Secure Communication Platform Research Manual

13th November 2020

Bachelor Of Science (Honours)
Software Development

Liliana O'Sullivan
C00227188

Paul Barry
Project Supervisor

Institiúid Teicneolaíochta Cheatharlach
INSTITUTE of TECHNOLOGY CARLOW
At the Heart of South Leinster

Table of Contents

List of Figures

List of Tables

1.Introduction

On the 6th of June, 2013, an article was published by 'The Guardian' (Glenn Greenwald, 2013) that, unbeknownst to the public, was the beginning of a series of disclosures that changed how the world views surveillance. Soon came another two articles about the PRISM program (Ewen MacAskill and Glenn Greenwald, 2013), which forced American telecom providers to disclose data on their customers. The world was now aware, and there are no possibilities of covering our eyes anymore.

Was the government the villain in this? Was Edward Snowden irresponsible? The answer is an individual one, though the thirst and desire for security echoed throughout the public. As seen in Figure 1[1], some users internet online behaviour changes as a result. A year after these disclosures, Apple announced its newest flagship mobile operating system would natively encrypt the data stored within its user's mobile devices (David E.Sanger and Brian X.Chen, 2014).

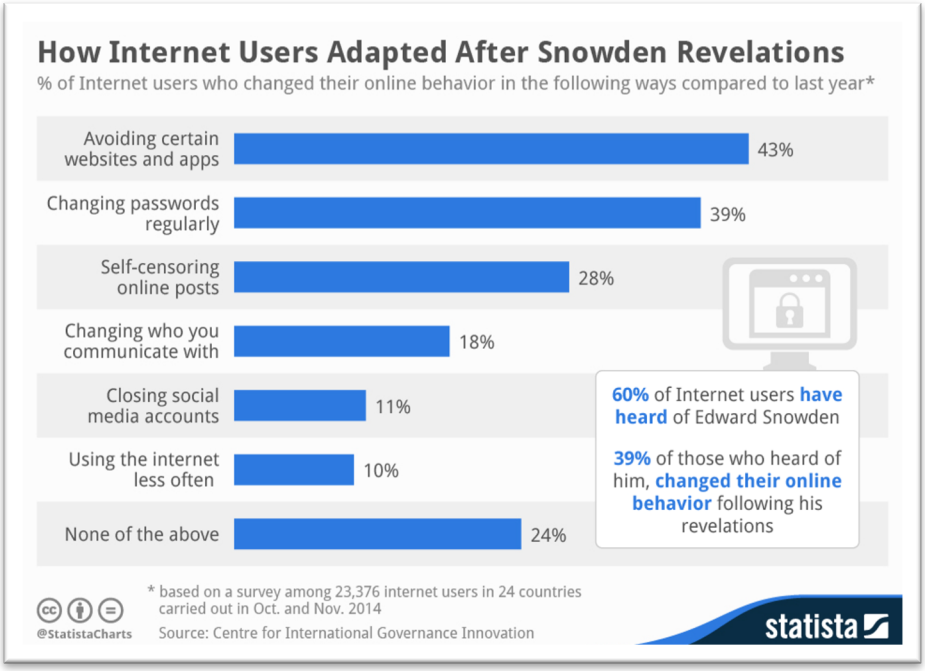

Figure 1 Adaption of Internet users after Snowden revelations

The need for private communication was clear. Many communications platform have stepped up, such as Signal or Jami. Private chat is not currently standard on all platforms, many of which enable a party outside the conversation of the two individuals to access chat logs.

[1] (Felix Richter, 2014)

Regarded by many as the notable feature of the year 2020, COVID-19 caused many disruptions, from economical to the simplistic parts of daily life, such as visiting the relative. A breakdown of its impact can be seen in Figure 2[2].

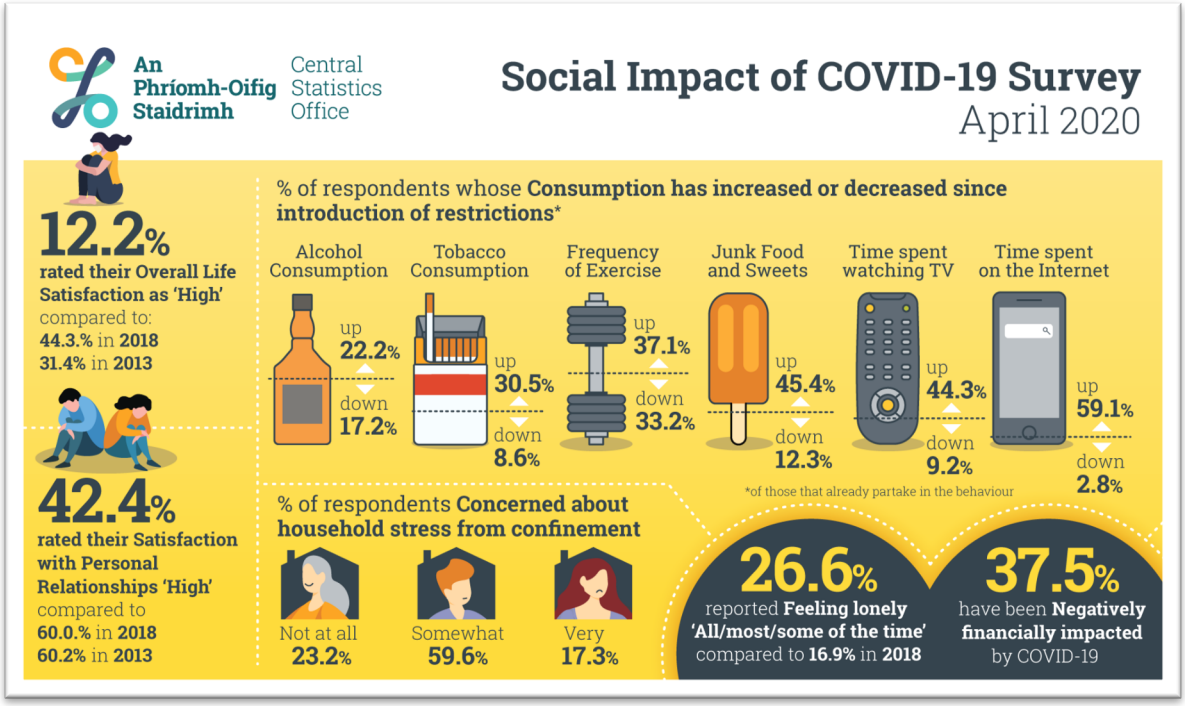

Figure 2 Social Impact of COVID-19 Survey CSO

Due to COVID-19, events of all kind got cancelled entirely or transitioned to an online model. Two notable events were two cybersecurity conferences, Blackhat and Def Con. Def Con announced its social aspects would occur within a Discord Server (Def Con Official, n.d.). For Blackhat 2020, the admins and presenters used Discord too for communication (Bazzell, 2020). The use of Discord sparked much disappointment from presenters and attendants alike (Alfred Ng, 2020). The motivation for this project was raised from this. A platform that multiple people can communicate without degrading user privacy or security.

[2] (Publication, 2020)

2. Privacy and Security

Privacy and security often are not considered very engaging unless depicted in a Hollywood production where systems are hacked into in a moment of brilliance and sophistication.

While the realities of Privacy and Security are much less glamourous, it is hard to deny the need for it in the modern age of technology. According to the United Kingdom's Office for National Statistics, the number of offences for 'computer viruses/malware' received an increase by 61% from 2019-2020, 'Hacking- Social media and email' had an increase of 55% and 'Action Fraud' saw an increase of 23% (Nick Stripe, 2020). The need for privacy and security speaks for itself.

Privacy and security are two interrelated terms, frequently used in conjunction and often interchangeably. Privacy is about the handling of personal information in its processing, storage or usage. It is about the misuse of an individual's information. Common ways to ensure user privacy includes applying government regulation (Eg: General Data Protection Regulation) and the application/management of the privacy policy within an organisation (DataPrivacyManager, 2020). Security is about protecting data from unauthorised third parties access. Security involves ensuring the integrity of data, such as accuracy, reliability and controlling availability to authorised parties. Methods used to increase security include using secure encryption, applying network security, providing multi-factor authentication options, breach response and access control. A visual representation can be seen in Figure 3[3].

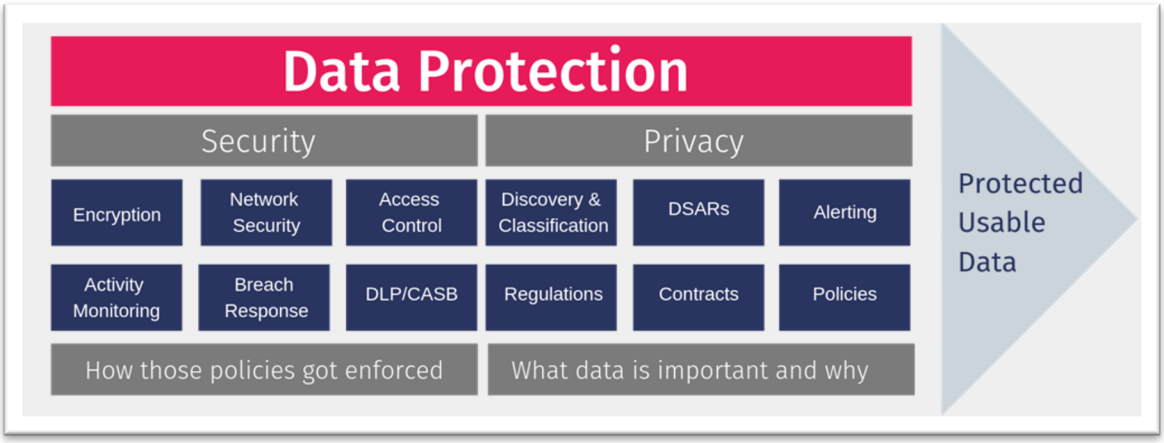

Figure 3 Data Protection

[3] (DataPrivacyManager, 2020)

3. Market Analysis

With an ever-increasing amount of people working remotely due to COVID-19, the requirements for security has increased globally. No more apparent than in the rise of Zoom (Eric S. Yuan, 2020). At the end of December 2019, Zoom had approximately 10 Million daily meeting participants. As of March 2020, this has increased to over 200 Million daily users. This rise in popularity was not without its complications. Multiple security vulnerabilities were discovered in Zoom, degrading user trust. Zoom responded with the acquisition of Keybase (Eric S. Yuan, 2020) and a ninety-day security plan, showcasing their plans to repair its vulnerabilities and improve security across the platform.

