# Abstract

This initial report sets out my plan and research for producing a web-based teaching simulation for a Baseband Modulation visualisation. The visualisation will simulate generating a signal, coding the signal using the line coding schemes defined in section 4.2.2, filtering out the high-frequency content from this signal, and finally displaying this signal’s power spectrum prior to transmission. If time permits, I will expand upon these goals as per section 5.2.

Students will be able to interact with the simulation by altering parameters. These parameters will include specifying the entropy of the generated signal, selecting which coding scheme to use, and altering the cut-off frequency of the low-pass filter.

I will introduce the project and give the reasons why it is needed in section 2. Discuss the specification of the objectives required to meet these aims in section 3. Give an overview of the background reading which led me to decide upon the tools and technologies I chose for this project in section 4. Give the must-have and nice-to-have requirements in section 5. Expand upon my approach and give a timeline for my planned progression in section 6. I will state the risks and mitigation strategies I plan on using in section 7. I will state any ethical considerations for the project in section 8. Finally, I will conclude the report in section 9.

# Introduction

Teaching visualisations are a key tool for helping students to understand complex systems. They provide a way for students to not just to see how concepts they have been taught work but to interact with these concepts and in doing so develop a greater understanding. By altering the parameters of the simulation, they can learn how these parameters affect the system and understand why these decisions may have been made when implementing real-world systems.

Dave Pearce has published over thirty Silverlight based visualisations helping students understand concepts including Basic Electronics, Communication Physical Layers and Communication Protocols and in doing so was awarded the [Insert full award name here]. This demonstrates the importance of these visualisations to students learning. However, as of 2019 the technology used to run these demos is no longer supported by any modern web browser [1].

However, as of 2019 Microsoft Silverlight, the application framework used to write and run these web applications has been discontinued. The software was only previous supported on Internet Explorer and as of June 2022 Internet Explorer has reached its end of life meaning these visualisations are no longer able to run on modern hardware making it difficult for students to be able to access these useful learning resources.

Because of this it is important that these visualisations are updated to be able to be ran on modern hardware that today’s students are able to access not just on university resources but also on their own devices whether that be a laptop or mobile phone.

The aim of the project is to develop a Baseband Communication Visualisation, based on Dave’s original demo built with modern web technologies. It will be able to run a Discrete Fourier Transform in both JavaScript and WebAssembly; allowing the user to choose which technology to use and present the user with the time taken to run the algorithm allowing the user to evaluate the benefits of WebAssembly for different numbers of sample points.

# Specification

[Objectives to achieve your aims, including, where appropriate, hardware and software]

# Overview of background reading

When researching for this project I divided by research into two main sections. The first of these is the technology and software engineering skills that would be required to best realise the project. With the goal of creating a deliverable that could be accessed by as many students as possible, for a long a time period as possible. The second research area focussed more on the theory behind the key stages of implementing a Baseband Communication protocol, such as the Discrete Fourier Transform and the Line Coding techniques required for creating a Baseband Communication Visualisation.

## Technology

When designing a teaching tool, it is important that it is accessible to all students regardless of what computing devices they may have available. It is important that the visualisation can be ran by students regardless of if they are on a Windows or Mac computer, or indeed if they are using a mobile phone as their primary computing device. According to a survey conducted in December 2022 by StatCounter [1] Android has a 44.6% Market Share. This beats out Window’s market share of 28.41%, which, according to Stat Counter has been decreasing year on year since at least 2009, where it had a Market Share of 95.42%. Looking at this data it is important that when I am developing my teaching visualisation that I should ensure that it is able to run on whatever operating system the student may be using.

Figure 1 - Operating System Market Share Worldwide December 2022 (Data taken from: https://gs.statcounter.com/os-market-share)

### Accessibility

The Covid-19 pandemic has re-confirmed the importance of developing tools and learning resources that are not just accessible to those with access to University resources. The tools we develop should be accessible to all students regardless of their location or the technical capability of their computing hardware. Due to this, the most logical platform to develop these tools for are web-based environments. All of the top 7 operating systems by market share are able to access web-based resources, additionally, developing these tools to be accessed from a webpage allows these tools to be developed once and run by students who are using a range of operating systems.

