Lappeenranta-Lahti University of Technology

School of Engineering Science

Software Engineering

Master's Program in Software Engineering and Digital Transformation

Hassan Shahbazi

Design and develop an environment for uploading and executing Chatbots using WebAssembly and WASI

Examiners: Professor Kari Smolander

Supervisors: Professor Kari Smolander

M.Sc. Jari Jaanto

ABSTRACT

Lappeenranta-Lahti University of Technology
School of Engineering Science
Software Engineering
Master's Program in Software Engineering and Digital Transformation
Master's Thesis 2021

Hassan Shahbazi

Design and develop an environment for uploading and executing Chatbots using WebAssembly and WASI

76 pages, 34 figures, 7 tables

Examiners: Professor Kari Smolander

Keywords: online services, language-independent, cloud API, chatbot, WebAssembly, WASI, Wasm

With the COVID-19 epidemic spreading across the globe in 2020-2021, Ninchat, being an expert in digital communication products, decided to improve its products even further and implement new features. The purpose of the project was to build a working reliable and performant chatbot environment for current and potential customers. The product should be sufficiently maintainable and scalable that it can be integrated with the current Ninchat's infrastructure. In this thesis, we discuss the best accessible approaches and choosing WebAssembly as an overall design for the described product. By the end of the thesis, the interfaces have been defined and the product has been assessed against technical and business requirements. The thesis succeeded in completing a practical final product, contributing not only to Ninchat but also to the Wasm-WASI open-source community. The final product, in addition to improving the researcher's practical knowledge and skills, can assist other researchers in developing similar applications.

ACKNOWLEDGEMENTS

My special thanks go to Professor Smolander for offering all guidance and feedback throughout the research, as well as for providing me with all the teachings I needed to submit the thesis. I would also like to thank Ninchat for all their support, without whom it would have been difficult to complete the project.

Finally, I want to thank my wife and my family for the unconditional love, support, and understanding they have shown me over the years.

Hassan Shahbazi
April 2021

Table of Contents

LIST OF SYMBOLS AND ABBREVIATIONS

ABI	Application Binary Interface
AI	Artificial Intelligence
AOT	Ahead-Of-Time
API	Application Programming Interface
ARC	Automatic Reference Counting
CIX	Commercial Internet Exchange
DSRM	Design Science Research Methodology
ECMA	European Computer Manufacturers Association
FAQ	Frequently Asked Questions
GC	Garbage Collection
HTTP	Hypertext Transfer Protocol
JIT	Just-In-Time
JSON	JavaScript Object Notation
ML	Machine Learning
NLP	Natural Language Processing
POSIX	The Portable Operating System Interface
RPC	Remote Procedure Call
SDK	Software Development Kit
WASI	WebAssembly System Interface
WASM	WebAssembly
WAT	WebAssembly Binary Text Representation

1 INTRODUCTION

1.1 Background

Since the internet era began, the idea of switching services to online alternatives gained a lot of attention. From 1991 with Commercial Internet eXchange (CIX), to now with Facebook and Twitter, many companies and governments have attempted to expand the range of online services available. Several countries, not only in the European Union but also on the four corners of the globe, working hard to increase their online offerings.

Year after year, the number of people with internet access is increasing steadily. 73% of people in Europe and Central Asia had access to the internet in 2016, which was almost 27% more than the world average. Still, the number of active internet users doubled in the past four years, surpassing 4.92 billion. (Roser, et al., 2015; Clement, 2020) The following chart illustrates the internet access rate in comparison to the world population, rocketing from 6.7% in 2000 to 63% in 2020. (Worldometer, 2020)

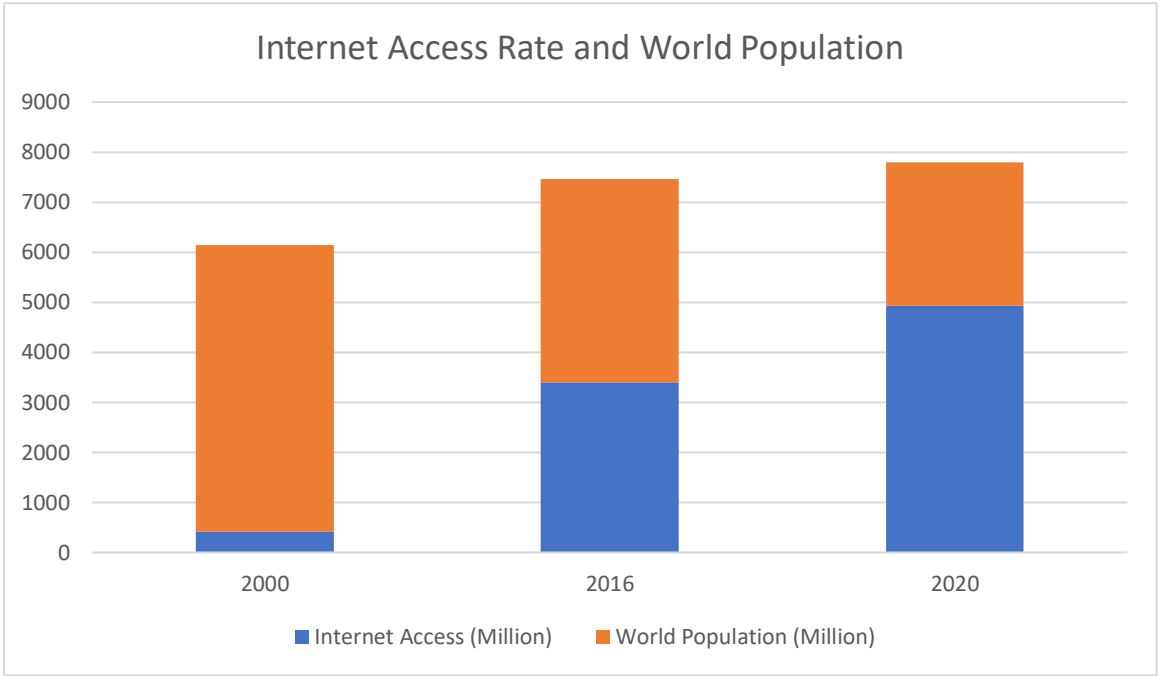

Figure 1 - Internet access rate since year 2000

Due to the COVID-19 epidemic that started in 2020 and has resulted in extensive and lengthy lockdowns in the European Union and the United States (AJMC, 2021), digital and remote

infrastructure has become a subject of increased interest among European and American. Although an untold number of services have been forced to stop on-site services due to the new conditions, online services have remained fully operational 24/7 with the ability to fulfill a large number of tasks simultaneously and accurately.

Finland, an internet-permeated country with the internet penetration of 95% (Kemp, 2020), offers a variety of online services such as the national postal service. Moreover, many private companies offer online services including but not limited to e-commerce, distance education, e-booking, and video consultation.

Ninchat, an award-winning startup based in Helsinki, is specialized in building digital products and infrastructures for internal team communications, customer relations, and live group discussions. Ninchat's web and mobile SDKs are widely used in different public and private sectors throughout Europe including Finnish healthcare (e.g. cities of Oulu and Rovaniemi health centers, health districts and private health service provides), Dutch Police, non-profit organizations such as Red Cross, etc.

Figure 2 - Ninchat team

Ninchat has provided a number of services to healthcare customers during the global epidemic such as remote appointments with doctors and nurses, video or chat counseling, products to manage time between numerous professionals, etc. (Sevon, 2020). However, with the growing demand for online services, the Ninchat management decided to explore offering a platform to host and execute also chatbots.

1.2 Goals, delimitations, and objectives

The thesis proposes a practical implementation for a performant environment that is integrated into the Ninchat infrastructures with regard to scalability, security, and maintainability. While the project is rather a complex software project, the thesis tries to explain and describe it in a way that non-technical stakeholders can also understand how the system works. By listing technical and non-technical requirements, explaining features, clearing the project's scope, addressing potential risks, and explaining internal and external components, the thesis aims to describe how the new service is initiated and controlled to communicate with Ninchat's internal services and external third-party APIs such as Amazon and Google.

Furthermore, since Ninchat's customers are the main users of the new product, the thesis intends to propose an intelligible interface for building bots compatible with the new product. The interface is clear, understandable, sufficient, and easy to translate into different languages. Also, the thesis illustrates how customers can utilize it to address technical requirements.

Likewise, this project is a great opportunity for bridging academic topics into high-quality real-world products. The first objective of the thesis is to offer the researcher the chance to gain practical experience and guide the academic courses on software design and quality assurance to a real-world project that is used by thousands. Additionally, considering the importance of documentation in software engineering that will enable future developers to maintain the project efficiently and with minimal effort by explaining the product's functionality, strengths and weaknesses, as well as unifying project-related information (AltexSoft, 2018), the thesis aims to facilitate updating the project in the future by insisting on documentation. In addition, the thesis provides a valuable tool for other researchers in order to conduct potential similar cases and would help them build products quickly and more reliably.

Some technical topics such as assessing the pros and cons of the technologies used or diving into technical details (such as how WebAssembly and WASI work on the kernel level) can easily exceed the scope and focus of the thesis. But the thesis provides a brief introduction to them and explains why they are being used for this project.

Most importantly, the thesis concepts are general, and they apply to similar cases in different contexts. Many uses are possible for WebAssembly. However, because Ninchat is the sole subject of this thesis and the fact that the new product would be hosted by Amazon, the implementations and designs presented in this thesis and the upcoming sections of this thesis may not be applicable to other designs and environments.

1.3 Research Questions

Without formulating research questions, developing knowledge wouldn't be possible. This is why defining these questions is one of the most important aspects of every kind of research. (Alvesson & Sandberg, 2013)

Previously discussed, and will be addressed in future chapters, Ninchat's thesis primarily focuses on building an environment that allows Ninchat's customers to submit and execute chatbots that act on behalf of an agent. The following research questions are defined in order to achieve the aims and objectives described in the previous section:

- **RQ1**: *How can customers build and submit chatbots to act on behalf of an agent?*
- **RQ2**: *How can Ninchat benefit from WebAssembly to provide a robust environment to run customers' Chatbots?*
- **RQ3**: *How different cloud APIs are accessible in the new environment?*

1.4 Structure of the thesis

The thesis paper consists of eight chapters in total. The first chapter introduces Ninchat, define research questions, explains its motivations, and gives an overview of the product that it is building. It

goes on to list definitions, objectives, and goals that will be mentioned again in the chapters following this one.

Chapter two includes a review of the literature. The chapter introduces the principles of and underlying technologies of the study domain, addressing WebAssembly and WASI approaches that helped to build the product, and defining Ninchat Lightbots and Chatbots.

In chapter three, Design Science Research Methodology (DSRM) is discussed while revealing why this thesis falls within the realm of Design Science. Then, it goes on to discuss what type of DSRM solutions the thesis belongs to and what steps were taken to complete it.

Chapter four describes requirements categories, the elicitation process and approaches, and lists functional and non-functional requirements. This chapter also covers stakeholders, success criteria, and scopes.

Chapter five discusses design and development topics. It introduces available cloud data APIs and then describes design decisions, assumptions, dependencies, and risks. Following that, the chapter explains technical decisions including operating environments and constraints. The rest of the chapter discusses the chosen I/O model and the streaming payload structure.

Chapter six introduces and describes how to use defined interfaces for binary communication through end-to-end examples that cover all aspects of binary communication.

Chapter seven discusses the results and evaluates the project's success and failure. Furthermore, it also includes potential future research and development.

Finally, the summary chapter describes technical and theoretical challenges and outcomes of the project for the student and Ninchat.

2 MAIN CONCEPTS

2.1 A Language-Independent Environment

Hosting and executing third party binaries raises the question of whether the host environment is able to run binaries independent of the language. While it may be tempting to limit the environment to support one language to reduce the interface complexity, it can be burdensome for customers. Not all customers have a diverse development team to develop chatbots in the supported languages. Many may reject the new product and the project will fail due to lack of enough interest. Therefore, it is critical for Ninchat to execute customers' chatbots in whatever language they are written in. In the next sections, the thesis attempts to propose a technical solution for building a language-independent environment using WebAssembly.

2.2 WebAssembly

Different trend analyses indicate that JavaScript has been one of the most popular programming languages for the past five years. According to (TIOBE, 2020) as of December 2020, JavaScript is ranked right behind C, Java, Python, C++, C#, and Visual Basic based on the number of active engineers around the world, the accessibility of online and on-site courses, and also third-party vendors. Various other statistics usually place it somewhere between rank 2 and 5. TypeScript, the new JavaScript derivation, is also very popular and among the top 10 most popular languages. (Tung, 2020) Furthermore, JavaScript not only rules in the front-end world, but it also offers powerful frameworks for server-side projects. (Fowler, 2020) That being said, it is hard to imagine the web, and the future of it, without JavaScript and its frameworks.

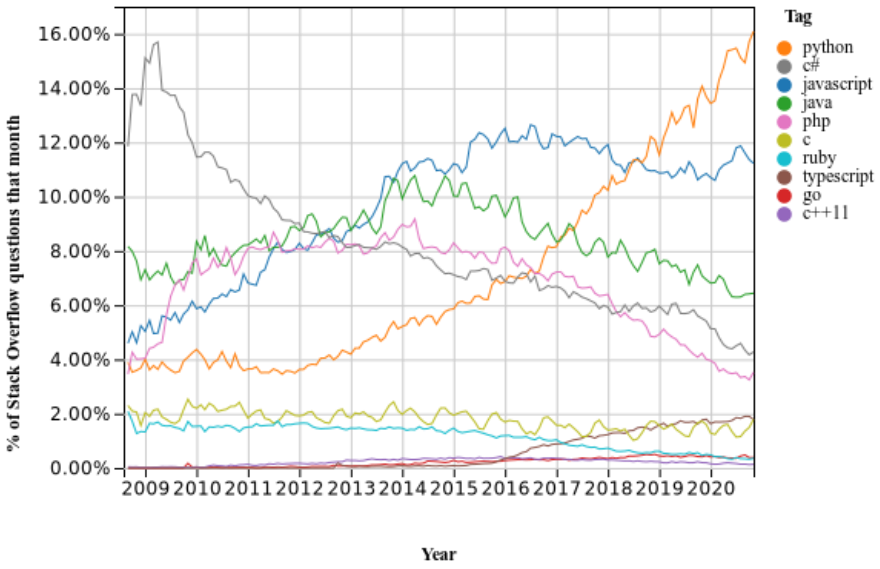

Figure 3 - Stackoverflow trends (Stackoverflow, 2020)

On the other hand, JavaScript is not a well-reputed language performance-wise. JavaScript has been demonstrated to perform poorly in comparison to an AOT-compiled language like C++, not only for complex problems but almost all issues. (Ogasawara, 2014).

There are fundamentally distinctive features and targets between JavaScript and a language like C++. On one hand, JavaScript was primarily designed as a lightweight language with the least possible learning curve required. To provide a declining learning curve, it uses many abstractions which are costly and require additional hardware resources. (Ignatchenko, 2018)

On the other hand, a compiled language such as C++ has no runtime source-code parsing, uses fixed-type variables, and allocates the memory during the compilation. Despite some issues, C++ avoids garbage collection, allowing developers to optimize memory usage as much as possible. (Stefanoski, et al., 2019; TIOBE, 2020)

This is why the comparison between JavaScript and an AOT-compiled language like C++ is not always precise, though the performance gap cannot be ignored. With more and more applications moving to be accessible online and as Web Applications, it is undoubtedly important to find an approach to bring AOT-compiled languages' performance advantages to JavaScript and Web Applications.

2.2.1 Wasm

WebAssembly (Wasm) is a binary instruction format that is designed to be executable on browsers. Just same as an assembler that generates machine instructions from source-code to be executed by an operating system, WebAssembly generates machine codes to be run on browsers. It takes advantage of common hardware capabilities to be fast and efficient. It supports a memory-safe sandboxed execution environment and comes with a pretty-printed textual format suitable for debugging. (WebAssembly, 2019). In simple words, Wasm is a standard for running AOT-compiled languages in browsers, as well as executing applications written in languages other than JavaScript efficiently and safely.

It started in 2015, mainly based on JavaScript implementation of assembly (asm.js). By February 2018, the Working Group published three public working drafts for the Core Specification, JavaScript Interface, and Web API. Since early 2019, it has been widely supported by major versions of browsers. (Mozilla Firefox, Google Chrome, and Microsoft Edge) (ComputerHope, 2019)

Despite aforementioned features and advantages, WebAssembly is not a self-contained binary that can be run on its own, and it is impossible to execute Wasm modules independently. To run a Wasm binary, JavaScript must be used. Only after JavaScript starts the module, exposed functions can be accessed via the host application. Because WebAssembly was designed primarily to execute binaries in a browser, dependency on JavaScript is not an issue. (Clark, 2019; Duin, 2020)

WebAssembly supports only scalar types limited to 32-bit and 64-bit integers and floats (`i32`, `i64`, `f32`, and `f64` respectively) (WebAssembly-Community-Group, 2017). Therefore, the glue code acts as a helper at the boundary layer and facilitates using of complex data structures such as Arrays and Strings. The glue code bonds the Wasm module with the browser and translates JavaScript and C/C++ functions, data, and object representations. (Wilson, 2018; Duin, 2020; Bees, 2018)

As an example, a simple `hello.c` which prints a simple “Hello World!” to the output, can be compiled to WebAssembly target using `emcc` tool. The result consists of the following outputs (Mozilla, 2021)

- `hello.wasm`: WebAssembly binary module.
- `hello.js`: JavaScript glue code for translating C into JavaScript.

