

APRIL 1, 2023



DEVELOPING A MONITORING SOFTWARE SOLUTION FOR USE IN HEALTHCARE RECORDS SYSTEMS

BY ADAM VIMPANY

Supervisor: Dr Tim Bashford

Project submitted as part of the requirements for the award of BSc (Hons) Computing
(Software Engineering)

ADAM VIMPANY 1810613
INDEPENDENT PROJECT: ACCA 6020
University of Wales Trinity St David

DECLARATION FORM AND FORM OF CONSENT

Declaration

I, Adam Vimpany, declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Form of consent

I, Adam Vimpany, hereby consent that my Project, submitted in candidature for the B.Sc. (Hons) Computing (Software engineering) degree, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

ABSTRACT

Healthcare records systems are a complex chain of interconnected systems and applications, the management and maintenance of which can pose a major problem for healthcare organisations. The aim of this project was to plan a software solution that could be used to improve the monitoring of webservices and XML drip-feeds for NHS Wales WelshPAS integration, and then implement it.

A literature review was performed to identify possible solutions that had already been developed and evaluate them for performance and applicability for this project. The main conclusions from the review were that a remote, agentless internal monitoring architecture would be most appropriate, and some further recommendations were made. These recommendations were presented to the stakeholders of the project. Rapid Application Development methodology was used to identify the user requirements, then plan and develop the software.

It was determined that the `sync_date` database field for each health board should be monitored for changes to track the connections being made, and a wireframe diagram was created to demonstrate the intended architecture of the program. The software was developed using NodeJS, as this had the advantage of being able to use an asynchronous approach which prevented blocking of connections during data retrieval. A range of packages were identified and utilised to ensure development time was used efficiently.

During development, users were consulted after each iteration, and some suggestions for improvements were made: these included the addition of email alerts and the creation of a centralised connections database. The application successfully tracked the connectivity for each health board and was designed with the intention that it would be easy to amend if changes were made in the future, as is currently intended by the organisation.

Areas for improvement or further work were identified, and these include the use of technologies such as React or Vue to improve the frontend, improved testing, and exploration of avenues identified in the literature review that could be introduced to expand the functionality of the application.

TABLE OF CONTENTS

Declaration form and form of consent.....	1
Declaration	1
Form of consent.....	1
Abstract	2
Chapter 1: Introduction	5
Background	5
Problem	6
Aim	7
Objectives	7
Chapter 2: Research and literature review.....	9
Introduction	9
Background	9
Interoperability and integration	9
Webservices and APIs.....	10
Database	11
Monitoring systems.....	11
Background.....	12
Research	13
Recommendations	14
Chapter 3: Development methodology	16
Methodology.....	16
Architecture.....	17
Software.....	18
Delphi	18

.NET	18
Node.JS.....	19
Packages	19
Node-Firebird	19
Crypto	20
Express.....	20
Easy-SOAP-request	20
Axios/Ajax/jQuery	20
Node Mailer	21
Docker.....	21
Chapter 4: Design and implementation.....	22
Design.....	22
Project requirements	22
Prototype	24
Implementation.....	26
Problems and successes.....	28
Result	31
Chapter 5: Evaluation	32
Chapter 6: Conclusions and recommendations	34
Chapter 7: Reflection	36
Chapter 8: References	38
Figure 1: Wireframe diagram	24
Figure 2: Prototype UI	25
Figure 3: User interface of monitoring application	31

CHAPTER 1: INTRODUCTION

Background

It is essential for modern healthcare records systems to be interoperable with a huge range of other software applications and systems. Each of the different health boards, hospitals, departments, clinics and disciplines in a modern healthcare setting will have differing requirements for software and data, therefore it would be an enormous task to develop one overarching system that would meet all of these needs. It is essential that these different health care applications can pull records from and send updates to records systems seamlessly to ensure that clinicians and staff are able to access the patient information they need to provide the best possible care [1]. The data managed by healthcare records systems, and required by applications, includes patient demographics; medical histories; hospital, ward or clinic admissions and discharges; and other special category data that is protected under the UK General Data Protection Regulation (GDPR) [2]. Healthcare records are especially complex, because they can also include diagnostic test results, long-form descriptive clinical information, referrals or images [3]. It is essential to ensure that all records for patients, no matter how complex, are secure and accurate, while maintaining a system that is interoperable.

In NHS Wales, the Welsh Patient Administration System (WelshPAS) is a centralised, cloud-based healthcare records system that runs in all seven NHS Wales health boards and administers millions of patient records. It is used to maintain a single, universal, up-to-date source of patient data which can be accessed by other healthcare systems and applications across Wales, using interfaces to send and receive messages.

The WelshPAS integration department have identified a need for a software solution that can be used to monitor the connectivity of interfaces for all servers and health boards and alert staff to problems with the system that may impact patient data or care.

Problem

WelshPAS uses SOAP webservice interfaces to handle both inbound and outbound requests for data from external applications and services. The WelshPAS web services handle over 100,000 requests per day. The applications run in IIS, with each individual health board being served by their own copy of the web service. In total, there are eight SOAP APIs (discussed in more detail in the Webservices and APIs section below), each with over 100 methods. The web services are vital to the running of WelshPAS and many other important healthcare applications.

An XML “drip-feed” pulls XML files from a database and adds them to a Microsoft message queue. These are filtered to the national integration engine, Fiorano, and the XML files are then routed to the destination systems. Inbound messages, such as demographic updates, are received in HL7, and then converted to an XML file and used to populate the WelshPAS master patient index (MPI).

The problem that was identified was that there was very limited monitoring of the interfaces used by WelshPAS. At the time, the XML feed used a “heartbeat message” as one of the only methods of monitoring. This message was automatically triggered every 20 minutes and travelled through the integration interfaces before looping back: if no message is returned, then this indicates a potential problem in the system and an email alert is triggered. This is a very basic method of monitoring as it only indicates that a problem may exist somewhere in the system, with no information about the origin of the problem, or the stage of the process at which the problem occurred.

The integration department required a more sophisticated, proactive, monitoring solution than relying on the “heartbeat message” as they found they were often reliant on feedback from downstream systems and user error reporting to identify problems. A worst-case scenario would be an end user in the process of using an essential healthcare application for patient treatment who identifies a problem receiving patient data and reports it to their IT service desk. Significant delays could happen as the ticket is passed through official channels: a preliminary investigation may be performed by their hospital or departmental IT staff to rule out isolated problems with local hardware or software, followed by escalation to the organisational service desk. If this is then compounded by misassignment of the ticket to a different department, then by the time the

error is investigated by them and reassigned to the integration department, there could have been a prolonged negative impact for even the most easily rectified of problems. This was a serious risk for such an integral part of the healthcare records system and could potentially affect patient care.

A more effective monitoring system enables the proactive identification and resolution of problems before users or other systems have been impacted, rather than reactively investigating user reports of errors or system downtime. Ideally, the monitoring software should give an indication of which servers, databases or health boards are affected, to aid investigation into the source of connectivity problems and therefore improve resolution times. The information provided by the monitoring system should allow department staff to proactively send alerts and updates to shareholders if problems are found and suspected to be serious or have a wide impact, which gives end users the opportunity to put in place actions to minimise impact on workflows or patient care while the problem is resolved. Successful deployment of monitoring software is expected to both improve the department's accountability and empower service users.