While Zoom might be considered an exception, if we look at the internet searches people are making worldwide for the past five years, we can see a common trend. In Figure 4[4] can be seen a worldwide search trend of 'VPN' for the past five years.

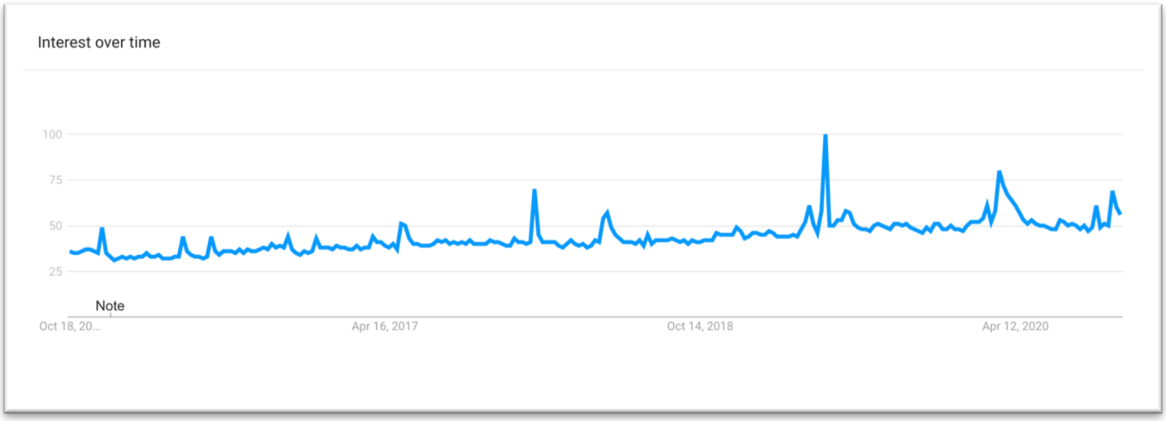

Figure 4 Google Trends VPN

There is a slow upward trend of individuals searching for VPN's. If we look at organisations that produce products with privacy and security at the forefront of their design, we can see an increasingly upward trend.

[4] (VPN, 2020)

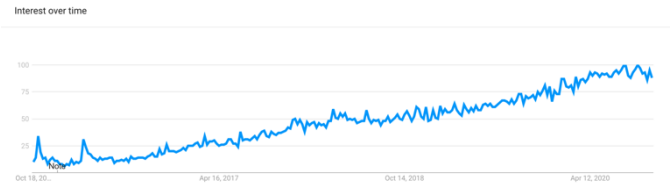

Figure 6 Google Trends ProtonMail

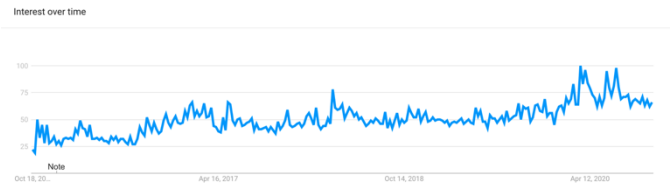

Figure 5 Google Trends, Signal

Figure 8 Google Trends DuckDuckGo

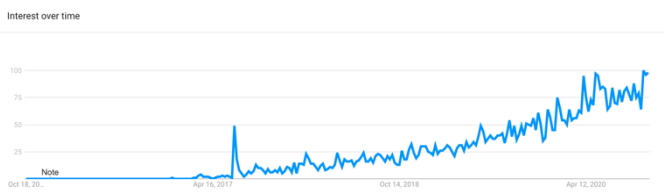

Figure 7 Google Trends ProtonVPN

In Figures 5[5],6[6],7[7],8[8] an uptrend in privacy and security-centric software can be seen, with a positive uptrend in individuals looking for such software, a market position where software can be designed to nurture their user's data, rather than the exploitation of it must exist.

[5] (ProtonMail, 2020)
[6] (Signal, 2020)
[7] (DuckDuckGo, 2020)
[8] (ProtonVPN, 2020)

4. Communication Platform Architectures

Platforms that enable conversations can be architecturally constructed differently. The most common are in one of three forms

- Centralised
- Federated
- Peer-To-Peer

Each platform has its advantages and disadvantages. Developers may choose one architecture over another, depending on their requirements. In general, most platforms in mainstream use are centralised.

4.1 Centralised

Centralised architecture is the most common, primarily due to the advantage of retaining control over many aspects of the system. New features can be implemented with relative ease, finding contacts is simplified, and it can provide lower latency with consistent and predictable performance.

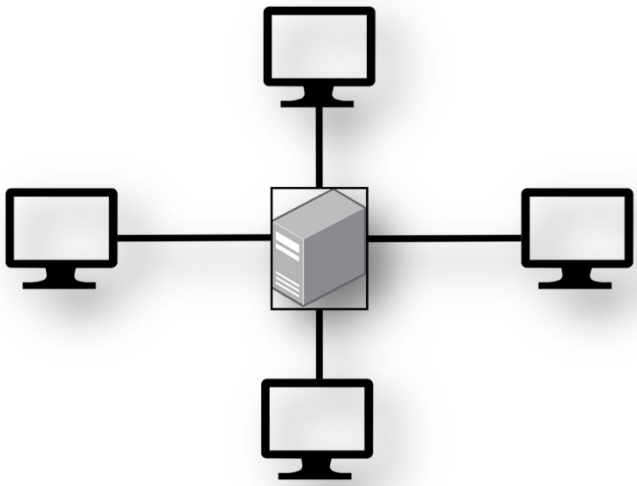

Figure 9 Centralised Architecture

This approach features one or more clients connecting to one or many central servers, as seen in Figure 9. In general, each client is limited in its logical capabilities and serves to send requests to the server. Servers wait for requests to be received from clients and replies with a response in return. These servers are generally deployed with large computing capacities using high-end enterprise hardware equipped with high-speed internet connectivity, often in data centres. Centralisation has its disadvantages; from a security point-of-view, the users place absolute trust in the server administrators. The platform users have no insights into the software running on the server or any malicious acts being committed. Internet Protocol (IP) addresses of the platform users are also exposed to the central server, though not often exposed to individuals the user chooses to engage with. In creating this architecture, setting up and maintaining these servers can be an expensive endeavour; not only in the hardware, there is also the need to maintain the environmental conditions the servers operate within, ensuring they receive adequate heat-management solutions.

4.2 Federated

In this approach, multiple independent servers exist, with a client that can communicate to each server using a shared protocol. Email is an example of a federated service; many emailing services exist, such as Gmail or Microsoft Office 365, that are all capable of talking to each other using protocols such as Simple Mail Transfer Protocol or Internet Message Access Protocol.
A client will connect to one or more independently organised hosted servers, as shown in Figure 10. These servers are considered a separate self-contained ecosystem. The client software is not much more sophisticated than a centralised client, with a similar approach of request from a client and respond from the central server being used.

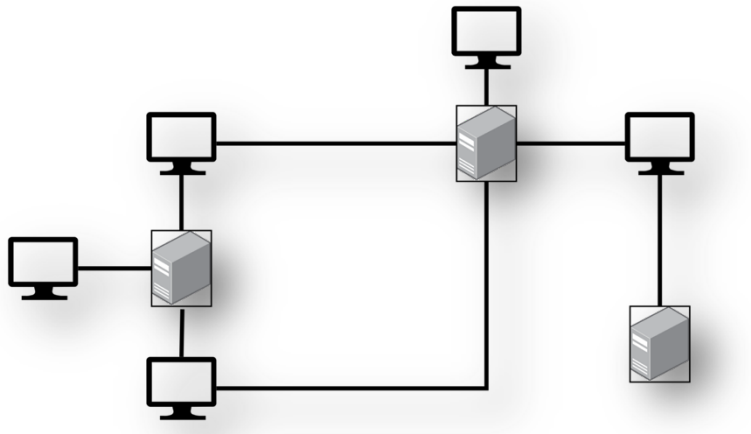

Figure 10 Federated Architecture

Often in this semi-distributed fashion, multiple third party clients can provide more native and customised experiences to their users. If we follow the federated email example, we will soon find that multiple email clients exist, such as Thunderbird, Microsoft Outlook, K-9 Mail, eM Client and many more. As the protocols used are often open-source to enable inter-polarity, the path to the development of clients is more accessible.

The prominent advantage of this approach enables the distribution of trust. This comes from hosting a personal server or through the usage of different public servers. This may also lead to complications as servers might be hosted by a hobbyist or an individual who is not informed of cybersecurity. The host of these servers has complete visibility of the IP addresses of its users, though they are not generally exposed to the other users of the platform.

Due to the distributed nature of this architecture, the ability to integrate new features becomes a near-impossible task, as the need to standardise and test it across the network becomes essential. Any new features generally are implemented on the client side.

4.3 Peer-To-Peer

Peer-To-Peer is designed with every client (known as a peer) connecting directly with one another without a third-party server, showcased in Figure 11. Peers find each other through the use of distributed networking technologies. Distributed Hash Table (DHT) is one example of this. DHT, for example, is an essential technology for the discovery of peers in Torrents.