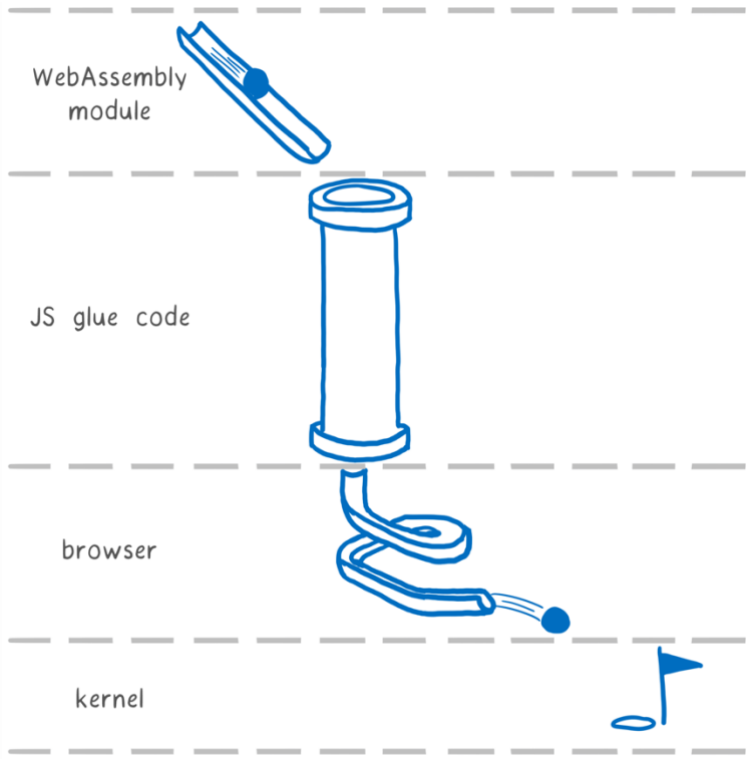

Figure 4 - Wasm and JS glue code (Clark, 2019)

Security

The idea of executing a compiled binary always comes with the question of safety and security. What happens if a malicious code gets embedded and run in the browser on behalf of the user?

To address the security concerns, WebAssembly offers to execute binaries in sandboxed environments where every request is routed only through the host application. It means a binary can have no access to the machine's resources, operating system calls, and I/O operations directly. If a Wasm binary needs to access system resources, the host is responsible to call the system directly or through other applications, and then delivers the results back to the binary. (WebAssembly-Community-Group, 2017)

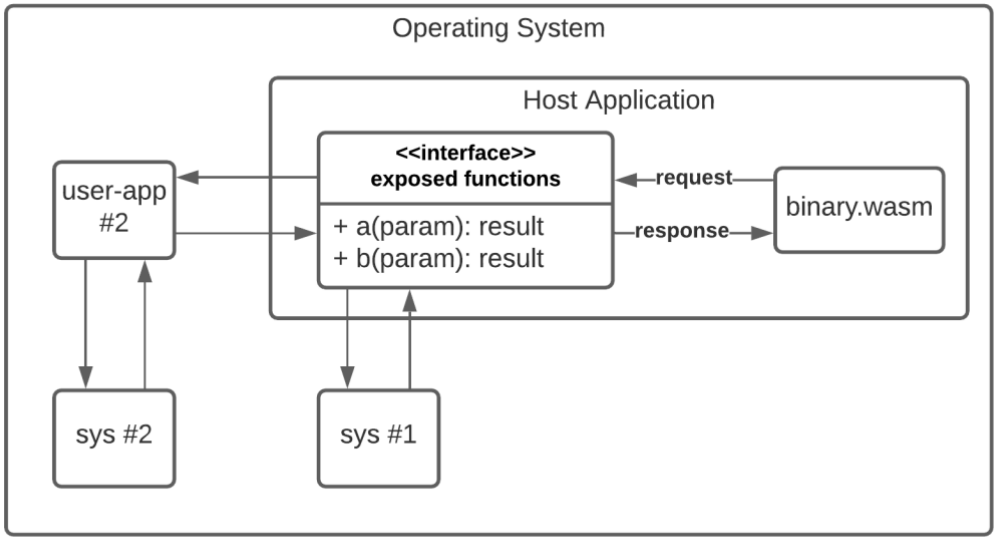

Figure 5 - Wasm sandboxed environment

Despite all described considerations, it is still possible that WebAssembly binaries are misused as Spywares or Malwares. Embedding in an extension or hidden in the source-code of a website, they can be used to take advantage of the visitors' resources and do hidden crypto mining. (Neuman & Toro, 2018) As an example, supporting multi-threading feature is on halt in the Spec 1.0 of WebAssembly, just after (Cimpanu, 2018) showed supporting threads with shared memory can result in the possibility of using WebAssembly to bypass browser mitigations for security vulnerabilities.

Performance

A significantly improved performance compared to pure JavaScript is the most notable advantage of WebAssembly. Many enterprises and resource-consuming applications such as Figma have taken advantage of WebAssembly to gain a smooth experience in their web applications. (Wallace, 2017)

WebAssembly is faster than JavaScript in fetching and decoding phases because it is more compact than JavaScript even when it is compressed. Furthermore, compiling and optimizing take less time with WebAssembly because it goes through optimization during compilation. And more importantly, executing WebAssembly is often faster because its instructions are more ideal for machines and there are fewer compiler tricks developers should know for writing performant code. (Clark, 2017).

On the other hand, since there is no garbage collection for WebAssembly, the memory must be managed manually. This can improve performance, though managing the memory manually can result in many obscure code situations. It can be the root for many known issues such as Memory Leaks, Dead Memory, Segmentation Fault, Dangling Pointers, Uninitialized Memory, and so on. (Backtrace, 2016) These problems are the reasons for the invention of garbage collection in 1959 by John McCarthy. GC has a great influence on the program and if implemented right, it can improve the performance. (Romanazzi, 2018) Moreover, some languages like Swift and Objective-c offer Automatic Reference Counting (ARC) instead of GC, which can supply the best of both manual memory management and tracing garbage collection. ARC allocates the memory to store information about the instance and frees the memory once it is not in use (or pointed) by any instance anymore. (Apple, 2020)

To have a better vision about the performance advantages of WebAssembly, the table and diagram below show how the same set of functions with the same level of complexity have significantly different performance on JavaScript and WebAssembly.

Table 1 - WebAssembly and JavaScript examples with the same level of complexity

JavaScript	WebAssembly

`function jsFib(n) {`
`  if (n === 1) return 1;`
`  if (n === 2) return 1;`
`  return jsFib(n-1) + jsFib(n-2);`
`}`
	`int fib(int n) {`
	`  if (n == 1) return 1;`
	`  if (n == 2) return 1;`
	`  return fib(n-1) + fib(n-2);`
	`}`

`function jsSumDouble(array, n) {`
`  var s = 0;`
`  for (var i = 0; i < n; i++) {`
`  s += array[i];`
`  }`
`  return s;`
`}`
	`double sumDouble(double *array, int n) {`
	`  double s = 0.0;`
	`  for (int i = 0; i < n; i++) {`
	`  s += array[i];`
	`  }`
	`  return s;`
	`}`

Figure 6 – Performance comparison between WebAssembly and JavaScript

2.2.2 WASI

Earlier in this section, we mentioned that WebAssembly modules cannot be run on their own. They need to be glued with JavaScript. Many applications cannot take advantage of Wasm on the server-side because of this issue. It is impossible, for example, to develop the heavy part of a Java application in Rust, generate the Wasm binary from it, and import the binary into the main application on the backend.

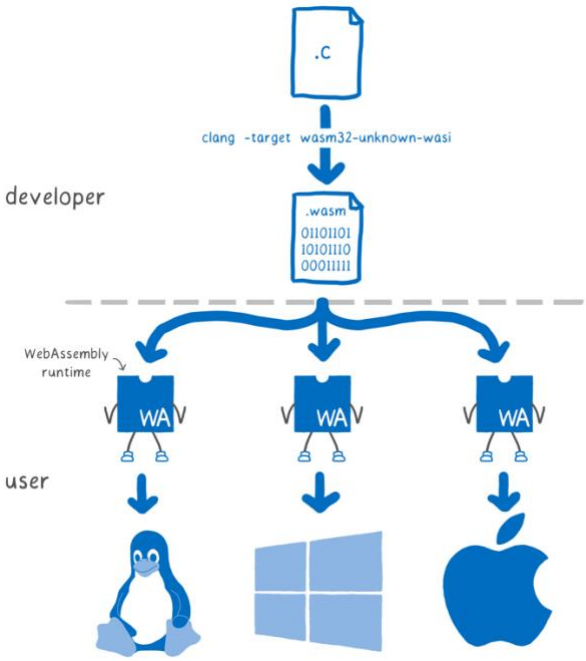

Figure 7 - WASI (Clark, 2019)

This is the reason for developing WASI, a system interface to run WebAssembly outside the browser. It is a conceptual operating system interface that lets WebAssembly binaries be run on any operating system. (Clark, 2019). Think of WASI as a layer of abstraction that translates Wasm instructions into system calls, letting any binary to communicate with operating systems and thus, with any other applications independently. It offers a set of interfaces to work with system calls remarkably like POSIX system ABIs for reading, writing, and polling system files and pipes (Snapshot, 2020).

Portability

The main advantage of WASI is the ability to build the code once and run it everywhere. (WASI, 2021) As depicted in **Error! Reference source not found.**, a generated WebAssembly binary with the WASI can be executed on any operating system with the help of runtimes. As an example, the current thesis supports WebAssembly binaries written in Rust, C, or Swift and can be run on Windows, Linux, and macOS.

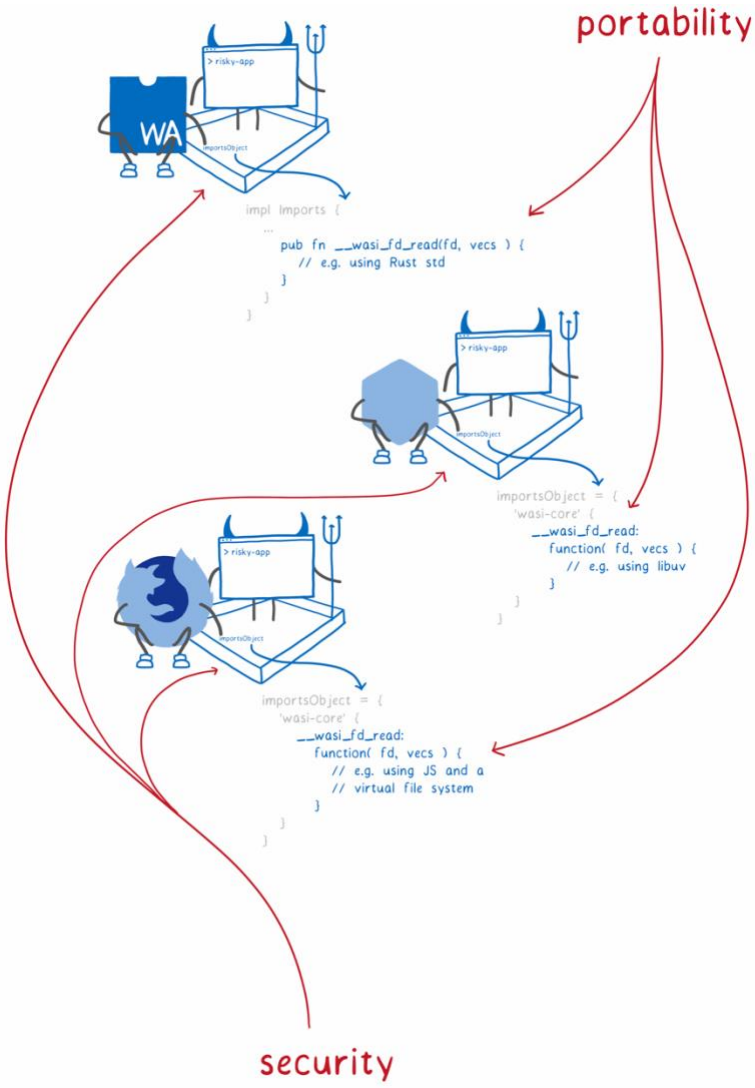

Figure 8 - WASI Portability and Security

Security

WASI is just an interface for Wasm to make it able to work without JavaScript, and that means Wasm binaries with WASI still have the security consideration explained earlier. They are run in sandboxed environments with no direct access to the machine's resources. Furthermore, WASI has extended those security considerations beyond by adding an extra layer of security to operating systems' capability-based approach. For example, for calling the `open` system API, the OS checks if the running program has enough permissions to access the file. Using WASI, however, the `open` API accepts only file descriptors that have permissions attached to them. With that, it is impossible to make a WASI binary to access sensitive files such as `/etc/passwd`. (Clark, 2019)

2.3 Chatbots

As the name suggests, a chatbot is a software designed to talk with a user, understand their inputs, collect any required information, and refer the user to the right path according to the given information. (Jenny, 2020) Chatbots can improve users' overall experience. Users do not need to wait in line to talk with an agent and are referred to the right specialist if necessary. Chatbots can also help businesses to reduce costs by serving more users and improve their experiences. (Expert-System-Team, 2020)

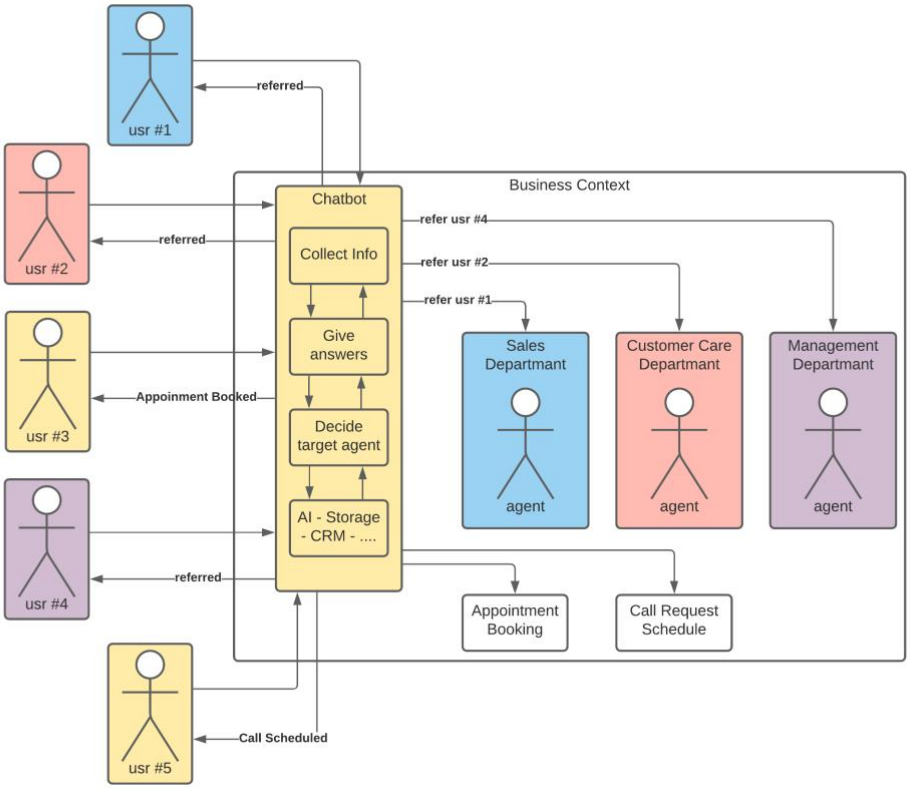

Figure 9 – Chatbots

Different types of chatbots are built on different technology and use different use cases, as we will discuss in the upcoming sections.

2.3.1 Rule-based Bots

Rule-based (or Linguistic based) chatbots can be very complex or as simple as decision trees. They are implemented either on the client's side or the server's side and can be configured to have correct answers saved for questions. They are built of several `if-then` logics to shape the conversation, and thus, are testable both automatically and manually. Rule-based chatbots are capable of supporting language conditions in order to consider the words and their orders, synonyms, different shapes of a question, etc. to ensure the accuracy of the chatbot. (Solutions, 2019).