Aim

This project aims to design, develop and implement a web application to track the connectivity of the SOAP webservices and XML drip-feed for all servers and health boards across Wales. The application must be able to be accessed by integration department staff and must use a dashboard view to display relevant metrics that determine the status of each service.

Objectives

The objectives for this project were identified as follows:

- Identify previous approaches to developing a monitoring software application and review their benefits and drawbacks
- Using the results of the literature review, highlight the main recommendations for the architecture and design of the project
- Identify an appropriate metric to monitor
- Identify and follow an appropriate development methodology

- Discover the requirements of the application
- Develop a prototype design that meets the user requirements
- Select appropriate software and technologies to develop the application
- Implement the application
- Review the application with users and implement improvements
- Identify how testing should be performed
- Release the software
- Evaluate the software.

CHAPTER 2: RESEARCH AND LITERATURE REVIEW

Introduction

This chapter describes the background to the project and the literature that was consulted before the project was undertaken. The WelshPAS infrastructure and theory behind its use is discussed in the background section, including the technologies used in the organisation to achieve interoperability. The next section discusses previous approaches taken to development of monitoring systems, their benefits and drawbacks, and their applicability to this project. Recommendations for the design stage are identified and discussed.

Background

To understand the rationale behind the project, it is important to understand what WelshPAS and the interfaces controlled by the department accomplish. This section discusses the principles behind the system and the technologies it employs.

INTEROPERABILITY AND INTEGRATION

As mentioned previously, this project ties into the principles of interoperability and integration. Interoperability, where the services of two different systems or applications are made to work together using only a common standard [4], is a huge concern for complex systems such as those in healthcare, as the timely exchange of data across applications is essential [1]. These different applications may use different frameworks, data structures or technologies for their functionality, and therefore connecting them to allow for communication can be time-consuming and difficult [5]. Networks that contain legacy software that does not adhere to modern technical standards, or that contain specialised components or hardware may add additional complexity [6], and this is particularly common in the healthcare field. Additionally, the data itself may be complex and difficult to standardise between applications. Solutions to the interoperability problem include integration and the use of middleware. Middleware is a third-party application that sits between two or more applications to control and transform communication requests to make them readable by the receiving application: this contrasts with interoperable systems where the

applications communicate with each other directly [4]. However, the addition of middleware interfaces between applications adds another additional layer of complexity.

To combat these problems, and minimise complexity, there has been a move in the wider healthcare industry to create a standardised approach for sending healthcare data across applications and systems. In particular, the Fast Healthcare Interoperability Resources (FHIR) were developed by HL7 international and are becoming widely adopted [3]. FHIR defines resources, and how these resources are accessed using RESTful Application Programming Interfaces (API) [3] (discussed in the next section). Systems used in NHS Wales will eventually move to this standard, however, updating or replacing existing infrastructure is time and resource intensive, and must be introduced cautiously to minimise disruption to working systems. For this reason, older and newer technologies will need to co-exist for the foreseeable future, by using both interoperable and integration methods. Any new applications that are created for use with older technologies will ideally be developed in a way that enables them to be quickly and easily adapted to become compatible with newer technologies. This is an important consideration for this project.

WEBSERVICES AND APIS

APIs are very common protocols that are used to allow the communication and exchange of data between systems and applications by providing a standard framework that defines how applications request, respond to and receive data [7]. They are essential for many applications, and are used to integrate healthcare applications in the healthcare industry: NHS Digital in England [8] and Digital Health Care Wales [9] both use a range of API technologies for their digital healthcare information systems. Some of these API architectures will be discussed here.

Simple Object Access Protocol (SOAP) is a protocol for webservices APIs that has been used since the 1990s [10] but is an aging technology that is falling out of favour since the introduction of Representational State Transfer (REST) APIs [10, 3]. SOAP follows Service-Oriented Architecture (SOA) principles, and applications host self-describable services that are accessed using webservices, usually through XML messages [11], but can also support other formats such as JSON or HTML [12]. SOAP is a very strict, standardised protocol [10], and is considered to be a secure method of communication, due to the way the SOAP protocol “wraps” messages [12].

RESTful APIs are the most used interface type in industry today [3], replacing SOAP due to the increased flexibility, reliability and scalability [13]. The REST architectural principles are five non-optional and one optional design constraints which define how the system should be designed to enable communication, without specifying the exact technologies to be used to achieve this [13]. It is a lighter-weight architectural style, rather than a protocol like SOAP [12]. FHIR, discussed in the previous section, is a RESTful specification [3].

WelshPAS interfaces use SOAP webservices to handle requests from external applications and servers. Other areas of the system use different webservices and APIs, creating additional complexity. Work is currently underway in NHS Wales to fundamentally reorganise and restructure the management and handling of data from the ground up, with the creation of a new National Data Resource (NDR) Data Strategy [14], and it is intended that these SOAP webservices will be replaced with a FHIR open API [9, 14]. The intent is to create a single standard approach within the organisation to reduce complexity and become more agile. It would be advantageous for any monitoring system to be designed to be flexible enough to be modified to work with the new technologies when they are introduced.

DATABASE

The backend for WelshPAS uses the opensource SQL Firebird relational database management system [15]. It is a standard SQL relational database that stores all patient information.

Monitoring systems

The aim of this project was to develop a web application to track the connectivity of the SOAP webservices and XML drip-feed for the WelshPAS. This section will discuss the rationale behind monitoring software methodologies, especially those applied to cloud computing architectures, in published literature and the different approaches that may be employed to meet these requirements. It was not possible to identify any published literature that directly addressed the monitoring of interfaces for use in healthcare information systems in a search of the ACM Digital Library database using keyword combinations including **healthcare**, **interfaces**, **monitoring**, **FHIR** and related terms, and so it was necessary to identify work that was sufficiently similar as to be applied to this project.

BACKGROUND

Selecting a monitoring software architecture is highly dependent on the service being monitored, and as such, there have been many different approaches taken to create monitoring systems for webservices, cloud computing solutions such as IaaS and other systems that may be applicable to the problem. There are many commercial monitoring frameworks that are offered, including docker stat [16] and Amazon cloud watch [17] which can be used to monitor cloud applications, however these are often tied to a cloud-provider or virtualisation architecture and so will not be useful options for this project. Another problem with commercial monitoring frameworks is that often the metrics they monitor are general-purpose, for example CPU or memory usage [18], which may not be useful for specialised systems and applications such as the ones involved in this project.

In their analysis of different architectures for monitoring cloud computing infrastructures, Alcaraz-Calero and Gutiérrez-Aguado identified four different approaches that were commonly taken by commercial monitoring software. Local monitoring is where plugins are hosted and ran on a local machine where they extract metrics directly [19]. The other approach is remote monitoring where there is no monitoring software installed directly onto the host resources. Remote monitoring can be either agent-less or agent-based. Agentless monitoring uses port-scanning or TCP connections to extract metrics from a source remotely, with no software installed onto the resource. Agent-based approaches use software installed onto the system, which then extract metrics remotely. They can be either passive or active: passive monitoring opens a port and extracts metrics periodically or when triggered by an event; active monitoring extracts metrics in response to a request using plugins [19]. Monitoring architectures can also be internal (IMA), where the monitoring software is located on the internal network infrastructure, or external (EMA), where the software is located on a dedicated machine outside of the network infrastructure. EMAs are mostly applicable for cloud software solutions, for example where an external client is accessing public cloud services provided by a third-party host [19], and so will not be discussed in detail here, as it is clear from the project requirements that IMAs will be more suitable.

