Interactive online debugger for introductory C++ programming courses

Alisher Anuarbekov, BEng in Electrical and Electronic Engineering

Submitted in fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering

School of Engineering and Digital Sciences
Department of Electrical and Computer Engineering
Nazarbayev University

53 Kabanbay Batyr Avenue,
Nur-Sultan, Kazakhstan, 010000

Supervisor: Nursultan Kabylkas
Co-supervisor: Aigerim Yessenbayeva

April 2022

Declaration

I hereby declare that this manuscript, entitled “Interactive online debugger for introductory C++ programming courses”, is the result of my own work except for quotations and citations which have been duly acknowledged.

I also declare that, to the best of my knowledge and belief, it has not been previously or concurrently submitted, in whole or in part, for any other degree or diploma at Nazarbayev University or any other national or international institution.

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Anuarbekov Alisher Ardakuly

Date: 14.04.2023

Abstract

The development process for every high-level programming language is heavily dependent on integrated development environments (IDEs) or code editors, which allow software engineers to write code efficiently in terms of time consumption, clean code, and debugging. These tools are in demand for numerous reasons, including auto-completion, extensions, built-in artificial intelligence that memorizes code patterns, built-in debuggers, and readability. Almost all of the listed bullet points share one common objective, which is to alleviate the coding process for advanced users.

However, during the learning process and acquisition of unknown information, priorities change. As a result, syntax and knowledge of the underlying processes of syntax take precedence over time efficiency and clean code. Moreover, the presence of those listed features can make the study process overwhelming and tedious for students, as they need to familiarize themselves with the concepts of IDEs, code editors, compilers, and debuggers before studying the programming language.

It is worth mentioning that visualization is crucial in the debugging activity, since the survey conducted as a part of our research shows that 75 students out of 96 students prefer visualization during studying rather than plain text. We believe, visualization is particularly important when students start dealing with computer science concepts such as memory allocation, data structure behaviors when core methods are called, and finally, with algorithms. IDEs and code editors provide simple tables as visualizations in debugging procedures, which may not be intuitive at first glance.

Another problem mentioned by professors is the distinction of hardware systems that students use. This distinction reveals the problem of installing components that work only for certain hardware systems. For instance, gdb is not suitable for new ARM circuits, whereas LLDB works properly. Meanwhile, for x86 GDB works correctly.

In several courses of the School of Engineering and Digital Sciences, first-year students who take introductory courses related to high-level programming languages tend to use IDEs and code editors. Some of those courses are Engineering 101, CSCI 151, and CSCI 152. Students in those courses have faced similar problems as mentioned above. They spend time ineffectively downloading and installing components, instead of spending the same time learning actual programming. The objective of this thesis is to present a resolution for developing a comprehensive application, which allows for the visualization of the debugging process. To achieve this objective, the optimal technologies for the application will be assessed, comparing network protocols like HTTP and WebSocket, as well as examining statelessness and statefulness. The role of threads in this process will also be explored. For compiling and debugging, GCC and GDB were utilized respectively. Furthermore, the concept of a Process will be introduced to run the GCC/GDB process separately, and Input/Output streams will be explained. Finally, the thesis will explore parsing GDB responses and rendering them as a React component to provide an easily understandable visualization.

Acknowledgements

I am filled with immense gratitude towards my supervisor, Nursultan Kabylkas, and my co-supervisor, Aigerim Yessenbayeva, for their direction and support throughout the entire academic year, which made my thesis possible. Their professionalism and patience during the master's program were greatly appreciated, as well as the time they spent discussing and providing suggestions, and the hours of work they put into the thesis.

I am grateful to Nazarbayev University for providing me with the opportunity to acquire theoretical knowledge and engineering skills.

Lastly, I would like to express my appreciation to my classmates for their support and advice, which proved to be valuable during the writing of my thesis.

Table of content

List of Abbreviations & Symbols

GNU	GNU's not Unix
GDB	GNU DeBugger
GCC	GNU Compiler Collection
IOC	Inversion of Control
JS	JavaScript
TCP	Transmission Control Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
OSI	Open Systems Interconnection
ACE	Ajax.org Cloud9 Editor
LLDB	Low Level DeBugger
URL	Uniform Resource Locator
ENG101	Engineering 101
IOC	Inversion of Control
ARM	Advanced RISC Machine
API	Application Programming Interface
DOM	Document Object Model
IDE	Integrated Development Environment
JSX	JavaScript XML
NPM	Node Package Manager
REST	Representational State Transfer

List of Tables

List of Figures

Chapter 1 – Introduction

Debuggers were invented four decades ago and have been helpful in aiding inexperienced programmers in understanding code flow and helping experienced programmers identify defects in the software. However, the complexity of debugging varies depending on the code being debugged, and it can be an arduous and overwhelming task for novices who have not yet fully grasped coding concepts. A survey conducted among engineering students at Nazarbayev University revealed that 63 out of 101 students did not comprehend debugging through Integrated Development Environments (IDEs) while taking ENG101 or CSCI151 courses.

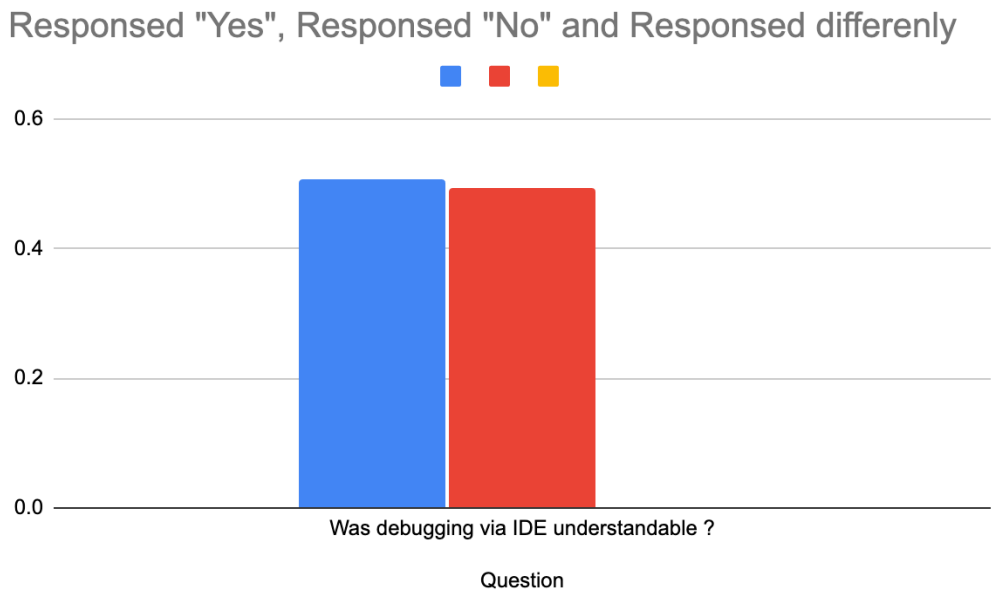


Figure 1. Conducted survey at Nazarbayev University about debugging through IDE

Nonetheless, when asked whether debugging is helpful in understanding coding, 60 out of 101 students responded positively.

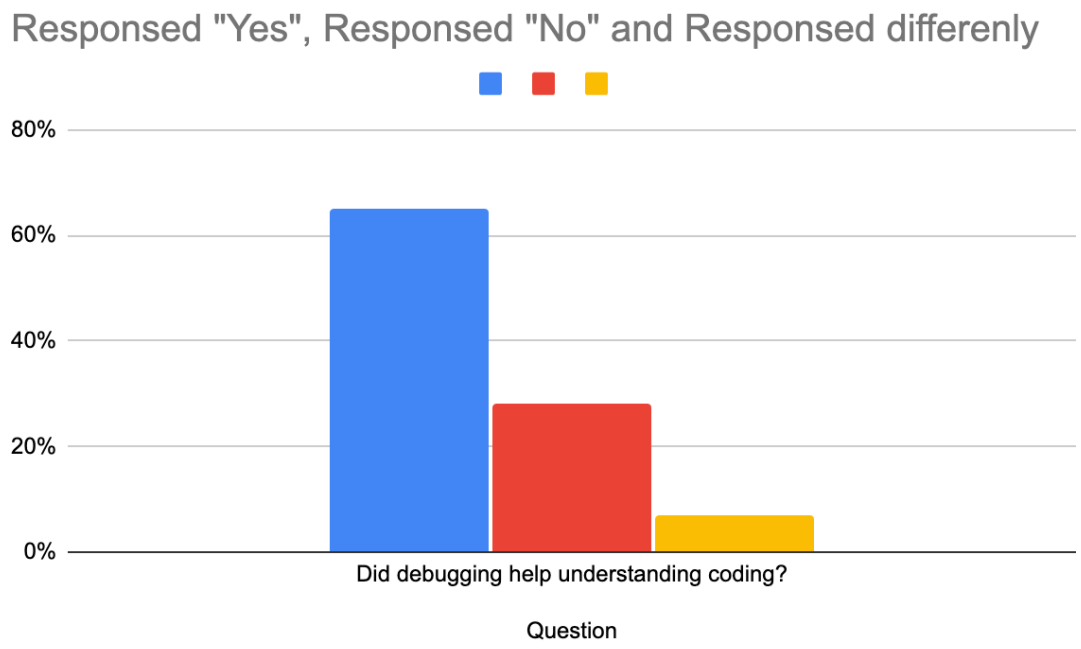


Figure 2. Conducted survey at Nazarbayev University about debugging

This suggests that debugging is an essential tool for acquiring coding skills, but it needs to be more user-friendly.

Furthermore, the thesis argues that students need to devote considerable time to understand what IDEs are and how to use them. This is supported by the fact that 43 out of 101 students did not know what an IDE is, and 56 out of 101 students confirmed that they had to configure the compiler, while 27% stated that they had to install additional debuggers that are compatible with their hardware system. To address these issues, the thesis proposes the creation of a full-stack web application that is preconfigured and ready to use, thus reducing the time spent on tasks, not necessarily related to the core programming concepts.

1.1 Background Information

To begin with, debuggers have been an essential tool for software developers since their inception in the 1970s. They were initially developed to help programmers locate errors in their code by allowing them to stop the program's execution at any point and examine its state. Over the years, debuggers have evolved to include a range of features such as breakpoints,

watchpoints, and memory inspection tools, making them indispensable to modern software development.

However, debugging is a skill that takes time and effort to develop. It requires practice and experience, and even the most experienced programmers don't get it right all the time [1]. This is where IDEs come in. IDEs are software applications that provide a comprehensive development environment, including a text editor, a compiler, a debugger, and other tools that facilitate software development. IDEs have become increasingly popular among software developers because they simplify the process of writing, testing, and debugging code.

Unfortunately, the complexity of IDEs can be overwhelming for novices. This underscores the need for more user-friendly development tools that reduce the time spent on unnecessary tasks and allow users to focus on coding. Fabio Petrillo's "Towards Understanding Interactive Debugging" is an example of efforts aimed at improving user-friendliness in tracking errors in code. He argues that traditional debugging methods, such as print statements or stepping through code, are not always effective in these situations, and that interactive tools are needed to allow developers to explore the system and diagnose issues more quickly and efficiently. Based on his findings, Petrillo proposes a set of design principles for interactive debugging tools, which he argues can help make these tools more effective and user-friendly. These principles include features such as the ability to explore the program state dynamically, the ability to visualize program data[4]. While this paper focuses on developing robust and large-scale solutions, it's worth emphasizing that our project aims to create a simple debugger that prioritizes user-friendliness, specifically for educational purposes rather than industrial applications.

Over the decades, hardware systems evolved, and there are now several manufacturers which produce their own CPUs. By the end of the first decade of the 21st century, there were two dominant technology companies - AMD and Intel - whose CPUs were installed in 80% of devices. However, in 2020 Apple has switched to an in-house designed ARM processor which is rapidly growing in numbers installed in MacBooks and several other products of the same

company. Thus, GDB, debugger became not suitable for newly launched ARM processors.

In light of these challenges, the thesis proposes the creation of a full-stack web application that is preconfigured and ready to use, with the aim of reducing the time spent on setting up an IDE and configuring the compiler and debugger for newbie students. However, the most important part is appealing visualization which will alleviate understanding programming concepts. This would make it easier for novices to learn how to debug and would help them acquire coding skills more efficiently.

1.2. Aims and Objectives

As stated in the previous section, this paper proposes interactive with proper visualization of debugging as a solution for mentioned problems. In the theory of cognitive load Mayer, R. E., & Moreno, R claimed that visual aids, such as diagrams and animations, can help to reduce cognitive load and enhance learning in multimedia environments[11]. They suggest that visual aids can help learners to better understand complex concepts, as they provide a way to represent information in a more intuitive and easily digestible form. This, in turn, can make it easier for learners to process and retain the information being presented, leading to improved learning outcomes[3]. In the case of programming, visualization can help developers understand the architecture of a system and trace the root cause of a problem to a specific module or component [2]. In the courses ENG101 or CSCI151, it can be helpful to understand concepts of pointers, memory allocations, iterations, and objects.

During the survey that I launched at Nazarbayev University, we have found that 27% of students have to install a debugger that specifically fits hardware systems. Nowadays, in the market there are two main debuggers for C, C++ and they are GDB and LLDB. GDB can not be installed to ARM processors, however LLDB does. We can now assume that 27% of students have to deal with installing LLDB instead of the default GDB.

Subsequently, the mission of this thesis is to:

1. Eliminate the need of installing a debugger that fits for system use

2. Eliminate configuring a compiler after installing IDE

3. Implementation of interactive debugger with excellent visualization that can be easily understood.

1.3. Literature Review

In this section, the view's of several papers to mentioned problems are discussed, including their advantages and drawbacks. Moreover, as this paper works on creation applications, the list nearly has similar applications.

1.3.1. Read the Debug Manual: A Debugging Manual for CS1 Students by R. Garcia, C. -J. Liao and A. Pearce

In "Read the Debug Manual," Garcia, Liao, and Pearce created a debugging manual for CS1 students that was presented in a conversational tone and removed technical jargon to make it easier for students to understand. The authors emphasize the importance of understanding the root cause of a bug before attempting to fix it and suggest a methodical approach to debugging that involves finding the problem, and testing possible solutions. The manual also includes tips for preventing bugs, such as code reviews and testing.

The manuals are divided into sections based on their role in the debugging process. The first two sections focus on promoting program comprehension and reducing frustration, respectively. The third section promotes testing, with boundary testing as a strategy. The last three sections support the location of errors, with the first subsection providing strategies for compile-time issues and the second subsection discussing code tracing. The third subsection presents strategies for finding runtime errors, including the use of print statements.