On the other hand, rule-based chatbots are powered by keywords and thus, they are incapable of learning from interactions. Though they are highly predictable and testable, they suffer from the lack of semantics analyses and are hard to scale up. In addition to scalability, performance optimization is also difficult in rule-based chatbots. (AI, 2020)

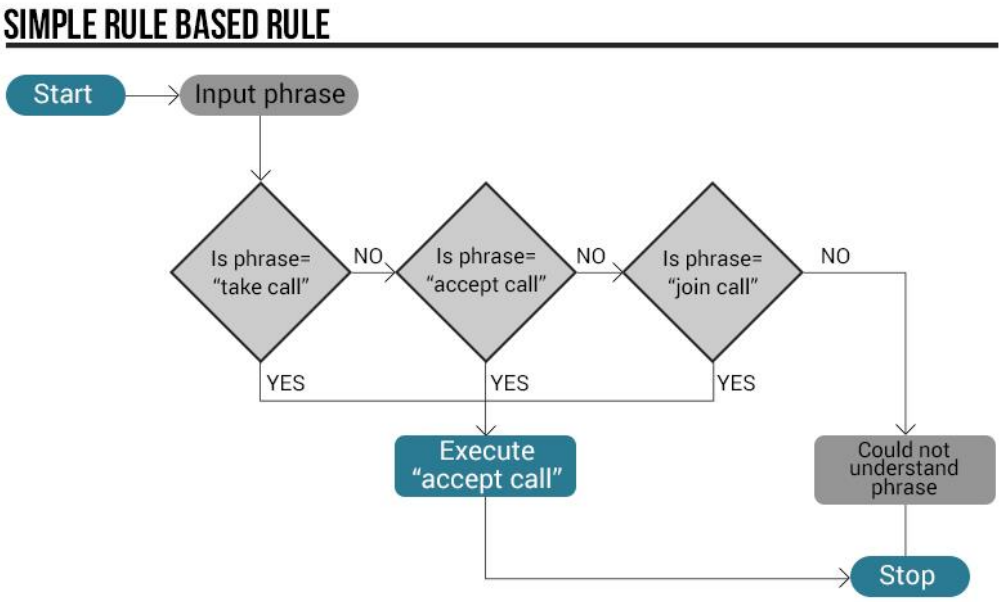

Figure 10 - Rule-based Chatbots (Team, 2019)

2.3.2 AI-powered Bots

On the other side, there are AI-powered chatbots that are capable of learning the language the same way a human does. Once they are trained enough (usually using FAQ or knowledge-based articles), they can communicate with users, answer multiple types of questions, and collect various information. (Flow.ai, 2019; Pettersen, 2020). Useful as support bots for answering customers' basic questions such as the opening hours or available services, AI-powered bots are particularly helpful to businesses with big sales or tech teams (20 staff or more) and a desire for supporting multi-lingual customers 24/7 over multiple channels (website, applications, social media, etc.). (Savina, 2019; Pettersen, 2020).

These chatbots benefit from deep-learning or conversational AI, where they use our voices to establish the conversations instead of using menus, touchscreens, or any other input devices. (Amazon, 2021). NLP engines can learn from every interaction and conversation and engage naturally in human-like personalized conversations. Also, NLP is able to understand the true meaning of users using semantic analyses. AI-powered chatbots are easier to scale up and the performance is optimized easier through reinforcement of learning. (AI, 2020)

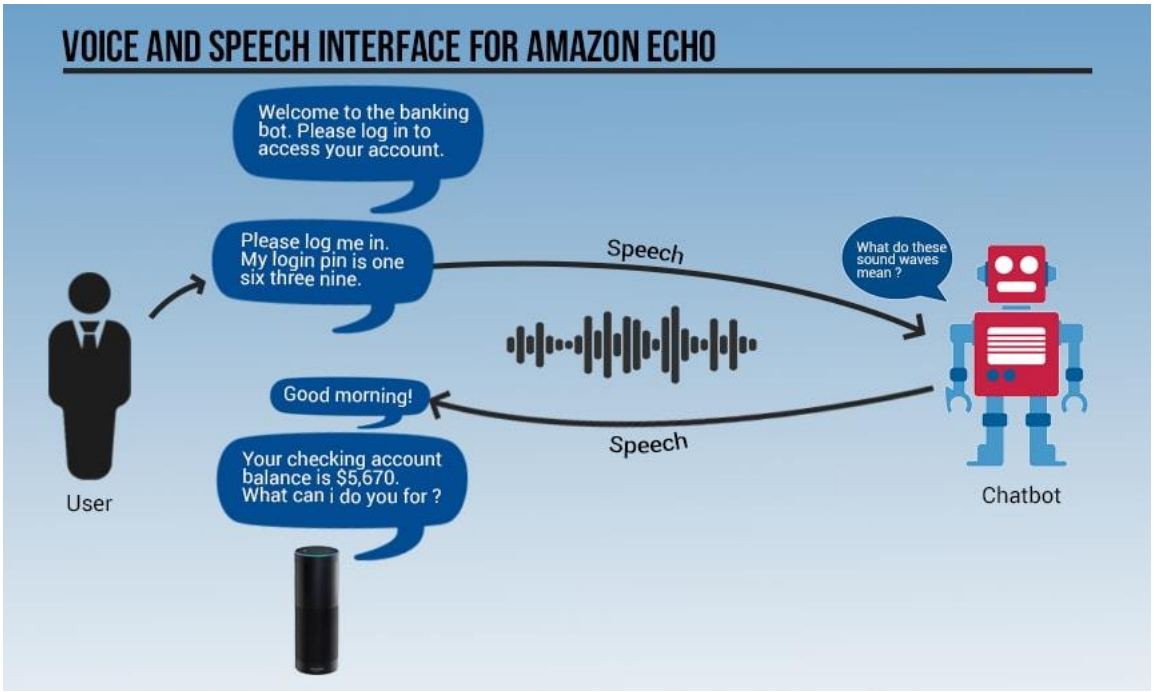

Figure 11 - AI-powered Chatbots (Team, 2019)

2.4 Lightbots

Ninchat Questionnaire (or Lightbot in marketing terms) is a rule-based chatbot developed by Ninchat in order to satisfy customers' demand for chatbots. As the name suggests, Lightbots consists of a set of predefined logic blocks which guides users through a set of simple questions. Lightbots can be used to support 24/7, gather users' information before the chat begins, collect data for statistics, or refer users to specialists.

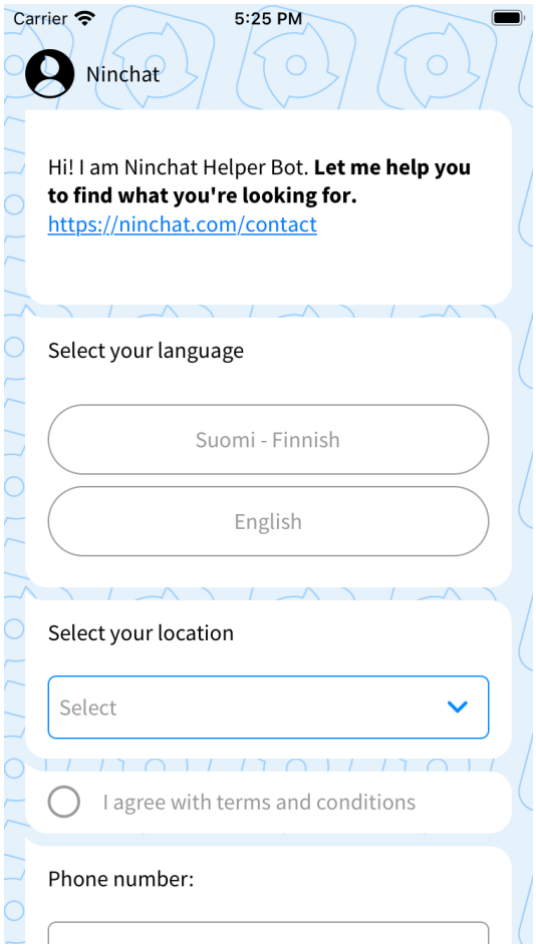

Figure 12 - Lightbots on Mobile

Lightbots are implemented on the clients' side on both Web and Mobile SDKs. They are configured by JSON files that are fetched when the SDK is initiated. Although the configuration files are shared between the different platforms (Web, Android, iOS), each platform manages and applies them independently.

Besides the gap in user experiences on different platforms which is due to individual implementations, Lightbots are incapable of offering advanced features such as storage. Some features such as text suggestions, interactive human-like conversation, and other AI-powered features are not supported by Lightbots.

3 RESEARCH METHOD

3.1 Design Science Research Method (DSRM)

Design Science is a research methodology in information systems that focuses on building and evaluation of artifacts and intends to improve the performance. (Wikipedia, 2020). Introduced in 2007 in the Journal of Management Information Systems, Design Science Research Method (DSRM) refers to a methodology to apply design science researches in information systems. The aim is to improve how the design science is applied as an approach for developing artifacts. (Pefers, et al., 2007)

In overall and as depicted in Figure 14, DSRM consists of six steps starting from problem and motivation identification. Following that, the solution's objectives are discussed and then, a roadmap for designing and developing the artifact is offered. In the next steps, the artifact is demonstrated to show how to solve defined problems and then it is evaluated against defined objectives. Finally, the research represents the problem, developed artifact, and its utility to other interested researches and professionals. With these steps in mind, the main goal of DSRM is to construct a new reality instead of describing an existing one by following the objectives below: (Pello, 2018)

- To benefit from acquired knowledge for solving problems and improving existing solutions.
- To create new knowledge, insight, or theory.

3.2 Steps

3.2.1 Problem Identification and Motivation

All the improvements in accessing the internet have led companies to offer more online services to keep customers, gain new customers, and succeed in the market competition. Having seen great potential in the market of building communication infrastructures and SDKs, Ninchat started providing smarter, more customized products among which chatbots have found the most attention and requests from customers.

Ninchat currently has a simple rule-based chatbot product in the basket (referred to in marketing terms as Lightbot), however, they are insufficient for smarter and more customizable versions. Even though Lightbots are rule-based questionnaires that can be customized using JSON files, the level of customization is limited. Moreover, Lightbots cannot offer advanced features, including storage or natural language processing (NLP), and due to their nature (developed by the clients typically without technical resources and limited budgets) it is challenging to create such new features for them. That is why Ninchat is interested in looking into new generation of chatbots.

3.2.2 Objectives of the Solution

In view of all the limitations, the design of the new product should primarily accommodate customization and provide the flexibility to add features in the future. With the goal of unifying the experience across various platforms (Web, Mobile, Embedded, etc.) the program should host and execute chatbots directly through Ninchat's backend.

Secondly, the new product must be able to support persistence to enable chatbots to save and load information to support more complex tasks. AI is expected to play a major role in helping customers extract information and generate smarter answers for users.

Lastly, Ninchat's new product should be easily integrated into the existing infrastructure and use readily available resources. It is critical that the solution’s design is scalable and reliable enough to be used by thousands of users simultaneously.

As a way to meet the objectives described, the thesis declares the functional and non-functional requirements of the new product. The thesis explains stakeholders and the impact they have on the final product, as well as defining the boundary and scopes of the project. In addition, it defines factors that make the product successful and describes possible technical and non-technical risks.

3.2.3 Design and Development

Upon identification of industrial demands and on the basis of discussions of objectives, the thesis presents a technical and practical solution to satisfy all requirements and objectives. The following diagram illustrates how the design meets discussed objectives and requirements and where it is placed in Ninchat's current backend.

On one side, the new product is in connection with Ninchat core where it receives events and is controlled as a part of Ninchat backend microservices. On the other side, the product is in connection with different external services to provide advanced features. These external services are not limited only to Amazon or Google, but any custom REST endpoint can be integrated into the system. The product is in connection internally with storage provided by Ninchat for saving and loading information required for communication with two other parties. With the current design, adding and removing external services are feasible at any time in the future, because they do not affect the new environment or Ninchat's backend.

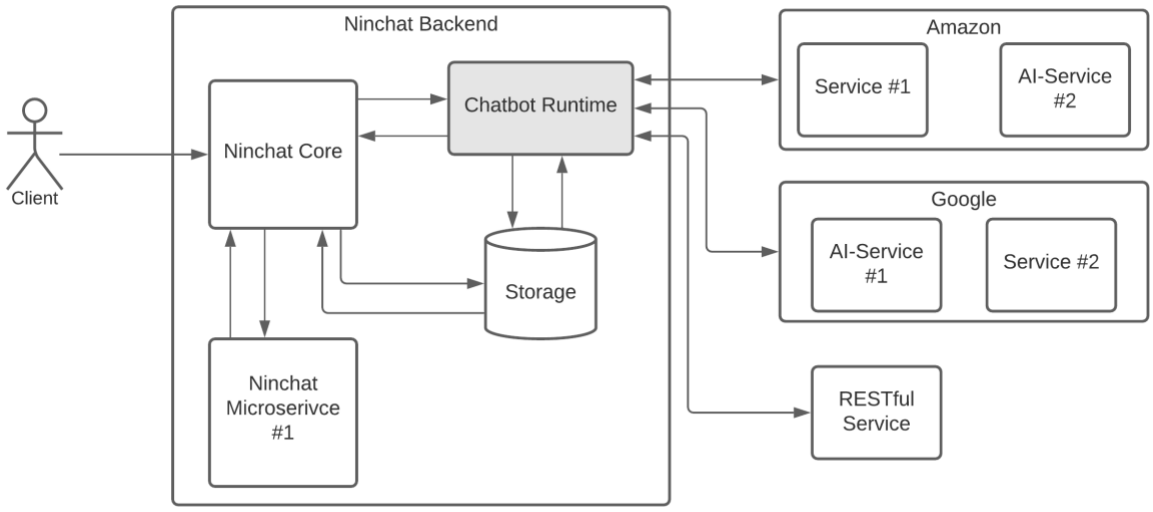

Figure 13 - Chatbot Runtime is palced in Ninchat backend

By presenting and discussing diagrams, justifying the product's design, and explaining how the system works internally and externally, the thesis seeks to help technical and non-technical stakeholders understand the design. For example, implemented components, communications protocols, and the

chosen I/O model are discussed in the upcoming sections. At the end, the data structure used for streaming purposes between the binary and the runtime environment is explained.

3.2.4 Demonstrations and Evaluation

This thesis introduces an interface for establishing communication between a Wasm binary and the Chatbot environment in order to demonstrate the designed system and prove that it works in real-world situations. ABIs for managing storage (store, load, delete) and for streaming data (send and read events) are included in the interface. Further, because Wasm binary practically can be developed in any language, the thesis defines ABIs in WAT format, the WebAssembly Text Format that must be translated by customers to the target language.

Empirical examples are presented in the thesis using these ABIs for real-world situations after the interface's introduction and explanation. They demonstrate how the product is successful in solving problems and meeting requirements. The chapter proves that the product is powerful, robust and flexible enough to be used by a wide range of customers in a production environment. Despite the fact that working with low-level ABIs may be a new experience for many, Ninchat may introduce language-specific helper libraries in the future, to enhance the ability to use the platform in different languages (Python, Go, JavaScript, etc.).

3.3 An Objective-Centered Solution

By definition, any design science project can begin at any stage between a one and six depending on the topic, objectives, and needs. Because this thesis research is set up in response to an industry need for which it is expected to develop and deliver an artifact, the thesis may be viewed as an objective-centered solution. (Pefers, et al., 2007)

This diagram also illustrates how objective-centered solutions begin at step two by defining the solution's objectives.

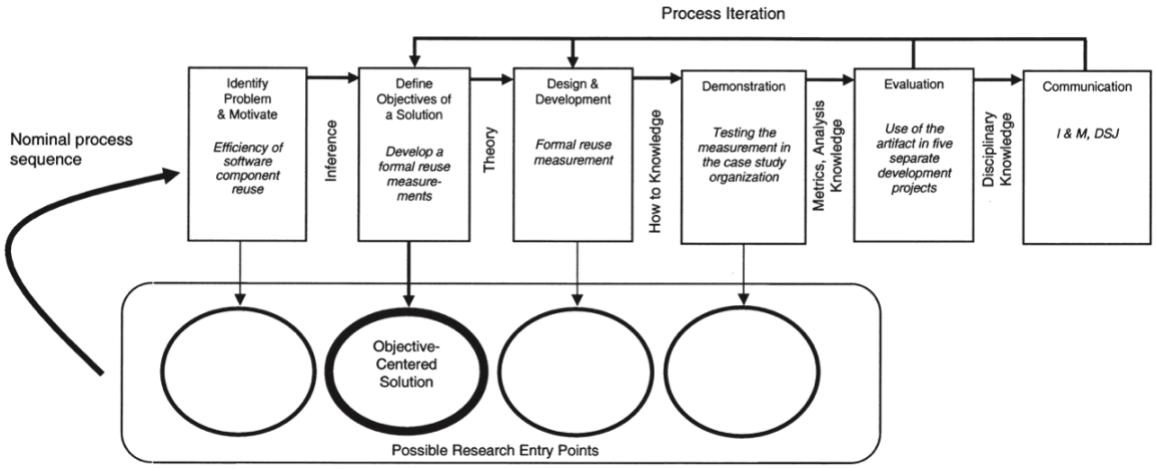

Figure 14 - Objective-Centered Solution (Pefers, et al., 2007)

4 SOLUTION OBJECTIVE

In spite of the fact Lightbots are widely used by many companies big and small, Ninchat seeks to enhance the overall user experience in using chatbots. Like mentioned earlier, Light bots are simple questionnaires with no advanced features. There is no way to add storage to Lightbots, or to support custom REST endpoints. Also, AI features like natural language processing and image processing are not supported. They can be used for customer service reasons, for gathering users' information, or as helpers for connecting users with the right specialists. All limitations considered, Ninchat has finally moved to design a large-scale project to host customers' chatbots and execute them as agent services from the Ninchat backend.