The next section will discuss how some of these methodologies have been employed in monitoring software solutions, as discovered during a review of relevant literature, and their value to this project.

RESEARCH

Some approaches to designing and implementing monitoring systems that were identified during a literature review will be discussed in this section and evaluated in terms of usefulness to the project problem.

Noor *et al.* developed an agent-based monitoring system for cloud applications in multi-cloud environments. The focus of this study was on monitoring metrics such as CPU and memory usage of containers within the cloud environment [18], so is only partially applicable to this project: it is possible to develop a similar application that could monitor these metrics for the WelshPAS interfaces, but there are different metrics that could be monitored that would be a more efficient and economical use of time to achieve the objectives of this project. These will be discussed later in this section.

Noor *et al.* deployed start-up packages onto every VM on their system across different cloud environments to register every call made to other services. This was advantageous as it allowed all of the different VMs and microservices metrics to be monitored in real-time [18]. However, Moeyersons *et al.* criticised this approach as being both intrusive and inefficient [20]. A similar approach applied to this project would be to deploy start-up packages onto every VM for all seven health boards. This would be an inappropriate solution for the problem posed in this report, as not only would it result in very high overheads, but it would also be time-consuming and complicated to deploy, requiring access to be granted and changes made to every VM in the system.

Moeyersons *et al.* improved on the work of Noor *et al.* by using sidecar containers in an agent-based system, which was found to be less intrusive, as it did not need to be installed onto every service, which also made it less resource intensive. It also has the benefit of not requiring access to each service in order to install packages on them, so can be applied to “black box” services [20]. Sidecar containers are an improved solution over that proposed by Noor *et al.* when applied to the monitoring problem: it may be possible to containerise the procedure using a

technology such as Docker [16], and this will be investigated in more depth in the Development methodology section. However, the approach taken in this study was still agent-based, which can be more resource intensive than agentless approaches [21].

Alcaraz-Calero and Gutiérrez-Aguado designed a complete set of monitoring systems for cloud computing systems that utilised a range of different approaches in order to compare their performance and effectiveness. These included both IMA and EMA, and agent-based and agentless methods, as well as some combinations of these approaches. After identifying the five most promising architectures that were broadly suitable for a range of scenarios for both providers and consumers of cloud computing systems, they were designed, implemented and then deployed into a real-world cloud computing architecture, and tested to determine the most secure and best performing solutions. IMAs performed better than EMAs in their experiments because they had lower overheads, however EMAs were found to be a more scalable solution. The most effective method in terms of performance was Extended and Adaptive IMA for both cloud provider and consumer. This method used both agent-based and agentless monitoring architectures to monitor the physical layer, virtualisation layer and message queue, and was useful for large deployments [19]. This approach is sophisticated but is likely to be too complex for this project, which will be a smaller deployment and has a tight timescale. However, monitoring of the messaging queue is a possible method of tracking the connectivity of the interfaces that could be applied to this project.

The study found that traditional IMA, which in this case was agent-based, was found to be very efficient in monitoring physical machines, but other traditional monitoring systems, usually agentless commercial solutions, were less useful in monitoring VMs in the cloud-computing infrastructure [19]. This is less of a concern for this project, as it will be used in a simpler architecture than the one being monitored in the study. A traditional IMA approach could therefore be considered.

RECOMMENDATIONS

It was difficult to find literature that related directly to software for use in healthcare records systems. However, there was a range of useful information found from studies of monitoring software with applications in cloud computing that was applicable to this project.

It was found that the most suitable approach for this project is a bespoke, remote, traditional IMA approach. The traditional IMA used in the study by Alcaraz-Calero and Gutiérrez-Aguado used an agent-based approach [19], however, like the software developed by Noor *et al.* and Moeyersons *et al.* this may prove to be both intrusive [18, 20] and more complex to develop for this project as it would require installing agents onto the resources for each health board. An agentless approach would be simpler to develop and have a lesser impact on the system performance [21].

Monitoring the messaging queue [19] is an interesting avenue to explore during the design stage, as it could be applied to this project.

A limitation of this literature review was that the metrics being monitored in all studies were the physical performance of servers and VMs (e.g. CPU usage, memory, etc.), which is not directly applicable for this project, as the physical performance of servers or VMs are not controlled by the integration team: this is the remit of other departments in the organisation. It would be beneficial to identify and monitor a metric that was specific to the interface only, however it may be the case that this metric is impacted by other hardware performance factors that are not explicitly highlighted by the software. This should be considered when designing the software.

CHAPTER 3: DEVELOPMENT METHODOLOGY

This section will discuss the development methodology that was chosen for this project and the justification for selecting it. It will summarise the key points that were identified in the literature review, and how these were used to inform the design of the application. The software and development materials that are needed for this project are explained and discussed.

Methodology

There are many development methodologies that are used by software developers, which each have their own advantages and disadvantages. The methodology chosen by DHCW and NHS Wales for use across the organisation is Agile [14], an iterative, incremental process which uses short “sprints” of development, interspersed with “scrums” to track the progress of the team. The aim of the methodology is to create a working product at the end of each phase, which can be presented to stakeholders who can evaluate the product and request changes, even late into the development cycle. The aim is to prioritise rapid development of a functional product that is acceptable to the end users, rather than time-consuming planning and documentation. It was developed in response to popular methodologies at the time, such as Waterfall, which were criticised as being inflexible and slow to react to changing requirements [22].

However, the Agile process was not the most appropriate methodology for the development of this project, where there was a single developer assigned to develop a small application, as it was not possible to divide the task between team members or hold scrums. The scale of the project and short timescales meant that it was likely that the project would be completed within one or two sprints, and it would be a poor use of organisational time to dedicate all potential users of the software to attending regular evaluations, particularly for a project such as this one where it was very unlikely that the requirements would change. Instead, Rapid Application Development (RAD) [23] was used.

RAD is a rapid development methodology with an incremental approach, like Agile and similar approaches such as Extreme Programming, but instead focusses on prototyping. RAD is especially useful for projects with simple requirements and low-complexity development, and therefore seems an ideal methodology for this project which has a limited scope, short-timescale,

single developer and a simple aim. It is also useful for programs where the user interface is a preliminary requirement [24], which is also a strong consideration for this project. RAD uses four phases for development: joint requirements planning (JRP), Joint Application Design (JAD), construction and cut over [25]. These phases are closely aligned with the design requirements envisioned for this project.

Architecture

In the previous section it was identified that an internal monitoring architecture (IMA) was most appropriate for this project. It was shown that EMAs do offer improved security over IMAs, but this was more of a consideration for software that would be accessed by a third-party client over the cloud. Although security is a strong consideration for this project, the software will only be used and accessible by members of the integration team or senior management within the organisation. EMAs also had a reduced performance over IMAs, which is a disadvantage for this project, as it is important that the software solution will have a minimal effect on the important system being monitored.

The Moeyersons *et al.* used an IMA approach that used sidecar containers, and this was identified as a possible solution. Docker is a commonly used technology that could containerise the software. However, the approach was agent-based which was determined to be less desirable for this project.