Overall, paper work provides a useful resource for novice programmers struggling with debugging. The manual is well-organized and includes practical tips and techniques that can be applied in a variety of programming languages and environments. Nevertheless, the manual has its own downsides, in terms of practical sides. Despite the fact that the results of research depict usefulness of provided strategies, only 18.4% of students used a manual. Moreover, some

strategies of the most important section of locating error were unfavorable, since it is easier to Google it. From the perspective of the present thesis, we aim to span a full class and give practical and visual explanations rather than theoretical bases with strategies, since visual aids alleviate the process of learning and memorizing concepts without much cognitive load[11].

1.3.2. Towards Understanding Interactive Debugging" by Petrillo, F. (2016)

Petrillo's paper explores the challenges of interactive debugging and proposes a theoretical framework for understanding it. The paper argues that interactive debugging is a complex and multifaceted process, and suggests that it is influenced by cognitive factors, such as perception, memory, and attention, as well as social and cultural factors. The paper provides an overview of existing research on interactive debugging, and identifies gaps in the literature that need to be addressed in future research. These gaps I highlighted as most relevant:

1. The need for more empirical studies on the effectiveness of different interactive debugging tools and techniques.
2. The need for more research on how novice programmers learn interactive debugging skills.
3. The need for more research on how interactive debugging can be integrated into programming education.
4. The need for a better understanding of the cognitive processes involved in interactive debugging.

In my opinion, Petrillo's paper makes a valuable contribution to the literature on interactive debugging. The theoretical framework that is proposed is useful for understanding the complex nature of interactive debugging, and the paper's identification of gaps in the literature provides a valuable starting point for future research. I think that further research into the role of context in interactive debugging is particularly important, as this could help to identify ways to support novice programmers who may struggle with debugging in certain contexts. Although

running and creating scripts using a specialized tool or training that Petrillo's paper introduced is quite manageable and can bring a lot of benefits in the long run, it still must be taught by professors and instructors at university. It is time consuming and overwhelming for novices who just started coding.

1.3.3. "VIDA: Visual Interactive Debugging" by Dan Hao (2008)

In this paper, Hao introduces VIDA, a visual interactive debugging tool for Java programs. The paper argues that traditional debugging tools can be difficult to use, especially for novice programmers, and that visual debugging tools can help to address this issue. The paper provides an overview of existing research on visual debugging and identifies gaps in the literature that VIDA aims to address.

The paper describes the key features of VIDA, including its ability to show the program state visually, to allow the user to interact with the program state, and to support the visualization of complex data structures. The paper also describes the evaluation of VIDA through a user study, which found that VIDA was effective in helping users to find and fix bugs in their programs.

Although this integrable tool represents everything we are looking to address, this tool has to be integrated into the Eclipse development environment, which in my opinion is a drawback of this tool. As an argument, I can say Eclipse requires you to configure a compiler and install a debugger that fits the hardware system. Since avoiding such overheads is the objective of this thesis, we must come up with more concise and an easy to start using solution.

Overall, the paper highlights the importance of visual debugging tools in helping programmers to understand the behavior of their programs and to locate and fix bugs. It suggests that VIDA is a promising tool for improving the debugging experience for novice and experienced programmers alike.

1.3.4. Web Based Programming Assistance Tool for Novices by Anmol More, Jitendra Kumar and Renumol V. G.

This research presents a web-based programming tool that aims to assist novice programmers in learning programming concepts and constructing programs. The tool provides various features such as syntax highlighting, code completion, and error detection, which are intended to make the programming process easier for novices. Additionally, the tool provides a graphical user interface that allows users to construct simple programs without having to write code.

The authors conducted a user study to evaluate the effectiveness of their tool. The study involved a group of novice programmers who used the tool to complete programming tasks. The results of the study indicated that the tool was effective in helping novices learn programming concepts and construct programs.

It also provides tips for each of C errors using relational databases and has the ability to go step by step and monitor the state of the program. It is the most relevant implementation of the program that I encountered during the research. However, this approach did not give the main objective of this thesis: visualization of the debugging process and avoiding just plain raw text.

Overall, the paper presents an interesting approach to assisting novice programmers in learning programming concepts. The web-based nature of the tool makes it accessible to a wide range of users, and the various features provided by the tool are likely to be helpful for novices. However, it is important to note that the study conducted by the authors was relatively small, and further research is needed to confirm the effectiveness of the tool. Additionally, it would be interesting to see how the tool could be further developed to assist novices in more advanced programming concepts.

1.3.5. "The Art of Debugging with GDB, DDD, and Eclipse" by Norman Matloff, Peter Jay Salzman, and Kai Nitzsche

Debugging is a vital skill for programmers to ensure software quality and reliability. This literature review covers two key topics in debugging: graphical debugging with DDD and IDE debugging with Eclipse.

Graphical debugging with DDD provides a visual interface for debugging, allowing programmers to view their code as diagrams, making complex code easier to understand. It also allows for faster and more efficient debugging and improved collaboration among team members. However, DDD may not be suitable for all code types and may require significant resources for larger codebases.

IDE debugging with Eclipse is another popular technique that provides built-in debugging tools, allowing programmers to step through their code line by line, set breakpoints, and inspect variables. Eclipse offers a user-friendly debugging interface, easier code navigation, faster debugging, and better integration with other development tools. However, it may require significant setup and configuration time, especially for beginners.

While graphical and IDE debugging have numerous benefits, they may not be suitable for all code types and bugs, and may require significant resources and setup time. However, combining these techniques with other debugging tools and techniques can effectively identify and resolve bugs in code.

While graphical debugging with DDD and IDE debugging with Eclipse have many benefits, there are also some potential drawbacks to consider. One potential drawback of graphical debugging with DDD is that it may not be suitable for all types of code. While graphical debugging can be helpful in understanding complex relationships between functions

and variables, it may not be as effective for simpler code. Additionally, DDD can be resource-intensive and may slow down the debugging process for larger codebases. Another potential drawback of IDE debugging with Eclipse is that it may require significant setup and configuration time. IDEs can be complex tools, and configuring them for debugging can be time-consuming, especially for beginners. Additionally, IDEs can be resource-intensive and may require a powerful computer to run smoothly.

1.3.6. "Debugging reinvented" by Andrew Ko, Brad Myers

The paper "Debugging Reinvented" by Ko and Myers introduces a new debugging approach based on causal reasoning, which involves identifying the root cause of a bug by reasoning backwards from the symptom of the bug. To implement this approach, the authors developed a tool called "Whyline" that allows developers to interactively (by asking questions) trace the causal chain of events that led to a bug, starting from the symptom reported by the user. The tool presents this information in a way that is easy to understand and navigate, allowing developers to quickly identify the root cause of the bug. The authors argue that traditional debugging methods, such as print statements and interactive debuggers, are often time-consuming, error-prone, and require a deep understanding of the code being debugged. In contrast, their approach using Whyline is more efficient and effective. The paper presents the results of a user study comparing Whyline to traditional debugging methods, which found that developers using Whyline were able to fix bugs more quickly and with less effort.

Overall, the paper provides a compelling argument for the benefits of causal reasoning and introduces a useful tool for implementing this approach. In my perspective, despite the fact that the paper argues against interactivity in debugging, I assume it will be extremely helpful to give users an opportunity to ask questions about possible problems in the code. The only issue can arise is accuracy of the answers. However, whyline uses algorithms such as dynamic slicing,

precise call graphs, and new algorithms for determining potential sources of values and explanations for why a line of code was not reached[10], this paper was introduced 15 years ago. This approach of asking questions and finding possible roots of the problem reminded me of an OpenAI solution that was launched in 2018, which is presumably much more precise in generating answers. Thus, integrating a limited version of OpenAI to a web application solution would be beneficial in learning the debugging process. It is worth considering for future work.

Chapter 2 – Suitable Network Protocols for the objectives of thesis

2.1. Transfer Control Protocol as a base of connectivity

The Transmission Control Protocol (TCP) is a core protocol of the Internet Protocol Suite that is responsible for ensuring reliable and ordered data delivery between applications running on different hosts. TCP achieves this by providing flow control, error detection, and retransmission of lost packets.

One of the key features of TCP is the three-way handshake process used to establish a connection between two devices. This process involves three steps: the client sends a SYN (synchronize) packet to the server, the server responds with a SYN-ACK (synchronize-acknowledge) packet, and finally, the client sends an ACK (acknowledge) packet to the server. Once this handshake is complete, a reliable connection is established, and data can be transmitted between the two devices using TCP.

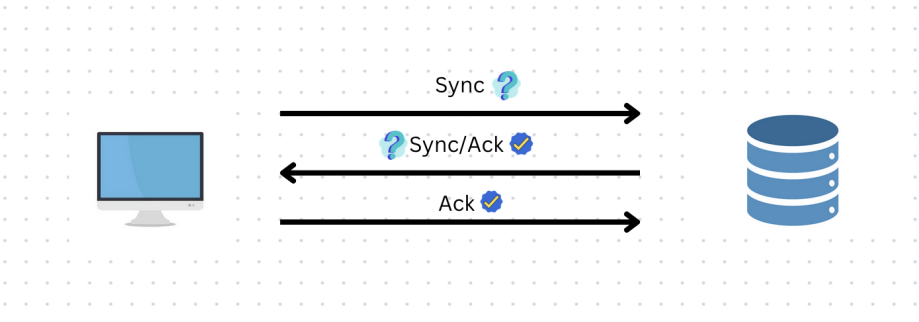


Figure 3. Three-way handshake establishment

In the perspective of this thesis, TCP serves as the foundation of the WebSocket protocol, providing the necessary reliability and error correction mechanisms for real-time communication over the web. WebSocket, a computer communications protocol, builds on top of TCP to provide a bi-directional, full-duplex communication channel between a client and server. It enables real-time data exchange and reduces the overhead of constantly establishing and tearing down connections. By leveraging TCP's connection-oriented features, WebSocket provides a low-latency and efficient means for bidirectional communication between web applications and

servers, enabling rich and interactive web experiences. TCP also includes several other features such as flow control and congestion control to optimize the data transfer process. One such feature is the window size, which specifies the amount of data that can be sent before receiving an acknowledgment from the receiver. The receiver's window size is communicated to the sender through the ACK packets, and the sender can adjust its transmission rate based on the receiver's window size to prevent overloading the receiver. This helps in optimizing the data transfer process and preventing data loss due to congestion.

TCP's reliability and error correction mechanisms make it an ideal protocol for use in web sockets, which require a stable and persistent connection. Overall, TCP plays a crucial role in enabling connectivity in web sockets and ensuring a stable and reliable data transfer.

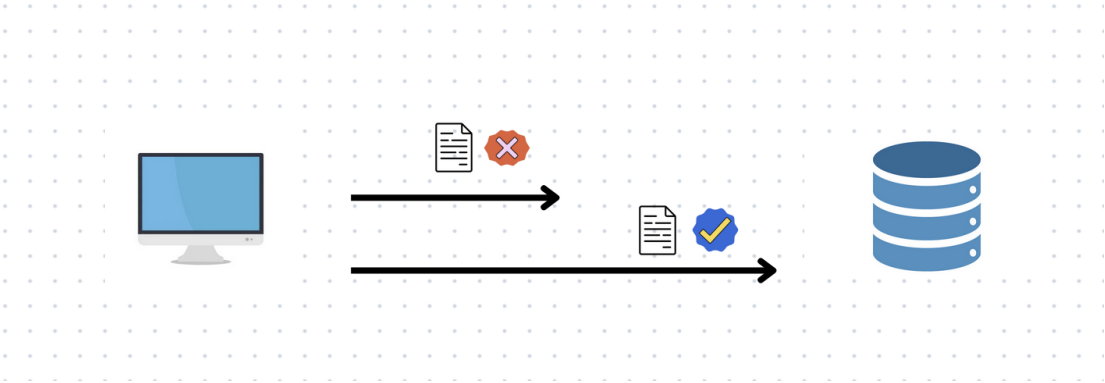


Figure 4. TCP reliability feature which resend lost data

2.2. Concept of HTTP and lack of state

Hypertext Transfer Protocol (HTTP) is the foundation of data communication on the World Wide Web. It is a client-server protocol where the client makes requests and the server responds to the requests. The protocol works on a request-response model, where a client sends a request to a server and the server sends a response back. HTTP is stateless, meaning that each request from a client to a server is treated as an independent transaction. The server has no knowledge of any previous interactions with the client. This design choice was made to keep the protocol simple and scalable, which has contributed to its success.

However, the lack of state can also pose challenges for certain types of web applications. For example, an online shopping website requires a user to maintain a shopping cart as they navigate through the website. Without state, the server cannot maintain the contents of the shopping cart across multiple requests. This problem is usually solved by using cookies or other mechanisms for maintaining state information. This limitation brings an analogous issue to code editor web applications too, since code may request user input and without a stateful connection server would not know to which session send request. Moreover, HTTP is unidirectional, meaning HTTP requests and responses flow in a single direction, from the client to the server or from the server to the client. It will arise a problem, considering the fact that for one connection code editor application will send multiple answers, such as debugging responses and user input requests.

Making elaboration to unidirectional HTTP protocol, when a client (usually a web browser) sends an HTTP request to a server, it expects a response in return. The client initiates the communication by sending a request that includes specific HTTP methods (such as GET, POST, PUT, DELETE) and a Uniform Resource Locator (URL) that identifies the location of the requested resource on the server. The server then processes the request, retrieves the requested

resource (such as a web page, image, or file), and sends a response back to the client. The response typically includes the requested resource, along with an HTTP status code that indicates whether the request was successful or encountered an error. Once the server sends the response, the communication is complete, and the connection between the client and server is closed. If the client needs to request additional resources, it must initiate a new HTTP request, starting the unidirectional communication process over again. Here comes new overhead in the face of a new TCP connection for each HTTP request. Having a brand new TCP connection is costly for servers which accept request, since it needs to waist resources for establishing each connection and unwise network bandwidth management.