Throughout this thesis, the aim is to create an environment that will enable production of a chatbot as well as advanced features like persistence and cloud APIs. The new product is expected to offer additional advanced features like integration with third-party services (NLP, image recognition, etc.) to encourage more customers to migrate from Lightbots to the new environment.

The following context view (Figure 15) provides an in-depth look at the components and boundaries of the project from a non-technical standpoint. The new environment (now called the runtime) lets any verified customer (see Table 2) develops a Wasm chatbot (which has conformed to the designed interface, see chapter 6) and then submits it to Ninchat.

Ninchat Core, which is responsible for handling all Ninchat microservices, including the runtime, starts the Wasm binary once it receives a request from the agent. The binary then begins communicating with the user on behalf of the agent. In addition, the runtime enables a binary to communicate with other endpoints and services such as Amazon, Google, etc.

In summary, the thesis objectives are to:

- Host and start Wasm binaries on behalf of customers' agents.
- Supply required infrastructures needed for large-scale organizations.
- Supply advanced features such as third-party cloud APIs and storage.
- Help customers improving services by gaining from automated assistants.
- Unify user experience independent of the platform.

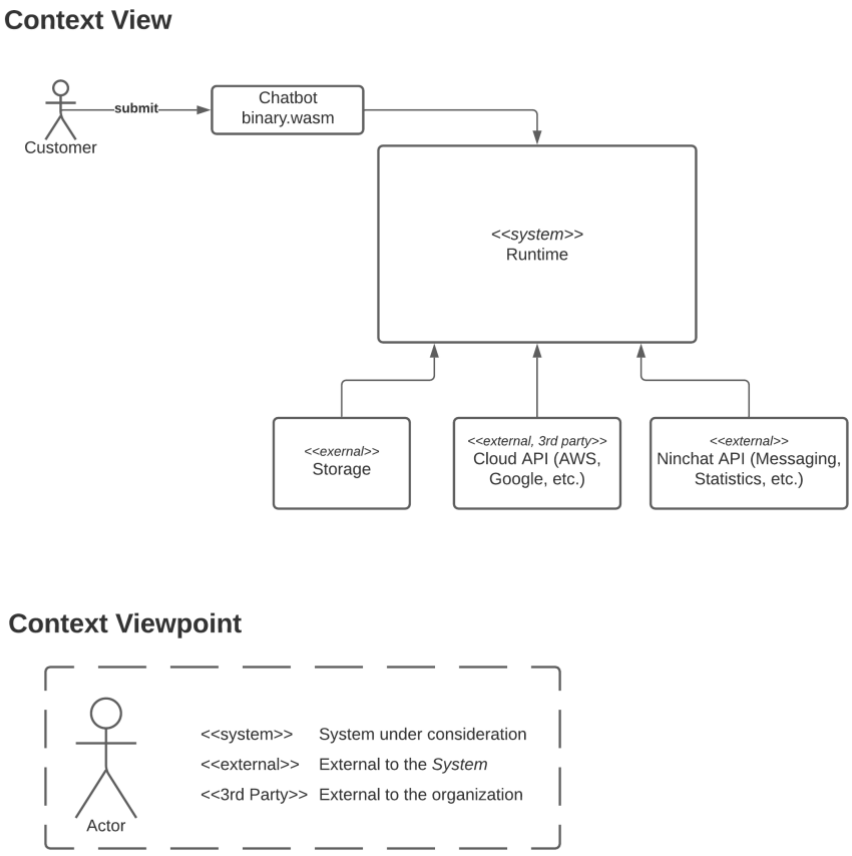

Figure 15 - Context View

4.1 Success Criteria

The technical solution is considered successful if it meets the aforementioned objectives and provides a high-quality environment for customers. An advanced runtime that is high quality and meets expectations is expected to gain customers' attention and encourage them to move from Lightbots to it.

With various functional and non-function requirements defined in the upcoming sections, it is possible to evaluate the final product success in detail. However, to define measurable factors we consider the environment to be successful if the following criteria are met. These criteria are measured once the product's development is complete, and it becomes officially available to customers. By collecting

customers' feedback, these criteria are continuously monitored to ensure the product's success continues and improves.

- The product satisfies quality attributes as well as other defined requirements.
- The product has the technical flexibility for adding features in the future.
- At least half of selected customers start migrating to the new environment.
- Only 5% of sessions face fatal errors or become inaccessible due to technical issues.

4.2 Requirements

Based on the degree of details, the thesis project has three types of requirements, which are represented in Figure 16. Business requirements contain the highest-level requirements and the least details, while system requirements contain the most details.

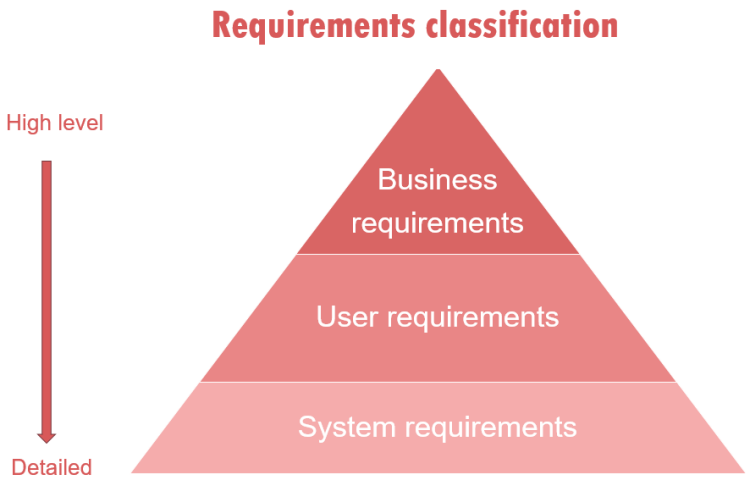

Figure 16 – requirements classification (Tkachenko, 2019)

4.2.1 Elicitation

Requirement elicitation is the process to extract requirements for a software project through communication with stakeholders. (Point, 2020) Elicitation can be the most communication intensive phase of the software development, which can be successful only through an effective partnership among team members and with customers. (Jain, 2020)

Despite the fact that a variety of approaches are available and usable for requirements elicitation, the thesis benefited mainly from brainstorming sessions in addition to oral and one-to-one interviews with Ninchat's CEO, CTO, and Product Owner.

Interviews

Since Ninchat is a key and first-class stakeholder of the thesis project (see chapter 4.3, the researcher conducted several open-ended interviews with Ninchat employees, both technical and non-technical. The aim of these sessions was to help the researcher collect the necessary requirements and understand the expectations of clients. These interview sessions examined many potential features, such as text suggestion or extracting metrics from a user's conversation, along with upcoming functional and non-functional requirements.

Brainstorming

With an emphasis on technical matters and taking into consideration the fact that many different ideas are shared and documented, brainstorming sessions were held to elicit functional requirements and quality attributes, as well as addressing design and development issues. When a brainstorming or interview session concluded, the researcher organized, prioritized, and finalized the shared ideas to be shared in the next session.

4.2.2 Business Requirements

The highest level of statements describing the objectives and goals of the project for the organization, while driven by the stakeholders of the project, particularly the management team in Ninchat (Product Managers), they aim to discover the motivations for initiating the project in the first place and resolve inconsistencies throughout the project development. (Wiegers & Beatty, 2013; Tkachenko, 2019; Spacey, 2018).

- A robust environment for hosting and executing customers' chatbots.
- User-friendly interfaces for building and submitting chatbots by customers.
- Access to Ninchat and several third-party services (Amazon, etc.) with all required resources.
- Advanced functionalities including administrations, observation, customization, etc.
- Access to statistics and reports for customers.

4.2.3 User Requirements

The next level of requirements (referred also as User Needs) aims to bridge business requirements to software requirements and to describe user goals or tasks that the user must take to use the product. In other words, they define what a user can do in working with the product. (Wiegers & Beatty, 2013)

- Standard API for communication with available services and resources.
- Documentation, guides, and examples for designing practical chatbots.
- Helper libraries for most popular languages to facilitate working with provided low-level API.

4.2.4 System Requirements

System requirements describe functions that a system as a whole must meet to satisfy stakeholders' expectations and needs. (Alan Faisandier, 2020) These requirements are the blocks used by developers to make the product and are usually described using the word "shall". (Parker, 2012)

There are two major categories of System requirements, Functional and Non-Functional requirements, as explained in the following sections.

Non-functional Requirements

Non-functional requirements (also known as quality attributes or Supplemental requirements), define the general software characteristics and specify criteria that can be used to judge the operation of the product. (Tkachenko, 2019; AlexSoft, 2018) The following table lists defined non-functional requirements for the thesis.

Table 2 - Non-functional requirements

Requirements	Notes
Performance	Synchronized API with an immediate response.
	Asynchronized API with an acceptable timeout for the response depending on the endpoint.
	No performance issues in other parts of the system.
Security	Use Ninchat's unified authentication mechanism.
	Available only for verified customers for earlier versions.
	Communications only through the runtime using the given interface.
	No direct access to the machine.
	Restricted and listed available endpoints.
Usability	An interface of exposed functions to do most with the least constraints.
	Helper libraries for different languages to facilitate building chatbots.
	A wiki with explanations about how the system works using relevant examples.
	Avoiding fatal failures by tolerating and handling errors.
Scalability	Expandable for new cloud data APIs.
	Scalable on Amazon alongside other Ninchat's services.
Availability	99.9% Availability. (maximum of 1.5 minutes downtime per day)
	Live patch with no interruption to the running services.

Functional Requirements

Functional requirements (also known as behavioral requirements) specify functionalities that define ways the product must behave. (AlexSoft, 2018). They are built by development teams and make users able to accomplish their tasks and hence, meet business requirements. (Wiegers & Beatty, 2013; Tkachenko, 2019).

Table 3 - Functional Requirements

Code	Feature	Requirements
FR-1	The customers submit a binary to Ninchat.	Available user interface for submission of binary.
FR-2	A binary is started on the Ninchat's backend.	Starting the binary.
FR-3	A binary calls Ninchat API.	Interface for exposed functions to communicate with Ninchat.
FR-4	A binary calls third-party API.	Interface for exposed functions to communicate with external API.
FR-5	Customers develop chatbots.	Accessible helper libraries to convert low-level interfaces to language-specific ones.
FR-6	Customers build chatbots using a helper UI.	Available user interface facilities for building *Chatbots.*
FR-7	Customers evaluate the binary before the submission.	Available sandboxed environment for running the binary before submission.

4.3 Stakeholders

Stakeholders are entities or persons that are involved with the outcome of the project and are affected by it directly or indirectly. (Shafqat, 2019). The Ninchat organization may be considered as the most significant stakeholder because it decided to develop the project. There are still other stakeholders, internal and external, with their own interests and constraints that could affect or be affected by the final products.

Table 4 - Stakeholders

Stakeholder	Interests	Attitudes	Constraints
Ninchat	Offer bug-free and high-quality services to new and current customers.	Supply all required resources.	Deadlines
Customer	Offer various high-quality and affordable digital services to end-users.	Gradually upgrade to new versions.	Product quality. User Experience.
Developer	Develop bug-free, scalable, and maintainable products using documents.	Follow best practices. Test before release.	Technical skills.
User	Use offered products for daily needs.	Give feedback about the product.	None identified.

4.4 Roles and Responsibilities

Ninchat is a startup company involving distinct roles with dedicated responsibilities that can influence the project.

Table 5 - Roles and Responsibilities in Ninchat

Role	Responsibility
Product Manager (PM)	Defines tasks, change priorities, and communicates with customers to get feedback about alpha and beta versions. He is also responsible to gather customers' requested features.
Chief Technology Officer (CTO)	Responsible for supervision of the technical aspects of the project, ensuring that it will not break current products, and it follows best practices. He is responsible for mentoring the developer in the integration and deployment processes.
Quality Assurance (QA)	Responsible for confirming that a version has the acceptable quality before it is released and delivered to the selected audiences.

4.5 Project Scope

Development of both the backend environment that fetches and executes customer binaries, as well as the application user interfaces that facilitate submission and testing is included in this project. But because providing both backend and frontend features can easily exceed the deadline for a thesis project, a limited set of features focusing only on the backend is offered for now.

Table 6 - Project Scope

Feature(s)	Release v. 1.0 (The Thesis Scope)	Release v. 2.0
FR-2	Download and start a binary	
FR-3, FR-4	Design an interface for binaries to communicate with the runtime	
FR-5	Develop helper libraries for most common languages	
FR-1, FR-6		User interface for submission and updating binaries
FR-7		User interface for testing binaries before submission

4.6 Risks

Risks are defined as uncertain events that are associated with upcoming events. Risks have the possibility of occurrence and may bring loss to the project. Due to great impacts on the success or the failure of the software project, risk identification is known as one of the most important tasks during the project design. (Geeks, 2020) Several risks have been identified for this project due to its nature as a software project. The following risks have been sorted according to the impact they have on the project.

Table 7 - Project Risks

Risk	Probability	Impact	Mitigation
The project face critical and major bugs and fatal errors.	Medium	High	Perform comprehensive automated and manual tests alongside gathering feedback.
The binary makes harmful API calls.	Low	High	Block any outgoing untrusted connections.
Customers do not use the new service.	Medium	Medium	Add UI facilities to create and test binaries for non-technical people.
The runtime cannot scale up for more customers.	Low	Medium	Continuously upgrade required infrastructures.
The project does not meet the deadline.	Low	Low	Use Agile to diminish the possibility of missing the deadline.

5 DESIGN AND DEVELOPMENT

Earlier sections mentioned Ninchat's plan to provide an environment for hosting and executing binary files submitted by customers. First, this section introduces available cloud APIs, followed by discussions of technology decisions, and finally, an overview showing how the cloud data APIs work in the real world.

5.1 Cloud Data API

The Ninchat service has the highest integration priority among cloud data APIs available to be integrated into the project. When communicating with users, it’s important for a chatbot to have access to Ninchat’s text messaging, statistics, and questionnaire services.

Additionally, Ninchat is hosted on Amazon, which means the application is able to utilize numerous services offered by Amazon. For instance, the S3 service can be used to host binaries, while DynamoDB is a powerful key-value database for chatbot configurations and temporary values. In addition, services such as Amazon Machine Learning and Natural Language Processing are also available for integration to create smarter chatbots.

A chatbot should be able to use Ninchat and Amazon services, as well as the needed URL, to send an endpoint's body and then receive the response. This feature allows chatbots to take advantage of nearly unlimited custom services like if they were standalone applications. This flexibility, however, can be abused in a way that violates security considerations, so strictly adhering to the security considerations discussed earlier is essential.

5.2 Design

Most of the design and feature elicitation was influenced by Lightbots. As part of the requirements' elicitation process, business priorities were listed, the most demanding features of Lightbots were discussed, and quality measures for the new project were decided. More details about these decisions are given in the following sections.

5.2.1 Design Decisions

Invalidation should be a possibility during the development phase of any decision that was made during the design phase. It is also difficult to list all the decisions that are taken during the implementation of a software project. Thus, the following list includes only important ones who had a significant impact on the development of the project.