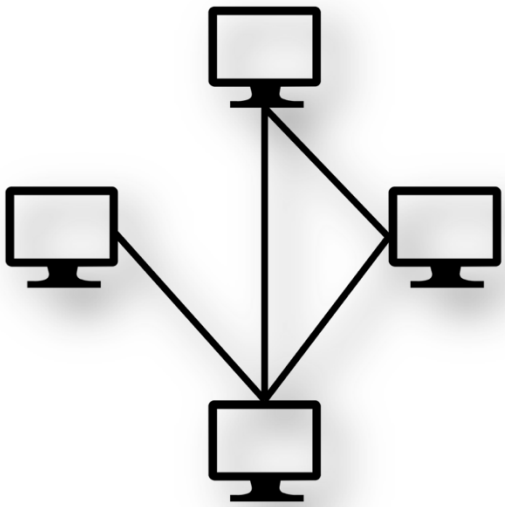

Figure 11 Peer-To-Peer Architecture

In the messaging space, peer-to-peer messaging can also be created using proximity-based networks such as Wi-Fi or Bluetooth. An implementation example of this is ‘Scuttlebutt’ or ‘Briar’. This approach of Peer-To-Peer messaging is generally low bandwidth, can only facilitate text-based messages. Proximity-based network messaging is generally used when internet connectivity is problematic such as a time of natural disaster.

The appeal of using peer-to-peer comes from its trustless design. In peer-to-peer connections, the user does not place trust in a third-party central server. The trade-offs come in the form of user experience generally. For example, messages are not sent unless both peers are online, or the stability of the connection is not on-par with a centralised high-spec server. The users IP address is also directly exposed to all other peers they connect to. As with all of the above architectures, IP address masking can be mitigated by using a Virtual Private Network or Proxy; however, these are not ideal solutions.

5. Case Studies

The following three case studies showcase the three different architectures discussed in the previous section as current production platforms.

5.1 Discord

Platform type - Centralised

Discord is a platform that combines the voice and video aspects from platforms like Skype with the text aspects of Internet Relay Chat (IRC) in a centralised design. Discord originally was developed for the online video-gamers target market; however, it has recently changed course towards a general-purpose group chatting platform (Librarian, 2020).

Discord came as inspiration when the founder noted how difficult it became to develop a strategy with their teammates in strategy games. They specifically found that services forced users to type in IP addresses just to chat or tended to be resource-heavy with security vulnerabilities built on outdated technologies. This led to developing a chat service with a simplified user experience and built on modern technologies (Tasos Lazarides, 2015).

All text-based messaging is enhanced with uploading files and sending GIFs all in-lined within the text window. The chat also has full support for the Unicode standard and supports the markdown standard enhanced for text formatting.

Users can send text-based messages to each other directly in private chat and create audio and video calls. Users are also able to share their computer's screen with audio when in a call.

For group chats, users can create 'servers' with multiple channels. These channels are similar to IRC text channels, with the ability to create audio call only channels. Within a server, roles can be created to aid in server administration. Roles can be used to enable sending text messages in restricted channels, aid in moderation and more. All roles are entirely customisable to the administrators of the Discord server.

Discord has developed a feature-rich Application Programming Interface (API) that enables developers to plug into the Discord ecosystem. This can be through using Discord to log into services (Eg 'Signup with Discord'), creating bots to create interactivity in text-based chat.

As per the inspiration, Discord utilised multiple modern technologies. Its web application is created in ReactJS. By developing it in ReactJS, they were able to port ReactJS to React Native to gain operability for iOS. Android was created natively due to React Native's performance difficulties (Fanghao (Robin) Chen, 2016). Utilising Electron, they were able to develop native Windows, Linux and Mac applications.

When Discord was created in 2015, it initially used MongoDB to store its user's messages. In under a year, once the 100 million stored messages were reached, performance became unpredictable. This is due to MongoDB storing all data and its indexes in memory. Once the data size outgrows the memory available, data begins to be moved in and out of memory in a Least Recently Used (LRU) ruleset. This unpredictability in performance, among other issues such as scalability, became the lead to the search for a possible replacement (Stanislav Vishnevskiy, 2017). The engineers at Discord choose to use Apache Cassandra for such a task due to its fault-tolerant design, swift writes, and its overwhelming has aided in maturing it into a stable platform.

At the heart of Discord servers is Elixir. The choice for Discord was clear; it desired to be a highly concurrent real-time system, this paved a clear path to the Erlang VM. Erlang is designed for highly concurrent distributed environments primarily aided by its fundamentally functional design aided with immutable data enforcement. To gain more performance from Elixir, the engineers at Discord created their own data structure developed in Rust and bridged it to the Erlang VM using 'Rustler' to develop further a more performant system (Matt Nowack, 2019).

To develop calling and text chat, Discord utilises WebRTC and WebSockets (Jozsef Vass, 2018). While on a browser, it is reliant on the implementation offered by the browser. In the Desktop (Electron), iOS and Android applications, a custom C++ application was built on top of the existing native WebRTC tailed to Discords needs. Having this customised version of WebRTC has enabled multiple features that would pose a significant challenge to implement otherwise. An example of this is avoiding sending audio data during periods of silence, which results in reduced bandwidth and Central Processing Unit (CPU) usage.

In Discords design, users do not directly connect in the standard peer-to-peer design of WebRTC. Instead, users connect to centralised servers operated by Discord. This provides more reliable connections and keeps the IP addresses hidden from the other parties involved in a call.

5.2 Element - Matrix

Platform Type - Federated

Element is a client that implements the Matrix protocol. Matrix is an open-source protocol for secure decentralised communication. It left beta on the 11th of June 2019. On the 19th of December 2019, Mozilla announced Matrix with Element would become its successor to IRC due to its "…excellent team collaboration and communication tools,.." (mhoye, 2019). The French government also recently completed its transition to using the Matrix and Element due to fears of foreign surveillance and released the app as an open-source application to the iOS and the Google Play Store (Matthew Hodgson, 2018). Matrix is an application-layer protocol for federated real-time communication, much like the current email protocols. It exposes Hypertext Transfer Protocol (HTTP) APIs and uses the JavaScript Object Notation (JSON) for data exchange. It uses SQL for local storage of information.

Element is a separate entity from the developers of the Matrix protocol. It is a for-profit company that design their services around hosting servers using the Matrix protocol. Element has a web client, mobile applications on iOS, Google Play Store and F-Droid. It provides desktop clients for macOS, Windows and Linux (element.io, 2020). The desktop application can be seen in Figure 12[9].
Element's Android application was created in Kotlin. Kotlin is a general-purpose

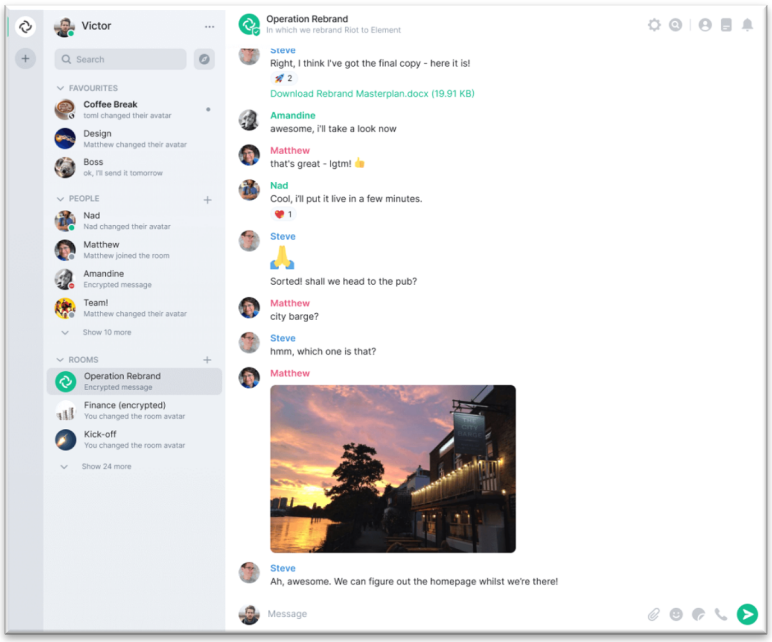

Figure 12 Element Client

programming language

[9] (element.io, 2020).

designed to be interoperated with Java by compiling Kotlin to java bytecode. JetBrains developed it, known for creating integrated development environments such as IntelliJ and PyCharm in 2011. On the 7th of May 2019, Google announced that Kotlin would become the preferred language for the development of Android apps (Frederic Lardinois, 2019).

The macOS, Windows and Linux desktop applications are achieved through the use of the Electron wrapper. Electron is maintained by GitHub and enables developers to build cross-platform applications with the combination of the Chromium rendering engine with a runtime of Node.js. Some well-known applications created using Electron include Visual Studio Code, Atom, Slack, Facebook Messenger and more (Electronjs, 2020). Elements iOS application is created in native Objective-C.

5.3 Briar

Platform Type – Peer-To-Peer

Briar is an open-source instant messaging platform implementing a Peer-To-Peer architecture. It is designed to be used by activists, journalists and anyone else who requires security in their messages. All message content is synced directly between devices. It does so by one of three means:

- Bluetooth
- Wi-Fi
- The Onion Router (TOR)

Its use of Bluetooth/Wi-Fi for connectivity removes reliance on existing centralised infrastructure, as displayed in Figure 13[10]. By directly connecting users, it aids in preventing surveillance and censorship by distributing data across multiple sources.

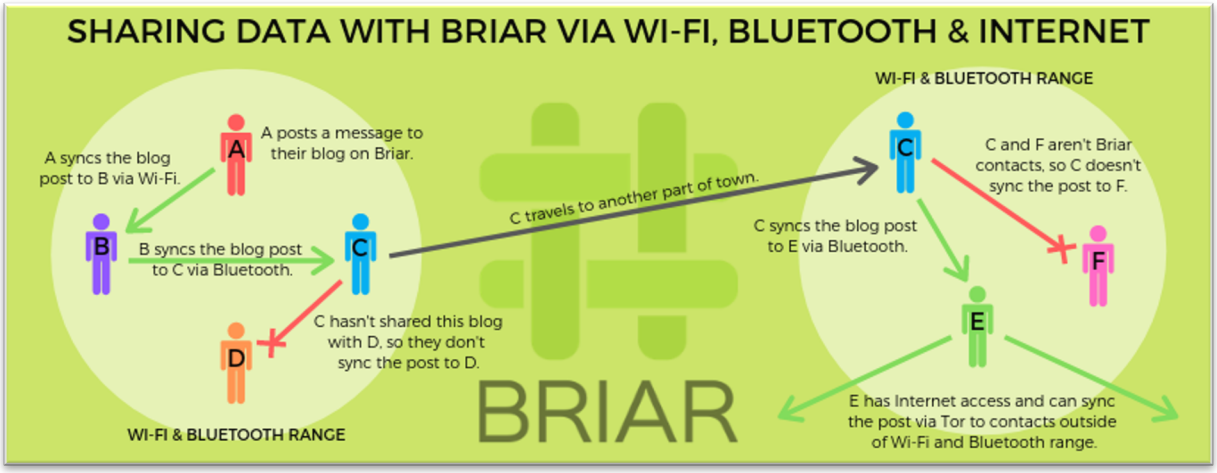

Figure 13 Sharing Data with Briar

If the user has access to an internet connection, they can connect to their peers with the security of the TOR network. TOR is an open-source volunteer-run network to bypass censorship, mass surveillance and ensure the anonymity of its users.

