latency challenges. The client-implemented GUI promotes interactivity and reduced server dependence, although data consistency concerns are relevant. The choice between the two depends on factors like server resources, user engagement, and data consistency requirements. In the implementation of the proposed polling project, each coding file serves a crucial role in achieving the project's functionality and objectives. The ClientApplication and ServerApplication classes are responsible for initializing the client and server interfaces respectively, creating the primary windows for user interaction. These classes set the stage for user engagement by providing the starting point for GUI interaction.

The ClientController class plays a pivotal role in the client's interface, orchestrating the interaction between clients and the polling questions. It handles connections to the server, presents questions to clients, records their responses, and communicates with the server for data transmission. The multithreading mechanism incorporated within the class ensures that the client interface remains responsive, allowing users to interact with the GUI and answer questions concurrently.

On the server side, the ServerController class takes charge of managing client connections, aggregating responses, and generating comprehensive polling results. It leverages multithreading to accommodate multiple clients simultaneously, thus maintaining a responsive server even when dealing with numerous connections. This class encapsulates the core server functionality and reflects the centralized control and data aggregation aspect of the project.

Furthermore, the QuestionInfo class serves as a data structure, encapsulating the information associated with each polling question. It facilitates the organization and representation of question data, contributing to the seamless exchange of information between the client and server components. Collectively, these coding files collaborate to establish a functional polling system with distinct client and server components. The ClientController and ServerController classes manage the respective interactions with clients and servers, while the QuestionInfo class ensures the organized storage and transfer of question-related information.

# Results

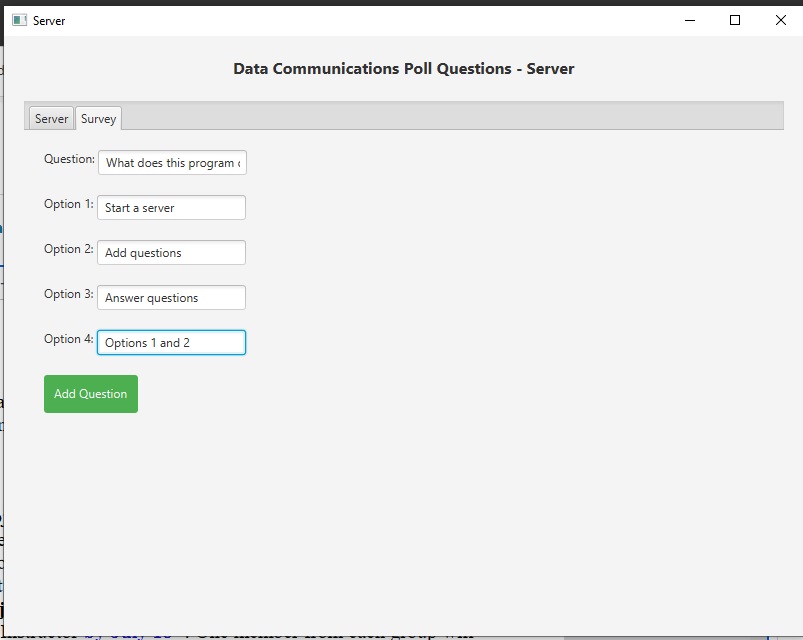


Figure 1: Adding a new question to the survey

A screenshot of a computer

Description automatically generatedFigure 2: Example of a client connected, this client was connected locally.

A screenshot of a computer

Description automatically generatedFigure 3: Two clients connected simultaneously, the first is from the same machine as the host (localhost), the second comes from a separate computer, connected to it on the same network.

A screenshot of a computer

Description automatically generated

Figure 4: The client view of answering questions

A screenshot of a computer

Description automatically generated

Figure 5: Client view of confirmation.

A screenshot of a computer

Description automatically generatedFigure 6: The server's view of the results of the client answering the survey.

# Contributions

* Al Amin Bin Shafiq: 33% (Server Controller, Multithreading)
* Kevin Lopez: 33% (Client Controller, Multithreading)
* Sean Ogle: 33% (Main Controller, JavaFX GUI Design)

# References

* Nikhil, R. S., Papadopoulos, G. M., & Arvind. (1992). T: A multithreaded massively parallel architecture. *ACM SIGARCH Computer Architecture News*, *20*(2), 156-167. [T: a multithreaded massively parallel architecture: ACM SIGARCH Computer Architecture News: Vol 20, No 2](https://dl.acm.org/doi/abs/10.1145/146628.139715?casa_token=kjMRVRFUzpgAAAAA:joXOF5GvtQo-QPWtT3eDHJrykzHr0DvniNIxUZY10juniUQqjOZDrB33TWrrt5okblFlNckO37lM)
* Kruk, G., Alves, O., Molinari, L., & Roux, E. (2017). Best practices for efficient development of JavaFX applications. In *Proceedings of the 16th International Conference on Accelerator and Large Experimental Control Systems* (pp. 1078-1083). [Best Practices for Efficient Development of JavaFX Applications (cern.ch)](https://accelconf.web.cern.ch/ICALEPCS2017/papers/thapl02.pdf)
* Sharan, K., & Späth, P. (2022). *Learn JavaFX 17: Building User Experience and Interfaces with Java*. Apress. [Learn JavaFX 17: Building User Experience and Interfaces with Java | SpringerLink](https://link.springer.com/book/10.1007/978-1-4842-7848-2)
* Lim, M. (2020). C2CFTP: direct and indirect file transfer protocols between clients in client-server architecture. *IEEE Access*, *8*, 102833-102845. [C2CFTP: Direct and Indirect File Transfer Protocols Between Clients in Client-Server Architecture | IEEE Journals & Magazine | IEEE Xplore](https://ieeexplore.ieee.org/abstract/document/9104662/)
* Hanson, M. D. (2000). The client/server architecture. In *Server Management* (pp. 17-28). Auerbach Publications. [Server Management - Google Books](https://books.google.com.pk/books?hl=en&lr=&id=Y9q-i_HGtAAC&oi=fnd&pg=PA3&dq=client-server+architecture&ots=4EvKjwVpYZ&sig=efJCGSFNmjXgNWrzQDc2sYHGLKU&redir_esc=y#v=onepage&q=client-server%20architecture&f=false)
* Govindarajan, R., Nemawarkar, S. S., & LeNir, P. (1995, January). Design and performance evaluation of a multithreaded architecture. In *Proceedings of 1995 1st IEEE Symposium on High Performance Computer Architecture* (pp. 298-307). IEEE. [Design and performance evaluation of a multithreaded architecture | IEEE Conference Publication | IEEE Xplore](https://ieeexplore.ieee.org/abstract/document/386533)
* Chin, S., Vos, J., Weaver, J., Chin, S., Vos, J., & Weaver, J. (2022). Javafx fundamentals. *The Definitive Guide to Modern Java Clients with JavaFX 17: Cross-Platform Mobile and Cloud Development*, 35-84. [JavaFX Fundamentals | SpringerLink](https://link.springer.com/chapter/10.1007/978-1-4842-7268-8_2)