Remote Procedure Call (RPC)

Remote Procedure Call (called also as Subroutine or Function Call) is the solution for establishing communication between a client and a remote server. The client starts by sending a procedure request or function call, which is then translated by RPC for the remote server. Once the server receives the request, it sends the response back to the client where the RPC translates the response again. During the process, the client gets blocked until it receives the response from the remote server. (Onsman, 2018)

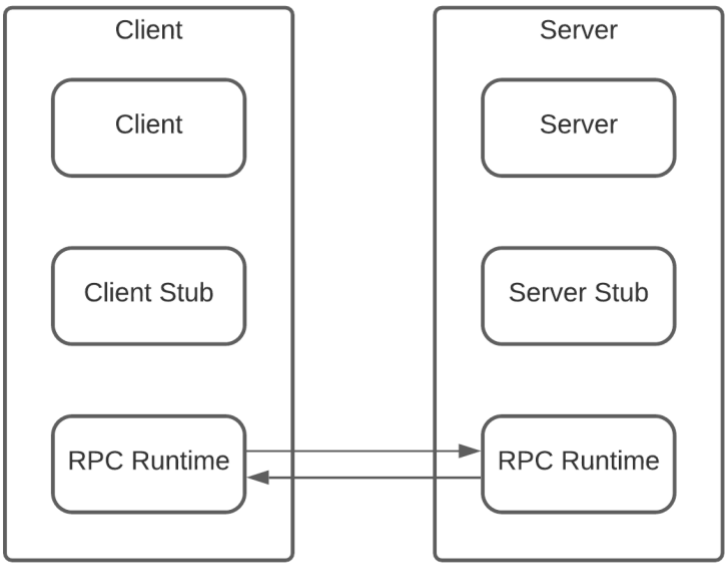

Figure 17 - Remote Procedure Call

Choosing the right RPC for the project was done over technical meetings, where the team came up with the two most popular and available options, *gRPC* and *REST.*

Representation State Transfer (REST) was introduced in the early 2000s by Roy Fielding as a standard and easy-to-use API. (Fang, 2021) REST is an architectural style for distributed media systems and comes with guiding constraints that must be satisfied by the interface. Additionally, because REST works on top of the HTTP layer, it does not hold any state by itself, which means that a request must contain all information needed for an operation. (Parameshwara, 2019)

On the other hand, gRPC is an open-source, performant, and well-supported solution introduced by Google in 2015. It aims to make abstraction easier and facilitates HTTP calls, and as a result, gRPC is quite helpful and popular for microservices that include multiple languages within a single project. While REST is built to work with HTTP/1.1, gRPC gains several benefits from HTTP/2.0 including the "request/response multiplexing" and "header compression". Supporting HTTP/2.0 in addition to these features make the gRPC able to provide bidirectional streams. (Winata, 2020; Parameshwara, 2019).

Although gRPC is a newer technology with a focus on performance, REST is an easy-to-understand solution that uses standardized HTTP language such as GET and PUT (Fang, 2021). In addition, with native support by the chosen programming language as well as being used widely by other Ninchat backend components, using REST can help to deliver the product to customers sooner. With respect to REST benefits and features, the project's level of complexity, and the fact that switching to HTTP/2.0 and supporting bidirectional communications is not a priority for this project, the technical team eventually decided to use REST over gRPC.

RPC Encoding Data Format

Once the decision regarding encoding was reached, the next step was to choose the RPC data format. Among them all, XML, JSON, and Protobuf are the most popular and commonly used formats. They all have their own benefits and drawbacks that make them best paired with one RPC protocol.

On one side, although XML and JSON are well-readable and are consumable by browsers, they lack validation and can be eavesdropped on. Furthermore, JSON is generally faster than XML in parsing

data, though they are still inefficient for decoding a massive amount of data. Neither XML nor JSON offer interfaces for RPC, but both are suitable options for working with REST. (P, 2018)

On the other hand, Protobuf is not a human-readable and browser consumable format, but it is faster than both XML and JSON and comes with validation support, data security, and cost efficiency for decoding a massive amount of data. Providing interfaces for RPC, Protobuf is mostly used as the payload format when using gRPC. (P, 2018)

Despite all the improvements that Protobuf has provided, it is still not the best match for REST. XML and JSON are different, but JSON is certainly a better option given that the amount of data is expected to be modest for this project. It is faster, much easier to use, more readable, and also is widely used by the Ninchat backend for communication between different components. In addition, since it is natively supported by the chosen programming language, it is perfectly matched for REST, and since it does not need to be implemented individually, it speeds up the development process.

Hypertext Transport Protocol (HTTP)

Having decided on the RPC and Encoding protocols, it was time to choose a hypertext transport protocol (HTTP). By playing a request/response model role, HTTP enables communication between clients and servers. (Mozilla, 2021) Even though there are various HTTP methods available, GET and POST are the most common ones. (W3School, 2021)

The project uses POST as the HTTP method over GET. GET is mainly used for *requesting* data from a specified resource, while POST is used to *send* data to a server in order to update or create a resource. GET requests are cacheable with the ability to keep history, but POST requests have neither. But one important point is that POST requests have no restrictions on the data length and types, while GET requests accept only ASCII characters with a maximum length of 2048. Another important point for choosing POST over GET is the fact that POST requests are more secured compared to GET requests, where the data is a part of the URL. (W3School, 2021)

Programming Languages

Due to the tight connection of the project with WebAssembly and WASI, the chosen programming language for developing the project must have enough support for these technologies. While there are a few numbers of languages with such supports by the time the report is written (including Rust, Go, C, and Swift), Rust has the first-class support of all. Rust is a fast, efficient, safe, and very robust programming language making it one of the most interesting options for system programmers. (Stackoverflow, 2020) Alongside official documentation and examples about writing performant Wasm binaries in Rust, there are various online materials for developers who are interested in WebAssembly in Rust. Additionally, compiling a Rust program to Wasm or WASI binaries is as simple as importing and starting a binary. With all these features, neither C nor Go can compete with Rust in the area of Wasm-WASI development. (Levick, 2019)

However, after discussions with the CTO, it was decided to choose Go over Rust for this project. First of all, Go is not as fast as Rust, but by benefiting from GC and native multi-threading features it is still one of the most performant programming languages available.

Secondly, though Go does have very limited support for generating Wasm binaries, it has enough level of support for importing and controlling them. Popular available WebAssembly runtimes have a respectable SDK and supportive community for Go.

Thirdly, while it is fast, Go is a very-easy-to-learn and develop programming language. When compared to Rust, the researcher needs to spend significantly less time to learn how to develop the project, import a Wasm binary, handle threads and avoid thread-safety issues, and understand and follow the language's best practices.

Fourthly, since the backend of Ninchat already consists of different languages (C, Python, and Go), it was a smart decision maintenance-wise to not introduce a new language to the backend. A new language means the technical team has to take care of one more language during every deployment.

And finally, the skill level of the technical team of Ninchat in Go is not comparable to Rust. Rust is a complex and new language mainly suitable for low-level programming. In case of any critical technical

questions and discussion about Rust, it was rare for the researcher to expect Ninchat teammates to be able to help him beyond available online forums and resources.

WebAssembly Runtime

WebAssembly runtimes are the key component to run a Wasm-WASI binary outside the browser. They can be used either as a command-line tool or a library embedded in larger applications. (Wasmtime, 2021).

There are two main projects that are useable for this thesis project, *wasmtime* and *wasmer.* Both libraries are open-source providing enough level of support for importing and controlling Wasm-WASI binaries in many languages including Rust and Go. (Wasmer, 2020; Wasmtime, 2020) The Go SDKs in both projects have support for executing exported functions and injecting implementations for binary interfaces (also called *imported functions*).

However, there are tiny differences between these two libraries. *wasmtime* is maintained and supported by *Mozilla*, (Callahan, 2019) while *wasmer* is more a community-driven solution. Although *wasmer* has a cleaner and easier approach for controlling the binary (including API for closing the binary), *wasmtime* has better support for WASI, though neither supports WASI completely. As an example, *wasmtime* has partial support for setting *Stdin, Stdout, Stderr* files, and it is more likely to expect full WASI support in *wasmtime* sooner than *wasmer.*

With these tiny differences in mind and with respect to the fact that *Mozilla* is probably a more reliable option than a community-driven project, the thesis project decided to choose *wasmtime Go SDK* for managing binaries in the product.

5.2.2 Assumptions and Dependencies

The thesis project relies on several assumptions and dependencies in order to deliver a successful product. The Ninchat backend is assumed to be sufficiently flexible and modular so that new services can be easily integrated into it without breaking other components.

In addition to integration, it is expected that the new service will be scaled alongside other components of Ninchat. Since scalability is one of the key requirements of this project, it will not succeed if Ninchat fails to provide scalability.

In addition to Ninchat, the project relies on Amazon and its services (such as storage). Therefore, it is assumed that Amazon provides high-quality services and can handle various failures as the infrastructure. If Amazon experiences any problems, all Ninchat services will fail, including the thesis project. It is also assumed that Amazon has practical services (such as ML and NLP) that can be integrated into the thesis project.

Moreover, the project is highly dependent on WebAssembly Spec 1.0 and how it is implemented in the chosen runtime and language. Supporting WASI is yet another dependency the thesis project has, as the chosen runtime does not support WASI fully at the time the report is written. Thus, any changes in the Wasm and WASI API will probably cause breaking changes in the project (for instance, customers' binaries may not be executed anymore).

5.3 Constraints

There are restrictions that have an effect on the final product in addition to design choices and dependencies, and they must be satisfied.

While the design constraints were already discussed in the previous section, the most important constraint is time. With a high demand for the product, a speedy delivery is crucial. The product might lose potential customers if the Ninchat technical team and the researcher are unable to meet the deadline.

Additionally, WebAssembly and WASI are relatively new topics in the software engineering world and as a result, there are not yet many companies that have used Wasm and WASI with a serious production project and thus the potential problems (however rare) are not yet fully understood.

Additionally, these new-comer technologies fall victim to incomplete SDK implementations and a lack of enough guides and documentation. There are a few examples, but most of them are no longer

maintained or are merely an experimental project without a viable solution. For instance, *Swiftwasm*, which aims to generate Wasm-WASI binaries from Swift projects, is available only on *Ubuntu* and cannot be used on Amazon servers (which run *Amazon Linux*).

Also, despite technical qualifications, the project still requires an in-depth understanding of software engineering principles to solve issues in developing the project.

5.4 Development

As mentioned earlier, the runtime is initiated with a request from Ninchat Core, and then the binary communicates with the runtime to send and receive data. *_start* is a part of WASI that intends to let the host application runs the binary once it is called. For example, calling *_start* from a Rust binary means the `main()` is called.

Upon starting the chatbot runtime, it starts the HTTP server as an RPC handler and listens to different endpoints. The RPC server is the component where Ninchat Core, which is the component responsible to manage Ninchat microservices, communicates with the runtime. The server may have a limited set of endpoints for the first versions of the product, but it is a flexible solution that can offer more endpoints in the future.

Among currently available endpoints, the RPC server includes *init* and *event* endpoints. When an *init* POST request is received by the RPC server, the runtime starts fetching the binary content from S3 and then calls *_start* ABI to let the binary initiates itself. Once initiated, the binary reads required configurations from S3 storage using defined imported functions and starts communication with Ninchat and other services to accomplish the customer's defined tasks.

In the meantime, Ninchat Core can send different events to the binary using the same RPC server and *event* endpoint. However, since all interactions between the runtime and the binary must be initiated by the binary, the binary must continuously poll for new events to get notified of new data.

The following diagram shows how these internal and external components are in connection to make the product usable. *RuntimeRPC* is the HTTP server that is in connection with *RPCEventHandler* and *RPCInitializer* that are mapped to *event* and *init* endpoints respectively. Also, the Wasm binary is in connection with *StorageHandler* and *EventHandler* to manage storage as well as to communicate with internal and external resources. *FetchBinaryHandler* and *BinaryStarter* tasks are in connection with *RPCInitializer* for downloading the binary and calling *_start* to initiate the binary.

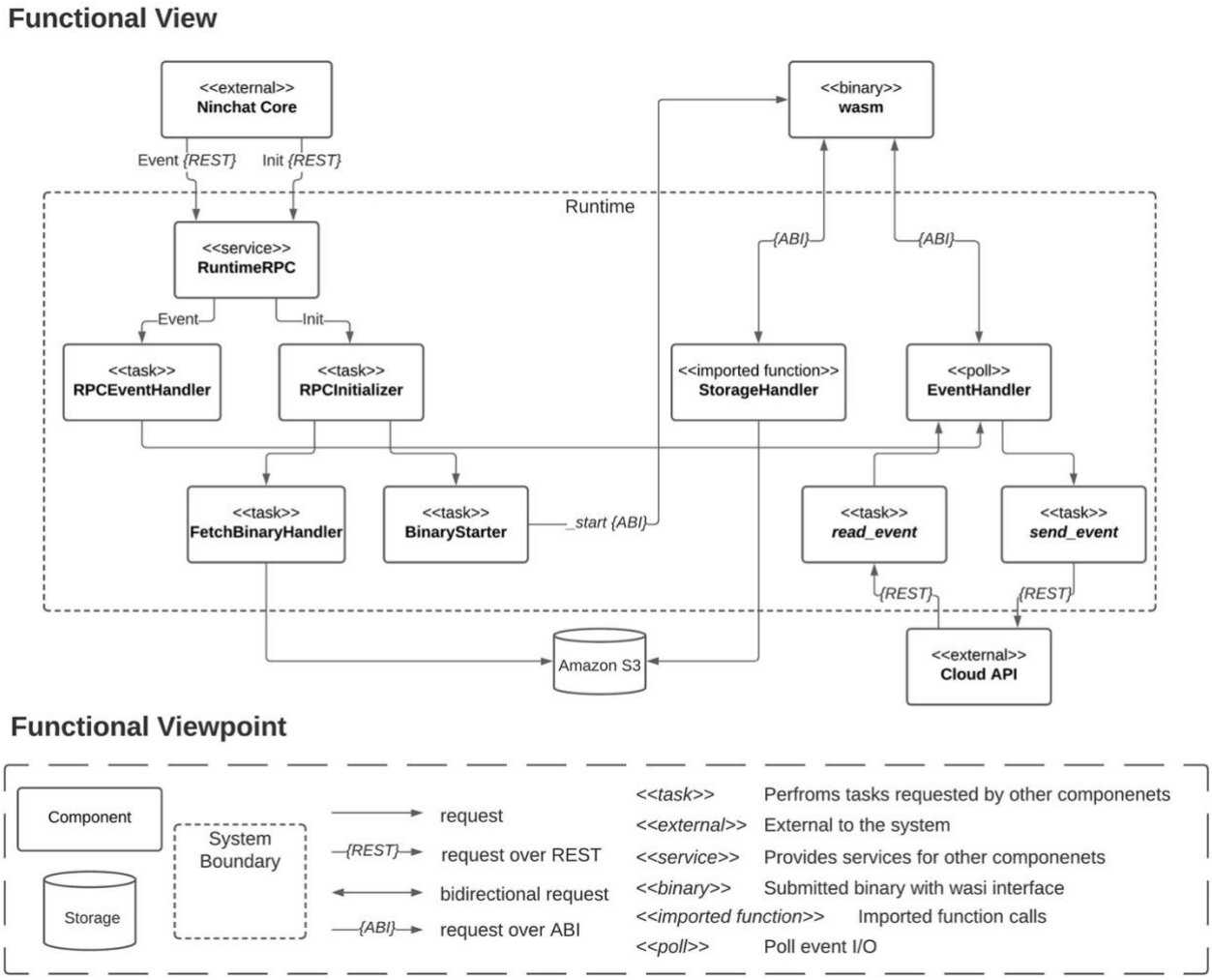

Figure 18 - Functional View

On the development side, new changes (including features developed or bugs that are fixed) are done on the developer's local machine with the help of virtualization technologies such as *VMware* and *Docker*. Once the code is ready to be used in the production, it is *committed* and *pushed* to the remote repository hosted on *GitHub*.

As soon as changes get online, automated tests start checking if the new changes have broken the codebase, and also if they follow agreed coding styles. In addition to automated tests and linting tools, every change is submitted to be *merged* into the production codebase as a *pull-request,* which is then reviewed and approved by technical teammates.

Thus, a change is accessible publicly only when both automated tests and manual reviews are passed. Once it is ready for deployment, it can be deployed to *dev, staging,* or *prod* environments according to agreed plans. It is always wise to deploy recent changes with supervision to avoid releasing incomplete, broken, or unwanted changes.