Briar is natively designed around the Android platform, leading it to be developed in Java. It uses an in-memory SQL database named “H2 Database Engine” (GitLab, n.d.). The use of a local database ensures that all data is stored on the user's device locally is essential to distributed philosophy.

[10] (Briar, n.d.)

6. Relevant Technologies

This section discusses technologies that are relevant to the development of this project.

6.1 Network Protocols

Network protocols are a set of rules governing how information is transmitted to ensure that multiple devices can communicate and have an ‘understanding’ of each other. Protocols define the syntax, semantics and synchronisation of the communication. While protocols used in networking can be broken into multiple layers, as shown by the Open Systems Interconnection (OSI) model, the protocols relevant to this project primarily reside in the upper host layers. The protocols discussed will mostly be in those layers.

6.1.1 HTTP(S)

Hypertext Transfer Protocol (HTTP) is one of the foundational protocols used on the internet to facilitate data exchange between devices. It is a set of standards maintained by the Internet Engineering Task Force (IETF), a non-profit which develops the internet protocols widely in use. When accessing a web page, the ‘http://’ commonly a prefix to a webpage specifies to the web browser to use the HTTP protocol for data exchange. Modern browsers, such as Google Chrome or Mozilla Firefox, generally do not require the ‘http://’ prefix as it is the standard method for communications across the internet.

Hypertext Transfer Protocol Secure (HTTPS) is an extension of HTTP, implementing security through encryption using Transport Layer Security (TLS). A user connects using HTTP to a web server by using the ‘https://’ prefix in the Unified Resource Location (URL). The concept of HTTPS has initially been in production by Netscape Communications, the developers of Netscape Navigator, in 1995 by implementing Secure Socket Layer (SSL). SSL is the predecessor to the modern TLS commonly used today. (Cloudflare, n.d.) As SSL evolved into TLS, it became standardised into HTTPS by the IETF in May 2020 (E. Rescolra, 2000).
The addition of TLS aims to provide privacy and data integrity. Once a HTTPS connection has been established, any parties attempting to eavesdrop on the traffic in transit generally will be unable to see the unencrypted data. In a practical perspective, if a user connects to Google.ie and places a search, no party outside of Google, and the user would be able to see the contents of the search that occurred. It is the standard protocol used to authenticate credentials for the processing of sensitive or private information. According to Firefox Telemetry, the

information statistics on Firefox users, at the time of writing, ~84% of sites loaded by Firefox defaulted to using HTTPS by default (letsencrypt.com, 2020).

6.1.2 WebRTC

Web Real-time Communication (WebRTC) is an open-source project that aims to provide web browsers with APIs to enable audio and video communication to occur within a webpage by connecting directly on a peer-to-peer basis without the additional installation of plugins or downloaded applications. WebRTC composes of multiple components that enable it to function (Mozilla, 2019).

ICE

Interactive Connectivity Establishment (ICE) is a method of connecting the two parties attempting to communicate together. It conducts connectivity checks to gain awareness of the environment present. Once it has completed collection, it begins initiating connections aiming to connect to the other party. It does so by sending information until a direct connection is established. This can take multiple seconds to establish a connection. As a result of this overhead, Trickle ICE was developed, which essentially conducts the connectivity checks in parallel to decrease the time taken of this entire process (WebRTCGlossary, 2017). It will first attempt to connect directly, such as in the case of being on a local network. In most scenarios, the parties are on a different network requiring the use of a STUN server or possibly a TURN server.

STUN

Session Traversal Utilities for NAT (STUN) is a tool used by ICE. It is a lightweight protocol used to establish the current public IP address and port number. It is achieved through the use of an existing third-party server known as a STUN server. The client will send a request to the STUN server. Once the server receives the request, it responds with the IP address and port number as observed by it. STUN is useful when a client is behind a router or additional software that may restrict connectivity.

TURN

Traversal Using Relays around NAT (TURN) is used in a scenario where STUN is not possible. A TURN server relays information through it to the other party. A TURN server may not always be used. It is generally used as a failsafe mechanism by the ICE protocol. Operating a TURN server is considered expensive due to the bandwidth usage associated with relaying information.

6.1.3 Web-Sockets

This is a communication protocol that provides the ability to both send and receive data simultaneously; this ability is known as full-duplex. This protocol enables interaction between a web browser and a web server. It is considered lightweight compared to HTTP as once a web server desires to send information to the web browser, it does not require a request first to be created by the web browser to facilitate the transfer. This can be especially useful for latency-sensitive applications such as an online game with real-time components. This can be additionally useful in a situation when the server must send updates in real-time, such as a text-based chat system (Kitamura, 2010). Following Samhita Alla's tutorial on Web-Sockets(Samhita Alla, 2018), a short experiment was created using Python, Flask and SocketIO. Once a browser is launched, it automatically attempts to connect to the local socket server, as shown directly below Figure 14, using the SocketIO JavaScript library.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/1.7.3/socket.iox.min.js"></script>

<script type="text/javascript">
  var socket = io.connect('http://' + document.domain + ':' + location.port);
```

Figure 14 Browser SocketIO Connect Code

Once a username and a message are entered in the HTML form Figure 15, this is forwarded to the server.

```
<form action="" method="POST">
  <input type="text" class="username" placeholder="Username" />
  <input type="text" class="message" placeholder="Message" />
  <input type="submit" />
</form>
```

Figure 17 HTML Form

```
var form = $('form').on('submit', function (e) {
        let username = $('input.username').val()
        let input = $('input.message').val()
        socket.emit('event', {
          username: username,
          message: input
        })
      })
```

Figure 15 Form Submit Code

The server replies with the data it received; this can be seen in Figure 17. In a non-experimental scenario, this logic would be more complex, such as sanitising the message before forwarding it to other peers. Once the server responds, and the event is handled in the browser, as showcased in Figure 18,

```
@socketio.on("event")
def eventHandler(json, methods=["GET", "POST"]):
    socketio.emit("response", json)
```

Figure 16 Server Logic

```
socket.on('response', function (msg) {
    if (typeof msg.username !== 'undefined') {
    $('div.message_holder').append('<div> <b style="color: #000">' + msg.username + '</b>: ' + msg.message + '</div>');
}
```

Figure 18 Response Handler

the browser appends the new messages to the previous messages and renders the results, as shown in Figure 19.

User1: To be, or not to be, that is the question
User1: Whether 'tis nobler in the mind to suffer
User1: The slings and arrows of outrageous fortune,
User1: Or to take arms against a sea of troubles
User1: And by opposing end them.

User1 | Message | Submit Query

Figure 19 Browser Display

6.2 Databases

Databases are an organised collection of data stored on a computer system (Oracle, n.d.). They are generally classified into SQL and NoSQL databases. While the functionality provided by a specific database can significantly vary depending on the system being used, they all generally provide the following

- Creating, retrieving, updating and deleting of entries
- Support for concurrency
- Support for recovering a database in the event of corruption
- Providing authorisation and controlling access to the data
- Remote access
- Enforcing user-created constraints to aid in data consistency
- Importing and exporting of data

Databases can be centralised or distributed. A centralised database is helpful for a small organisation with ‘lightweight’ requirements. The centralised database design is relatively simple to understand and has low maintenance. A single system residing within a single location contains the database. If this system fails, the entire data may be lost.

In a distributed system, multiple ‘nodes’ are spread across different locations, which are geographically separate from each other. These systems have increased difficulty in maintaining data consistency. These systems are very performant due to the possibilities of creating distributed workloads and has superior fault tolerance due to the distribution of data instead of it being held in a central location. These systems are also capable of achieving higher uptimes.

Databases often employ redundancy to increase data availability to increase performance and aid in fault tolerance through the use of Replication. This involves sharing updates to the database among multiple network-attached distributed databases. This is generally achieved in either a ‘Master-Slave’ or a ‘Multi-Master’ relationship.

There is a single ‘Master’ process in a' Master-Slave' relationship that maintains control over one or many slaves. Only the Master has permission to create updates to be applied. The Master process records updates to be applied by the slaves. These updates are propagated to the networked-attached slaves, who return a confirmation once they successfully applied an update.

Databases scalability can be classified into Vertical and Horizontal. This can be visualised in Figure 20[11]. Vertical scaling, also known as scaling up, involves adding additional resources to an existing machine. This is done by improving one or multiple components such as its Central Processing Unit (CPU), Random Access Memory (RAM), storage capacity, network bandwidth or any other component. Vertical scaling can quickly become limited by hardware limitations, for example, the amount of memory a motherboard can support. Once a limit has been reached, it requires purchasing entirely new hardware to migrate, which can cause downtime, with downtime already compounded with non-hardware limited upgrades such as a simple addition of memory to machine (Craig S. Mullins, 2018).

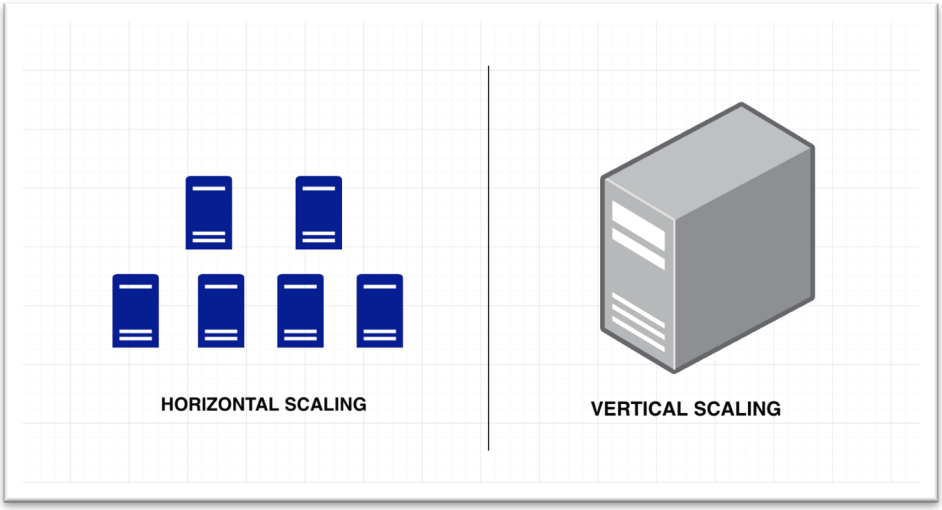

Figure 20 Horizontal vs Vertical Scaling

Horizontal scaling, also referred to as scaling out, is the process of adding additional hardware to a system, generally in the form of 'nodes' in a distributed database design. Removing 'nodes' from the network is known as scaling in. As hardware cost declines, it becomes cost-effective to adopt many 'commodity' hardware computers in an interconnected network and distribute the workload among them. This also comes with the simplicity to upgrade in the future by adding an additional 'node' to the network. This approach is limited by the software's ability to efficiently take advantage of distributed computing resources (Craig S. Mullins, 2018).
Horizontal and vertical scaling can be combined. The network-connected machines in a horizontally-scaled system can be upgraded. The ideal situation is a combination of both horizontal and vertical scaling by applying a balance.

[11] (Chandan Kumar Singha, 2019).

6.2.1 SQL

Structured Query Language (SQL) acquired its name from the language used in accessing the data stored. SQL can be considered a lightweight database equivalent of server-side scripting. SQL can be considered high level, aiding in its simplicity. When creating a query, an attempt is made to specify the desired query; instead of how to get it, SQL will devise how to obtain the data in the most efficient manner, this makes SQL nonprocedural. Another stand-out feature is its robust JOIN clause. This enables developers to join data spanning multiple tables through a predefined relationship with a relatively simple syntax. An example of a join clause can be seen in Figure 21.