Alcaraz-Calero and Gutiérrez-Aguado used monitoring of the messaging system in their approach to implementing monitoring architectures for cloud computing systems. This is a possible useful approach for this project, as monitoring the messaging queues that send the XML files would meet the objective of tracking the connectivity of the interfaces.

They identified extended and adaptive internal monitoring architecture as being the best performing architecture in terms of performance. This approach was sophisticated and interesting, however would be far too complex to implement within the timescales required for this project. Traditional IMA was simpler, but also performed well in the experiment, and an agentless IMA would meet the objectives of this project. Therefore, the software is intended to be a remote, agentless traditional IMA.

Software

Selecting an appropriate tech stack for use with this project is fundamental as it will affect every part of the design. Within DHCW, there were three realistic options that could be used in this project, which will be discussed in the next sections.

DELPHI

WelshPAS and the interfaces themselves have been written with Delphi. Delphi is an object-oriented programming (OOP) language that was based on Borland's turbo pascal for Windows, designed by Anders Hejlsberg [26], who went onto develop C# for Microsoft. Use and popularity of Delphi has declined according to the TIOBE index, falling from number six in the year 2001, to the thirtieth in July 2020 [27] (although it has recovered to eleventh place as of April 2023 [28]).

This programming language offers itself as a RAD environment due to its inbuilt support for prototyping and rapid UI [29]. This would be a benefit for this project, as it will use a RAD methodology. Additionally, it may be beneficial to program the software using a language that is in common use in the organisation and therefore will be well-supported by other team members for the future.

.NET

.NET is an opensource web application framework created by Microsoft in 2002 [30]. It is used throughout the NHS Wales and DHCW for many different applications, albeit not on the WelshPAS or WelshPAS integration team. The framework could certainly be used to develop the application, but it would not be the optimum choice as has been suggested that .NET is better suited to larger projects where the ability to scale up is important [30]. This is not the case for this application, and so other options should be considered. In addition, the limited use of .NET within the team that will be using and supporting the software in the future is disadvantageous and will limit the opportunity to collaborate if there are any major roadblocks encountered during the development process.

NODE.JS

NodeJS/JavaScript was chosen as the main development environment. Initially released in 2009, NodeJS was built upon the Google innovation of the V8 engine [31], the main benefit of which was allowing JavaScript to be used as a server-side programming language [32]. Being both cross platform and open source offers advantages for this project, for example, the ability to use JavaScript both server and client side is desirable. However, arguably the biggest benefit of using node is the asynchronous non-blocking event loop, which allows connection and retrieval of data without being delayed by other connections [33]. This approach also allows for the software to be scaled up without drastically affecting the speed of the application, for example, if further requirements are added for additional monitoring in the future, which is currently a real possibility given the work in progress by the NDR [14]. The implementation of this asynchronous approach in NodeJS is much less complex than making a .NET multithreaded application. In addition, NodeJS has previously been used in FHIR API work in a healthcare records system and has been shown to be efficient and stable [34]. The disadvantage will be that it will be poorly supported by the team, as there have been no practical applications of NodeJS within WelshPAS or WelshPAS integration to my knowledge.

Packages

Before starting development, it was essential to determine if the chosen environment had tools available that could meet the objectives of the project. This was particularly important as the project was tightly constrained by time. Libraries, modules and packages were used in the development of the project to enable development time to be used efficiently in meeting the specialised requirements of the project, and not “reinventing the wheel”. Below is some of the key modules that were identified and employed in the application. The rationale for their use will be discussed in more detail in Chapter 4: Design and implementation.

NODE-FIREBIRD

As mentioned in the Database section of the report, WelshPAS uses the Firebird database engine. **The Pure JavaScript and Asynchronous Firebird client for Node.js** [35]

provides asynchronous methods to obtain data necessary for monitoring. This allows for the data to be changed without needing to stop the program and update the connection strings.

CRYPTO

The NodeJS inbuilt **Crypto** module was necessary to apply AES-256 encryption and decryption to database passwords. **IV** and **Key** were used to encrypt the database passwords within the module, before storing them in the central database repository. When the password fields are pulled down, they are run through the decryption function and added to the relevant database connection object. It was important to use a 2-way encryption method such as AES, because other commonly used methods for password storage in databases, such as hashing/salting, are not reversible [36].

EXPRESS

The **Express** framework [37] was used as the application middleware for routing. The **app.get()** method was used for routing the data objects for the ajax webservice to call by wrapping the functions that retrieve the database and webservice data. The express framework was used as it is asynchronous and time efficient for implementing the REST API calls. It is also easy to understand as it negates the need to understand lower-level protocols [38].

EASY-SOAP-REQUEST

To achieve the objectives for this project, the **easy-soap-request** library was used. The package simplifies SOAP requests by delivering methods that allow NodeJS to call and retrieve data from an XML based SOAP web service. It is essentially a wrapper of **Axios**, discussed below.

AXIOS/AJAX/JQUERY

A combination of **axios** [39], **ajax** [40] and **jQuery** [41] was used to deliver the content from the server to the client, which can then dynamically update the front end. The REST calls are built using **jQuery** using **Ajax** to retrieve the data via the **express** middleware. The function calls for

the key data are all made from the client in the form of SOAP **get** requests. The use of **ajax** is essential so that data on the webpage is updated without the requirement to refresh the page. As soon as the data is retrieved from the server, the html elements are instantly updated, as set out in the jQuery code.

NODE MAILER

This package was included to meet an additional requirement that was introduced later in the development cycle (discussed in the Implementation section). It triggers email alerting [42], should values pulled down from database or SOAP requests meet a certain condition. This package is also used if a function throws an error, for example, if a web service is not responding and timing-out.

DOCKER

Docker could be run as a microservice, to allow all images to run in one container. The benefit of using Docker to containerise an application is that it makes is easier to run on other systems. It is a more efficient method than the alternative of running the application as a VM: spinning up a VM is more time- and resource-intensive than using Docker [16].

The next chapter will discuss the design of the application.

CHAPTER 4: DESIGN AND IMPLEMENTATION

Design

This section discusses the design decisions that were made, and the factors that were analysed when considering how the program should function.

PROJECT REQUIREMENTS

The project requirements were derived from a gap analysis on the ability of the team to react proactively in the event of an outage or downtime to an interface or one of its hosts. As discussed in the Background section of the report, the team may only discover an interface has gone offline when the team receives a Servicepoint ticket reporting that there is an issue. It was determined that if the team had a tool that monitored these APIs and displayed it to a dashboard, the time to resolution for incidents specifically relating to the interfaces could be greatly reduced. The requirements of the application were identified from this gap analysis and used to create an appropriate design.

The main requirements of the application are:

- The project will follow a RAD development methodology
- The application will be a remote, agentless IMA
- The project will use NodeJS and Javascript
- The application will have minimal impact on the performance or functionality of WelshPAS and associated interfaces
- Database passwords will be handled securely using AES-256 encryption
- Must be approved by the SIT and UAT teams, and sent to the Change Management Board before the live application is rolled out
- The application should be designed to be able to accommodate the future plans of the organisation.