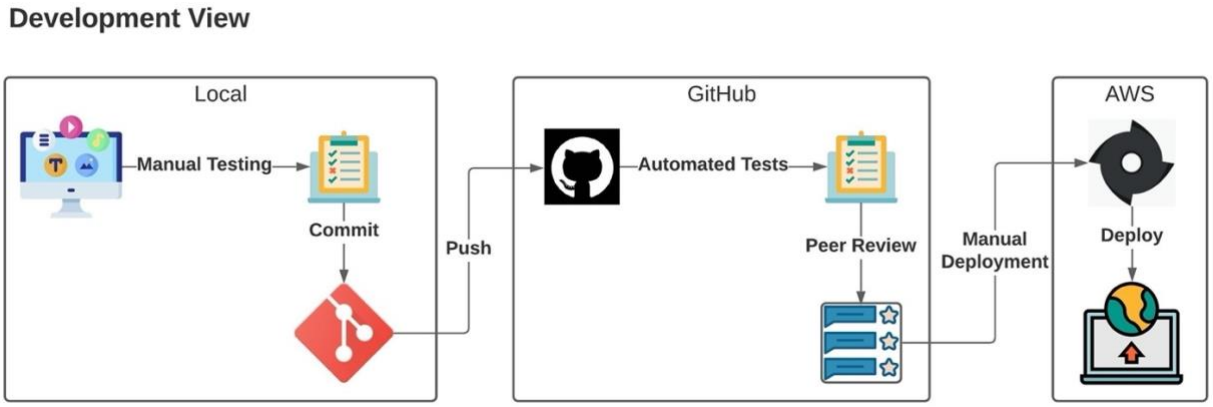

Figure 19 - Development View

5.5 I/O Model

Defining a protocol for delivering data to binary was one of the challenges in developing the product. Because the binary must initiate all communications with the runtime and that the runtime cannot push data to the binary, the communication protocol would have an effect on the development of both the runtime and a binary. Since the protocol is the backbone of the project, any decisions taken must be supported with strong justification. Even if there is a chance for the model to change in the future, the change will not be free and backward compatibility must be considered.

WASI offers a communication interface between a number of embedded systems, while Wasm relies on function calls from both the runtime and an embedded binary. In WASI, asynchronous tasks with an unpredicted size of data are practically suited to read, write, and poll ABIs with a similar approach to sending and receiving events in POSIX.

In order to leverage the WASI ABI more effectively, the runtime first tried providing standard function calls in combination with WASI:

- Minimize exposed interfaces by taking advantage from WASI.
- Manage events using a shared or multiple file descriptor.
- Manage incomplete events and occurred errors more efficiently.

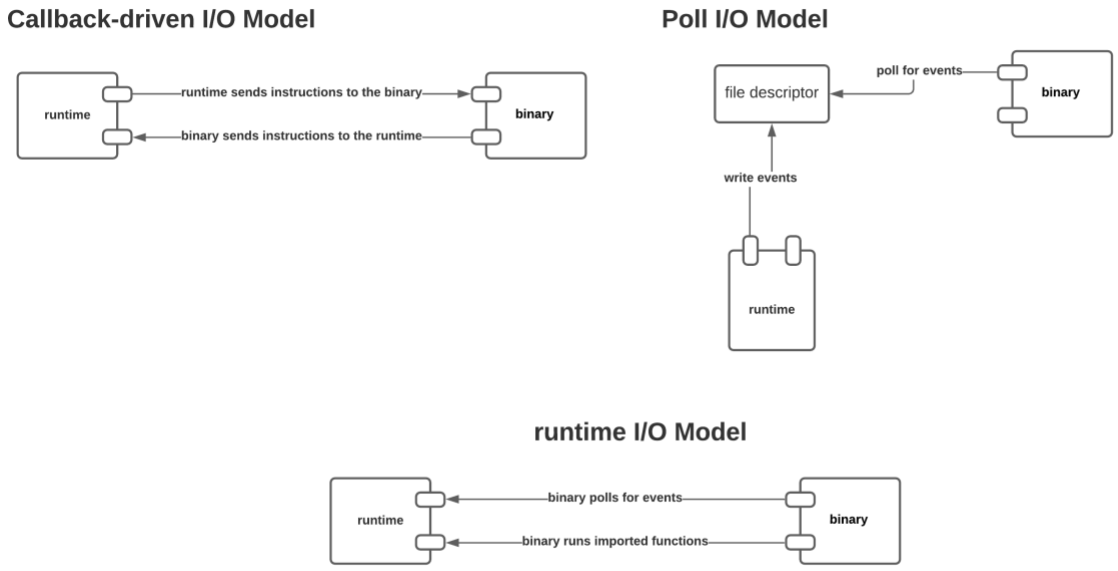

Figure 20 - I/O Model

However, due to limitations imposed by the *wasmtime Go SDK*, reading and writing through POSIX pipes are not yet supported. Thus, the runtime must define the WASI ABI as imported functions and simulates the same mechanism. The approach makes it easier for the runtime to migrate to the WASI version once the support is added in the future. With all in mind, the following activity diagrams depict internal steps of how the underlying communications are done.

Communication through a normal imported function

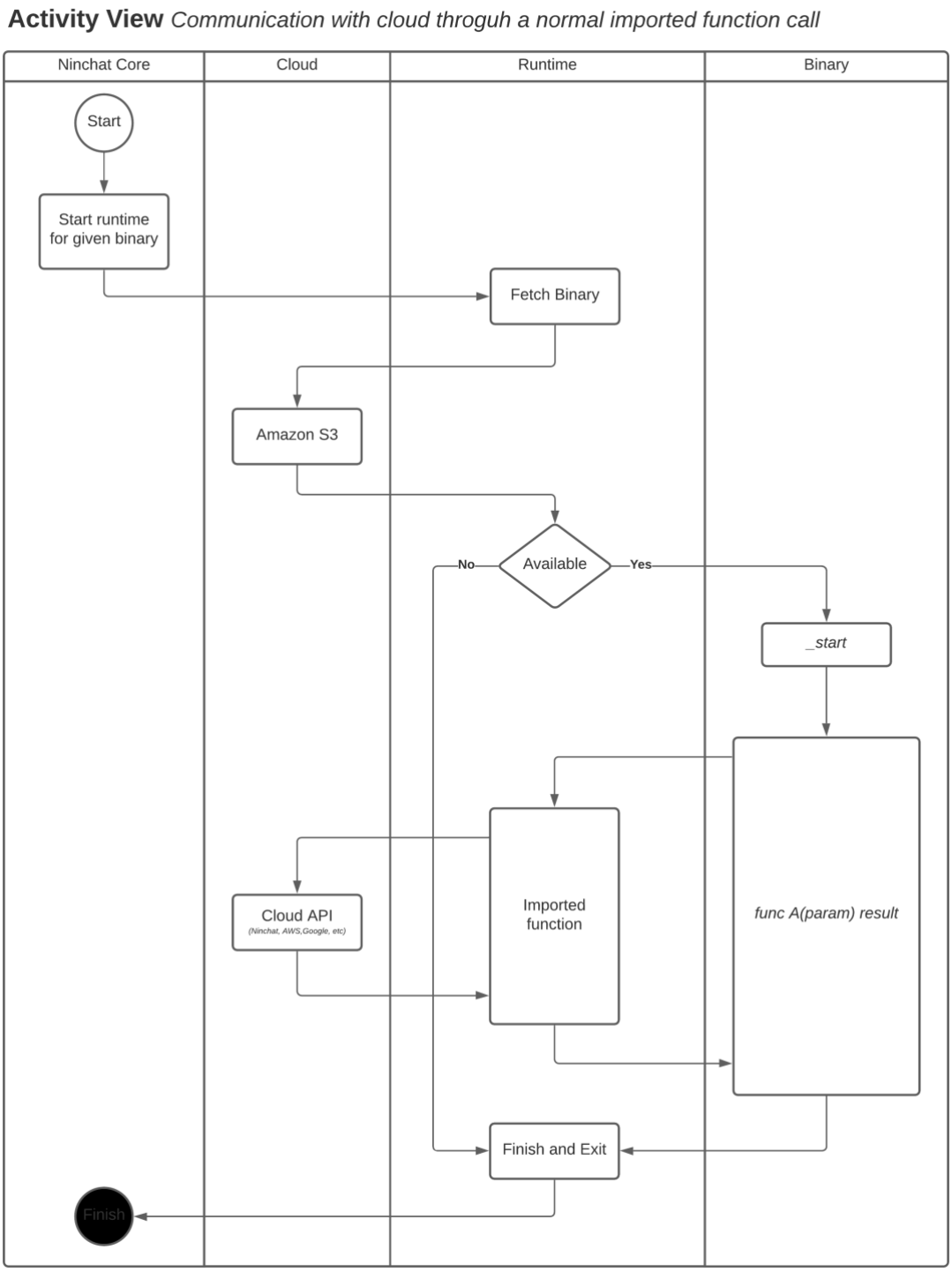

Figure 21 - Activity View, Imported Function Call

Communication by polling events

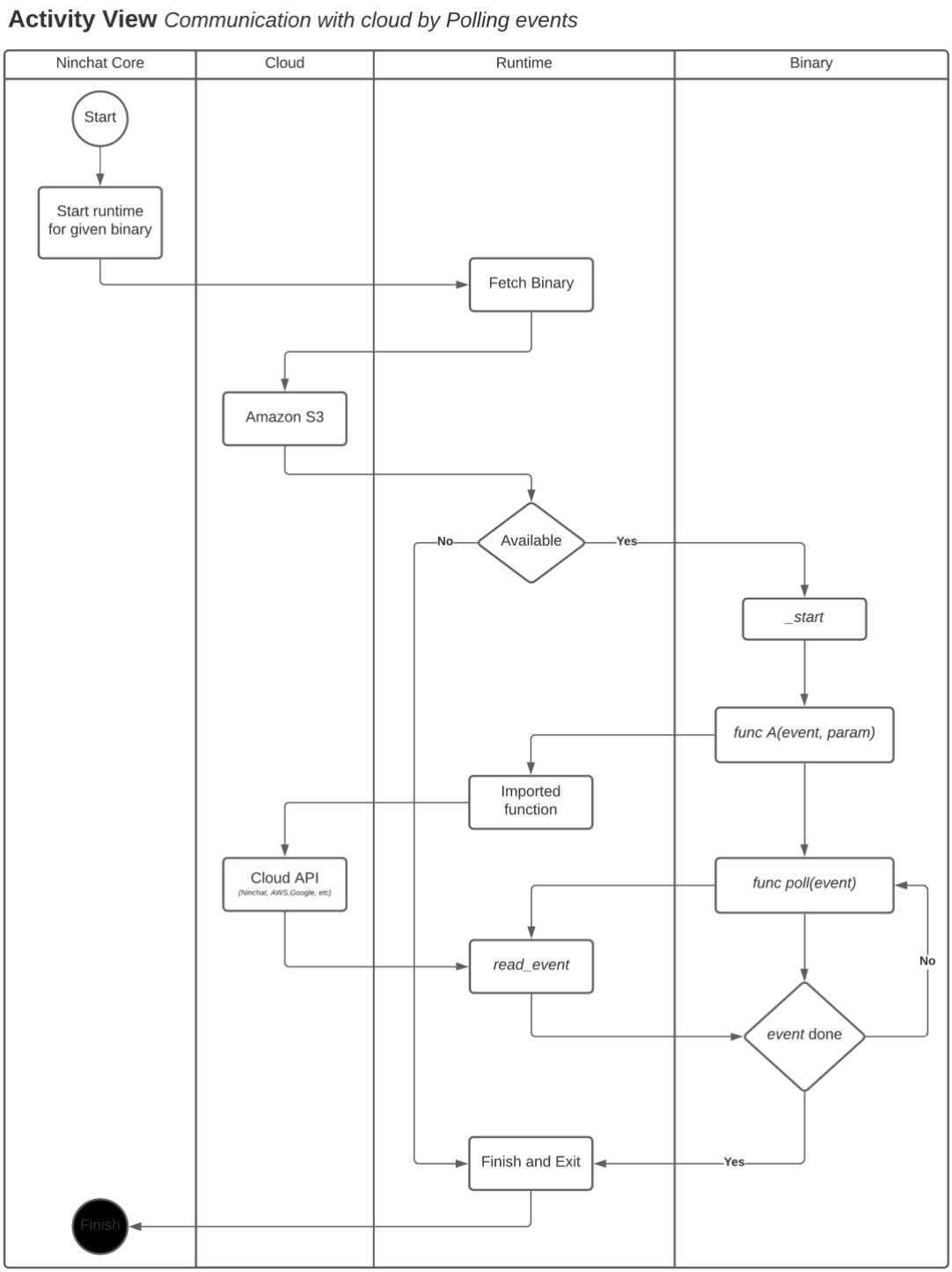

Figure 22 - Activity View, Poll Event

5.6 Event Payload

In accordance with the defined *polling* approach in the previous section, the binary must receive data from the runtime in a blocking approach. However, delivering the data with an unpredictable size would be a problem if the binary is not ready to receive it. Thus, the data communication must be done over a specific data structure that contains the size as a header.

The *event payload* described below is the structure to hold streaming data between a binary and the runtime. The structure is capable of transmission of an unlimited size of data over a limited buffer space to address the issue. The structure lets both parties know the size of the data as well as the target that the data should be sent or is received from.

The first 4-byte indicates the size of the payload lets both parties know what buffer size it needs for receiving the data. Also, the size lets the other party knows if the payload is chunked into smaller parts to fit into the provided buffer size.

The payload starts from the 5th byte and itself consists of three main parts. The first 2 bytes are known as the *event code,* which is used to identify the target that the data must be sent or is received from. Then, there are 2 bytes reserved for later uses and after that, the data is placed.

The following diagram shows the payload with the header, code, reserved, and the data sections.

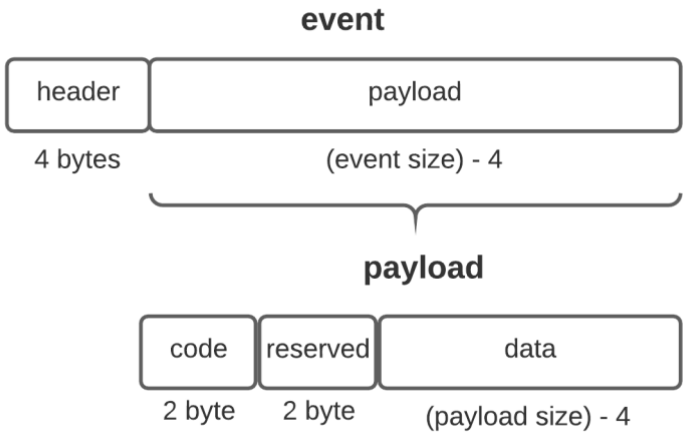

Figure 23 - Event Payload

6 DEMONSTRATION

Following the previous sections that presented pictures of the Wasm system design and introduced WebAssembly to make the project feasible, this section shows the interface that Wasm binaries need to agree to in order to communicate with the Wasm runtime. These functions are defined in WAT, a text format for Wasm binary format, which should be converted into a supported language to form a complete binary.

All ABIs use *'ninchat'* as the module name to distinguish from other interfaces such as WASI. So far, only Rust supports multiple returns, thus, the runtime uses output parameters instead, which are supported by other languages and their runtimes. Additionally, all ABIs return 0 in case of success and a non-zero value in case of failure.

6.1 Storage Management

The interface, as the name suggests, is supposed to help a binary deal with a key-value database. It consists of *Store, Load,* and *Delete* ABIs as described in the following

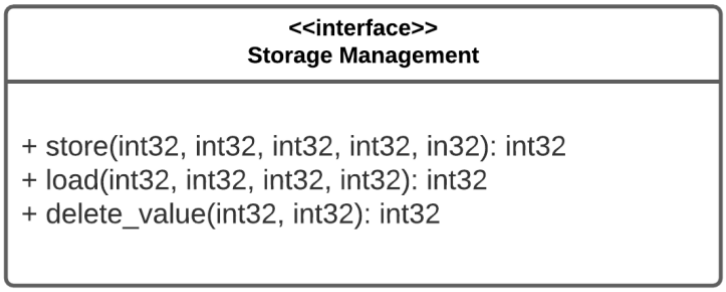

Figure 24 - Storage Interface

6.1.1 Store

```
(import "ninchat" "store" (func (param i32 i32 i32 i32 i32) (result i32)))
```

The ABI is used to save a value associated with a key in the database. All parameters are *input* type and point to addresses in the memory. The first two parameters are the key's content and length offsets, and the second two ones are for the value. Also, the last parameter determines the value must be updated or ignored in case the key already exists in the database.

6.1.2 Load

```
(import "ninchat" "load" (func (param i32 i32 i32 i32) (result i32)))
```

The ABI is used to load a value for the given key from the database. The first two parameters are the key's content and length and thus, are *input* type. The last two ones are *output* types pointing to the value's content and length in the memory.

6.1.3 Delete Value

```
(import "ninchat" "delete_value" (func (param i32 i32) (result i32)))
```

The ABI is used to remove a single key with its associated value from the database. It accepts the key's content and length offsets in the memory.

6.2 Event Management

The interface, as the name suggests, is supposed to help a binary receive data from and send responses back to the runtime. It is the core part of the project, as it makes cloud data API integrations possible. Also, since the WASI is not yet fully supported by *wasmtime GO SDK*, the interface defines normal imported functions instead.