```
SELECT
  *
FROM Invoice
INNER JOIN Customer
  ON Customer.Id = Invoice.CustomerId
```

Figure 21 SQL Example

SQL is known as a relational database. This is having data stored in tables containing rows and columns. Each entry is stored as a row in the table, and each column represents the specific data type and name associated. Tables have relationships to other tables through the use of primary and composite keys, often an Id. This relationship is known as a schema. The schema generally must be defined before any data can be added to the database.

Any data stored in a SQL database must be structured. Time must be spent focusing on the development of an effective schema to ensure data consistency. A poorly devised schema can present challenges in future development due to the rigidity of constraints applied in the schema.

Examples of popular SQL databases include MySQL, MariaDB, Microsoft SQL Server and more. As many of these database technologies have existed for multiple years, MySQL for example, was initially released in 1995 making it twenty-five years old at the time of writing. They can be considered mature technologies and have relative stability in most application environments.

6.2.2 NoSQL

NoSQL, coming from the phrase ‘Not Only SQL’, has gained recent popularity due to its advantages over a NoSQL database that has been heralded as the default de-facto standard for databases. If the requirements for data storage are not defined/are unclear/there is a large amount of unstructured data in use, developers may not have the luxury of creating a schema that can accurately model their needs, or it may not be possible. This is where NoSQL databases fit.

NoSQL is designed to be flexible in its schema. Many NoSQL databases achieve their flexibility in different approaches. They can be loosely classified into the following types

- Document
- Key-Value
- Graph
- Column Store

There rise for non-relational databases came with the evolution of the internet. Coined as the ‘Web 2.0’ or ‘The Participatory Web’ (Reisdorf, 2012), refers to the development and evolution of websites that focus on user-generated content with interoperability with other services. These services developed a need for Big Data. Big Data is processing “..*large, diverse sets of information that grow at ever-increasing rates*” (Troy Segal, 2019). NoSQL databases are capable of scaling out much easier than a traditional SQL database system, due to the trade-offs being made, such as having ACID (Atomicity, Consistency, Isolation, Durability) transactions (Adam Fowler and Eugene Ciurana, 2017). Instead, NoSQL databases focus on high availability and flexibility in the schema or lack of a schema (Terblanche, 2020).

Document

Document-based databases store data in documents. Variations exist between the database being used. Generally, the formats are JSON, Extensible Markup Language (XML), or YAML Ain't Markup Language (YAML), with JSON being the most popular. An example of what a document may look when stored in JSON can be seen in Figure 23. A document can be contained in a Collection, which contains multiple documents.

```
{
    "id": 1,
    "age": 25,
    "name": "John",
    "friends": [
      "Anthony",
      "Christa",
      "Robert",
      "Christopher"
    ]
}
```

Figure 23 Document Example

The structure of a document is not predefined and can vary from other documents. This can be advantageous, especially when defining a schema can become difficult. A document is similar to how developers create an instance of an object in the programming equivalence; a class is easier described in a JSON format than in a relation row and column table. Querying multiple documents often employs the use of an Id value created by the database in use. The most popular document-based database is MongoDB (DB-Engines, 2020).

MongoDB stores its documents in a Binary JSON (BSON) format, which has full interoperability with JSON. An example can be seen in Figure 22[12] of how JSON is converted to BSON in MongoDB. MongoDB inc develops MongoDB. It is developed primarily in C++ with an initial release in 2009, making it eleven years old (DB-Engines, 2020). Natively MongoDB uses a JavaScript shell to interface with an administrator. In the annual StackOverflow developer survey, MongoDB ranked as the most used NoSQL database by developers in their workplace (StackOverflow, 2020).

```
{"hello": "world"} →    \x16\x00\x00\x00          // total document size
                        \x02                     // 0x02 = type String
                        hello\x00                // field name
                        \x06\x00\x00\x00world\x00  // field value
                        \x00                     // 0x00 = type EOO ('end of object')


{"BSON": ["awesome", 5.05, 1986]} →     \x31\x00\x00\x00
                                        \x04BSON\x00
                                        \x26\x00\x00\x00
                                        \x02\x30\x00\x08\x00\x00\x00awesome\x00
                                        \x01\x31\x00\x33\x33\x33\x33\x33\x33\x14\x40
                                        \x10\x32\x00\xc2\x07\x00\x00
                                        \x00
                                        \x00
```

Figure 22 JSON to BSON

[12] (MongoDB, n.d.)

An experiment of MongoDB was created using Python, Flask and HTML. This is a showcase of a basic 'Login' system by querying MongoDB to check if a user exists with an entered email and password. In MongoDB, a database was created name 'login_db', containing a collection titled 'users'. Two entries were populated. Each entry contains a username and password field, with an Id generated by MongoDB by default, as shown in Figure 24.

Figure 24 login_db.users collection

A Python Flask server was created to serve a basic login page to a web browser, with server code in Figure 25. Some code in the home function is excluded for brevity. Login.html, the content being served to the web browser, can be seen in Figure 26. Once the form has been completed and submitted to the server, it queries the database for the hash of the entered password. If the entered credentials match an existing user, a success message is returned; otherwise, an invalid credentials message with a form to re-enter the credentials is returned, as shown in Figure 27.

```python
app = Flask(__name__)

@app.route("/", methods=["POST", "GET"])
def home():
        ...
        return render_template("login.html")


if __name__ == "__main__":
    app.run(debug=True)
```

Figure 26 Flask Server

```html
<form action="/" method="POST">
    <label for="username">Username</label>
    <input type="text" id="username" name="username">
    <br><br>
    <label for="password">Password</label>
    <input type="password" id="password" name="password">
    <br><br>
    <input type="submit" value="Submit">
</form>
```

Figure 25 HTML Login Form

In a production scenario, more processing would occur, such as the creation of user session or the sanitising of user input.