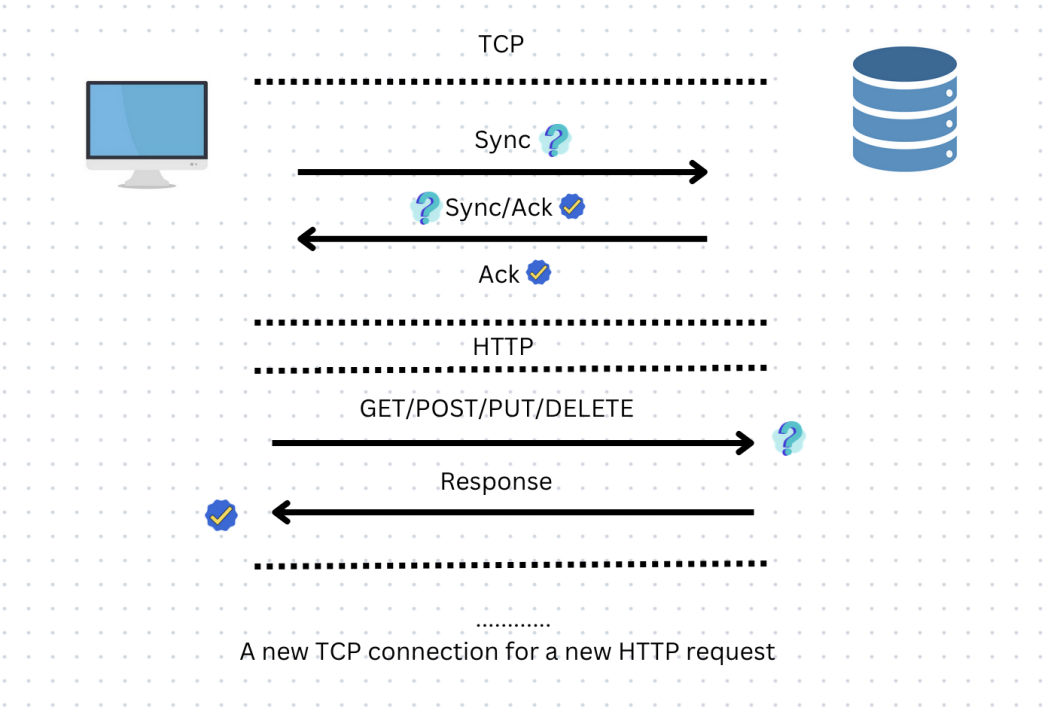


Figure 5. Unidirectional HTTP and TCP's Three-Way handshake

However, HTTP still will be useful, after all Web Socket connection starts from an HTTP request with a UPGRADE flag in it. In addition, the lack of state also makes it difficult to implement certain types of security measures, such as authentication and authorization. To address this issue, various mechanisms have been developed, such as session tokens and JSON

Web Tokens (JWTs).

Overall, the concept of HTTP and its lack of state have had a significant impact on the design and development of web applications. While stateless HTTP has contributed to the scalability and simplicity of the protocol, the lack of state can pose challenges for developing code editor web applications.

2.3. Concept WebSocket, importance of state, and bidirectional communication

WebSocket is a protocol that enables bidirectional, real-time communication between a client and a server over a single, long-lived connection. Unlike HTTP, which follows a unidirectional, request-response model, WebSocket allows for a persistent, full-duplex communication channel between client and server, where data can be sent in both directions at any time.

The key feature of WebSocket is its ability to maintain a persistent connection, which allows for efficient and low-latency communication between client and server. This is in contrast to HTTP, where a new connection must be established for each request, resulting in higher latency and increased network overhead.

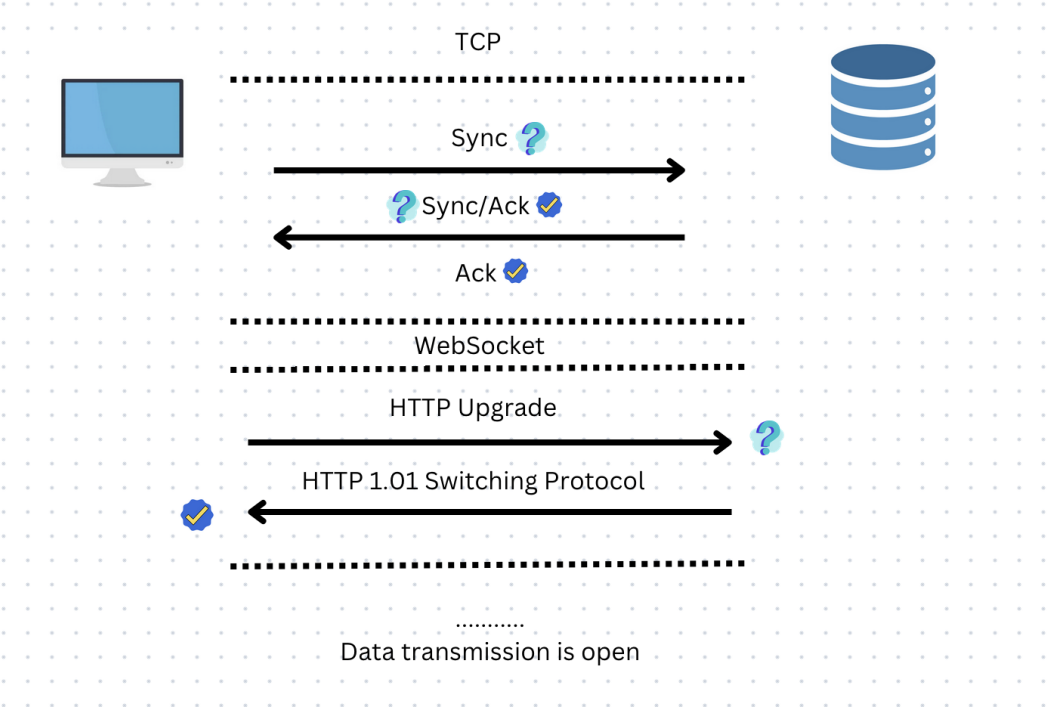


Figure 6. WebSocket connection establishment

For each WebSocket connection, a new TCP (Transmission Control Protocol) connection is not established. WebSocket uses an existing TCP connection to initiate and maintain a persistent, full-duplex communication channel between client and server.

When a WebSocket connection is established, it begins with a standard HTTP handshake over an existing TCP connection. The client sends an HTTP request to the server, requesting an upgrade to the WebSocket protocol. If the server accepts the request, it sends an HTTP response indicating the successful upgrade, and the connection is then switched to the WebSocket protocol.

The use of an existing TCP connection also allows WebSocket to take advantage of TCP's reliable transmission mechanisms, including error checking, flow control, and congestion control. This helps ensure that messages are delivered reliably and in the correct order, even in the presence of network congestion or packet loss.

The importance of state in WebSocket is critical to its functionality, as it enables the server to maintain information about the client's connection, such as session IDs or user-specific

data. This information is stored in memory and is accessible by the server for the duration of the WebSocket connection. The use of state is essential for WebSocket-based applications, such as online games, chat applications, or real-time collaborative editing tools, as it enables the server to maintain context about the client's state and respond accordingly.

Bidirectional communication is another critical aspect of WebSocket. This allows for real-time updates to be sent from the server to the client without the need for the client to initiate a request. This enables more responsive and dynamic user interfaces, as changes can be pushed to the client in real-time, rather than relying on the client to request updates at regular intervals.

If we come to the need of Web Socket from the perspective of code editor application, the key features of this network protocol will become handy. Firstly, the presence of stateful connection with memorizing Session ID by server makes the server know which client to send requests to about user input. Secondly, bidirectional communication makes it possible for the client side to send and also receive requests from the server. It will be the essence of the establishing connection for the debugging process too. Making points clear, let's assume the C++ code requires input from the user, in this case the server ought to send a request to the client that code which under running requires user input. In this case, the server must possess knowledge about Web Socket Session ID, and ability to send data to the client during processing business logic. As noticed above, the debugging process also needs to be taken care of by WebSocket's features. When code is sent to the server, it will be debugged by GDB and its output should be read by the server and sent to the client for further processing. However, the GDB will output one output for each step or for each break in the code, meaning the server will send multiple responses as a part of one Web Socket session.

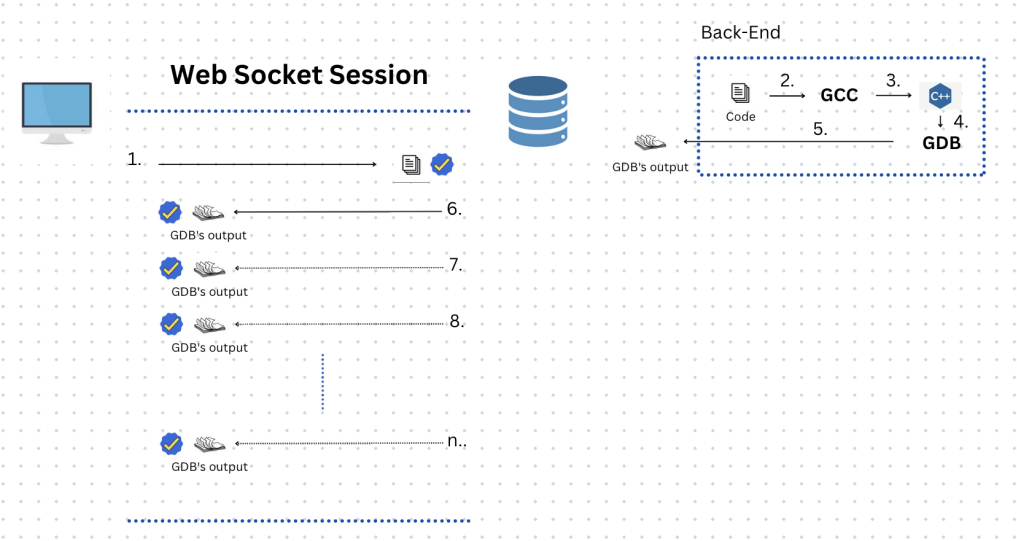


Figure 7. Multiple responses from GDB as part of a Web Socket Session

In summary, WebSocket is a protocol that enables bidirectional, real-time communication between a client and a server over a single, persistent connection. The use of state and bidirectional communication are critical aspects of WebSocket, allowing for efficient and low-latency communication and enabling more responsive and dynamic user interfaces. Listed features are ideally fit for this thesis needs, since statefulness gives application ability to send in both directions multiple messages as part of one single connection.

2.4. Objectives of utilizing concurrency and threads

Concurrency and threads are critical concepts in the field of computer science that enable programs to perform multiple tasks simultaneously. Concurrency refers to a program's ability to handle multiple tasks simultaneously, whereas a thread is an independent sequence of instructions that can execute concurrently with the main program. By splitting a program into smaller, independent parts or threads, each can execute separately, allowing the program to perform more than one task at a time. Multithreading is the process of simultaneously executing multiple threads within a program, which can enhance performance and create more responsive user interfaces. Worth to be mentioned that concurrency and multithreading are not interchangeable. Whereas concurrency is a term that defines the program doing multiple tasks at

the same time, multithreading is the technique by which concurrency is achieved. The utilization of concurrency and threading is essential in modern computing as it permits faster and more efficient processing of tasks. These concepts have diverse applications, ranging from operating systems and database management systems to video game development. However, for accomplishing the purpose of this thesis, we will use multithreading not for boosting application's performance, but rather for its convenience to achieve subtasks. As we mentioned, code that is running and debugging will produce some output to the standard output stream. These outputs might be error related outputs when the program throws runtime exceptions and results of code. Consequently, these outputs must be monitored and scanned when code will send output to the output stream. Then scanned output must be sent to the client. Thus, separate threads that will monitor and catch results of C++ programs will streamline those processes.

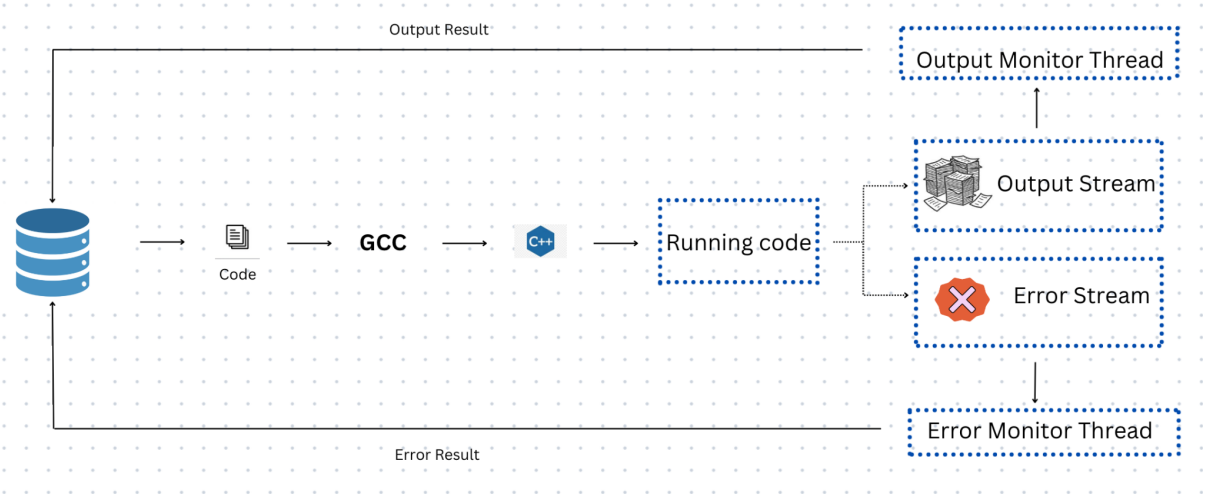


Figure 8. Standard Output Monitoring Threads Layout

Due to the fact that Java was utilized as developing back end applications, ServiceExecutor will be used as coordinator of threads we created and responsible for closing those threads. There are two main tasks which will do application and they are compiling/running and debugging. For each session of Web Socket and for each task we will create separate threads for capturing output results.

2.5. Roles of GCC, GDB and Standard Input/Outputs

GDB (GNU Debugger) and GCC (GNU Compiler Collection) are two powerful tools that are widely used in the field of software development. GDB is a debugger that allows developers to track down and fix errors in their programs, while GCC is a compiler that is used to transform source code into executable files. GDB is a command-line tool that is available on many operating systems, including Linux, macOS, and Windows with x86. It provides developers with a range of debugging features, such as setting breakpoints, examining variables, and stepping through code line-by-line. With GDB, developers can pause the execution of their programs at specific points and inspect their state, making it easier to identify and fix bugs. GCC, on the other hand, is a compiler that supports multiple programming languages, including C, C++, and Objective-C. It is an open-source tool that is available on many platforms, including Linux, macOS, and Windows with x86 architecture. GCC translates high-level source code into machine code that can be executed by a computer. It performs various optimization techniques to produce efficient and high-performing code.

Code that is compiled by GCC and running and GDB produce responses, such as the result of code that is completed and GDB's output for each breakpoint. However, where are all those outputs sent ?. Considering the fact that compiling, running and debugging are separate processes, they are able to be logically partitioned in business logic. The Process class is typically used in situations where a Java program needs to interact with an external process, such as running a command-line utility, launching another application, and ideally suitable for running GDB/GCC processes. The Java program can start the external process using the Runtime.exec() method, which returns a Process object that represents the running process. Once the Process object is obtained, the Java program can use its methods to interact with the process. For example, the getInputStream() method can be used to read the process's output, while the getOutputStream() method can be used to write to the process's input. Availing of those streams we can retrieve or write data to GDB/GCC processes via separate Threads which we discussed earlier.