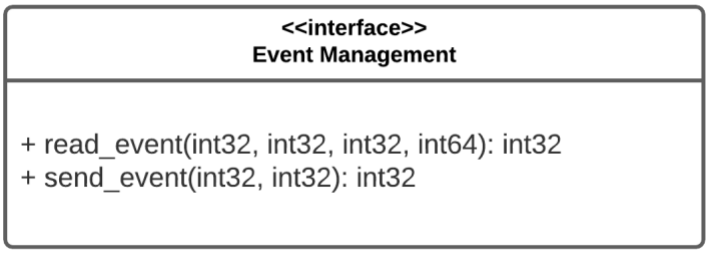

Figure 25 - Event Interface

6.2.1 Read Event

```
(import "ninchat" "read_event" (func (param i32 i32 i32 i64) (result i32)))
```

The ABI is used by the binary to poll streams from the runtime. The first parameter is an *output* one pointing to the payload's offset in the memory. The second parameter is set by the binary to let the runtime knows the available buffer size (which then the runtime decides if the data must be parted). The third parameter is *output* again and sets the length of the value. And the last parameter, which is an *int64,* is a timeout that lets the operation be ended if the payload is not received in the given period.

6.2.2 Send Event

```
(import "ninchat" "send_event" (func (param i32 i32) (result i32)))
```

The ABI is used by the binary to sends streams to the runtime. All parameters are *input* types, where the first one points to the payload's offset in the memory and the second one is the length of payload. (This let the runtime knows if the data is parted and must be polled).

6.3 End-to-End example 1

The following are the algorithm and code examples (written in Swift) that illustrate the functionality of the Wasm binary and its runtime. The ultimate goal is to send an event to an external resource by reading configurations from the storage. The diagram illustrates the algorithm used for this simple example.

After the binary has been launched, the first step is to load configurations from the storage by means of the customer key. The configuration is written in JSON format that allows for customer-specific values to be stored. It is available through the customer-specific key and is intended to help enhance flex.

The binary exits if the configuration is not valid. Otherwise, it processes the configuration and extracts code for communication with the Amazon Machine Learning service. If the code is negative, it means the Wasm binary is not permitted to use the service.

However, if the code is valid, the binary tries to send a simple *init* event to Amazon through the runtime. If it receives the success code (0), then it can continue loading more code and send more events.

All functions above are demonstrated in the following pictures that are more suitable for reading purposes. The last picture shows the main body of a Swift Wasm binary and contains calling all defined functions.

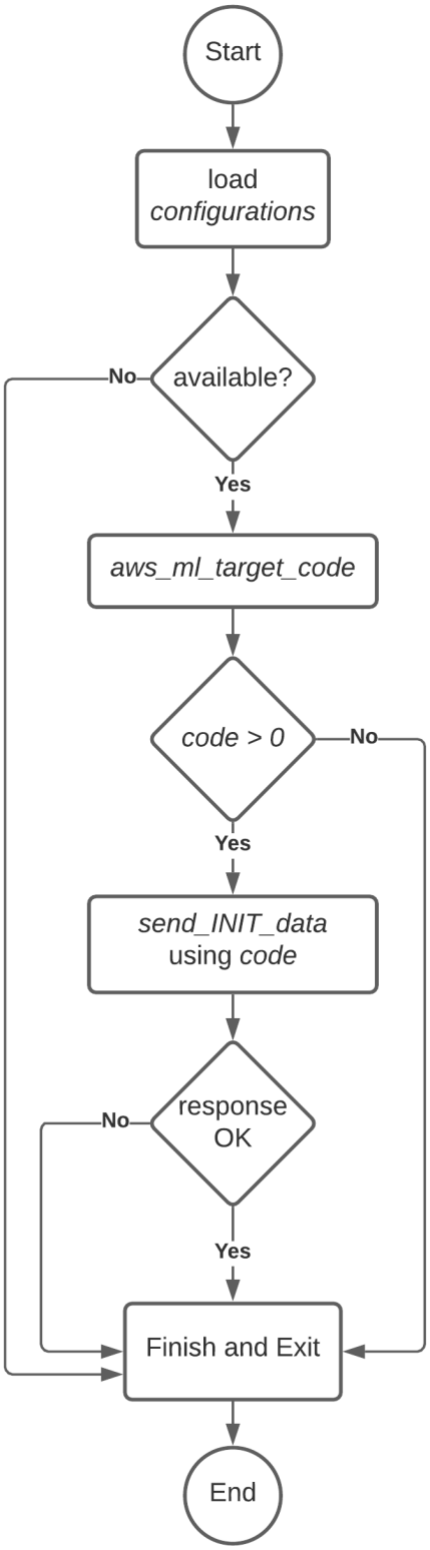

Figure 26 - End-To-End Wasm example 1

example-1.swift

```swift
func loadConfigurations() -> String? {
  // define variables
  let configKey = /* CUSTOMER_CONFIG_KEY */
  let configKeyPointer = UnsafeMutablePointer<CChar>.allocate(capacity: configKey.count)
  let configVal = UnsafeMutablePointer<CChar>.allocate(capacity: BUFF_SIZE)
  var configValLen: Int32 = -1

  // allocate pointer
  configKeyPointer.assign(from: configKey, count: configKey.count)

  // call 'load' ABI to fetch configurations from the runtime
  let loadResult = load(configKeyPointer, Int32(configKey.count),
                        configVal, Int32(BUFF_SIZE), &configValLen)

  if loadResult < 0 {
    // 'load' failed
    return nil
  }
  // convert pointer to String
  return String(cString: configVal)
}

```

Figure 27 – Wasm example 1: Loading Configurations

example-1.swift

```swift
func extractAWSMLCode(config: String) -> Int32 {
  // convert String to Data
  guard let data = config.data(using: .utf8) else { return -1 }

  do {
    // convert Data to map [String:Any]
    let json = try JSONSerialization.jsonObject(with: data,
                                                options: .allowFragments) as? [String:Any]

    // extract 'aws_ml_target_code' from the map
    // or return -1 if not fount
    return Int32(json?["aws_mml_target_code"] as? Int ?? -1)
  } catch {
    // JSON convertion failed
    return -1
  }
}

```

Figure 28 - Wasm example 1: Extracting AWS Code

example-1.swift

```swift
func send_AWS_init(code: Int32) -> Int32 {
    // define local variables
    let value = ""
    let header = value.count + EVENT_PAYLOAD_HEADER
    let reserved = 0

    // event_payload header
    let headerPointer = UnsafeMutablePointer<Int8>.allocate(capacity: EVENT_PAYLOAD_HEADER)
    headerPointer.initialize(to: Int8(header))

    // event_payload code
    let codePointer = UnsafeMutablePointer<Int8>.allocate(capacity: EVENT_PAYLOAD_CODE)
    codePointer.initialize(to: Int8(code))

    // event_payload reserved
    let reservedPointer = UnsafeMutablePointer<Int8>.allocate(capacity: EVENT_PAYLOAD_RESV)
    reservedPointer.initialize(to: Int8(reserved))

    // event_payload value
    // value is empty for this request
    let valuePointer = UnsafeMutablePointer<CChar>.allocate(capacity: BUFF_SIZE)
    valuePointer.assign(from: value, count: value.count)

    let pointer = UnsafeMutablePointer<Int8>.allocate(capacity: value.count+8)
    pointer.initialize(from: headerPointer, count: EVENT_PAYLOAD_HEADER)
    pointer.advanced(by: 4).initialize(from: codePointer, count: EVENT_PAYLOAD_CODE)
    pointer.advanced(by: 6).initialize(from: reservedPointer, count: EVENT_PAYLOAD_RESV)
    pointer.advanced(by: 8).initialize(from: valuePointer, count: value.count)

    // call 'send_event' ABI
    return send_event(pointer, Int32(value.count+8))
}

```

Figure 29 - Wasm example 1: Sending an event to the runtime

example-1.swift

```swift
import Foundation
import c_header // contains ABI headers

let BUFF_SIZE = 256 // Maximum Buffer Size
let EVENT_PAYLOAD_HEADER = 4 // Event Payload 'header' size
let EVENT_PAYLOAD_CODE = 2 // Event Payload 'code' size
let EVENT_PAYLOAD_RESV = 2 // Event Payload 'reserved' size

let configurations = loadConfigurations()
if configurations == nil {
  exit(-1)
}

let awsCode = extractAWSMLCode(config: configurations!)
if awsCode < 0 {
  exit(awsCode)
}

let awsInit = send_AWS_init(code: awsCode)
if awsInit == 0 {
  /* AWS initiated */
}
```

Figure 30 - Wasm example 1: Start the binary with all functions defined above

6.4 End-to-End example 2

In the first example, we loaded configurations and sent events to an external resource. In the example below, we read the events received from the external resource and update the settings to avoid a duplicate request.

This diagram illustrates how the binary waits for a specified time for an event with the *user_name*. If it is unable to receive the event within that time window, the runtime returns timeout to allow the binary to continue execution.

To make sure that there are enough memory resources allocated for all data and that it is able to receive the data, the binary first reads 4 bytes of the payload to find out its size. Once the binary has received the data, it can process it.

If provided data contains a valid *user_name*, the binary would update the configuration storage indicating the *user_name* is loaded and should not be requested again.

For each instruction, if the operation fails, the environment returns a non-zero value, and the binary exits with the error code.

All functions above are demonstrated in the following pictures that are more suitable for reading purposes. The last picture shows the main body of a Swift Wasm binary and contains calling all defined functions.

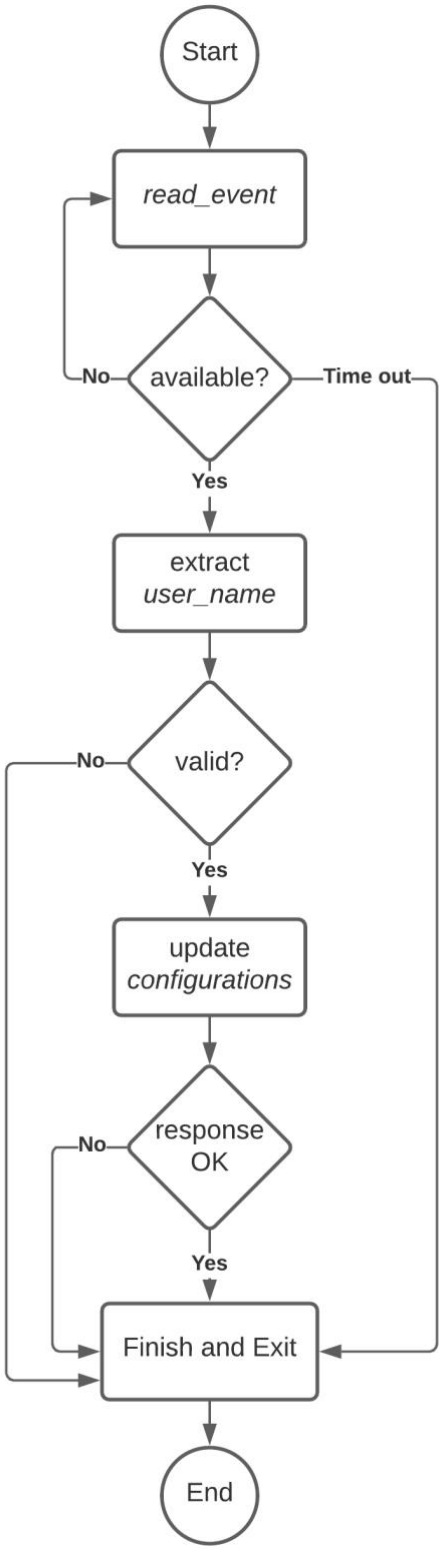

Figure 31 - End-to-End Wasm example 2

example-2.swift

```swift
func read_event(timeout: Int64) -> String? {
    // define local variables
    var result: String?
    var bufSize = EVENT_PAYLOAD_HEADER
    var readHeaderOnly = true

    // poll the ABI
    repeat {
        // define memory buffer with maximum length
        let value = UnsafeMutablePointer<CChar>.allocate(capacity: BUFF_SIZE)
        var valueLen: Int32 = Int32.min

        // read event from the runtime with timeout in seconds
        if read_event(value, Int32(bufSize), &valueLen, timeout * 1000000000) < 0 {
            // error
            break
        } else if valueLen == -1 {
            // timeout
            break
        }

        // read header only to get the size of data and set to variable
        if readHeaderOnly {
            bufSize = Int(value.pointee)
            readHeaderOnly = false
        }
        // read rest of data according to given size
        else {
            // convert pointer to byte array
            let bytes = Array(UnsafeBufferPointer(start: value, count: bufSize)) as! [UInt8]

            // convert event_payload 'data' to String
            result = String(data: Data(bytes[6..<bufSize]), encoding: .utf8)
            break
        }
    } while true

    return result
}
```

Figure 32 - Wasm example 2: polling events

example-2.swift

```swift
func updateConfigurations(value: String) -> Int32 {
    // define local variables
    let key = /* CUSTOMER_CONFIG_KEY */
    let keyPointer = UnsafeMutablePointer<CChar>.allocate(capacity: key.count)
    let valuePointer = UnsafeMutablePointer<CChar>.allocate(capacity: value.count)
    valuePointer.assign(from: value, count: value.count)

    // allocate pointer
    keyPointer.assign(from: key, count: key.count)

    // call 'store' ABI to update configurations
    return store(keyPointer, Int32(key.count), valuePointer, Int32(value.count), 1)
}
```

Figure 33 - Wasm example 2: updating configurations

example-2.swift

```swift
import Foundation
import c_header // contains ABI headers

let BUFF_SIZE = 256 // Maximum Buffer Size
let EVENT_PAYLOAD_HEADER = 4 // Event Payload 'header' size

let user_name = read_event(timeout: 5)
if user_name == nil {
  exit(-1)
}

let updateCode = updateConfigurations(value: "1")
if updateCode < 0 {
  exit(updateCode)
}

/* Configurations Updated */

```

Figure 34 - Wasm example 2: Start the binary with all functions defined above

7 DISCUSSION

As an objective-centered solution, the thesis was driven primarily by an industrial need to provide a technical solution to meet Ninchat's customers' demand for supporting chatbots. The first step was to define goals and objectives, and also research questions that would be addressed by the thesis:

- **RQ1**: *How can customers build and submit chatbots to act on behalf of an agent?*
- **RQ2**: *How can Ninchat benefit from WebAssembly to provide a robust environment to run customers' Chatbots*
- **RQ3**: *How different cloud APIs are accessible in the new environment?*

In order to answer questions and accomplish objectives, the thesis identified stakeholders and addressed their needs, identified risks and prioritized them, proposed a technical solution, and proven its overall functionality. An enterprise-wide solution such as this required a deep knowledge of quality assurance, architecture and low-level programming as well as ABI design skills. In the upcoming sections, we will discuss research findings, define project successes and failures, and identify potential future research projects.

7.1 Results and Outcomes

This thesis was not only a project with outcomes limited to Ninchat, but also an opportunity to contribute to the open-source community and help others. As discussed earlier, WebAssembly and WASI are new technologies which suffer from incomplete implementations and documentation. As an example, *Swiftwasm* and *wasmer* added support for the CentOS Linux family (including CentOS, Fedora, Amazon Linux, RHEL) and other distributions such as Gentoo Linux after a continuous contribution from the researcher during the thesis. That's a big step towards reaching the Wasm-WASI community's goal of releasing on popular distributions and attracting users with skills in other programming languages than Rust.

The thesis also provided the opportunity for contributors to contribute to the repository, as well as the opportunity to fill the gap in documentation and examples through the publication of online essays.

Through this, hundreds of developers have been able to use *Swiftwasm* to develop Wasm-WAS-compliant applications.

One of the most notable outcomes of the thesis is the final product and its architecture. Tested mostly by automated tests, the final product and its architecture proved to be functional when deployed to the Ninchat's backend. Binaries written in *Swift*, *Rust*, and *AssemblyScript* were examined, and the given records were transferred to the Ninchat DynamoDB storage in all cases.

Additionally, we developed practical binaries in different programming languages to work perfectly with the product, and despite all obstacles in working with low-level concepts such as pointers, the interface has been proven usable in any programming language.

Lastly, the thesis was a great opportunity for the researcher to apply theoretical skills to real-world problems. The project involved working heavily with non-trivial software development topics like pointers, memory management, binary optimizations, etc., which have improved the researcher's overall skills. Moreover, the elicitation of requirements, building an application from scratch using diagrams and models, deploying a microservice to the cloud, and developing standard ABIs have been among the most valuable experiences gleaned by the researcher so far.

7.2 Successes and Failures

First of all, the thesis project showed how to design and generate enterprise software using both WASI and WebAssembly technologies. It followed defined quality measures and successfully integrated into Ninchat's infrastructure, making the product available to all Ninchat customers just like other microservices.

Secondly, the product satisfied technical requirements. It was developed according to best practices and implemented using T.D.D approach with a test coverage of about 90% on all internal and external components. It was tested on different binaries, and it performed as planned.

The thesis was also a successful experience for the researcher as well as Ninchat in terms of communication with stakeholders. Information gathered during several elicitation sessions has

enabled both parties to understand the product context better and to adjust their expectations. Ninchat's design and development teams have shown positive attention toward the product's functionality and infrastructural requirements during the entire development process.