```python
if request.method == "POST":
    user = userDB.find_one(
        {
            "username": request.form["username"],
            "password": hashlib.sha1(
                request.form["password"].encode("utf-8")
            ).hexdigest(),
        }
    )
    if user is not None:
        return f"Successful login for {request.form['username']}"
    else:
        return (
            "<h3>Invalid Credentials</h3><p>Please try again</p><hr>"
            + render_template("login.html")
        )
```

Figure 27 Login Server-Login

Key-Value

Key-Value databases can be considered the simplest of all types. They can be likened to a map/dictionary or an associative array in different programming languages. They employ a data structure that consists of a unique key that is used to associate a value, this can be visualised in Figure 28. The key can consist of any type of data, and the value associated with the key also can be of any type. They are especially useful when storing large amounts of data that do not require complex queries. This database achieves the best results from horizontal scaling (Amazon Web Services, n.d.). Redis is a popular, in-memory implementation of this type of database. It was ranked as the number four position for top developer tools of 2019 by stackshare.io, a website about sharing technology stacks of different organisations (StackShare Team, 2020).

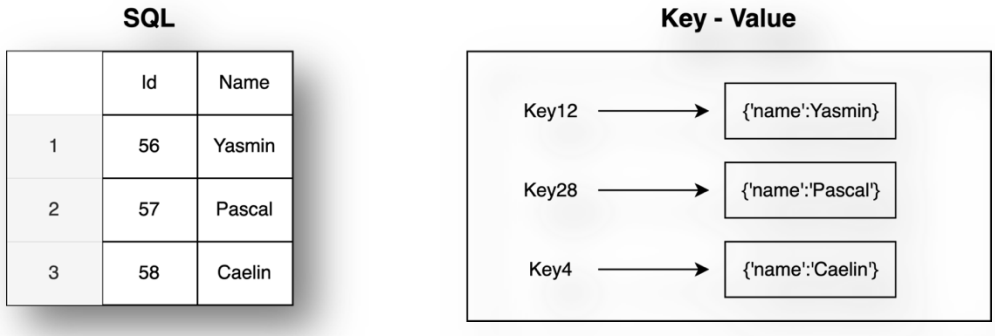

Figure 28 SQL vs Key-Value

Graph

Graph-based NoSQL consists of the use of two entities (PhoenixNAP, 2020)

- Nodes for storing data
- Edges for storing the relationship between nodes

A visual representation can be seen in Figure 29. In this example, each node holds information about a person. The edges define the relationships a person has with another. As this type of storage can be very data-specific, it is not commonly used. Graph datastore is particularly useful when needing to find patterns in relationships. An example of a use case might be a social network to store data on the relationship between its users. RedisGraph and Neo4j are examples of such databases.

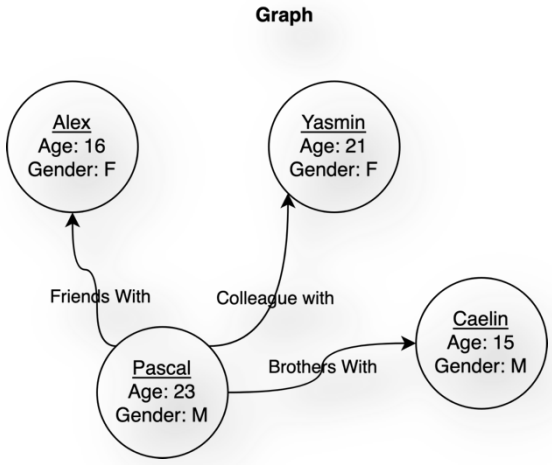

Figure 29 Graph NoSQL

Column Store

Column store, also known as Wide Column, stores data in tables with rows and columns. Data is stored in a ‘cell-like’ structures in columns known as Column Families. Column families can be grouped into super column families. Virtually unlimited number of columns and column families can exist and do not require a complete schema to be initially created. Columns can be created at runtime as needed. A visual representation can be seen in Figure 30. A popular implementation of Column Store is Apache Cassandra.

	Person					
	Name	**Gender**	**Address**		**Education**	
1	Lily	F	**City**	Portlaoise	**Heigher Level**	ITCarlow
			County	Laois	**Secondary**	Scoil Chríost Rí
					Primary	Whitehill Cork
2	Ted	M	**City**	Naas	**Heigher Level**	Trinity College
			County	Dublin	**Secondary**	St.Mary's CBS
					Primary	St.Patrick's NS

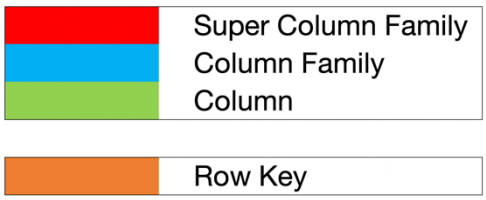

Figure 30 Column Store

Apache Cassandra (Cassandra) is an open-source database developed using Java. It is used by over 40% of the fortune 100 companies, including Apple, GitHub, Microsoft, Netflix, Spotify and many more (Apache, n.d.). Cassandra is best known for its performant abilities (Rabl, et al., 2012). It can have consistently high operations per second, with powerful capabilities to horizontally scale out nodes (Jonathan Ellis, 2013). It designed on a master-less approach to decentralisation, intending to eliminate a single-point-of-failure in a cluster. Cassandra’s high performance is based on the philosophy of three principles (Alex Bekker, 2018)

- Storage space is cheap
- Writes are fast
- Networking is an expensive endeavour

Once a node receives a write request, the node will write to the commit log. This is used storing information about its in-memory cache. The write request is concurrently placed into a table in memory known as a ‘MemTable’. This table gets written to disk cache into ‘SSTables’. Once the memtable is written to disk, the Commit Log is purged of its information. If a read request is received, Cassandra first checks its memtable followed by multiple different caches Cassandra creates, such as the row key cache or the partition key cache.

7. Security

One of the foundational requirements of security is trust. There is no escaping trusting an entity in security. The trust might be put into the centralised server that a user connects to or in the application-developer to implement best practices during development. Trust, once damaged, can become challenging to regain.

Transparency is fundamental in gaining user trust. An organisation/project must be transparent to its users, in general, as transparent as possible. Transparency can come in many forms, such as undergoing independent security audits or open-sourcing the applications' source code.

Open-source applications can be debated in security. On one side, the scale of developers now available to improve a codebase expands to the entire developer community. On the other side, a malicious individual now has access to a deeper understanding of the internals of a system which before may have remained a shrouded mystery.

Encryption aids in protecting data from unauthorised parties. It can be implemented at different stages, such as encryption-in-transit through the use of HTTPS or encryption-at-rest by enciphering the data in its local storage. Encryption can be described as the "*..process that scrambles readable text so it can only be read by the person who has the secret code, or decryption key*" (Alice Grace Johansen, 2020).

End-To-End Encryption (E2EE) is about encrypting data in transit from the origin to the destination. It prevents any listeners, including a central server, from intercepting the data in the middle of the endpoints. This differs from the general encryption implementation employed, which focuses on protecting data in transit between the user and the central server. With E2EE, the users hold their decryption keys other parties are unable to decipher the required keys. E2EE in protecting user data can be considered more robust than encryption in transit alone (Proton Team, 2018).

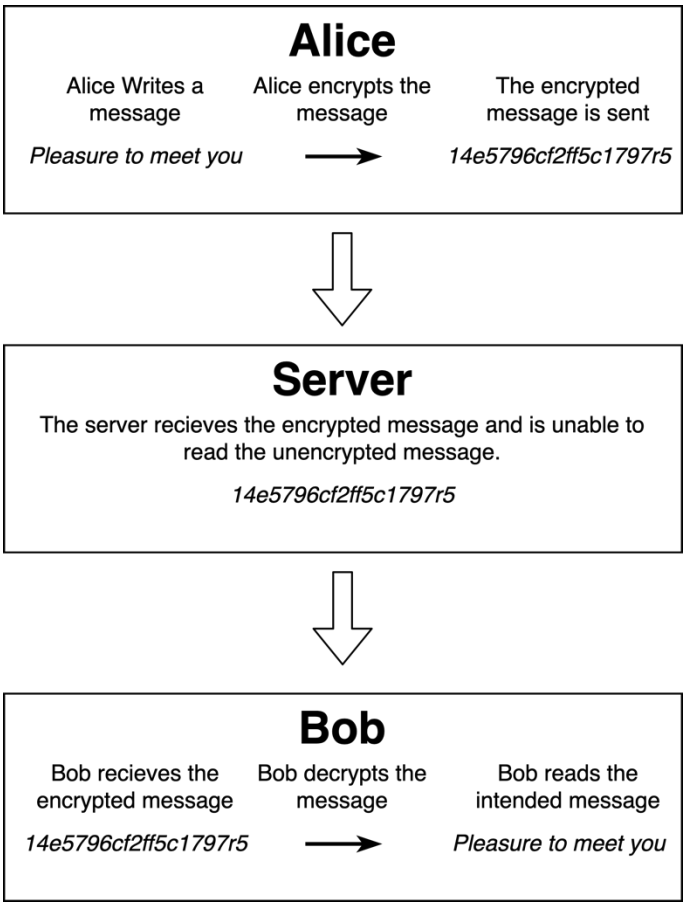

Figure 31 End-To-End Encryption

To exemplify E2EE, see Figure 31. In it, Alice desires to send Bob a message. Alice encrypts the message to be sent, using a method only Bob can decrypt. The encrypted message from Alice is forward to the server. The server receives the message; however, it is the encrypted form, and the server is unable to understand the message Alice intends for Bob. The server passes the encrypted message to Bob. Once Bob receives the message from the server, he decrypts the message to view the original message. In this scenario, the message remained a secret between Alice and Bob. If a third party attempts to listen at any point, they will receive the encrypted message, which they are unable to decrypt.

Zero-knowledge encryption is using encryption to make data inaccessible to the service provider while it is at rest. Any data associated with the user can only be decrypted using their private key. With this form of encryption, if a data breach occurs, this ensures that the data remains protected (Ben Wolford, 2019).

The difference between E2EE and Zero-knowledge encryption is the medium in which it is. E2EE is intended to obfuscate information in transit, as it passes traverses a network between the recipients and senders devices. Zero-knowledge encryption is about obfuscating information once it has finished traversing a network, known as at rest.

8. Supplemental Updates

Throughout the project, additional research was required. Included within this section is additional information on new or existing topics where information was gathered.

Web-Sockets

As a simplification, a Web-Socket can be in one of two states

- Open
- Closed

An open connection can send and receive data; alternatively, it could be called an ‘alive’ connection. A connection can have a configurable timeout associated. If this time is reached, the client will send a keep-alive connection ping, hence the name of an ‘alive’ connection, that, if responded to, will prevent the connection from being closed. A closed connection is no longer capable of sending or receiving data. If the exchange of data is desired at a state of a closed socket, a connection to the server or client will need to be re-established. Web-Socket connections are lightweight to keep open, and a server often can handle hundreds of thousands of active connections.

Python

FastAPI

FastAPI can be described as

> ..a modern, fast (high-performance), web framework for building APIs with Python 3.6+..
>
> *Sebastián Ramírez, creator of FastAPI*

It is a web framework designed for API development; At its' centre, it is about using modern Python features such as type hints and conformance to standards. FastAPI is much like a `wrapper' for the Starlette framework. Starlette is an Asynchronous Server Gateway Interface (ASGI) framework. Starlette supports the generation of OpenAPI specification (Formally known as Swagger). FastAPI further utilises the *pydantic* package to enforce Python type hints of a defined JSON schema. Combining pydantic with Starlette's OpenAPI documentation provides a high-performance and standards-oriented framework for creating APIs.

At the beginning of the project life-cycle, Python's Flask was the prospective technology; A hesitation of Flask was in place due to Flasks' and Pythons' inherent concurrent inabilities due to the Global Interpreter Lock within Python's architecture. With the interest of a comparison, a micro-benchmark was devised.

To test the response time and concurrent capabilities of Flask and FastAPI, the Locust Load Testing tool was used. This is a Python-based tool that uses the Greenlet package to create Green Threads to swarm the intended server with the desired amount of concurrent users executing Python-defined instructions. Green threads (Also known as a virtual thread) vary from standard native operating system threads by the space they are scheduled within. A green thread is scheduled exclusively within a virtual machine or a runtime library instead of using the underlying operating system. This attempts to simulate a concurrent environment without true parallelism occurring. This micro-benchmark simulated a login request. The task to be completed by each user can be seen in Figure 32.