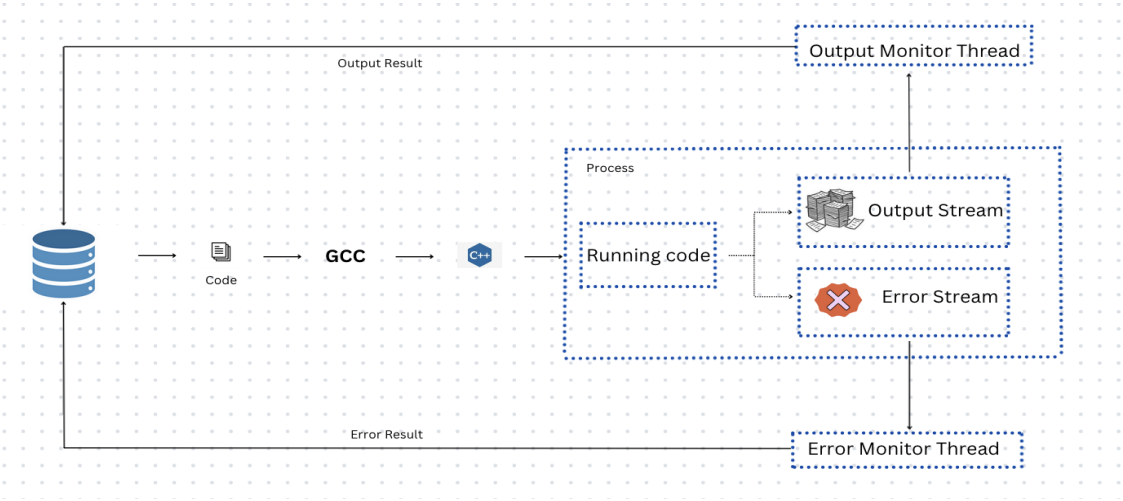


Figure 9. Process class wrapped into GCC/GDB code blocks

2.6. Availing of JavaScript, React.js, and ACE API

As stated in paragraph "Aims and Objectives" the purpose of this thesis is to create a full stack web application, since we need to contemplate about front end technological stacks. Nowadays, there are numerous technologies that can accomplish required thesis tasks, all of them have their pros and cons. Nevertheless, there are two main technologies or base tools that regardless of the used high level technologies will be utilized. They are HTML which stands for Hyper Text Markup Language and CSS which stands for Cascading Style Sheets. They are used for building the structure of web pages and styling of web pages respectively. It is worth making a resolution on the structure of the web pages. At first site it might be simple to gain conceptually what structure of web page is, however web browser where the web page is rendered structure one using particular method. Method is closely related to N-ary Tree data structure, which has nodes represented by HTML elements and it is structured hierarchically from top to down.

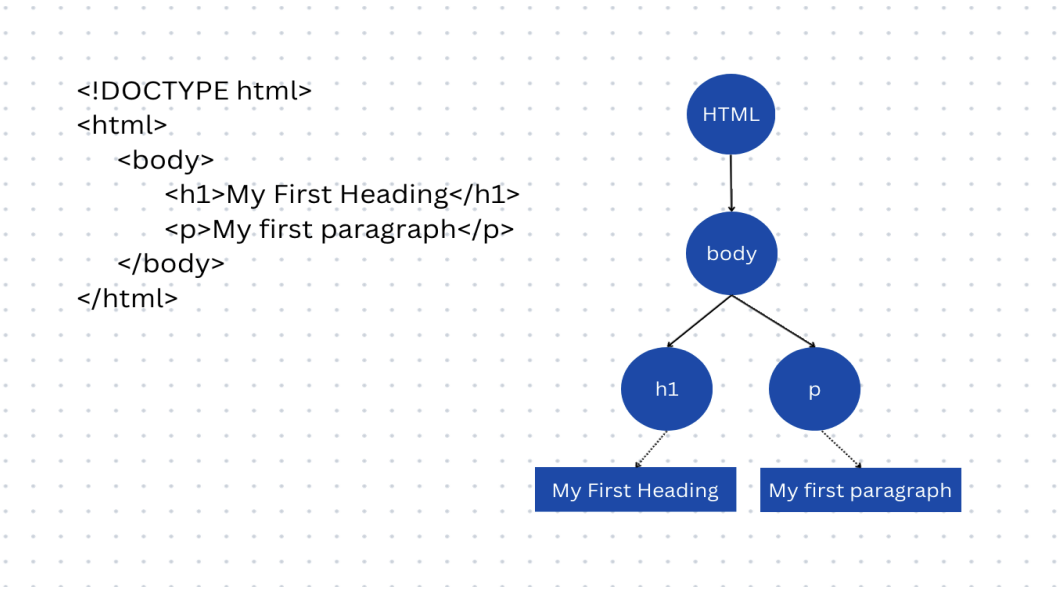


Figure 10. Document Object Model is structure of the web page

This tree-like structure is called Document Object Model, or DOM, and is a crucial component of web development that facilitates the creation of dynamic and interactive web pages. As a hierarchical structure of objects, the DOM represents the HTML elements of a web page, allowing developers to manipulate them through a scripting language such as JavaScript. This provides a powerful toolset for creating dynamic web content and user interfaces that can adapt to user input and respond to events in real-time. Overall, the DOM is an essential aspect of modern web development that enables the creation of engaging and inclusive web experiences. Obviously, as stated, the aim of the thesis is to create an interactive debugger for C++ code, and it is by definition a dynamic web application. Thus, as noticed above we need scripting language that will modify DOM for our needs based on responses from the server. JavaScript language can be an indispensable tool for accomplishing this task of manipulation DOM elements. Nevertheless, manipulating DOM directly using scripting language raises performance issues, given that changing into DOM, even small changes, will render the entire web page. So, imagine the user will step down from one line to next during the debugging process by the road updating the state of the variable. Consequently, we need to visually depict it on the specific part of the web page by modifying DOM, which will lead to inevitable rendering of the whole web page.

Fortunately, technological stacks are constantly evolving and engineers from Facebook have launched React.js framework to overcome the discussed limitation. React.js introduced concept - The Virtual DOM, or VDOM is a technique used in modern web development that involves creating a lightweight representation of the actual DOM of a web page. This virtual representation is used to manage and manipulate the content of the web page, without having to update the actual DOM directly. The library works by creating a virtual representation of the DOM, which is then used to manage and update the actual DOM as needed. This approach allows for faster and more efficient updates to the web page, as the virtual representation can be updated quickly, and then the changes can be applied to the actual DOM in a more optimized way. One of the main advantages of the Virtual DOM is that it allows for more efficient rendering of web pages. By only updating the parts of the web page that have changed, rather than re-rendering the entire page, the Virtual DOM approach can help to reduce the amount of time and resources needed to render web pages. This can be particularly important for web applications that require real-time updates or dynamic content.

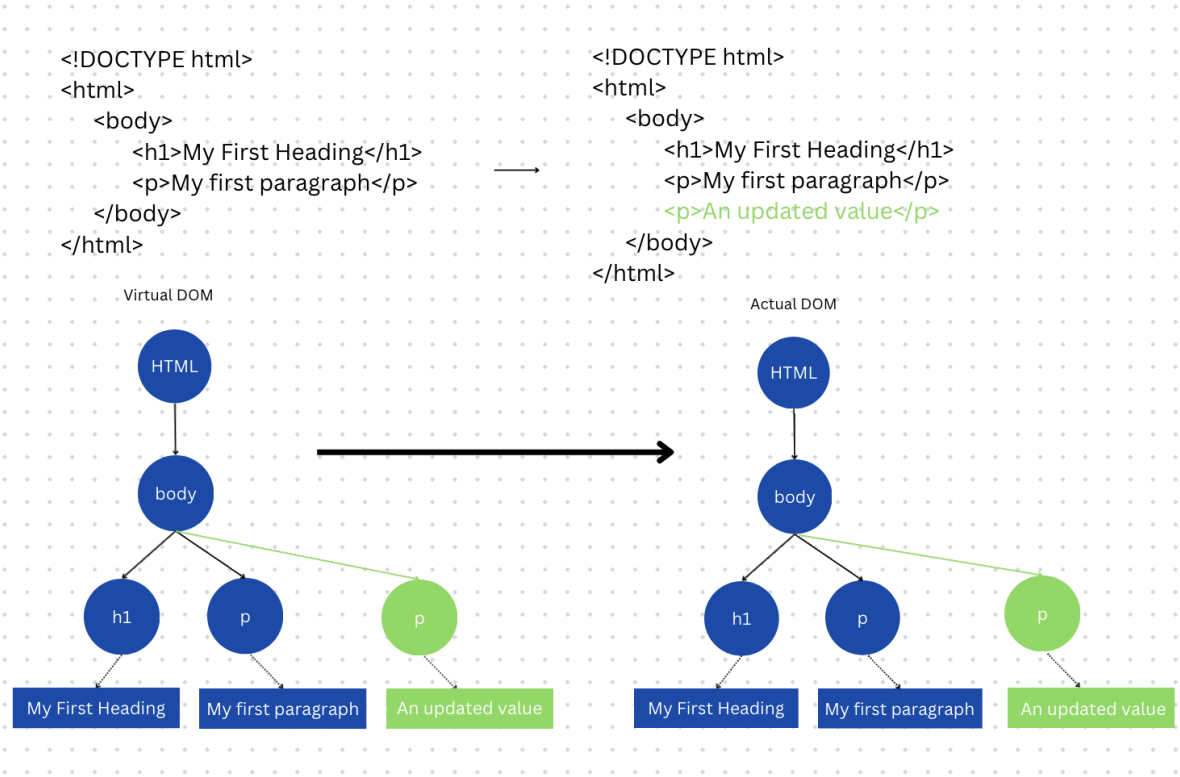


Figure 11. Virtual DOM updates a specific component of Actual DOM

Moving to additional tools that will play an essential role in the developing application, ACE API needs to be discussed. ACE is a widely used code editor that was developed as an open-source web-based editor. It provides a lightweight and customizable environment for developers to write code. ACE stands for "Ajax.org Cloud9 Editor". One of the primary features of ACE is its support for over 60 programming languages, such as C type languages, which we are considering for the ENG101/CSCI151 courses. ACE offers syntax highlighting, code folding, and auto-completion capabilities, which are standard features in modern code editors. ACE is highly extensible, which makes it possible for developers to customize and extend the editor to suit their specific needs. This extensibility has led to the creation of many plugins and extensions that provide additional functionality to the editor. Nevertheless, for the purpose of this thesis tools such as auto-completion, extensions, or plugins need to be deprecated. To summarize, ACE will be the backbone for the code editor field in the web application.

2.7. Parsing GDB responses and rendering it to Virtual DOM

As we stated earlier GDB or GNU Debugger is a powerful command-line tool used for debugging programs written in various programming languages, such as C, C++. When you use GDB to debug a program, it provides you with various types of output to help you identify and fix issues in your code.

Type	Description
Program output	GDB allows you to execute your program inside its environment, and it will show you the output generated by your program on the console.
Breakpoint output	When you set a breakpoint in your code, GDB will stop the program's execution when it reaches that point. It will then show you the values of the variables at that point and let you interact with the program.
Error messages	GDB will also provide error messages if there are any issues

	with your code, such as segmentation faults or undefined references.
Stack trace	If your program crashes or encounters an exception, GDB can provide a stack trace showing the sequence of function calls that led up to the error.
Memory contents	GDB can display the contents of memory locations, allowing you to inspect the values stored in variables or other data structures.

Table 1. GNU Debugger's output

Here the list of possible output types from GDB and as it can be noticed two of them highlighted with green. For tracing the status of variables, collections and in which function programs are we need Breakpoint outputs, Memory contents, and Stack tree respectively. Precisely, via those outputs we will render the visualization part of debugging, such as the variable box, its type, value stored in it, and its scope. If we have collections, such as array and loop users will be able to see which index currently the pointer is, what value in the array, and etc.

For the sake of demonstration let's assume that we have this particular C++ code that we will debug using our web application.

```
1 #include <iostream>
2
3 int main() {
4
5   int array[7] = {0};
6  int i = 0;
7
8  while (i < 7) {
9     array[i] = i;
10    i++;
11 }
12
13  return 0;
14 }
```

Figure 12. C++ reference code for debugging

We will omit the part of compiling and all the business logic regarding output capture and sending it back to the client, but focus on only GDB's responses and how we will render it using front end tools. If we start debugging from line 3 and onwards and step line-by-line, GDB will start outputting the current status of variable "i" and "array" collection and our purpose is to parse it and render it to a specific area on the web page.

On every step from line 3, we will execute "***info locals/variables***" and "***ptype*** *arg*" commands. They will output variable names and their associated values, and variable types in such way:

- array = {-12514496, 32767, 0, 0, 143219904, 143219904}
- i = 0
- type = int[7]
- type = int

Do not pay attention to initial values of array, these outputs are recorded in step 3, where those two variables will not even be declared & initialized. After every recording, we capture them and then format them in a specific way based on their data structures. For instance, for primitive data types we need format output in this way [type | scope | address | name | values] and simple collections such as array we need format output in such way [type| scope | address | name | [values]]. Thus, for line 3 the format will be following [int | main | address | i | 0]. However, gdb's output will be dependent on C++ code. If C++ code will have 10 variables of various types they must be sent to the client as one chunk of message. For that purpose, we will create an additional object that will represent all formatted outputs. Then, we will send this formatted output to the client for further parsing and rendering it to the web page. Parsing formatted outputs is the easiest thing among the mentioned steps we need to pass through. We will iterate through all outputs parse with disclaimer "|" and took advantage of ordering patterns of formatted outputs to know where types, scopes, addresses, names, and values are located. Then

we will render for each one of them appropriate figures on the web page with all mentioned information. Here when components of React.js will be mounted the React.js will update virtual DOM and Actual DOM behind the bars. We just need to obey the specification of developing by React.js. We will not go into details of React.js and its implementation, since it will be presented in the Chapter 4 of the thesis report.

2.8. Summary

From all those discussions we made, we can bring together and summarize statements and concepts we introduced. We constituted grounds of utilizing concepts concurrency, threads, and tools such as GCC, GDB and preference of using one network protocol over another.

To sum up, for compiling and running actual code we will first establish a WebSocket connection between server and client to open up message transmission. Secondly, the user will send an actual code in text representation to the server. Thirdly, the server will preprocess it to distinguish real code from user input, and then convey code to the service layer. Service layer then compiles it using Process object and Runtime object. Subsequently, threads will be kicked off and begin monitoring outputs from running C++ code. Finally, the thread will pass on the outputs to the client using the WebSocketSession object. As agreed, there is a second plot of what can happen during programming. C++ codes might contain user input, when it happens we must spot it and send a request to the client, and when client responds with user input, we will accept it on the server side. Preprocess will play its role and we will differentiate user input and send it to C++ code's standard input stream.