Despite the success of WebAssembly and WASI in terms of executing language-independent binaries on the server, the project failed to work directly with WASI. Since neither the *wasmer* nor the *wasmtime Go SDKs* at the time the report was written supported WASI ABI, the project could not use them for its first public version. These limitations have a result in defining all ABIs as *imported functions* instead of the normal approach.

7.2.1 Future Research and Developments

The thesis project can be used as a jumping off point for creating WebAssembly compatible projects. There are hundreds of research and development opportunities for developing new projects or updating the existing project. In addition, since Wasm-WASI projects require both academic knowledge and practical skill, the thesis can be used as a successful reference for future cases.

Also, as discussed in earlier sections, the project is expected to include user interfaces in the following versions, which will enable customers to be able to test and submit Wasm binaries before releasing a public version. Interfaces must be implemented in accordance with Ninchat’s current design language and be compatible with the latest technologies and best practices.

Furthermore, there appears to be potential for research-and-development projects for adding WASI support to *wasmtime* and *wasmer* SDKs, not only for Go but for all supported languages. It would have a major impact on the open-source and WebAssembly communities.

Regarding WASI support, it is vital for the thesis to update the defined interface as soon as WASI is fully supported by *wasmtime*. Replacing imported functions with WASI ABI may result in breaking changes and should be considered by all stakeholders.

Additionally, since working with low-level ABI (such as pointers, memory objects, etc.) may have difficulties in different languages and with considering the fact that Wasm supports only scalar types

(`int` and `float`), one prominent potential topic to work on in the future is developing helper libraries for different languages, so Ninchat's customers would not deal with low-level concepts. Helper libraries can also avoid breaking changes when moving from custom imported functions toward WASI ABI in the next versions of the environment.

And finally, one other potential area for research is about finding practical and useful external services to be integrated as external cloud APIs. For example, it was once mentioned in stakeholders' meetings that the project can use AI-powered services to suggest automatic answers to users, as well as to extract useful information from users' messages and replies.

8 SUMMARY

In response to the continuously increasing demand for remote digital services throughout the world and particularly in Finland, Ninchat decided to offer a new service by hosting, providing, and running intelligent chatbots. A chatbot manages multiple users on behalf of an agent, available to respond to a query non-stop. The thesis set out to provide a practical method for Ninchat to meet client demand through a practical solution.

In keeping with the Design Science Research Method, this thesis begins with a description of the technical terms and concepts that were used to create the product, specifically WebAssembly and the WASI. Additionally, concepts like Chatbots and Lightbots were defined and explained.

Upon describing concepts and technologies, requirements and features from the stakeholders' point of view are identified, listed, and prioritized. Each requirement is gathered using different elicitation techniques, which are categorized into business, user, and system requirement groups, according to their level of detail. Furthermore, the criteria by which to evaluate the product's success, the scope of the product, and potential risks are presented.

Several diagrams are used in the thesis to build the product. Among them are Activity and Functional diagrams. Once the internal steps and boundary constraints are understood, the thesis presents protocols and data structures for communication between binary and the system.

In conclusion, the thesis lists and describes the interface of the binary with examples, which demonstrate a practical implementation of Wasm using the binary interface.

BIBLIOGRAPHY

Adams, K. et al., 2018. *The HipHop Virtual Machine.* New York, Association for Computing Machinery.

AI, M., 2020. *Why AI-powered Chatbots are superior to Click-based Chatbots.* [Online]
Available at: https://laptrinhx.com/why-ai-powered-chatbots-are-superior-to-click-based-chatbots-197489401/
[Accessed 24 January 2021].

AJMC, 2021. *A timeline of Covid-19 developments in 2020.* [Online]
Available at: https://www.ajmc.com/view/a-timeline-of-covid19-developments-in-2020
[Accessed 23 January 2021].

Alan Faisandier, G. R. C. A. R. T. R. A. A. S., 2020. *System Requirements.* [Online]
Available at: https://www.sebokwiki.org/wiki/System_Requirements
[Accessed 17 January 2021].

AlexSoft, 2018. *Functional and Nonfunctional Requirements: Specification and Types.* [Online]
Available at: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/
[Accessed 16 January 2021].

AltexSoft, 2018. *Software Documentation Types and Best Practices.* [Online]
Available at: https://blog.prototypr.io/software-documentation-types-and-best-practices-1726ca595c7f
[Accessed 23 January 2021].

Alvesson, M. & Sandberg, J., 2013. *Constructing Research Questions: Doing Interesting Research.* 1 ed. s.l.:SAGE.

Amazon, 2021. *What Is Conversational AI?.* [Online]
Available at: https://developer.amazon.com/en-GB/alexa/alexa-skills-kit/conversational-ai
[Accessed 24 January 2021].

Apple, 2020. *The Swift Programming Lanaguage.* [Online]
Available at: https://docs.swift.org/swift-book/LanguageGuide/AutomaticReferenceCounting.html
[Accessed 14 January 2021].

Backtrace, 2016. *Memory Management Bugs: An Introduction.* [Online]
Available at: https://backtrace.io/blog/backtrace/introduction-to-memory-management-errors/
[Accessed 17 January 2021].

Bees, J., 2018. *Gluing the Web and WebAssembly Together.* [Online]
Available at: https://www.toomanybees.com/storytime/gluing-the-web-and-webassembly-together
[Accessed 2021 January 24].

Callahan, D., 2019. *Debugging WebAssembly Outside of the Browser.* [Online]
Available at: https://hacks.mozilla.org/2019/09/debugging-webassembly-outside-of-the-browser/
[Accessed 25 March 2021].

Cimpanu, C., 2018. *Changes in WebAssembly Could Render Meltdown and Spectre Browser Patches Useless.* [Online]
Available at: https://www.bleepingcomputer.com/news/security/changes-in-webassembly-could-render-meltdown-and-spectre-browser-patches-useless/
[Accessed 14 January 2021].

Clark, L., 2017. *What makes WebAssembly fast?.* [Online]
Available at: https://hacks.mozilla.org/2017/02/what-makes-webassembly-fast/
[Accessed 15 January 2021].

Clark, L., 2019. *Standardizing WASI: A system interface to run WebAssembly outside the web.* [Online]
Available at: https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/
[Accessed 1 February 2021].

Clement, J., 2020. *Global digital population as of October 2020.* [Online]
Available at: https://www.statista.com/statistics/617136/digital-population-worldwide/
[Accessed 23 January 2021].

ComputerHope, 2019. *WebAssembly.* [Online]
Available at: https://www.computerhope.com/jargon/w/webassembly.htm
[Accessed 14 January 2021].

Duin, M., 2020. *Going from JavaScript to WebAssembly in three steps.* [Online]
Available at: https://engineering.q42.nl/webassembly/
[Accessed 23 January 2021].

Expert-System-Team, 2020. *Chatbot: What is a Chatbot? Why are Chatbots Important?.* [Online]
Available at: https://www.expert.ai/blog/chatbot/
[Accessed 24 January 2021].

Fang, A., 2021. *REST vs gRPC: Understanding Two Very Different API Styles.* [Online]
Available at: https://rapidapi.com/blog/rest-vs-grpc-understanding-two-very-different-api-styles/
[Accessed 6 February 2021].

Flow.ai, 2019. *Different types of chatbots: Rule-based vs. NLP.* [Online]
Available at: https://flow.ai/blog/kb-different-kinds-of-chatbots
[Accessed 24 January 2021].

Fowler, T., 2020. *Is JavaScript Front End or Back End?.* [Online]
Available at: https://careerkarma.com/blog/javascript-front-end-or-back-end/
[Accessed 24 January 2021].

Geeks, G. f., 2020. *Different types of risks in Software Project Development.* [Online]
Available at: https://www.geeksforgeeks.org/different-types-of-risks-in-software-project-

development/
[Accessed 6 February 2021].

Ignatchenko, S., 2018. *Why is C++ so much faster than Javascript, but harder to code?.* [Online]
Available at: https://www.quora.com/Why-is-C-so-much-faster-than-Javascript-but-harder-to-code
[Accessed 14 January 2021].

Jain, A., 2020. *Software Engineering | Requirements Elicitation.* [Online]
Available at: https://www.geeksforgeeks.org/software-engineering-requirements-elicitation/
[Accessed 27 January 2021].

Jenny, 2020. *What is a Chatbot?.* [Online]
Available at: https://www.getjenny.com/what-is-a-chatbot
[Accessed 24 January 2021].

Joseraj, J., 2019. *The JVM Architecture Explained.* [Online]
Available at: https://dzone.com/articles/jvm-architecture-explained
[Accessed 26 March 2021].

Kemp, S., 2020. *Digital 2020: Finland.* [Online]
Available at: https://datareportal.com/reports/digital-2020-finland
[Accessed 23 January 2021].

Levick, R., 2019. *Why should you use Rust in WebAssembly?.* [Online]
Available at: https://opensource.com/article/19/2/why-use-rust-webassembly
[Accessed 25 March 2021].

Meijer, E., Wa, R. & Gough, J., 2001. Technical Overview of the Common Language Runtime. *Microsoft Research,* p. 2.

Mozilla, 2021. *Compiling a New C/C++ Module to WebAssembly.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/WebAssembly/C_to_wasm
[Accessed 23 January 2021].

Mozilla, 2021. *HTTP.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP
[Accessed 25 March 2021].

Neuman, R. & Toro, A., 2018. *In-browser mining: Coinhive and WebAssembly.* [Online]
Available at: https://www.forcepoint.com/blog/x-labs/browser-mining-coinhive-and-webassembly
[Accessed 14 January 2021].

Ogasawara, T., 2014. *Workload characterization of server-side JavaScript.* s.l., s.n.

Onsman, A., 2018. *Remote Procedure Call (RPC).* [Online]
Available at: https://www.tutorialspoint.com/remote-procedure-call-rpc
[Accessed 6 February 2021].

P, S., 2018. *When to use gRPC over REST.* [Online]
Available at: https://medium.com/@sankar.p/how-grpc-convinced-me-to-chose-it-over-rest-30408bf42794
[Accessed 7 February 2021].

Parameshwara, S., 2019. *REST vs gRPC.* [Online]
Available at: https://devsuhas.com/2019/12/22/rest-vs-grpc/
[Accessed 7 February 2021].

Parker, J., 2012. *Business, User, and System Requirements.* [Online]
Available at: https://enfocussolutions.com/business-user-and-system-requirements/
[Accessed 6 February 2021].

Pefers, K., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S., 2007. A Design Science Research Methodology for Information System Research. *Journal of Management Information Systems,* pp. 45-77.

Pello, R., 2018. *Design science research — a short summary.* [Online]
Available at: https://medium.com/@pello/design-science-research-a-summary-bb538a40f669
[Accessed 25 January 2021].

Pettersen, K., 2020. *How customer service chatbots are redefining customer engagement with AI.* [Online]
Available at: https://www.intercom.com/blog/customer-service-chatbots/
[Accessed 24 January 2021].

Point, T., 2020. *Software Requirements.* [Online]
Available at: https://www.tutorialspoint.com/software_engineering/software_requirements.htm
[Accessed 27 January 2021].

Romanazzi, S., 2018. From Manual Memory Management to Garbage Collection. *ResearchGate.*

Roser, M., Ritchie, H. & Ortiz-Ospina, E., 2015. Internet. *Our World in Data.*

Saldien, J., 2009. Development of the Huggable Social Robot Probo. On the Conceptual Design and Software Architecture.. *Uitgeverij VUBPRESS Brussels University Press,* p. 62.

Savina, N., 2019. *Five Different Types of Chatbot.* [Online]
Available at: https://medium.com/voiceui/five-different-types-of-chatbot-17bb255b23b4
[Accessed 24 January 2021].

Sevon, S., 2020. *Fight COVID-19 with secure messaging.* [Online]
Available at: https://ninchat-blog.blogspot.com/2020/12/fight-covid-19-with-secure-messaging.html
[Accessed 23 January 2021].

Shafqat, 2019. *Stakeholders in Software Project.* [Online]
Available at: https://xalitech.com/stakeholders-in-software-project/
[Accessed 6 February 2021].

Snapshot, W., 2020. *WASI Snapshot document.* [Online]
Available at: https://github.com/WebAssembly/WASI/blob/master/phases/snapshot/docs.md
[Accessed 17 January 2021].

Solutions, A., 2019. *Types of Chatbot Technology.* [Online]
Available at: https://medium.com/voice-tech-podcast/types-of-chatbot-technology-72d095df2540
[Accessed 24 January 2021].

Spacey, J., 2018. *50 Examples of Business Requirements.* [Online]
Available at: https://simplicable.com/new/business-requirements
[Accessed 6 February 2021].

Stackoverflow, 2020. *What is Rust and why is it so popular?.* [Online]
Available at: https://stackoverflow.blog/2020/01/20/what-is-rust-and-why-is-it-so-popular/
[Accessed 25 March 2021].

Stefanoski, K., Karadimce, A. & Dimitrievski, I., 2019. PERFORMANCE COMPARISON OF C++ AND JAVASCRIPT (NODE.JS -V8 ENGINE). *Research Gate.*

Team, C. I., 2019. *Difference between Rule based and AI-based Chat-bot.* [Online]
Available at: https://www.cosoit.com/difference-between-rule-based-and-ai-based-chatbot
[Accessed 24 January 2021].

TIOBE, 2020. *TIOBE Index for December 2020.* [Online]
Available at: https://www.tiobe.com/tiobe-index/
[Accessed 19 January 2021].

Tkachenko, I., 2019. *FUNCTIONAL VS NON-FUNCTIONAL REQUIREMENTS: MAIN DIFFERENCES & EXAMPLES.* [Online]
Available at: https://theappsolutions.com/blog/development/functional-vs-non-functional-requirements/
[Accessed 16 January 2021].

Tung, L., 2020. *Programming language popularity: Python overtakes Java – as Rust reaches top 20.* [Online]
Available at: https://www.zdnet.com/article/programming-language-popularity-python-overtakes-java-as-rust-reaches-top-20/
[Accessed 14 January 2021].

Venners, B., 2000. *Inside the Java Virtual Machine.* 2 ed. s.l.:McGraw-Hill Osborne Media.

W3School, 2021. *HTTP Request Methods.* [Online]
Available at: https://www.w3schools.com/tags/ref_httpmethods.asp
[Accessed 19 February 2021].

Wallace, E., 2017. *WebAssembly cut Figma's load time by 3x.* [Online]
Available at: https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x/
[Accessed 17 January 2021].

WASI, 2021. *WASI, The WebAssembly System Interface.* [Online]
Available at: https://wasi.dev/
[Accessed 24 January 2021].

Wasmer, 2020. *@wasmer/wasi.* [Online]
Available at: https://docs.wasmer.io/integrations/js/reference-api/wasmer-wasi
[Accessed 25 March 2021].

Wasmtime, 2020. *WASI tutorial.* [Online]
Available at: https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-tutorial.md
[Accessed 2021 March 2021].

Wasmtime, 2021. *Introduction.* [Online]
Available at: https://docs.wasmtime.dev/
[Accessed 7 February 2021].

WebAssembly-Community-Group, 2017. *Types.* [Online]
Available at: https://webassembly.github.io/spec/core/syntax/types.html
[Accessed 24 January 2021].

WebAssembly-Community-Group, 2017. *WebAssembly.* [Online]
Available at: https://webassembly.github.io/spec/core/intro/introduction.html#security-considerations
[Accessed 14 January 2021].

WebAssembly, 2019. *WebAssmebly.* [Online]
Available at: https://webassembly.org/
[Accessed 24 January 2021].

Wendland, A. R., 2020. *WebAssembly as a Multi- Language Platform,* s.l.: Harvard College.

Wiegers, K. E. & Beatty, J., 2013. *Software Requirements.* s.l.:Microsoft Press.

Wikipedia, 2020. *crt0.* [Online]
Available at: https://en.wikipedia.org/wiki/Crt0
[Accessed 27 March 2021].

Wikipedia, 2020. *Design science (methodology).* [Online]
Available at: https://en.wikipedia.org/wiki/Design_science_(methodology)
[Accessed 4 February 2021].

Wilson, N., 2018. *when should we need a glue code?.* [Online]
Available at: https://github.com/WebAssembly/design/issues/1201#issuecomment-380442028
[Accessed 24 January 2021].

Winata, P., 2020. *What is gRPC? Protocol Buffers, Streaming, and Architecture Explained.* [Online]
Available at: https://www.freecodecamp.org/news/what-is-grpc-protocol-buffers-stream-architecture/
[Accessed 7 February 2021].

Worldometer, 2020. *World Population by Year.* [Online]
Available at: https://www.worldometers.info/world-population/world-population-by-year/
[Accessed 23 January 2021].