An important design element that was determined during the requirements stage was what method or metrics should be used to monitor the interfaces. Monitoring the messaging queue, as

implemented by Alcaraz-Calero and Gutiérrez-Aguado [19], was rejected due to the perceived complexity, and because of potential future changes to the system as the organisation moves to REST APIs. A decision was made to instead monitor the database table that generates and sends the data that gets transformed to XML before being sent downstream. The rationale for this is that if there is a large number of records sitting in the database table, it signals that there is either a potential problem with the interface that is pulling the records off, or that a health board user has ran a very large change against PAS. An example of where this might happen would be if an authorised user from a health board were to change multiple GP records, then this would then touch all the associated patients which would subsequently generate a large number of transactions in the table. It is important that this increased traffic generates an alert, as it will enable the team to be proactive in resolving any issue that is affecting the interface (e.g. a crash, host server is down) as soon as possible. Additionally, it will allow downstream systems to be warned if there is an imminent influx of XML files which may affect the downstream flow. It is a simple metric to monitor, which is beneficial for this project. An additional benefit of this approach is that this table can be monitored even if there are other changes to the design of the interfaces, as is currently planned for the future: there will be minimal changes needed to the monitoring software to adapt it.

The next section contains diagrams and prototypes that were delivered during the Joint Planning Phase.

PROTOTYPE

Wireframe diagram

A wireframe diagram was created and approved during the Joint Planning Phase to demonstrate the intended architecture of the application.

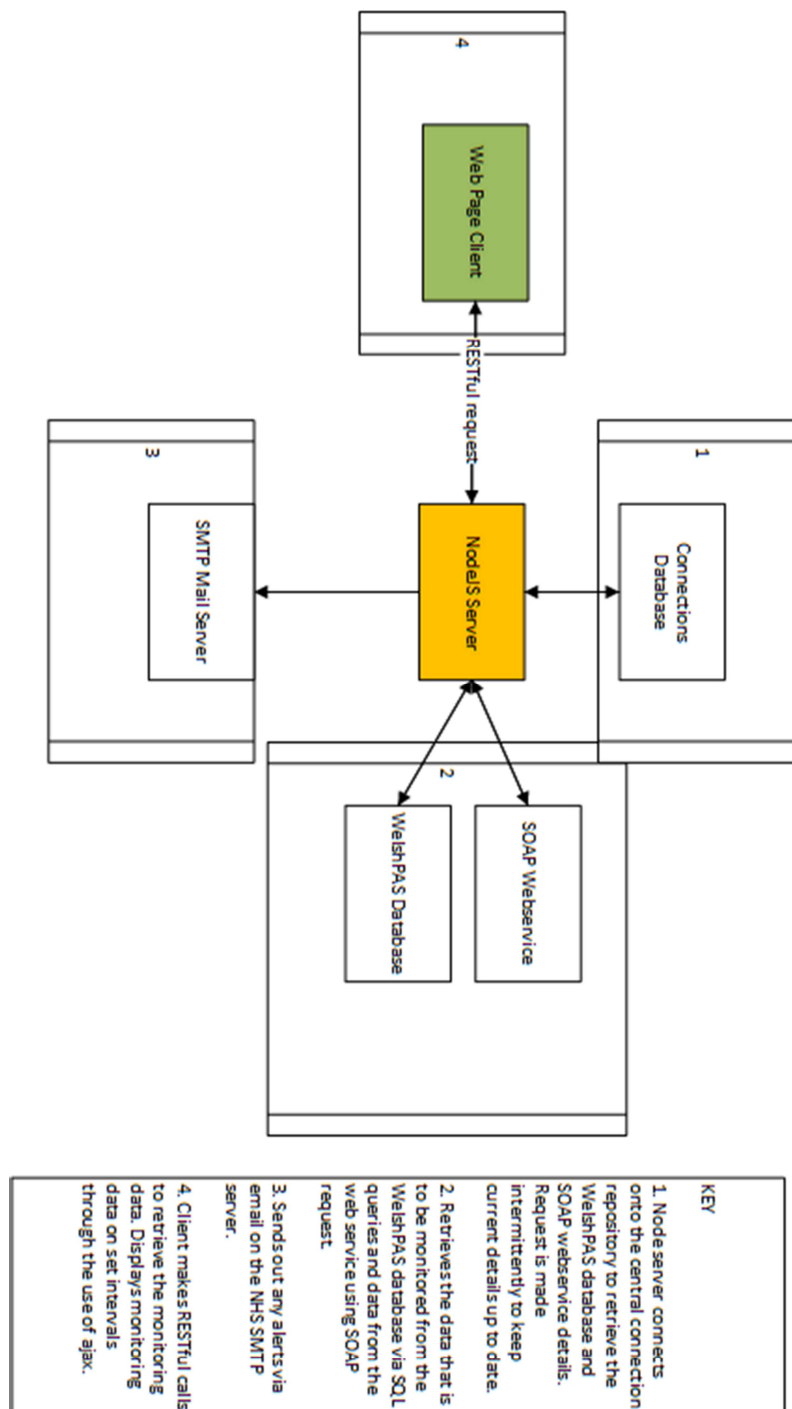


Figure 1: Wireframe diagram

Prototype UI

A basic prototype user interface (UI) was created using wireframe.cc [43] to demonstrate how the application will be displayed in the web browser. The aim was to have all information displayed on a single page, so that all required information is available “at a glance”. The prototype design was approved during the Joint Planning Phase.

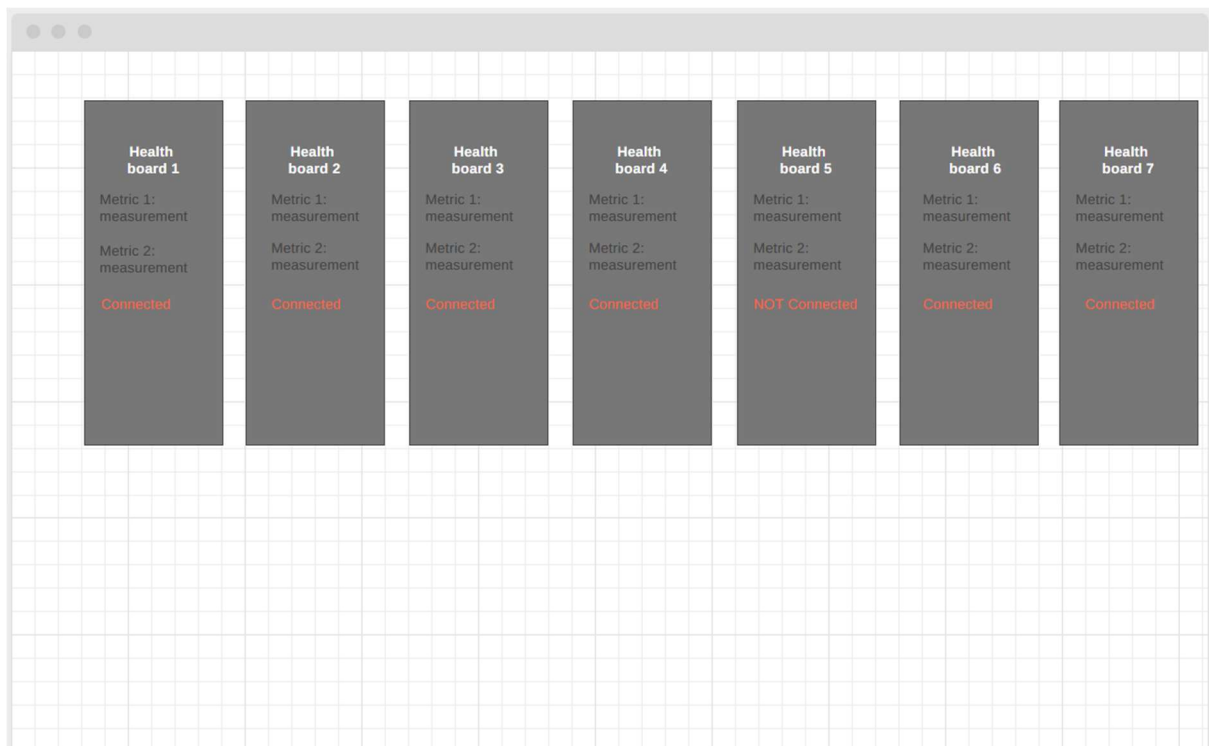


Figure 2: Prototype UI

Implementation

Using JavaScript and NodeJS, it was possible to handle the event loop of using asynchronous functions and call backs where required. The RESTful framework was used for making the API calls from the client to the server with the help of **express** middleware routing. Tools from the basic JavaScript library were used to parse and slice strings, which was particularly useful for the SOAP calls where the returned XML data needed to be formatted for display on the web page.