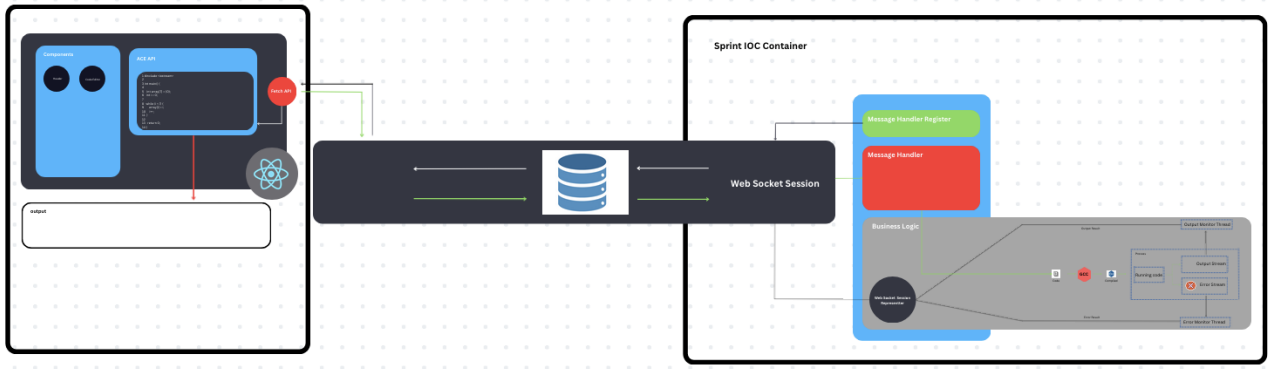


Figure 13. The layout of the compiling and running concept

On the client side code we are availing of frameworks and a third party API to display it on the code editor's output field.

For the debugging purpose, we will establish a brand new WebSocket connection for conveying messages. Yes, the message will be in the text form, and they will be formatted using a specific pattern which client side and server side are familiarized with. Then the client will send code with a signal of debugging and then the server side will accept it though a dedicated endpoint. Finally, a process which we described earlier will take part and constantly will be sending output results to the client for further rendering. The rendered components will precisely visualize each step and will bring a correct perspective of the state of the C++ code.

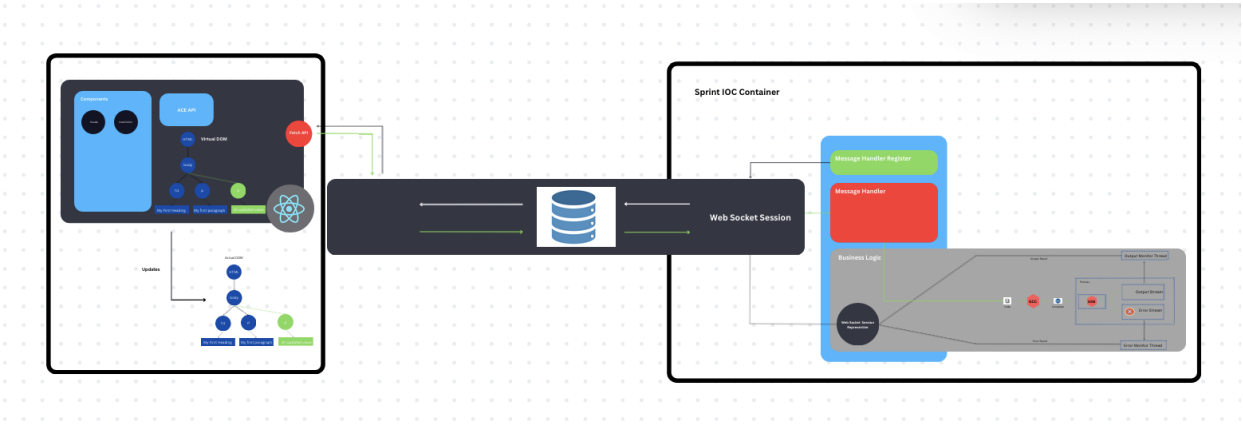


Figure 14. The layout of the compiling and debugging concept

Chapter 3 – Implementation of Spring Boot based Back End

3.1. Web Socket Configuration and MessageHandler registration

Before diving deep inside the implementation part, it is worth introducing a framework that will be the backbone of server side code and be in charge of many managing works related to package control, dependency control, building project, executable file etc. Spring Framework is an open-source application framework for Java that provides a comprehensive programming and configuration model for modern Java-based enterprise applications. The framework provides a wide range of features such as inversion of control, dependency injection, aspect-oriented programming, and many other useful tools for building enterprise-level applications. Spring framework consists of numerous projects like Spring MVC, Spring Boot, Spring Data JPA, Spring Cloud and components such as Spring Core. Each one of them is aimed to facilitate the development process and achieve implementing enterprise-level applications. Thesis would not light into details of Spring Framework and its tools, however let's mention that every application's needs can vary a lot. Thus, we will not utilize Spring Frameworks's all projects and components, rather some of them. They are Spring Core as its IOC container, Inversion of Control, Dependency Injection, and Autowiring are extremely helpful. Moreover, Spring Web and Spring Boot also will be used in order to tackle requests from clients and get the embedded tomcat server. Several dependencies ought to be mentioned too, such as spring-boot-starter-websocket and spring-boot-starter-test in order to be able to establish WebSocket connection and run unit tests after implementation is done. Quick disclaimer - we will not use Test Driven Approach for this application development since it was hard to predict how our application will be implemented back then.

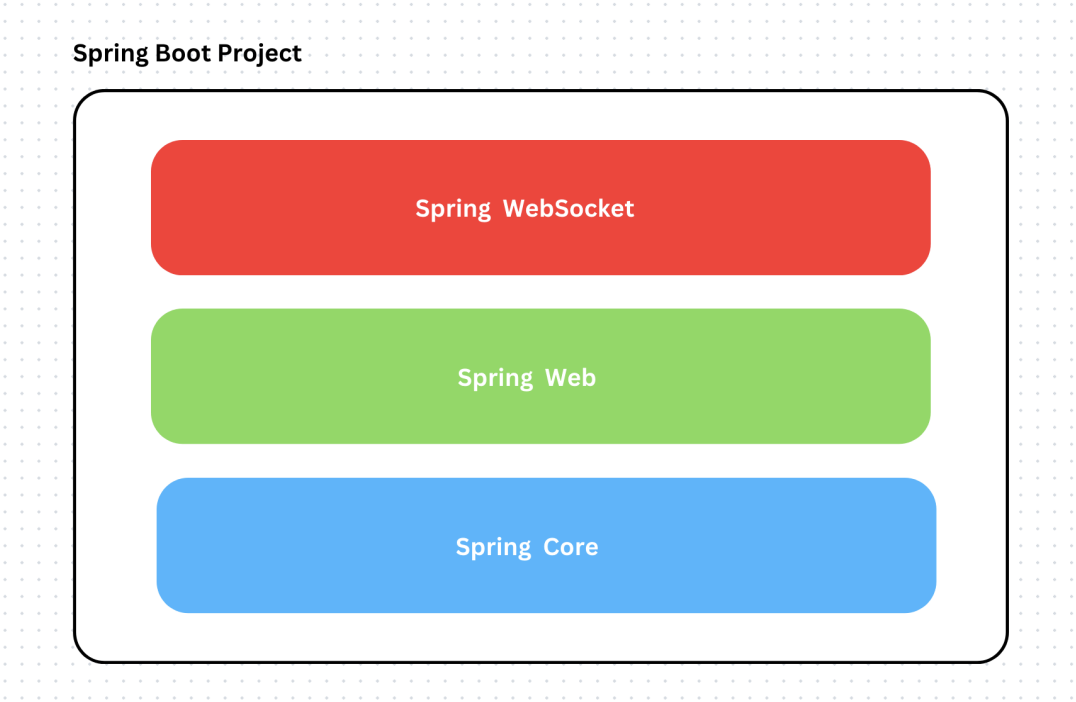


Figure 15. Spring Framework projects and dependencies

It must be mentioned that Spring's core concept is that everything is a bean and IOC container or Spring container manages life cycles of beans. As an example, web socket configuration and message handler classes are treated as beans. But, before creating our beans let's add dependencies we mentioned and depicted in the figure. Here as depicted in Figure №14 we included spring-boot-starter-web, spring-boot-websocket, spring-boot-starter-test. It might be intimidating to see how dependencies are added to a project, nevertheless it is quite structured via HTML like code and incredibly intuitive. This file is called pom.xml by the Maven project manager. Maven is a package and project manager that alleviates the process of adding and managing dependencies to projects, since it has its own database of one and compatibility of versions.

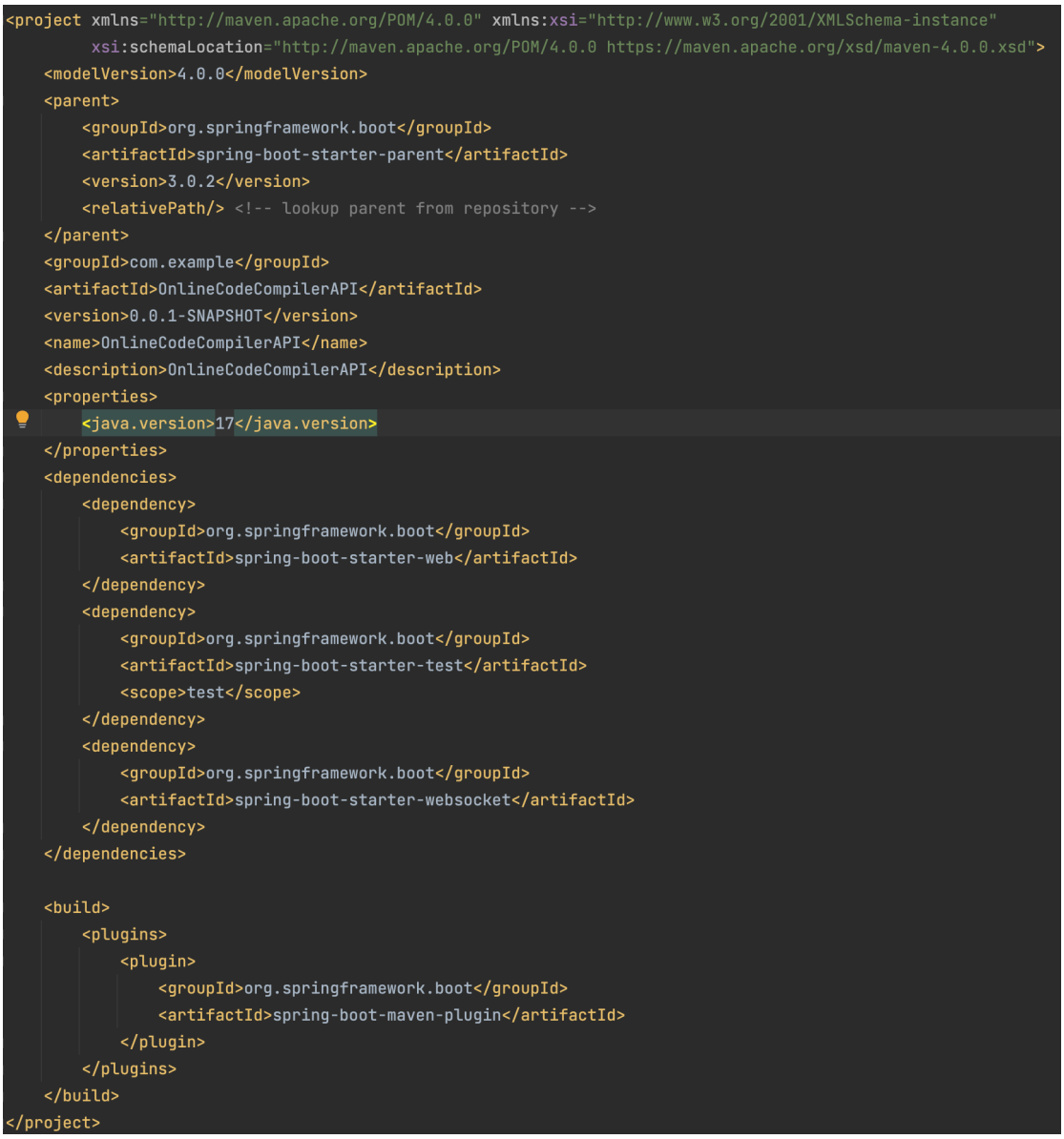
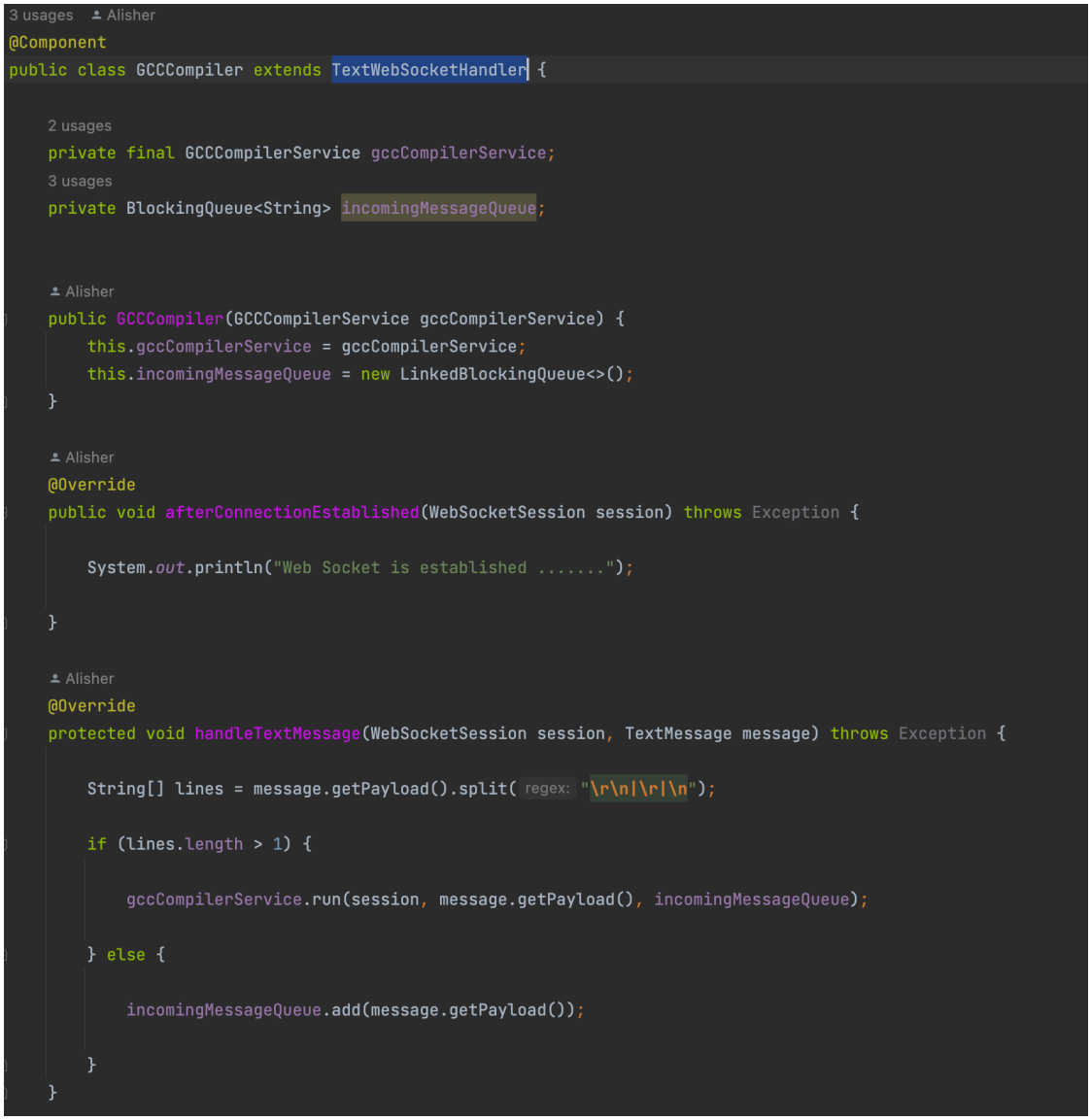