```
from locust import HttpUser, task
class QuickstartUser(HttpUser):
    @task
    def test(self):
        self.client.post("/", data={
            "username": "admin",
            "password": "password"
        })
```

Figure 32 Locust Micro-benchmark

This micro-benchmark was executed with two thousand users, beginning at zero and ramping up at fifty users every second. Once the two thousand users were reached, the test continued to run for five minutes. As Flask and FastAPI are foundationally frameworks and not servers, they require a server to run upon. FastAPI was executed on Uvicorn, while Flask on Gunicorn. The results of this can be seen in table 1 and 2.

# Requests	# Fails	90%ile (ms)
39,495	14,105	30,000

Table 2 FastAPI Result

# Requests	# Fails	90%ile (ms)
248,604	0	2,200

Table 1 Flask Results

As Flask did not net a desired result, FastAPI was chosen. FastAPI did not fail a single request; this raised the question: *How many more users could*

FastAPI handle? More benchmarks were executed, scaling the simulated users vertically up to seven thousand and fourteen thousand yielded the results shown in table 3 and 4.

# Requests	# Fails	90%ile (ms)
623,656	4,071	5,100

Table 4 FastAPI 7,000 Users

# Requests	# Fails	90%ile (ms)
580,292	31,145	8,400

Table 3 FastAPI 14,000 Users

With a ninetieth percentile response time of 5,100ms and 4,071 fails at seven thousand users, it was not desirable to push FastAPI further.

Tkinter

The Tkinter package comes as part of the Python standard library. It stands for `Tk interface'. It is an interface to the Tcl's programming language Tk GUI toolkit. The Tk toolkit became the framework the reference client was created upon. Paired with the ‘Websocket-client’ package, ‘PyCryptodome’ package for cryptography and the Python standard library for JSON parsing, the required capabilities of a client were fulfilled.

API Development

Documentation

As part of developing an API for developers came the need to create developer-focused documentation. The OpenAPI standard appears to be one of the most popular standards, additionally overseen by the Linux Foundation. It initially was named the Swagger Specification only to become a separate project. It is a specification that aims to describe how RESTful API's should be consumed. Multiple tools exist that can produce visualisations or generate test cases based on this. FastAPI supports the generation of OpenAPI documentation automatically by using Starlette as a back-end that natively incorporates this generation.

API Tools

Within a specific field, certain tools aid in development. In the Data Science field, a Data Scientist may utilise Jupyter Notebook. For API development, tools exist that aid in developing the application. A specific set of software tools often built on Electron or Chromium, allow the designing and testing of APIs' created. A popular

example of this tool is `Postman'. It allows the sending of requests through a graphical user interface (GUI). For example, a developer may need to send a multi-part form to a specific URL using a PUT HTTP method. This where these tools come into usage.

A developer can send complex or simplistic requests to a specified URL with complete control over the HTTP header. Additional information is provided once the request is responded to, such as

- HTTP Status
- Time for the request to complete
- Overall size of the request
- Body Size
- Header size and much more

Elixir

Elixir is a functional general-purpose programming language that aims to provide the Erlang feature-set with syntax heavily inspired by Ruby. It runs on Bogdan's Erlang Abstract Machine (BEAM) virtual machine (VM) and is extensible to native Erlang functions through the bytecode of the BEAM VM. Like Erlang, Elixir is particularly effective within high-concurrency environments with desired fault-tolerance handling. Elixir uses the same Supervisor model like Erlang to handle processes, with the `let it crash' philosophy.

Cryptography

Cryptography itself can come in multiple forms. For this project, the particular topics of interest are in the form of Symmetric-key cryptography and Asymmetric-key (Also known as Public-key) cryptography. To note, from this point forward, the content the encryption or decryption is performed upon will be referred to as `plaintext'. Plaintext that is encrypted is known as `Ciphertext'.

Symmetric Cryptography

Symmetric cryptography is the usage of a singular key for both encryption and decryption of plaintext. This type of cryptography is, on encryption and decryption time alone, overall faster by design due to the use of a singular key and key length is often shorter. A visualisation of this is showcased in Figure 32

Figure 33 Symmetric Cryptography Diagram

This type of cryptography is often used in file or database encryption due to the performance gain over asymmetric and the requirement for a singular key for file control. Examples of Symmetric algorithms are listed below.

- AES/Advanced Encryption Standard (Rijndael)
- DES/3DES
- Serpent
- Twofish

Asymmetric Cryptography

Asymmetric Cryptography uses a two-key system instead of a singular key by Symmetric Cryptography. The two keys are generally referred to as a Public and Private key. A disadvantage of the Symmetric key system is key transmission; every key share posses the risk of interception by an unintended third party. Asymmetric cryptography solves this by allowing the sharing of the public key. Once generating cryptographic keys, effectively two keys are created. The private key is kept secret and should not be shared or publicly exposed. The public key can be freely shared with third parties. The public key is used to create ciphertext that can only be decrypted to plaintext if the private key is known. The process of using asymmetric cryptography can be seen in Figure 34.

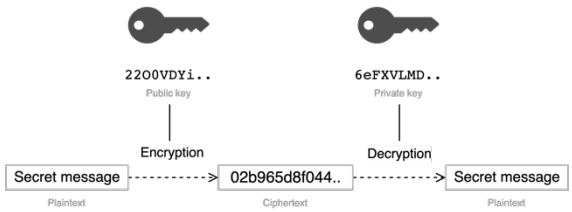

Figure 34 Asymmetric Cryptography Diagram

PEM Format

The PEM (Privacy-Enhanced Mail) is a file format for storing and sending cryptographic keys and certificates. It has become standardised by the IETF (Internet Engineering Task Force). As exchanging binary data (Such as a private key) can be problematic over a network, PEM is used to solve this issue. PEM stores the files in a base64 encoded format.

The file begins with five dashes and a `BEGIN', followed by a label and an additional five dashes. The file ends in the same format using an `END' in place of the `BEGIN'. PEM is used to store keys by the reference client. An example of a PEM file can be seen in Figure 35.