The University of York publishes a ‘Minimum PC specification for taught students’ web-page [2]. It is important that the visualisation is able to run fluidly on a laptop with those specifications, in order to ensure that all students following that guidance are able to engage fully with the content.

### Languages

The World Wide Web Consortium define the standards and best practices used in web development [3]. They define four languages for running code in the browser, these four languages are HTML, CSS, JavaScript and WebAssembly [4]. These four languages are able to run in all modern browsers [5]. However, only JavaScript and WebAssembly can be used to implement the interactivity required for this project.

Web Development often uses a JavaScript framework [6]; these are collections of code libraries and components which can be used to help provide a foundation for Web Developers to build their web-sites from. There are various frameworks of JavaScript used for web development, such as React, which is maintained by Meta [6], Angular, which was developed by Google [7], and Vue, which is an independent community-driven project [8]. However, each of these frameworks add complexity and overhead to developing web-apps. Additionally, if future developers wish to maintain/ update the code-base for future cohorts of students or modify the visualisation to introduce new concepts then they would need to be well versed in these frameworks as well as JavaScript.

According to the 2022 State of JavaScript Survey, which was created to was created to identify upcoming trends in the web development ecosystem in order to help developers make technological choices [9] there is clear segmentation between these front-end frameworks. Although React is used by 81.8% of respondents [10], when we consider the interest of JavaScript developers React drops to 47.2%, which may mean that many developers would be less interested in maintaining and updating the visualisation if I were to use this Framework.

Because of this I chose to develop the visualisation with vanilla JavaScript, to ensure that the code can be read and maintained by as many future developers as possible.

TypeScript is a strongly typed programming language that builds on JavaScript [11], it is popular with developers as it allows for type syntax to be added to variables, and structures. However, it is translated back into JavaScript before run time. I could develop the project with TypeScript, however, like with the above frameworks it may discourage developers who are unfamiliar with TypeScript’s syntax from maintaining and updating the visualisation.

### Development Environment

When deciding which development environment to use for the project I had to ensure that it would be suitable for web development. This meant having native syntax highlighting for JavaScript, CSS and HTML. Additionally, I knew I would be writing in an additional language for WebAssembly so the development environment would have to support syntax highlighting for languages such as C, Rust, or Java.

There are many development environments which fit the first requirement, those being WebStorm [12], which is made by JetBrains and is an Integrated Development Environment build for JavaScript. Brackets [13] w

Which is a Web Development Environment build by Adobe. However, due to these environments being build from the ground up for web development they do not support writing in languages usually not usually used for web development, such as C or Rust, which I may use for writing the WebAssembly Fourier Transform.

This leaves the final option being Visual Studio Code which supports syntax highlighting for nearly all languages thanks to a large extension library. Additionally, Visual Studio Code supports live server extensions which automatically refresh the page whenever changes are made to the code. This is great for extremely fast prototyping. Due to this I decided to develop the project with Visual Studio Code.

### WebAssembly

WebAssembly (wasm) is a recent technology adopted in 2019 by the World Wide Web Consortium. It is notable as the second standard for executing code in all modern browsers [14].

WebAssembly allows for code to be written in many languages and then compiled into a low-level binary format. This would theoretically allow for massive computational time savings when delivering computationally intensive programs to the user. This is because you can write the code in an extremely high-performance language, like C, or Rust, and convert it to a binary format for the user to run on their machine far faster than JavaScript could run.

As JavaScript was not designed to be a high-performance language and is compiled by a Just in Time compiler, ahead of execution, it would not be the language of choice for something computationally intensive, such as a Fast Fourier Transform. WebAssembly on the other hand would be complied to binary code far ahead of time and would be downloaded to the user’s machine when they load the web page and be immediately ready for fast execution.

I am planning to use the Fast Fourier Transform in my visualisation to convert a function between the time and frequency domains. This is a very computationally intensive operation, given the large number of sample points needed to capture the higher frequencies of the data generated. These operations could utilise the performance advantages of WebAssembly to allow the function to be programmed in a more appropriate language, and then compiled to byte code before being run by the user's browser, theoretically showing the user the output much quicker than with JavaScript.