Figure 16. Maven's pom.xml file for managing dependencies in the project

Then, we can get into creating our very first beans. They are message handler class and web socket configuration classes. To handle messages received over a WebSocket connection, we need to register a message handler. A message handler is responsible for processing incoming messages and sending responses back to the client. First GCCCompiler class was created and extended TextWebSocketHandler class to implement several methods that will be responsible for processing incoming text messages coming from WebSocket sessions. Then, the crucial point is to annotate the class with @Component, to make it a bean and transmit it to Spring IOC Container to manage. Set of annotations were created in the Spring framework to work as

configuration mechanisms and substitute old-fashion XML files.



```
3 usages  Alisher
@Component
public class GCCCompiler extends TextWebSocketHandler {

    2 usages
    private final GCCCompilerService gccCompilerService;
    3 usages
    private BlockingQueue<String> incomingMessageQueue;


    Alisher
    public GCCCompiler(GCCCompilerService gccCompilerService) {
        this.gccCompilerService = gccCompilerService;
        this.incomingMessageQueue = new LinkedBlockingQueue<>();
    }

    Alisher
    @Override
    public void afterConnectionEstablished(WebSocketSession session) throws Exception {

        System.out.println("Web Socket is established .......");

    }

    Alisher
    @Override
    protected void handleTextMessage(WebSocketSession session, TextMessage message) throws Exception {

        String[] lines = message.getPayload().split( regex: "\r\n|\r|\n");

        if (lines.length > 1) {

            gccCompilerService.run(session, message.getPayload(), incomingMessageQueue);

        } else {

            incomingMessageQueue.add(message.getPayload());

        }
    }
```

Figure 17. Implementation of "GCCCompile" bean with TextWebSocketHandler

Here, the most important points to take note are the "GCCCompilerService" bean that is included as a property of the GCCCompiler bean and function "handleTextMessage". Service bean contains business logic for processing messages coming from "WebSocketSession" and function "handleTextMessage" that referencing service's method run. Moreover, it contains preprocessing logic in the face of separating user input messages from real code messages via determining how many lines of text message contains. Exactly the same way we can achieve implementation of GDBDebugger bean, but with customized preprocessor in handling incoming

messages.

Once we have implemented the WebSocketMessageHandler, we can register our message handler with the "Web Socket Handler Registry" by calling the "registerHandler" method. This method takes the Web Socket Handler instance as a parameter and specifies the URL path that the handler should be mapped to. By registering a message handler, we are essentially telling the WebSocket server which class should handle incoming WebSocket messages and how to map those messages to specific endpoints within your application.

```
Alisher
@Configuration
@EnableWebSocket
public class WebSocketConfiguration implements WebSocketConfigurer {

    2 usages
    private final GCCCompiler gccCompiler;
    2 usages
    private final GDBDebugger gdbDebugger;

    Alisher
    @Autowired
    public WebSocketConfiguration(GCCCompiler gccCompiler, GDBDebugger gdbDebugger) {
        this.gccCompiler = gccCompiler;
        this.gdbDebugger = gdbDebugger;
    }

    Alisher
    @Override
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
        registry.addHandler(gccCompiler,  ...paths: "/nu/editor/compile").setAllowedOrigins("*");
        registry.addHandler(gdbDebugger,  ...paths: "nu/editor/debug").setAllowedOrigins("*");
    }

}
```

Figure 18. Implementation of WebSocketConfiguration class

It can be seen that registerWebSocketHandlers registers two endpoints "/nu/editor/compile" and "nu/editor/debug" for gccCompiler and gdbDebugger beans respectively. Notably, two annotations are utilized above to class declaration. They are @Configuration and @EnableWebSocket annotations, and without any words it can be understandable that first annotation is used to set this class as configuration bean and second one to switch on web socket network protocol.

3.2. Business logic of compiling and running C++ code

As I mentioned in the previous section, we integrated a service layer which contains business logic for GCC compiling and running C++ code. In a RESTful API, the architecture is typically divided into three main layers: the controller layer, service layer, and data layer. However, we are not adhering to REST API architecture and not even exposing resources of the database to clients using HTTP, but using WebSocket network protocol. So, what is the relevance of the service layer to thesis web applications? Reason is that the service layer is present in every application, regardless of architecture. It must be considered as a name convention for server side code. Nevertheless, we are taking the term "service layer" for our utilization. Thus, the service layer is responsible for handling business logic and data processing. It performs operations such as data validation, authentication, authorization, and other domain-specific operations. It interacts with the data layer to retrieve and manipulate data as needed. Service layer components are typically implemented as stateless beans in Spring Framework. However, it must be stated that we are not dealing with any kind of databases and data validations yet. Business logic in our service layer is for compiling and running C++ code using GCC and mentioned in the above introduction paragraph Process class with Threads.

In order to compile code, first we necessitate code itself from the client. As perceived in the above section, the server receives code in textual representation. Then, the controller layer, in quotation marks, sends textual representation with the session object that constitutes WebSocket session to the service layer. In the service layer, we create a new file with .cpp extension and then using a buffered writer we insert code to the created file. Ready to use C++ code we then compile using good-old GCC. However, how we can access the capabilities of GCC, it is only available using a terminal on our local machines. Java created a laconic method to make this possible. Process class in combination with a Runtime object we enclose our executable code to some kind of box and gain access to the terminal respectively. Then using this Runtime object we run on terminal "***g++ -g main.cpp -o executable***" command which produces compiled code.

Process instance has build-in methods that signals us status of the command that we run. One of them is the integer result of the "***waitFor()***" method, which "0" means the command runs successfully and "1" means Runtime object encountered a problem while executing the mentioned command.

```
String result = "";

try {

    File main = new File( pathname: "main.cpp");

    BufferedWriter writeCodeToFile = new BufferedWriter(new FileWriter(main));

    writeCodeToFile.write(code);

    writeCodeToFile.close();

    //------------------- Compile Code ------------------//

    Process process = Runtime.getRuntime().exec( command: "g++ -o compiled main.cpp");

    int execCode = process.waitFor();
```

Figure 19. Implementation of compilation of the C++ code

Consequently, if we have gotten a result of "0" which means we have executable code in our hand and we are ready to go straight to running the code. In order to achieve running code executable, Process and Runtime objects must be available once again, but with "/executable" command. As displayed in Figure №18, there are several other codes related to ExecutableService and Threads. However, we will not define them in this paragraph; it must be stated that they will monitor outputs and send inputs to the C++ code that is running inside the process object.

```
    //------------------- Run Code -------------------//

    if (execCode == 0) {

        process = Runtime.getRuntime().exec( command: "./compiled");

        ExecutorService executorService = Executors.newFixedThreadPool( nThreads: 2);
        OutputStreamMonitorThread userInputAndResult = new OutputStreamMonitorThread(session, process, incomingMessageQueue);
        ErrorStreamMonitorThread errors = new ErrorStreamMonitorThread(session, process, incomingMessageQueue);

        executorService.execute(userInputAndResult);
        executorService.execute(errors);

        executorService.shutdown();

    }

} catch (Exception error) {

    error.printStackTrace();

}
```

Figure 20. Implementation of running of the C++ code

It might not be distinguished right away, but all those logic of "GCCCompiler" service are wrapped inside the try catch statement. Precisely, try catch statements are necessary for exception handling in case we have code that does not meet requirements of C++ code. Thus, we handle this exception using a high level Exception class and for now just printing errors.

3.3. Separate Threads to monitor Inputs/Outputs of GCC

As we discussed in previous sections, concurrency is an exceptional concept that fits for precisely this task. We have created two threads for reading standard output and errors of a program that is running and managed them altogether through the ExecutorService class as it can be noted in Figure №18. Let's dive into details of those threads and implementations of them.

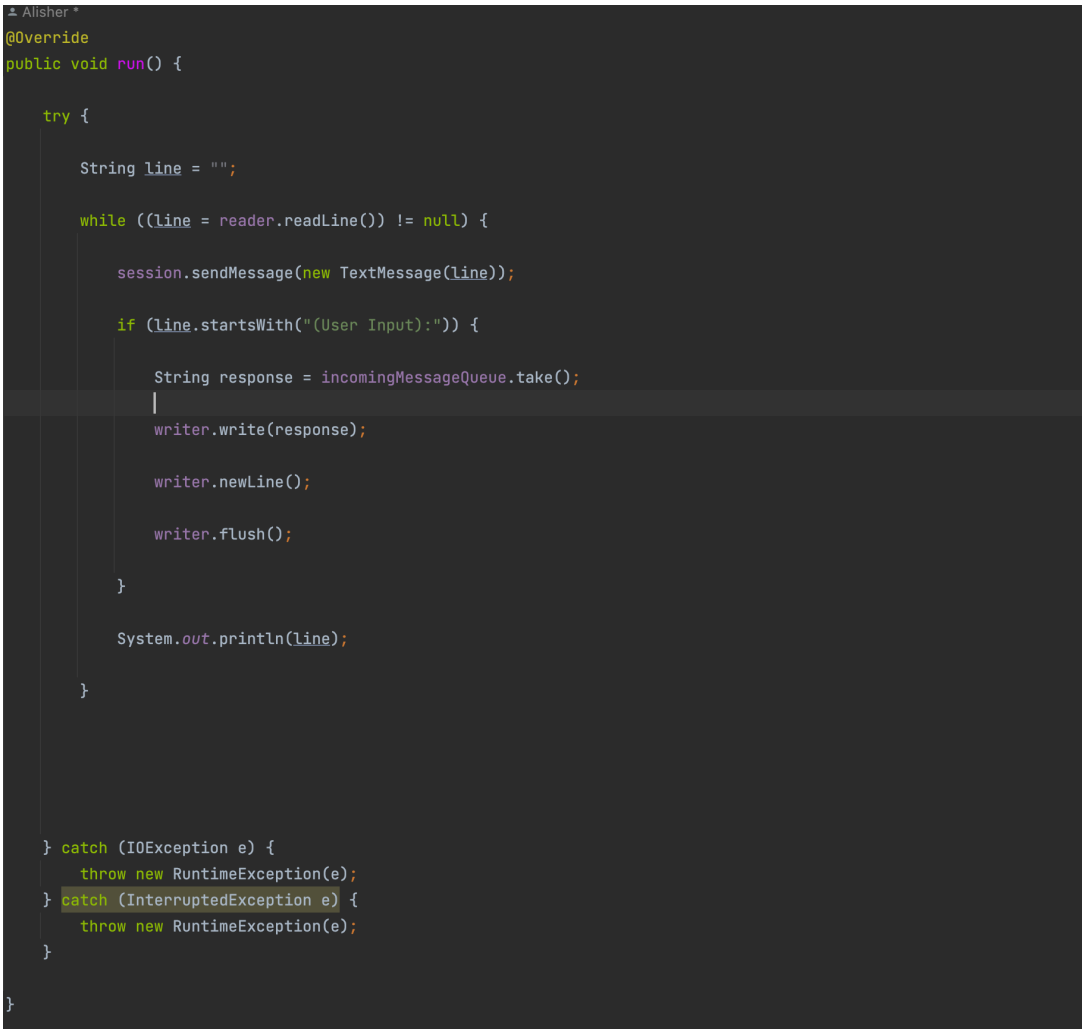


Figure 21. Implementation of the "OutputStreamMonitorThread" thread

Here in the code snippet we omitted declaration of the thread, however it must be said that every brand new thread ought to extend the Thread class which is kindly provided by Java. Then, in the constructor we accepts following dependencies:

```
1. private final WebSocketSession session;
2. private final Process process;
3. private final BlockingQueue<String> incomingMessageQueue;
4. private final BufferedReader reader;
5. private final BufferedWriter writer;
```

"WebSocketSession" and Process classes, as we stated numerous times, represent web socket sessions and separate boxes where code is running. BufferedReader and BufferedWriter

were considered earlier in the implementation of the service layer. They are in charge of reading and writing to the C++ code streams that are running inside the process object efficiently. Nevertheless, almost all of them we sighted previously, BlockingQueue is quite an interesting data structure and its reasons for using it are even more captivating.

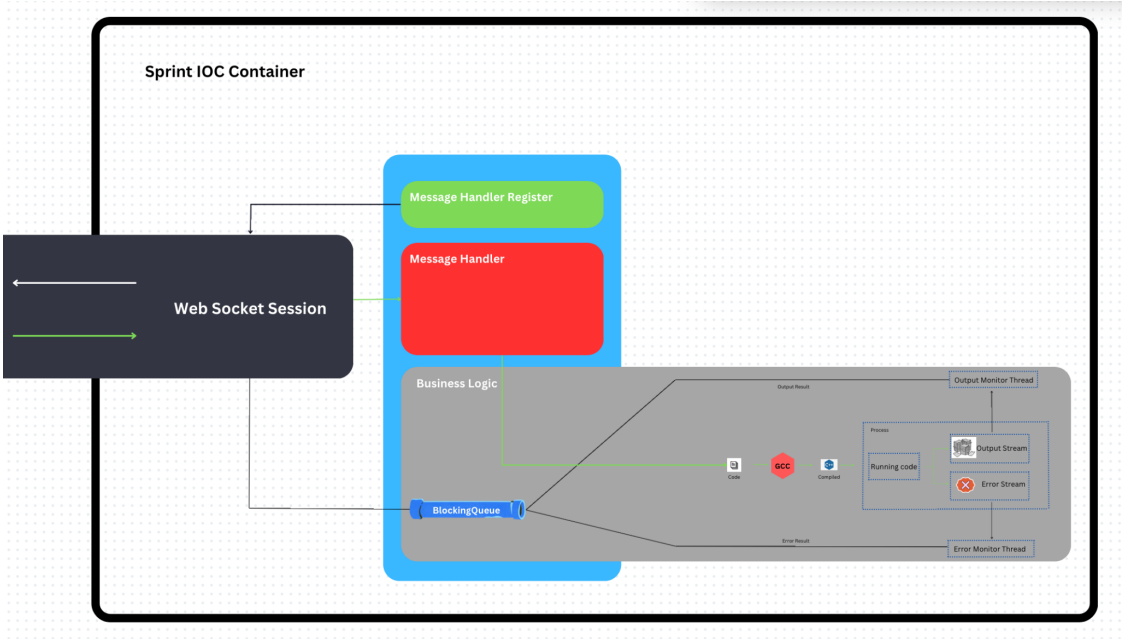


Figure 22. The aim of using the BlockingQueue data structure

When a client sends a code that contains a user input thread must act differently rather than just sending output to client, it must request client for user input. We can send output of the result and request for user input through WebSocketsSession's "***sendMessage()***" method, however when client respond to server it will receive it through "***handleTextMessage()***" in controller layer. As we discussed we have a preprocessor that distinguishes real code from user input based on the lines of received text message. Then, distinguished user inputs will be inserted to BlockingQueue and will be sent to the service layer and then to the thread. Finally, using buffered writer user input will be transited to C++ code's input stream. Absolutely, it is tedious at first sight, however this comprehensive approach is dictated by disadvantages of WebSocket dependency of Spring framework. We add and immediately delete after transmitting user input to C++ code in order to keep user input up to date for each user input request from

the server.