An important part of the implementation was to ensure that each route for the data calls were separated into their own units. **Count** and **SyncDate** for the database functions were kept separate to ensure that their timings could be controlled. This was considered to be important because a scenario could be foreseen where, for example, a decision by management was made to update the webpage every 5 seconds using **Count** but set the **SyncDate** update for every 60 seconds. This would be a simple change when these are kept separate, as it is possible to control that by passing the function calls into the **setInterval** Arguments. Another possible reason could be if it was decided that a very busy health board, for example Swansea Bay University Health Board, needed to report faster than a very quiet one such as Powys, the same change can be made.

The application mainly uses JSON, but also uses XML for the SOAP calls. As mentioned previously, discussion with the WelshPAS integration team established that the database table that generates and sends the data that gets transformed to XML before being sent downstream should be monitored. These metrics can indicate either a problem with the interface that is pulling the records off, or that an action has occurred that generated a very large number of transactions in the table. An alert should be generated to warn of any imminent influx of XML files which may affect the ability of the downstream applications to handle the data.

The **Pure JavaScript and Asynchronous Firebird client for Node.js**, discussed in the previous section, provides asynchronous methods to achieve this by pulling down **SELECT** statements to get the data required for monitoring. The original plan for the application was to store the connection details to be used in a config file. If the connection details needed to be changed, a user could enter the config file and update the connection object details being used by

the software. However, this would mean that the software would need to be restarted manually anytime the active site for WelshPAS changed. This could be problematic, as not only would users need to understand how to access and amend the config file, but they would also need to remember to do this every time it was changed. It was recognised during review of an iteration that this was not ideal, and so the concept of using a centralised database table that stores all pas active and inactive connections was discussed. This centralised database would have useful applications elsewhere in the organisation, and as such, a request was made to bring the introduction of this database forward. The NodeJS was a successful demonstration of the usefulness of the centralised database approach for the organisation. As additional monitoring software is planned to be introduced over the next 12 months for other non-interface aspects of WelshPAS, this suggestion was approved and will be implemented in production. This too is reliant on users to ensure that the table is updated whenever WelshPAS failover, but as the centralised database will impact more than just this project, it will be easier to enforce and regulate the updates.

A function was created that runs an SQL query against the new centralised connection table on a timer and pulls down the most up to date connection details in the repository. With this improved method, the program does not need to be stopped in order to update connection objects. It also means that support staff on the team are able to log onto the database and change details, rather than requiring developers to modify config files: a more beneficial use of organisational resources in the long run. However, it was important to be cognisant of the possibility of losing connection to the central connection repository, for example due to the host failing, and as a contingency measure the objects were also stored in a config file to be manually entered in case of serious failures to ensure continuity.

Initially, the plan was to display major issues via the client side using window alerts. For example, if records were over 50,000 for a health board, a windows alert would be displayed using the **ajax/jQuery** functions. The downside of this approach is that a browser alert halts the event loop until it is closed. This is not ideal, as occasions where message count is high but not in need of intervention are foreseeable. Moreover, the alert would flag repeatedly until there was a reduction in the message count. As such, it was suggested that window alerts were reserved for major failures only. An example of where a windows alert could be used would be where

results do not return from the rest calls because the database is offline. This would be a catastrophic problem for WelshPAS and considered a major incident under ITIL incident management criteria [44]. If this scenario were to occur, the service would be failed over to the secondary database instantly. WelshPAS operates using server pairs with replication, so the alert in this application would be cleared as soon as the secondary database is updated in the central database repository. Given this, most of the logic for the checks on data (e.g. **if message count >50000 do X**) was moved to the server side to stop any interruption to the event/loop dashboard unless there is a catastrophic failure.

Additionally, during a review it was decided that email alerts would be added as an extra requirement for the application. Outbound email alerts were implemented using the NHS SMTP server, so that alerts will be delivered without interrupting the program. A recommendation will be made to the mail list recipients to set up a separate mail folder to filter for these alerts only, so that they are not lost amidst a cluttered inbox.

Access to ubuntu virtual machines is not an option for the WelshPAS department. As NodeJS is cross platform, it was essential to consider whether to simply build on windows or instead use Windows Subsystem for Linux (WSL2). Microsoft itself recommends that JavaScript-based frameworks are better suited to WSL as it currently supports better performance speeds, system call compatibility and alignment between development and deployment environments [45], and this therefore informed the decision to use WSL2. Additionally, during the joint planning phase, use of a docker container to run the application was considered, which is a scenario where use of WSL2 is also beneficial [45].

Another essential requirement of the project was to ensure that database passwords were treated securely by the application. The in-built **crypto** module was simple to use to apply AES-256 encryption and decryption for the database passwords. This was one of the most straightforward tasks to accomplish during implementation.

PROBLEMS AND SUCCESSES

One of the biggest issues encountered during implementation of the application was setting up the development environment onto the work laptop. NHS Wales uses a company proxy, which resulted in numerous issues such as SSL certificate errors, download restrictions when

downloading packages from the NPM store and issues pulling down the schema. Some fixes were completed by changing the npm config registry and setting a npm config proxy. Issues with account permissions were also encountered, which were referred to the service desk and rectified.

The major problem that was encountered during the planning stage was to identify a way of tracking that the XML drip-feed interface had stopped. As discussed previously, the drip-feed pulls records from a database table that is populated every time there is a change on WelshPAS that needs to go to a downstream system. A solution that was developed was using the records field called **sync_date**. This field is updated each time a record is touched, so in relation to the table being monitored, this only happens when the record is written to the table as the table has no write access for any users to change it. By writing the SQL query to select the first record from the table after ordering by ascending, it would mean that the oldest record (which is always pulled first from the table by the feed) would always be the oldest record. Each time a record is pulled by the interface, a new record becomes the oldest and should have an older **sync_date**. Each time the function runs this query, the sync date is saved to the relevant health board object property. This **sync_date** is now stored in memory by the program. The next iteration should pull back a different **sync_date**, which indicates that the interface is working because records are going out as intended. If, however, the **sync_date** on subsequent iterations does not change, then this may suggest that the interface is not pulling down the records and must be inspected. As the interface processes over 100 messages every few seconds, it would be very rare that there was a period of more than a few seconds with the same **sync_date**. If the RESTful call from the client to the function that is pulling down the data is running every 30 seconds, it would be very unusual if the **sync_date** remained the same and the interface was working as intended. This was a successful solution that addressed the aims of the project and was an essential factor in the application functionality.