```
-----BEGIN RSA PRIVATE KEY-----
fIohV3etQecGfYTkcRQZ5Y8bzI4MBIV4DFDscb6ZRBv7FRtlCMhQPuCfe7yR3CRD
C9JQ6CN2J1ofCLtpHFhcAgfaFXR77S8QcqiJlMpLsSSrWtf2OQNTJJeCR1VZUotG
mt0Ybittw0aE28ZzxcEYmZadzKporDgp/dy5DKbKrBTRxUe7yw4HEWJWXgX+G9f2
12K9FVeoWYlsKM2syKNeN7XkzCDOX6GKu0EeM8HT6FN8VJ8uu2OP33TVXQk=
.....
-----END RSA PRIVATE KEY-----
```

Figure 35 PEM Example

Cassandra

Cassandra is a distributed NoSQL Column-Store database with a highly rigid schema. Cassandra queries are completed through the use of the Cassandra Query Language (CQL). CQL on a high level can be likened to SQL with the lack of relational functions such as JOIN or GROUP BY.

Cassandra is particularly effective at write performance and horizontally scaled distribution with fault tolerance. Cassandra achieves the write performance by storing much of the information in main memory. Once a write request arrives, Cassandra will write the request to a `commit log' and in-memory to the Memtable. The Memtable behaves like a cache for temporary storage. At different conditions, the Memtable is written from main memory to the disk into an SSTable (Sorted Strings Table). This is an optimised immutable table that provided persistence of

data. The commit log is used to prevent Cassandra from being too bound to the SSTable; As the SSTable is immutable, a modification or update will need a new SSTable to be created.

Another reason Cassandra has a performance advantage is through the column-store design. Assuming a user table exist, such as in table 5.

Id	Username	Age
56	Sunbeam	24
57	GentleMitten	20

Table 5 User Table

If we desire to read all ages from the table in SQL, a query may look like this '`SELECT Age FROM Users`'. A traditional SQL database will read all the columns from left to right. This includes the redundant username column. A Cassandra query will read the Age column downwards, never having to read the Username column. This is the reason Cassandra is a `Column-Store' database. This read difference is highlighted in orange in Tables 6 and 7.

Id	Username	Age
56	Sunbeam	24
57	GentleMitten	20

Table 7 SQL Read

Id	Username	Age
56	Sunbeam	24
57	GentleMitten	20

Table 6 Cassandra Read

9. Further Research

This document outlined multiple points of research. However, further research could be done in all areas of the project. In particular

- WebRTC – Developing a working prototype to experiment with its usage, particularly in the usage of TURN servers.
- Encryption – Further search into more technical details of how the cryptography functions programmatically and mathematically.
- Databases – Development of a functional prototype to showcase Column Store in Cassandra.

10. Bibliography

Adam Fowler and Eugene Ciurana, 2017. *DZone.* [Online]
Available at: https://dzone.com/refcardz/nosql-and-data-scalability-20?chapter=1
[Accessed 7 November 2020].
Alex Bekker, 2018. *ScienceSoft.* [Online]
Available at: https://www.scnsoft.com/blog/cassandra-performance#write
[Accessed 5 November 2020].
Alfred Ng, 2020. As Defcon goes virtual, organizers step up efforts to prevent online harassment. *CNet.*
Alice Grace Johansen, 2020. *Norton.* [Online]
Available at: https://us.norton.com/internetsecurity-privacy-what-is-encryption.html
[Accessed 4 November 2020].
Amazon Web Services, n.d. *aws.amazon.com.* [Online]
Available at: https://aws.amazon.com/nosql/key-value/
[Accessed 1 November 2020].
Apache, n.d. *Apache Cassandra.* [Online]
Available at: https://cassandra.apache.org/
[Accessed 5 November 2020].
Bazzell, M., 2020. *Privacy, Security, & OSINT Show 185.* [Sound Recording].
Ben Wolford, 2019. *ProtonMail.* [Online]
Available at: https://protonmail.com/blog/zero-access-encryption/
[Accessed 7 November 2020].
Briar, n.d. *BriarProject.org.* [Online]
Available at: https://briarproject.org/how-it-works/
[Accessed 5 November 2020].

Chandan Kumar Singha, 2019. *Medium.com.* [Online]
Available at: https://medium.com/@cskkman/scaling-load-balancing-4a2447fa4529
[Accessed 31 October 2020].
Cloudflare, n.d. *Cloudflare.* [Online]
Available at: https://www.cloudflare.com/learning/ssl/what-is-ssl/
[Accessed 29 October 2020].
Craig S. Mullins, 2018. *DZone.com.* [Online]
Available at: https://dzone.com/articles/what-do-we-mean-by-database-scalability
[Accessed 31 October 2020].
DataPrivacyManager, 2020. *DataPrivacyManager.* [Online]
Available at: https://dataprivacymanager.net/security-vs-privacy/
[Accessed 28 October 2020].
David E.Sanger and Brian X.Chen, 2014. Signaling Post-Snowden Era, New iPhone Locks Out N.S.A.. *The New York Times.*
DB-Engines, 2020. *DB-Engines.* [Online]
Available at: https://db-engines.com/en/ranking
[Accessed 1 November 2020].
DB-Engines, 2020. *DB-Engines CouchBase vs MongoDB.* [Online]
Available at: https://db-engines.com/en/system/Couchbase%3BMongoDB
[Accessed 3 November 2020].
Def Con Official, n.d. *Def Con 28 Safe Mode - Discord Server.* [Online]
Available at: https://defcon.org/html/defcon-safemode/dc-safemode-discord.html
[Accessed 17 October 2020].
DuckDuckGo, G. T., 2020. *Trends.Google.com.* [Online]
Available at: https://trends.google.com/trends/explore?date=today%205-y&q=duckduckgo
[Accessed 27 October 2020].
E. Rescolra, 2000. *tools.ietf.org.* [Online]
Available at: https://tools.ietf.org/html/rfc2818
[Accessed 29 October 2020].
Electronjs, 2020. *electronjs.* [Online]
Available at: https://www.electronjs.org/
[Accessed 27 October 2020].
element.io, 2020. *Element.* [Online]
Available at: https://element.io/
[Accessed 27 October 2020].
Eric S. Yuan, 2020. [Online]
Available at: https://blog.zoom.us/a-message-to-our-users/
[Accessed 1 April 2020].

Eric S. Yuan, 2020. Zoom Acquires Keybase and Announces Goal of Developing the Most Broadly Used Enterprise End-to-End Encryption Offering. *Zoom Blog.*
Ewen MacAskill and Glenn Greenwald, 2013. NSA Prism program taps in to user data of Apple, Google and others. *The Guardian.*
Fanghao (Robin) Chen, 2016. *blog.Discord.com.* [Online]
Available at: https://blog.discord.com/using-react-native-one-year-later-91fd5e949933
[Accessed 26 October 2020].
Felix Richter, 2014. *Statista.* [Online]
Available at: https://www.statista.com/chart/3002/how-internet-users-adapted-after-snowden-revelations/
[Accessed 27 October 2020].
Frederic Lardinois, 2019. *techcrunch.* [Online]
Available at: https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/?guccounter=1&guce_referrer=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnLw&guce_referrer_sig=AQAAABIDD3mswPX7mYXSOtp3mFbnm2fWHi68o6NELgjztNoesK7JNF10QKpJBS5U6XAXlXq
[Accessed 27 October 2020].
GitLab, B., n.d. *BriarProject.org.* [Online]
Available at: https://code.briarproject.org/briar/briar/-/blob/master/bramble-core/src/main/java/org/briarproject/bramble/db/DatabaseModule.java
[Accessed 26 October 2020].
Glenn Greenwald, 2013. NSA collecting phone records of millions of Verizon customers daily. *The Guardian.*
Jonathan Ellis, 2013. *DataStax.* [Online]
Available at: https://www.datastax.com/blog/2012-review-performance
[Accessed 5 November 2020].
Jozsef Vass, 2018. *blog.discord.com.* [Online]
Available at: https://blog.discord.com/how-discord-handles-two-and-half-million-concurrent-voice-users-using-webrtc-ce01c3187429
[Accessed 26 October 2020].
Kitamura, M. U. a. E., 2010. *HTML5Rocks.* [Online]
Available at: https://www.html5rocks.com/en/tutorials/websockets/basics/
[Accessed 31 October 2020].
letsencrypt.com, 2020. *letsencrypt.com.* [Online]
Available at: https://letsencrypt.org/stats/
[Accessed 29 October 2020].

Librarian, D., 2020. *support.Discord.com.* [Online]
Available at: https://support.discord.com/hc/en-us/articles/360041360311
[Accessed 26 October 2020].
Matt Nowack, 2019. *blog.discord.com.* [Online]
Available at: https://blog.discord.com/using-rust-to-scale-elixir-for-11-million-concurrent-users-c6f19fc029d3
[Accessed 26 October 2020].
Matthew Hodgson, 2018. *matrix.org/blog.* [Online]
Available at: https://matrix.org/blog/2018/04/26/matrix-and-riot-confirmed-as-the-basis-for-frances-secure-instant-messenger-app
[Accessed 27 October 2020].
mhoye, 2019. *discourse.mozilla.org.* [Online]
Available at: https://discourse.mozilla.org/t/synchronous-messaging-at-mozilla-the-decision/50620
[Accessed 27 October 2020].
MongoDB, n.d. *Mongodb.com.* [Online]
Available at: https://www.mongodb.com/json-and-bson
[Accessed 5 November 2020].
Mozilla, 2019. *MDN Web Docs.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Protocols
[Accessed 31 October 2020].
Nick Stripe, 2020. *ons.gov.uk.* [Online]
Available at:
https://www.ons.gov.uk/peoplepopulationandcommunity/crimeandjustice/bulletins/crimeinenglandandwales/yearendingmarch2020#computer-misuse
[Accessed 28 October 2020].
Oracle, n.d. *Oracle.* [Online]
Available at: https://www.oracle.com/database/what-is-database.html
[Accessed 31 October 2020].
PhoenixNAP, 2020. *PhoenixNAP.* [Online]
Available at: https://phoenixnap.com/kb/nosql-database-types
[Accessed 1 November 2020].
Proton Team, 2018. *ProtonMail.* [Online]
Available at: https://protonmail.com/blog/what-is-end-to-end-encryption/
[Accessed 4 November 2020].
ProtonMail, G. T., 2020. *Trends.Google.com.* [Online]
Available at: https://trends.google.com/trends/explore?date=today%205-y&q=ProtonMail
[Accessed 27 October 2020].

ProtonVPN, G. T., 2020. *Trends.Google.com.* [Online]
Available at: https://trends.google.com/trends/explore?date=today%205-y&q=ProtonVPN
[Accessed 27 October 2020].
Publication, C. S., 2020. *Central Statistics Office.* [Online]
Available at: https://www.cso.ie/en/releasesandpublications/ep/p-sic19/socialimpactofcovid-19surveyapril2020/
[Accessed 27 October 2020].
pypi, 2018. *Pocketsphinx Python.* [Online]
Available at: https://pypi.org/project/pocketsphinx/
[Accessed 31 October 2020].
Rabl, T. et al., 2012. *Solving Big Data Challenges for Enterprise ApplicationPerformance Management,* s.l.: VLDB Inc.
Reisdorf, G. B. a. B., 2012. *The Participatory Web,* s.l.: ResearchGate.
Samhita Alla, 2018. *codeburst.io.* [Online]
Available at: https://codeburst.io/building-your-first-chat-application-using-flask-in-7-minutes-f98de4adfa5d
[Accessed 5 November 2020].
Signal, G. T., 2020. *Trends.Google.com.* [Online]
Available at: https://trends.google.com/trends/explore?date=today%205-y&q=%2Fm%2F012pq75w
[Accessed 27 October 2020].
StackOverflow, 2020. *StackOverflow.* [Online]
Available at: https://insights.stackoverflow.com/survey/2020
[Accessed 3 November 2020].
StackShare Team, 2020. *StackShare.* [Online]
Available at: https://stackshare.io/posts/top-developer-tools-2019
[Accessed 1 November 2020].
Stanislav Vishnevskiy, 2017. *Blog.Discord.com.* [Online]
Available at: https://blog.discord.com/how-discord-stores-billions-of-messages-7fa6ec7ee4c7
[Accessed 26 October 2020].
Tasos Lazarides, 2015. *toucharcade.* [Online]
Available at: https://toucharcade.com/2015/09/14/ex-fates-forever-developers-making-discord-a-voice-comm-app-for-multiplayer-mobile-games/
[Accessed 26 October 2020].
Terblanche, K., 2020. *Medium.com.* [Online]
Available at: https://medium.com/dvt-engineering/when-to-use-a-nosql-database-over-a-sql-database-daac89059d8b
[Accessed 7 November 2020].

Troy Segal, 2019. *Investopedia.* [Online]
Available at: https://www.investopedia.com/terms/b/big-data.asp
[Accessed 7 November 2020].
VPN, G. T., 2020. *Trends.Google.com.* [Online]
Available at: https://trends.google.com/trends/explore?date=today%205-y&q=VPN
[Accessed 27 October 2020].
WebRTCGlossary, 2017. *WebRTCGlossary.* [Online]
Available at: https://webrtcglossary.com/ice/
[Accessed 31 October 2020].