A popular design tool called Figma [15] is currently doing this [16] to deliver an extremely computationally intensive design tool to users via a web browser. Previously something this powerful would only be able to run as a native, compiled desktop application, such as Photoshop.

When deciding upon which language to write my code in for WebAssembly I had to consider three main factors. As my goal was comparing the performance of WebAssembly to JavaScript I had to choose a programming language that has good performance. This means that it should be directly translated to machine code with little alteration of the code by a compiler or virtual machine. The second factor was memory management, I wanted to have full control of the memory being allocated so that I could ensure a fair trial with every trial having the same number of bytes of memory being allocated. Additionally, I did not want to have a garbage collector adding an additional variable to the trial that I could not control. The garbage collector would also increase the download size of the wasm code being downloaded to the user’s machine, this would dramatically increase the time taken to load the page.

Due to these factors this narrowed down by my choice of language to either C, C++, or Rust. The final factor I considered was support from the language community for WebAssembly. If I wanted to ensure that the project could be maintained by other developers in the future, the language needed to have strong tools for interacting with the webpage’s Document Object Model as well as having tools for translating rust code into WebAssembly code. Rust is the clear choice in this regard as despite being the most desired and frequently used language for WebAssembly, according to the 2022 State of WebAssembly Survey [17]. It has a large number of Open-Source tools for building WebAssembly Code [18], as well as tools for Interacting with JavaScript and the DOM [19]. Due to this I decided to use Rust as the language when developing the WebAssembly part of this project.

## Baseband Communication

### Discrete Fourier Transform

### Line Coding Techniques

When deciding which line coding techniques to include I wanted to ensure I was selecting those most useful to the students who were going to use the visualiser. This meant they had to be distinct, introducing new concepts such as return to zero. They should be used in the real world whilst being easy to understand for students who had previously never been introduced to the concept of line coding. Due to this I decided upon initially adding the following, five, line coding schemes:

* Non-return-to-zero level
* Non-return-to-zero mark. This introduces the concept that data may not just represented by a single voltage level but may be represented with a bit transition.
* Return to zero. This introduces the concept of return to zero coding to the students, showing that the data does not need to remain at a single level for the entire time period.
* Biphase-L. Commonly referred to as Manchester Coding, I intend on implementing the line coding technique defined by IEEE 802.3[20], which is implemented in their wired Ethernet standards. This technique is commonly used as regardless of which symbol is generated there is always a bit transition, this means the signal is self-clocking. This introduces a useful coding scheme in the real world and can also teach the importance of a signal being self-clocking.
* Bipolar, Duobinary signal [21]. This concept can introduce the advantages of a line coding signal having little or no DC-component to the students.

# Description of requirements

In order to achieve the aim of the project I have some must have requirements, these are essential to have in order to be able to thoroughly test my hypothesis. Additionally, I have listed some nice to have requirements that I would like to add to the project if I have time, these would extend the functionality of the visualisation as well as introducing additional concepts in order to expand the teaching goals of the project.

## Must have requirements

These requirements must be met in order to adequately create a teaching visualisation tool that is able to determine whether WebAssembly is an efficient tool for processing large data when developing web-based teaching visualisations. I will list the requirements in the order that they would be applied in when implementing a Baseband Communication protocol.

### Binary signal generator

The first of which is a binary signal generator able to generate a large, random, sample of data to be coded and transformed. This data must be able to clearly displayed to the user so they are able to follow the process of the data as it passes through the baseband visualiser.

### Line coding

The second requirement is a line coder which must be able to encode the data generated by the binary signal generator using at least five distinct line coding techniques, it is important for the student to be able to select the line coding scheme they wish to use so they are able to see how the encoding scheme changes the output frequencies.

The must have line coding techniques are:

### Discrete Fourier Transformation

The third requirement is to be able to pass this encoded signal through a Discrete Fourier Transform (DFT) algorithm. This is in order to represent the signal the signal as a function of its frequency content in order for the higher frequencies to be filtered out.