It was not determined that the central connection repository would be rolled out and used for pulling down the connection details for the databases and webservice until late in the project. This meant that my use of EJS templating had become redundant. EJS was being used to push some of the data from the health boards directly to the front end without going through the REST calls Ajax. EJS allowed the data to be updated on the front end every time the software was

stopped, and a different connection detail was added, e.g. from **NDC dater enter** to **CDC dater enter**. EJS was also used to write the SOAP web service URL that was being used. When the need to manually stop the program and update config details was removed with the creation of the central repository, these EJS entries needed to be changed to also go through a SOAP call, so that ajax could update the webpage dynamically as the software would remain running. To do this, they were incorporated into the existing **DBcount** and **DBconninfo** functions as part of a JSON object with the existing data. The client-side code for the ajax REST calls needed to change slightly to handle the extra data in response, but despite the extra work required, it was a fast refactor and a positive change that will improve the maintainability of the application when it is released.

A potential problem that exists with the application currently is that when the software first runs, it must first complete the function that gets all the connection details. This function will then update the connection objects with the data that are pulled down. As the Nodejs event loop is asynchronous, if this is not handled correctly, it is possible for the client REST calls to attempt to pull back the monitoring data before each of their connection details objects have been populated. This is not a “showstopper” bug [46] as it only has the potential to happen when the software first starts up. The REST calls are all made client-side with a timer delay so will not kick in until the web browser needs serving as a temporary solution. This issue has only occurred a handful of times during start-up, and future testing will need to be performed when running from IIS. This is not anticipated to be a major issue as the code has been written so that it should very rarely need to be restarted (e.g. pulling down the new connections while software is running). In addition, there will only be a small number of users, all of whom are technically adept, the risk of the problem occurring is low, and the impact of the bug is anticipated to be negligible.

RESULT

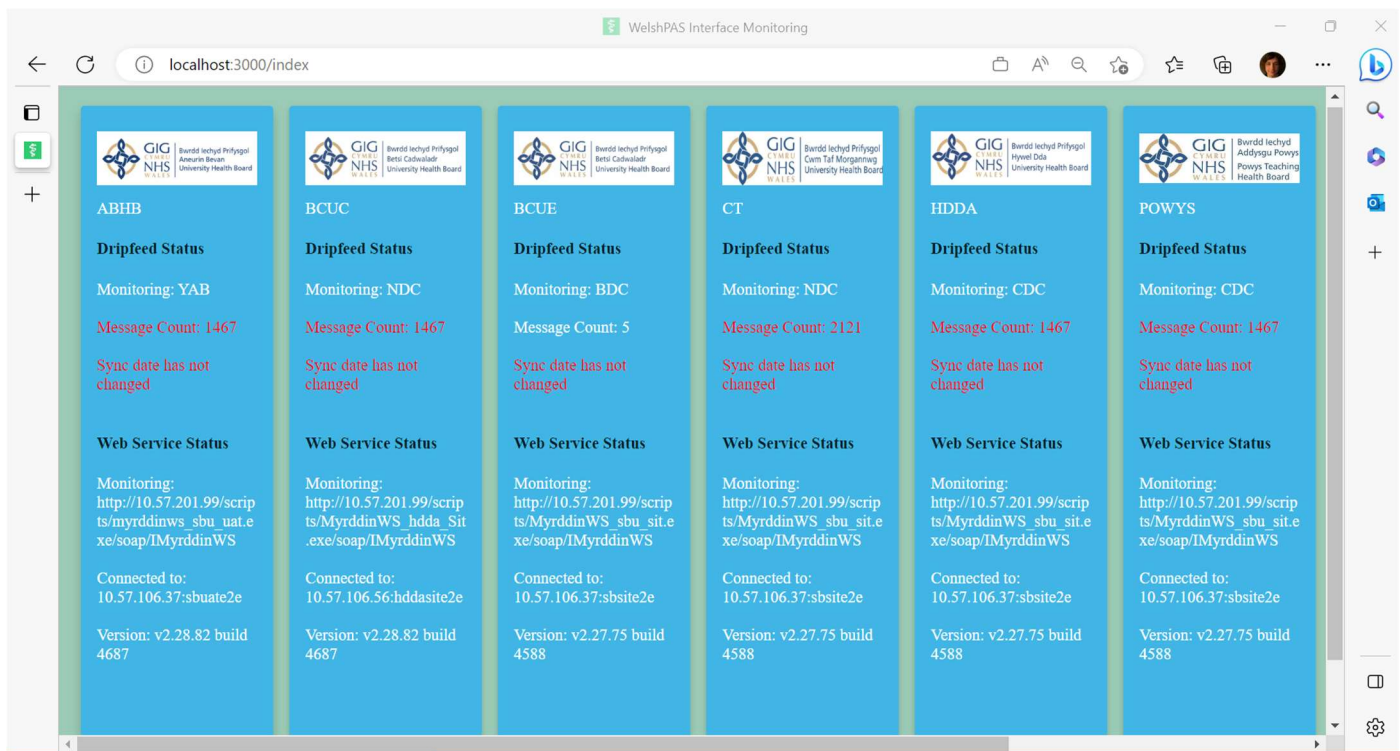


Figure 3: User interface of monitoring application

The above figure depicts the final application web application user interface, which is successfully connected to the seven health boards and their drip-feeds and Webservices. The message count and sync dates are successfully tracked, and information about the webservices is visible.

The next section will evaluate the development process and whether the application met the objectives of the project.

CHAPTER 5: EVALUATION

The application meets the requirements and objectives of this project. The application is a basic, remote agentless monitoring system, that uses the **sync_date** field of the database tables to track the functionality of both the webservices and drip-feeds of the seven health boards. A benefit of using this approach is that it would be possible to adapt the software in the future, for example when the interfaces move from SOAP to RESTful open API. However, a number of interesting, more sophisticated approaches to monitoring were found using a literature review that may be useful for future development of this or similar projects.

The successful use of NodeJS was particularly noteworthy for the team, as most development work is usually performed in Delphi. Proof of successful development using a different technology stack will encourage the adoption and future use of this technology for future projects.

Evaluation took place concurrently with the user design and construction phase, as is prescribed by the RAD methodology. RAD was a useful framework as there were some unanticipated changes and extra requirements that were identified during the construction phase and were responded to. A different, more heavyweight methodology such as waterfall may have resulted in a less refined final product and extended the development time. Any flexible, rapid development methodology would probably have a similar outcome.

Evaluation by the team identified areas of improvement over the design as originally envisaged. One of these areas was to add the functionality to send automatic email alerts in response to conditions that suggest a major incident. The other, more unexpected consequence of the project was the fast-tracking of the centralised connection database to be implemented for use across the WelshPAS team, and to replace the need for manual changes to config files in this application. The use of the monitoring software in this project as a demonstration of the benefits of this change that could be echoed across the department was a surprising and positive outcome.