3.4. Business logic of debugging C++ code

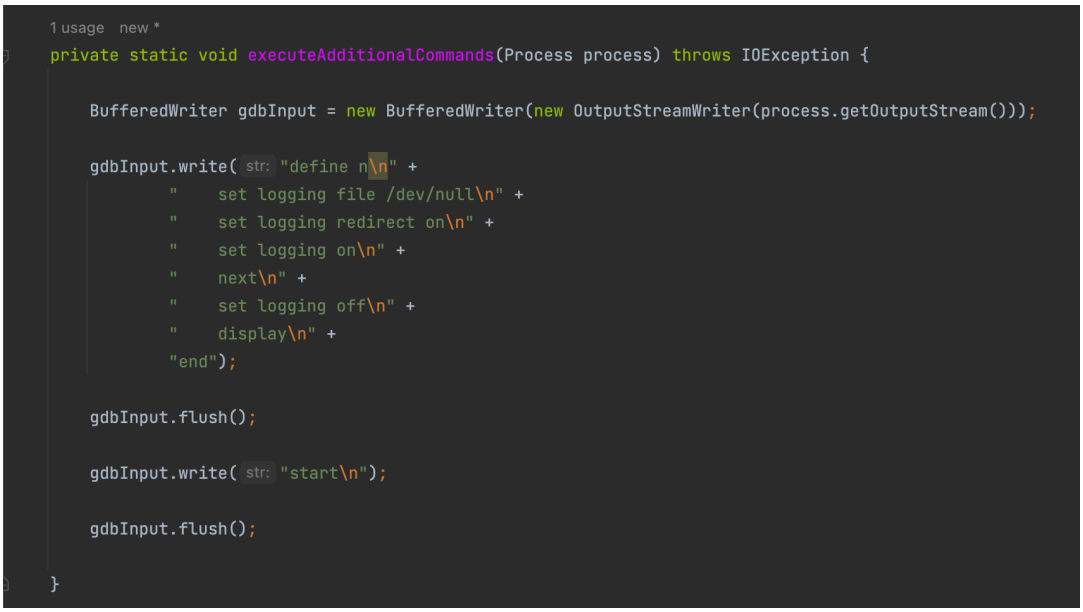
In the code snippet that will be provided below, I will explain each line of the code in the service layer.

```
Alisher *
public void debug (WebSocketSession session, String code) {

    try {

        File main = new File( pathname: "main.cpp");

        BufferedWriter writeCodeToFile = new BufferedWriter(new FileWriter(main));

        writeCodeToFile.write(code);

        writeCodeToFile.close();

        //------------------- Compile Code -------------------//

        Process process = Runtime.getRuntime().exec( command: "g++ -g main.cpp -o compiled ");

        int execCode = process.waitFor();

        //------------------- Run Code -------------------//

        if (execCode == 0) {

            process = Runtime.getRuntime().exec( command: "gdb -q compiled");

            //how to execute "start" command on GDB
            executeAdditionalCommands(process);

            ExecutorService executorService = Executors.newFixedThreadPool( nThreads: 2);

            DebugOutputMonitorThread outputThread = new DebugOutputMonitorThread(session, process);

            DebugInputSenderThread inputThread = new DebugInputSenderThread(session, process);

            executorService.execute(outputThread);

            executorService.execute(inputThread);

            executorService.shutdown();

        }

    } catch (Exception error) {
```

Figure 23. Implementation of compiling and debugging C++ code

This code defines a Java class called "GDBDebuggerService" that contains a debug method which takes in a WebSocketSession and a string code. The purpose of this method is to compile and debug the code using GDB (GNU Debugger). Firstly, the method creates a File object with the name "main.cpp" and writes the input code to this file using a BufferedWriter. It

then compiles the code by running the command "***g++ -g main.cpp -o*** *compiled*" using the "***Runtime.getRuntime().exec()***" method. If the compilation is successful, the method starts GDB with the compiled code using the command "***gdb -q*** *compiled*". The "-q" flag is used for a "quiet" start of the debugging process, meaning to escape unnecessary welcome messages from GDB. It then executes some additional GDB commands using the private method executeAdditionalCommands, which defines a custom GDB command n that switches off logging options and then executes the next command.



```
1 usage  new *
private static void executeAdditionalCommands(Process process) throws IOException {

    BufferedWriter gdbInput = new BufferedWriter(new OutputStreamWriter(process.getOutputStream()));

    gdbInput.write( str: "define n\n" +
            "    set logging file /dev/null\n" +
            "    set logging redirect on\n" +
            "    set logging on\n" +
            "    next\n" +
            "    set logging off\n" +
            "    display\n" +
            "end");

    gdbInput.flush();

    gdbInput.write( str: "start\n");

    gdbInput.flush();

}
```

Figure 24. Additional configuration for GDB.

The method then creates two threads, one for reading the output of the GDB process (DebugOutputMonitorThread) and another for sending input to the process (DebugInputSenderThread). It uses an ExecutorService with a fixed thread pool size of 2 to manage these threads. The DebugOutputMonitorThread reads the output of the GDB process and sends it to the WebSocketSession using a TextMessage, while the DebugInputSenderThread sends input commands to the process by reading from the WebSocketSession. Finally, the method shuts down the ExecutorService and catches any exceptions that may occur during the process.

3.5. Separate Threads to monitor Inputs/Outputs of GDB

Implementation of threads for interacting with GDB is slightly different. We still need to implement "***run()***", since we override it to launch Threads themselfs. Here is a high level overview of algorithms that we will utilize.

In the service layer, we also need to run "start" and initial "local info". This will make GDB set a temporary break at "int main() {" line and will check what local variables were initialized. Then the output thread will skip 4 lines, because after executing "next" GDB will output information about the temporary break and addresses in the memory that were allocated to variables. Then the output thread will start capturing outputs "start", an initial "local info" and subsequent "next" and "local info". However, thread must distinguish local info outputs from next outputs. Pattern is following so that when input thread will execute "next" it will output in "line_ number code_itself" pattern, but when input thread will execute "info locals" the pattern will be in "variable_name = value". Thus, we can check only if output starts with a number or not to decide what we captured. Based on this decision we will send a line to the client or we will further process outputs. The last case, the output thread needs to capture all the outputs from "info locals" and then store its arrays. Then the thread will iterate through captured elements and on each one of them will execute the "ptype" command to find out type. Eventually with all this information in hand it will store it in a special instance of class called "GDBOutputFormatter.class". This class is in charge of storing values and formatting them in the pattern that was discussed in the previous section. After formatting the values we send them to the client for further parsing.

Chapter 4 – Implementation of Reac.js based Front End

4.1. Functional component for header

Here we introduce for the first time in this thesis, the client side implementation. This is a simple React functional component called "Header". The component exports a function that returns a JSX element that will be rendered on the webpage. In this specific example, the function returns a header element with a div element containing a paragraph element with the text "Online Code Editor". The import statement at the beginning of the code imports the React library and the "index.css" file, which contains styles for the component.

```
1   import React from "react";
2   import "./index.css";
3
4   export function Header() {
5
6
7
8       return (
9           <header>
10              <div>
11                  <p>Online Code Editor</p>
12              </div>
13          </header>
14      );
15
16  }
```

Figure 25. The header implementation of web page using React.js

When the page is rendered, it will show the header with the text "Online Code Editor" followed by the content of the page.

4.2. Functional component for main part

4.2.1. States using UseState hooks

In React.js, state management is a fundamental concept that enables the creation of dynamic and interactive user interfaces. The useState hook is a built-in feature of React.js that provides a way to add state to functional components. It allows developers to store and update data within a component, triggering a re-render of the component when the state changes. By

using useState, developers can create components that are more modular and reusable, as state can be passed down as props to child components. Additionally, useState simplifies the process of handling user input and application logic, reducing the amount of code needed to manage state changes. Overall, useState is a key tool for developing complex, data-driven applications in React.js, and helps to streamline the development process by providing a simple and intuitive way to manage state within functional components.

In our case, use utilize useState hooks to store states of following aspects of the component called "ACE":

State Name	State Description
socket	This state variable holds a WebSocket connection to the server for sending and receiving data.
code	This state variable holds the C/C++ code that is displayed in the editor.
userInput	This state variable holds the user input that is required by the C/C++ code (if any).
output	This state variable holds the output generated by the compiled code.
isDisabled	This state variable is used to disable the "Enter" button until the user provides input.

Table 2. Stats of the "Ace" functional component.

So, to integrate this concept into our code we simply need to import it from the "react" framework.

```
import AceEditor from 'react-ace';
```

Then, we can use in our functional component called "ACE" like this:

```
export function Ace() {
    const [socket, setSocket] = useState(null);
    const [code, setCode] = useState("");
    const [userInput, setUserInput] = useState("");
    const [output, setOutput] = useState("");
    const [isDisabled, setIsDisabled] = useState(true);
```

Figure 26. Implementation of states of the "ACE" functional component.

Here, we also defined the initial value of each state when the component will be mounted. Despite the fact code's initial state represented as empty array, we define later value of code's initial value as this:

```
`#include <iostream>
 int main() {
     std::cout << "Hello, world!" << std::endl;
      return 0;
}`;
```

This will initial state for the code editor field and when the component will be mounted code editor will display this code snippet from which point users can begin coding.

4.2.2. UseEffect after component is mounted

The useEffect Hook is a feature of the React.js library that enables developers to execute side effects within functional components. Side effects refer to interactions between the component and its surrounding environment, such as fetching data from an external API, modifying the DOM, or subscribing to events.

To use useEffect, developers pass two arguments to the Hook: a callback function and an optional array of dependencies. The callback function is executed after the component is rendered, and it can return a cleanup function that will be called before the component is unmounted. The dependencies array is an optional list of values that the callback function

depends on. If any of these values change, the callback function will be re-executed. Here is how our useEffect works for compiling and running:

```
useEffect(() => {

  const initialCode = `#include <iostream>

int main() {
  std::cout << "Hello, world!" << std::endl;
  return 0;
}`;

  setCode(initialCode);

  const connection = new WebSocket("ws://127.0.0.1:8080/nu/editor/compile");

  connection.onopen = () => {

    console.log("Web Socket Connection is established");

  }

  connection.onmessage = (event) => {

    const data = event.data;

    if (data.startsWith("(User Input): ")) {
      //make button active
      setIsDisabled(false);
    }

    setOutput(prevOutput => prevOutput.length > 0 ? `${prevOutput}\n${data}` : data);

    console.log(output);

  }

  connection.onerror = (event) => {

    console.log(event.target);

  }

  setSocket(connection);

  return () => {

    connection.close();
    setSocket(null);

  }

}, []);
```

Figure 27. useEffect hook to compile and run purposes

This is a code snippet that makes use of the useEffect hook in React. The purpose of this code is to set up a WebSocket connection, listen for messages sent over the connection, and update the state of the component based on those messages. The useEffect hook takes two arguments: a callback function and an array of dependencies. In this case, the array of dependencies is empty, which means that the callback function will only run once, when the

component mounts. Within the callback function, a constant initialCode is defined that contains a C++ code snippet. The setCode function is then called, which updates the component's state with this code snippet. Next, a new WebSocket connection is created by calling the WebSocket constructor with the URL of the server to which the connection will be made. The onopen, onmessage, and onerror event listeners are then defined for the connection. The "onopen" event listener simply logs a message to the console indicating that the WebSocket connection has been established. The "onmessage" event listener is called whenever a message is received over the WebSocket connection. The data property of the event object contains the message that was received. If the message starts with the string "(User Input): ", the setIsDisabled function is called with an argument of false, which presumably enables a "send" button. The setOutput function is then called with a callback function that takes the previous value of the output state variable and appends the new message to it. The final step is to log the updated value of the output state variable to the console. The "onerror" event listener is called whenever there is an error with the WebSocket connection. In this case, the function simply logs the event target to the console. Finally, the setSocket function is called with the connection object as an argument. This updates the component's state with the WebSocket connection. The useEffect hook also returns a cleanup function that is called when the component is unmounted. This function closes the WebSocket connection and sets the socket state variable to null.

4.2.3. Built-in browser based Web Socket API

The built-in browser-based WebSocket API is a feature provided by modern web browsers that enables client-side JavaScript code to establish and maintain a persistent, bidirectional communication channel with a server over a single TCP connection. The WebSocket API allows for real-time, low-latency data transfer between the client and server, making it ideal for use cases such as the code editor for running C++ code. The WebSocket API provides a simple and straightforward programming interface that consists of a few key classes

and methods. The WebSocket class is used to create a new WebSocket object, which can then be used to open a connection to a server and send and receive messages. The WebSocket object also provides several event handlers that can be used to handle various events related to the WebSocket connection, such as the "onopen", "onmessage", "onerror", and "onclose" handlers.

To establish a WebSocket connection, a client-side script must first create a new WebSocket object and pass the URL of the server to which it wishes to connect as an argument. Once the WebSocket object is created, the client can send messages to the server by calling the send() method, and receive messages from the server by handling the "onmessage" event.