### Low-pass filter

A low-pass filter must be applied to the Frequency domain signal returned from the DFT algorithm. This is essential in order to cut off the extremely high frequencies that would be generated when trying to load a cable with a signal that changes instantaneously from zero volts to a higher voltage. These extremely high frequencies could cause undesirable coupling and crosstalk with other nearby cables so it is important that these frequencies are filtered out before transmission.

### Inverse Discrete Fourier transform

The final must have requirement is to use the Inverse Discrete Fourier Transform algorithm to transform the filtered DFT signal back into the time domain, this is important as it allows the student to see the impact that the low-pass filter has had on the original signal and allows them to understand how the signal will be loaded onto the cable for transmission.

## Nice to have requirements

Additionally, I will follow up these must have requirements with objectives that would be beneficial to include if time permits, in order to extend the scope and learning objectives for the project.

### Eye Diagram

Firstly, if time permits, I would like to present the user with an eye diagram overlaying the filtered signal for all possible sequences of binary ones and zeros. This would allow the student to be able to visually compare different line coding schemes by visually understanding where the receiver would need to sample the signal to receive the correct interpretation of the signals value. For example, in the diagram below, the receiver could read the signal at half of the time period and correctly interpret the desired signal value. However, if the receiver were to read the signal at one quarter of the time period then the receiver would not always interpret the correct desired signal value.

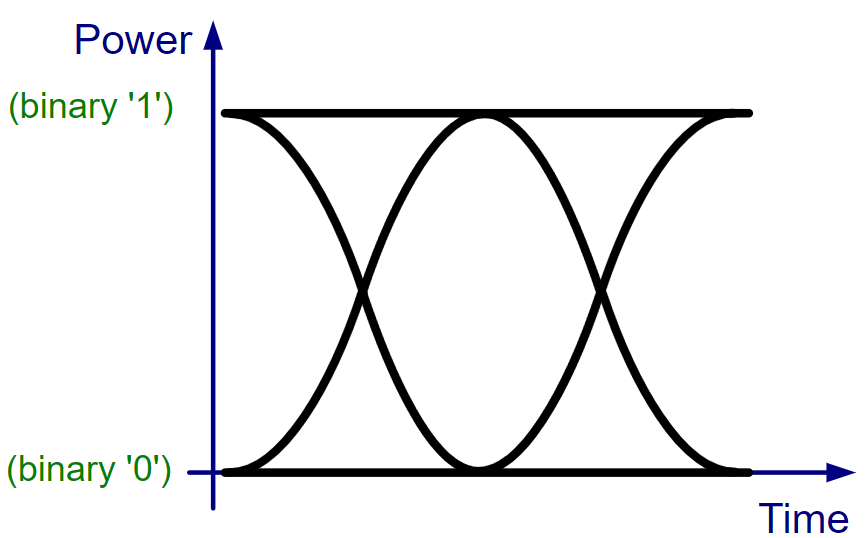


Figure 2 - Example of an eye diagram   
(Taken from: https://en.wikipedia.org/wiki/File:On-off\_keying\_eye\_diagram.svg )

### Noise

I would like to be able to allow the user to add variable amounts of gaussian noise to the original signal. This would allow the student to better be able to visualise how the signal would look in a real-world situation. Additionally, it would help the student understand how different line coding techniques can be used to mitigate the effects of noise. Finally, it would help the student visually understand how the eye diagram is an important tool when designing a communication protocol; as if there is no clear time period where the receiver can correctly sample the signal and interpret the correct signal value then there will be errors across the communication channel.

### Entropy

I would like to add the ability for the student to select the entropy of the original signal. They would do this by altering the probability that a binary one is generated. This would allow them to more easily understand how different line coding schemes affect the output signal. Additionally, by allowing the user to select the probability with a slider, the visualisation can provide feedback as to how their input affects the entropy of the signal by displaying an automatically updating entropy equation. Such as the one shown below.

Where is the probability of the symbol being generated. And, is the number of symbols used in the alphabet for the communication protocol; for the case of a binary signal this would be two.

### Additional Line Coding Techniques

If time permits I would wish to include additional line coding techniques for the student to be able to select from. In particular, I would wish to include techniques such as 4B5B line coding. This techniques maps for input bits onto 5 output bits for transmission [22]. This technique ensures that there will always be enough bit transitions to produce a self-clocking system regardless of the input bits. Due to this 4B5B encoding is used in the USB Power Delivery specification [23]

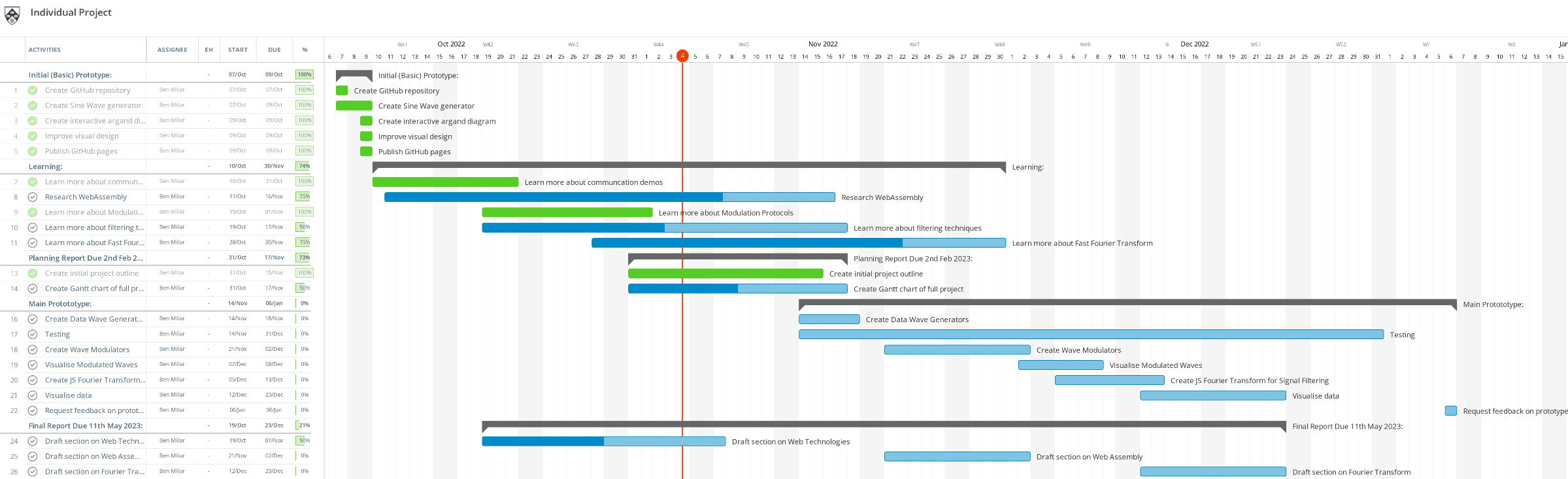
### Inverting Bits

The user should be able to click on the signal in order to invert an individual bit. Doing this will allow the user to understand how each bit affects the final signal and how one-bit alteration could have large effects on the final output signal.

### Cable simulator

[Show the output of the signal along a cable of X length with Y capacitance per unit length and Z inductance per unit length]. Show the Eb/No or C/No along this cable with X dB/m loss.

# Timetable

Not up to date, just for formatting 

# Risks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk | Severity | Probability | Seriousness | Mitigation |
| Losing project code due to laptop crashing. | 10 | 3 | 30 | Back-up code on GitHub, keep an offline backup. |
|  |  |  |  |  |
|  |  |  |  |  |

# Ethics statement

After consideration of the University’s code of practice and principles for good ethical governance no ethical issues were identified in this project.