4.2.4. ACE API integration to React.js project

The integration of ACE API into React.js project is fairly straightforward. All it needs to download a form of ACE for React application into the application using a terminal with project location specified. Package is called "react-ace" and to download it uses the command "***npm install react-ace***". Then we just imported it using the "import" statement " ***import AceEditor from 'react-ace'*** ". In addition to importing the ace editor itself, we also imported themes and necessary C++ language managing tools. Eventually, it is ready to use by referencing it as AceEditor in our JSX field.

```
import React, {useState, useEffect, useRef} from 'react';
import AceEditor from 'react-ace';
import 'ace-builds/src-noconflict/';

import 'ace-builds/src-noconflict/mode-c_cpp';
import 'ace-builds/src-noconflict/theme-monokai';

import "ace-builds/src-noconflict/theme-github";
import "ace-builds/src-noconflict/ext-language_tools";
```

Figure 28. Import AceEditor from "react-ace" package.

Since we are utilizing functional components, we need to return JSX and one will contain all HTML like language code, including ACE and its configuration using attributes.

```
return (
    <div className="main">
      <div className="editor">
        <AceEditor
            mode="c_cpp"
            theme="monokai"
            enableSnippets={true}
            editorProps={{ $blockScrolling: Infinity, enableDebug: true }}
            value={code}
            onChange={setCode}
            name="code-editor"
            width="70%"
            height="60%"
        />
        <div className="buttons">
          <button className="run" onClick={handleSendCode}>Run</button>
        </div>
        <AceEditor
            mode="c_cpp"
            theme="monokai"
            value={output}
            name="output-editor"
            width="70%"
            height="25%"
        />
        <div className="userInput">
            <input type="text" name="input" placeholder="Enter User Input, If code required to do so:" onChange={hanldeChangeUserInput}/>
            <button id="submit-button" type="submit" onClick={handleSendUserInput} disabled={isDisabled}>Enter</button>
        </div>
      </div>
    </div>
);
```

Figure 29. AceEditor element inside JSX with attributes configuration

4.3 Implementation of parsing GDB responses and rendering

In the back-end logic we processed and formatted output from GDB and sent it to the client. Formatted output was in String representation. The obligation of client side code is to parse it using a formatted pattern and store it somewhere. We know from the React specification that data is the state of the component. Accordingly, we need to create a new state using the "useState()" hook. State will contain an array with a custom object. About custom objects we told numerous times throughout the thesis. It will have patter [type| name | value]. Then parsed and stored values must be rendered. It can be rendered using the "forEach" statement with components inside it. For that we need to create a special component that will contain all that information such as type, name and value. Although rendering is pretty simple, styling will cause headaches, since we need to consider distinction in type. Arrays can not be styled as primitive data types. Moreover, we need to consider responsiveness of the components, since they must be appealing and concise.

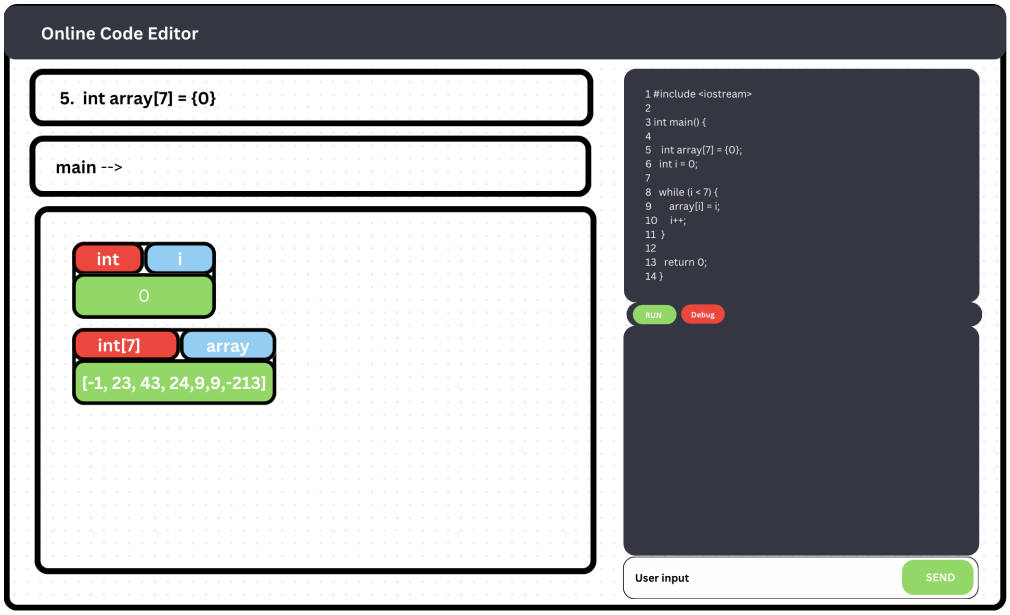


Figure 30. Web application representation during debugging

In the first bucket, we will have the line in which GDB currently is. Below that bucket, we have scope in which program is. Then, we have visualization itself. As you can see, we have two variables, one for variable "i" and one for an array named "array". The Array has random initial values, since GDB is currently in line 5 which as depicted it declares and initializes the array, however variable "i" is already initialized with value 0.

Chapter 5 – Results & Discussion

Aim and objectives we claimed to achieve were alleviating the process of studying programming concepts through elimination overhead of installing and configuring additional tools to commence studying, and provide novices with excellent visualization of debugging process. By the result of the conducted survey, we have found the problem definitely present. To achieve those goals, the thesis proposed and implemented full stack web applications which presumably meet the needs of newbie students. Since students are not required to install anything but are only required to open a web application in the browser, we do not need to demonstrate that the first challenge was successfully completed. We achieved it via processing all requests centralized in one machine, where we have installed GDB and GCC.

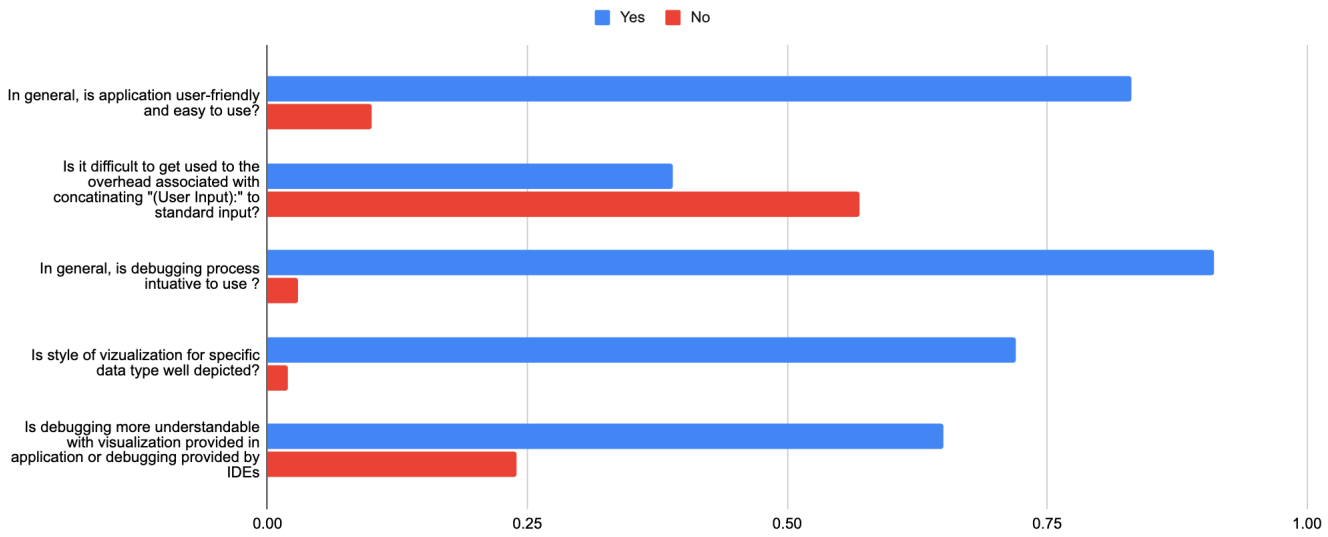


Figure 31. Result of the survey after user's hands on

For confirmation of solving the second challenge, we have conducted a survey in which 10 volunteers took part among engineering departments. 10 students tested the implemented application on codes with limited features and after provided their feedback. Code was with limited features, because application's debugging could only process certain types of data types. Overall, collected data confirms that the problem was resolved, however there is space to grow.

We can plainly see that 91% of users reported that debugging is straightforward in comparison to the 50.6% of users who previously claimed that debugging via an IDE is intelligible in question No. 3 of Figure 24 and Table 4. In question No. 5 depicted that users inclined to utilize debugging via visualization instead of debugging with IDE.

№	Question	Yes	No	Responded differently
1	In general, is application user-friendly and easy to use?	83.10%	10%	3% - need to get used to debugging process, 3.9 - inconvinient to use standard input
2	Is it difficult to get used to the overhead associated with concatinating "(User Input):" to standard input?	39%	57%	4% - inconvinient to use at first, but easy to get used
3	In general, is debugging process intuative to use ?	91%	3%	6% - uderstandable, but hard to get used to different format of debugging
4	Is style of vizualization for specific data type well depicted?	72%	2%	10% - address field it exessive, 4% - need to add scope for variable 2% - need to fine grain page responsivness
5	Is debugging more understandable with visualization provided in application or debugging provided by IDEs	65%	24%	11%
6				

Table 3. Result of survey after user's text

In our study, we solicited feedback from users on various aspects of their experience. We allowed comments for each question to fully capture their insights. Of the total respondents, 6% of experienced users reported difficulties adjusting to a new debugging format. It is worth noting, however, that this is a small proportion of the sample. Our survey also revealed shortcomings in data visualization, as some information was deemed excessive while other relevant data was lacking. In addition to debugging, users expressed concerns about the process of coding standard input, which requires a special flag "(User Input):" to identify input lines. Furthermore, students made themself clear that lack of other data types and collections rendering during debugging could reveal more issues. Overall, users provided positive feedback about their experience, and we can confidently assert that we are headed in the right direction with the valuable feedback received.

Chapter 6 – Conclusion and Future Work

In this thesis, we have analyzed problems that arose among students during grasping programming concepts in 2 undergraduate courses at Nazarbayev University. The problems were associated with understanding the debugging process, how it was used during class, and finally redundant work which must be done before commencement of classes. After discussing with authorized people and students, the thesis followed towards resolving stated issues by interactive web application with visualization of debugging process. In subsequent sections we have researched the impact of visualization on the learning process, analyzed how visualization could be helpful toward understanding programming concepts, and discussed several implementations. Thesis highlighted their advantages, drawbacks, relations to our work, and possible takeaways. From the collected information and careful research on possible technologies, the thesis introduced the architecture of the web application, network protocols, programming concepts, frameworks and business logic. Ultimately, the thesis delivered the implementation of web application with rigorous explanations. As mentioned, in order to test the minimum viable product, the thesis invited 10 volunteers from the engineering department at Nazarbayev University and beyond. Survey gave valuable results with positive feedback and revealed defects. Based on those defects and takeaways from analyzed literature, we enclose the following. The results say the application ought to implement the rest of data types and structures other than integer and array. It must be processed by GDB and rendered by the front-end to give full ability to its users. Additionally, survey dictated that the application have responsiveness issues when rendered components become more than screen can comprise. Code could have standard inputs and survey has shown that users feel it is inconvenient concatenating every time a specific flag that is not part of C++ language. Another issue which is not stated by users, nevertheless it is worth mentioning is performance issues. In each web socket session beans in Spring Context must have only one instance of the bean. Spring allows us to accomplish

it using additional annotation configuration. At the current level of the development, creation of the beans has not been analyzed yet. Moreover, applications require to be tested in both levels, such testing units for localized functionality and integration test that tests for proper interaction of components of the application. This was all about future work regarding the implementation part, however there is hand on testing that must be accomplished. We tested a minimum viable product with a limited number of recipients, however in literature "Towards Understanding Interactive Debugging" authors introduced an interesting method for observing effectiveness of their object of research. The authors picked 5 freelancers and 2 students and tested debugging tools. They provided 19 bag localization and collected through data of 110 breakpoints and 7000 invocations. Then they analyzed correlation with time spent on each task with consideration of the user's level of experience and number of breakpoints in order to find effectiveness of the tool. In this thesis, we could compare old fashion debugging with IDE with debugging with visualization, by defining proper metrics such as time spent on finding predefined bugs. Moreover, with consideration of the user's experience.

To conclude, we have implemented a minimum viable product and conducted preliminary tests. Moreover, we have introduced a space for developing products further.

Bibliography

[1] R. Garcia, C. -J. Liao and A. Pearce, "Read the Debug Manual: A Debugging Manual for CS1 Students," *2022 IEEE Frontiers in Education Conference (FIE)*, Uppsala, Sweden, 2022, pp. 1-7, doi: 10.1109/FIE56618.2022.9962675.

[2] F. Petrillo, Z. Soh, F. Khomh, M. Pimenta, C. Freitas and Y. -G. Guéhéneuc, "Towards Understanding Interactive Debugging," *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Vienna, Austria, 2016, pp. 152-163, doi: 10.1109/QRS.2016.27.

[3] Mayer, Richard E. and Roxana Moreno. “Nine Ways to Reduce Cognitive Load in Multimedia Learning.” *Educational Psychologist* 38 (2003): 43 - 52.

[4] F. Petrillo, Z. Soh, F. Khomh, M. Pimenta, C. Freitas and Y. -G. Guéhéneuc, "Towards Understanding Interactive Debugging," *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Vienna, Austria, 2016, pp. 152-163, doi: 10.1109/QRS.2016.27.

[5] Badiozamany, Sobhan and Haozhu Wang. “Debugging : the Difference between Novices and Experts.” (2010).

[6] A. Li, M. Endres and W. Weimer, "Debugging with Stack Overflow: Web Search Behavior in Novice and Expert Programmers," *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, Pittsburgh, PA, USA, 2022, pp. 69-81, doi: 10.1145/3510456.3514147.

[7] L. Gugerty and G. Olson. 1986. Debugging by skilled and novice programmers. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '86). Association for Computing Machinery, New York, NY, USA, 171–174. https://doi.org/10.1145/22627.22367

[8] A. More, J. Kumar and R. V. G., "Web Based Programming Assistance Tool for Novices," *2011 IEEE International Conference on Technology for Education*, Chennai, India, 2011, pp. 270-273, doi: 10.1109/T4E.2011.55.

[9] A. More, J. Kumar and R. V. G., "Web Based Programming Assistance Tool for Novices," *2011 IEEE International Conference on Technology for Education*, Chennai, India, 2011, pp. 270-273, doi: 10.1109/T4E.2011.55.

[10] A. Ko and B. Myers, "Debugging reinvented," *2008 ACM/IEEE 30th International Conference on Software Engineering*, Leipzig, Germany, 2008, pp. 301-310, doi: 10.1145/1368088.1368130.

[11] Richard E. Mayer & Roxana Moreno (2003) Nine Ways to Reduce Cognitive Load in Multimedia Learning, Educational Psychologist, 38:1, 43-52, DOI: 10.1207/S15326985EP3801_6

Appendix