# Conclusion

[1] Microsoft *Silverlight End of Support* support.microsoft.com [Online]. Available: <https://support.microsoft.com/en-us/windows/silverlight-end-of-support-0a3be3c7-bead-e203-2dfd-74f0a64f1788> [Accessed 25 January 2023]

[1] StatCounter *Operating System Market Share Worldwide* sstatcounter.com [Online]. Available: <https://gs.statcounter.com/os-market-share> [Accessed: 22 January 2023]

[2] University of York *“Minimum PC specification for taught students” York.ac.uk* [Online]. Available: <https://www.york.ac.uk/it-services/purchasing/private/minimum-spec-students/> [Accessed: 20 January 2023]

[3] W3C *“About W3C” W3.org* [Online]. Available: <https://www.w3.org/Consortium/> [Accessed: 19: January 2023]

[4] W3C *“World Wide Web Consortium (W3C) brings a new language to the Web as WebAssembly becomes a W3C Recommendation” W3.org* [Online]. Available: <https://www.w3.org/2019/12/pressrelease-wasm-rec.html.en> [Accessed: 19 January 2023]

[5] Mozilla *“WebAssembly”* developer.mozilla.org [Online]. Available: <https://developer.mozilla.org/en-US/docs/WebAssembly> [Accessed: 18 January 2023]

[6] Trio *“JavaScript Frameworks: What Are They and How Do They Work?”* trio.dev [Online]. Available: <https://www.trio.dev/blog/javascript-framework> [Accessed 25 January 2023]

[6] React *“React” reactjs.org* [Online]. Available: <https://reactjs.org/> [Accessed: 19 January 2023]

[7] Angular *“Angular”* angular.io [Online]. Available: <https://angular.io/> [Accessed: 19 January 2023]

[8] Vue *“Vue”* *vuejs.org* [Online]. Available: <https://vuejs.org/about/faq.html> [Accessed: 19 January 2023]

[9] State of JS *“State of JS About” 2022.stateofjs.com* [Online]. Available: <https://2022.stateofjs.com/en-US/about/> [Accessed: 20 January 2023]

[10] State of JS *“Ratios over time” 2022.stateofjs.com* [Online]. Available: <https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/> [Accessed: 20 January 2023]

[11] TypeScript *“TypeScript” typescriptlang.org* [Online]. Available: <https://www.typescriptlang.org/> [Accessed: 20 January 2023]

[12] JetBrains *“WebStorm” jetbrains.com* [Online]. Available: <https://www.jetbrains.com/webstorm/> [Accessed: 20 January 2023]

[13] Brackets *“Brackets” Brackets.io* [Online]. Available: <https://brackets.io/> [Accessed: 20 January 2023]

[14] Mozilla *WebAssembly developer.mozilla.org* [Online]. Available: <https://developer.mozilla.org/en-US/docs/WebAssembly> [Accessed: 24 November 2022].

[15] Figma *Figma* figma.com [Online] Available: <https://www.figma.com/> [Accessed : 20 January 2023]

[16] Evan Wallace CTO Figma *WebAssembly cut Figma’s load time by 3x* [Online]. Available: <https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x/> [Accessed: 24 November 2022].

[17] Scott Logic *The State of WebAssemly 2022* scottlogic.com [Online]. Available: <https://blog.scottlogic.com/2022/06/20/state-of-wasm-2022.html> [Accessed: 22 January 2023]

[18] The Rust and WebAssembly Working Group *wasm-pack* github.com [Online]. Available: <https://github.com/rustwasm/wasm-pack> [Accessed: 22 January 2023]

[19] *Rust and WebAssembly* rustwam.github.io [Online]. Available: <https://rustwasm.github.io/docs/book/reference/tools.html> [Accessed: 22 January 2023]

[20] *IEEE Ethernet,* IEEE Standard 802.3

[21] Wikipedia. *“Line Coding”, Wikipedia.org* [Online]. Available: <https://en.wikipedia.org/wiki/Line_code> [Accessed: 19 January 2023]

[22] Wikipedia. *“4B5B”, Wikipedia.org.* [Online]. Available: https://en.wikipedia.org/wiki/4B5B[Accessed: 19 January 2023]

[23] USB Implementers Forum *USB Power Delivery* usb.org [Online]. Available: <https://www.usb.org/document-library/usb-power-delivery> [Accessed: 20 January 2023]