The software has run for a trial of 12 continuous hours before it was manually stopped, and no errors were identified. The specification set by management has been met, and although some bugs have been noted, functionality of the application appears to be good. It is anticipated that the eventual deployment of the application will introduce a few issues initially, as while the SIT

and UAT databases are all hosted over two datacentres all on the same DHCW network, some health boards host their own servers. This means that firewall requests will need to be raised with the health boards before it can go into production. Additionally, it is intended that IIS will be used to host the application, and although the use of Docker has not been conclusively rejected by the organisation as yet, it is likely that IIS will be used. When the IIS is set up, the URL host name will be changed from the default. Once hosting is complete and final approval has been granted by management, the application will proceed to the next stage of development: the testing team will roll out the application onto the test environment.

During the evaluation by users for the next iteration, the suggestion was made that the timings could be controlled using environmental variables instead. A benefit of this would be that it would not be necessary to amend the code every time the length of one of the timers needed adjusting.

CHAPTER 6: CONCLUSIONS AND RECOMMENDATIONS

It is essential that different health care applications can pull records from and send updates to health care records systems, such as those used in WelshPAS, seamlessly to ensure that patient information can be used to support patients and provide the best possible care. The WelshPAS is part of a healthcare records system in NHS Wales and uses a number of different interfaces to achieve interoperability with the health care applications that use it.

A need was identified by the WelshPAS integration team for a way to track the connectivity of the SOAP webservice and XML drip-feed for all servers and health boards across Wales, to enable a more proactive approach to identifying and resolving problems.

A literature review was performed and identified that a remote, traditional IMA with an agentless approach appeared to be most suitable for the project, however the applicability of the research to this project was less than was hoped. The RAD methodology was identified as most appropriate for the project.

A design was created, and consideration was made to the future organisation changes expected, such as the adoption of open API and FHIR. The department decided to track the `sync_date` field of the database for changes that would indicate possible problems in the webservice or drip-feed. The project was implemented using NodeJS due to its support for asynchronous non-blocking event loops. This was the first project on the WelshPAS integration team to use this technology.

During development, some areas for improvement were identified from user feedback, including a request for email alerting. This functionality was introduced using the NHS SMTP. It was also identified that the creation of a centralised database of connection strings would be highly desirable for use in this project and beyond, and the introduction of the database was fast-tracked by WelshPAS as a result. The code was refactored to use the database rather than config files, but the ability to use config files was left in as a contingency in case of future problems.

Some avenues for further investigation have been identified, that would either improve the application, or inform the approach taken to similar work in the future.

An agent-based approach could be investigated for use in the system, or alternatively a combination approach such as that taken by Alcaraz-Calero and Gutiérrez-Aguado [19]. Further

work to identify other metrics that could be used within WelshPAS for systems and application could be performed and the functionality of the application expanded.

If time was less of a constraint, experimentation with a frontend framework such as Vue or React [47] may have improved the application. A good use for these technologies might be to develop a graphical representation of the data being monitored in the form of a dynamic graph display.

A limitation of this project was that there was little formal test schedule, and that most testing of the software will be passed to the testing team. Future projects should investigate an appropriate test schedule for the project.

CHAPTER 7: REFLECTION

During this project, I was fortunate to have a very clear goal and objective for the project from the outset. This meant that it was easy to focus on the design and implementation of the application without being side-tracked by “dead-ends”, and I could use my time for this project very efficiently.

Although I was disappointed that some of my more complex suggestions for the project that I found to be quite interesting were passed over by the department during the requirement discovery and planning stage, it meant that the final application that resulted met all the needs of the users, which ultimately is the most important factor. Using RAD to constantly develop and review the application with the users was a useful approach, which allowed me to prioritise their needs over my personal expectations for the software, and receiving productive feedback improved the software functionality in the end. It was easier to do any refactoring while still in the development stage, and not further down the line when things were more set in stone.

As such, this project has made me realise that development is a team effort, even when you are the sole developer on a project. I was particularly pleased that I was able to use NodeJS, a first for this team, and demonstrate that it could be used successfully in a project. This may offer the opportunity for colleagues in WelshPAS to use the technology more widely, which could be beneficial due to some of the advantages offered by NodeJS over some older technologies. Another unexpected consequence was that the centralised connections database was moved forward, which I hope will also benefit the department and reduce their time spent when the connections are moved and need to be updated.

During this project I encountered many difficulties that were not accounted for. One of the first problems I found was that setting up the development environment in the highly secure and restrictive NHS environment. With time and patience, this was resolved, and I am now aware of how to solve many common roadblocks that might occur in similar situations in the future.

One area where I would have liked to have spent more time was in the testing phase. Although there is a dedicated test team, and I challenged my own code informally, I would have liked to have performed unit testing on the software. This would not have been a good use of

organisational time however, and due to the time constraints, the testing of the software was deprioritised.

In conclusion, this project was both challenging and rewarding, and allowed me to apply my knowledge in a new way. I was able to demonstrate my ability to plan, collaborate and problem-solve, which resulted in a functional piece of software that will become widely used in NHS Wales WelshPAS integration.

CHAPTER 8: REFERENCES

- [1] B. M. H. K. G. G. & P. B. Suranga N. Kasthurirathne, “Enabling Better Interoperability for HealthCare: Lessons in Developing a Standards Based Application Programming Interface for Electronic Medical Record Systems,” *Journal of Medical Systems*, vol. 39, p. 182, 2015.
- [2] M. v. G. C. L. C. L. P. M. A. Ö. & G. S. Timo Jakobi, “The Role of IS in the Conflicting Interests Regarding GDPR,” *Business & Information Systems Engineering*, vol. 62, pp. 261-272, 2020.
- [3] T. B. & G. Grieve, *Principles of Health Interoperability: FHIR, HL7 and SNOMED CT (Fourth Edition)*, Cham, Switzerland: Springer Nature, 2021.
- [4] A. T. M. Van Steen, *Distributed Systems: Principles and Paradigms (Second Edition)*, Upper Saddle River, NJ: Prentice Hall, 2006.
- [5] J. Coenders, “Interoperability in the Networked Design infrastructure,” Delft University of Technology, Delft, The Netherlands, 2012.
- [6] A. B. N. G. P. G. V. I. V. N. M. P. Gordon S. Blair, “The Role of Ontologies in Emergent Middleware: Supporting Interoperability in Complex Distributed Systems,” *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 410-430, 2011.
- [7] M. P. R. & Y. B. Chhetri, “Recommending reference API documentation,” *Empirical software engineering*, vol. 20, no. 6, pp. 1558-1586, 2015.
- [8] NHS Digital, “Our API technologies,” NHS Digital, 25 11 2022. [Online]. Available: <https://digital.nhs.uk/developer/guides-and-documentation/our-api-technologies>. [Accessed 22 03 2023].
- [9] NHS Wales Digital Health and Care Wales, “APIs, standards and terminology,” NHS Wales Digital Health and Care Wales, [Online]. Available: <https://dhcw.nhs.wales/information->

services/health-intelligence/apis-interoperability-standards-and-terminology-services/.

[Accessed 22 03 2023].

- [10] J. Davidse, "API Fundamentals," IBM Developer, 16 12 2020. [Online]. Available: <https://developer.ibm.com/articles/api-fundamentals/>. [Accessed 24 03 2023].
- [11] D. E. G. K. A. L. N. M. H. F. N. S. T. & D. W. Don Box, "Simple Object Access Protocol (SOAP) 1.1," in *W3C Note 08 May 2000*, 2000.
- [12] M. F. Z. J. M. Z. S. A. Ali Mushtaq, "Mobile Cloud Computing with SOAP and REST Web Services," *Journal of Physics: Conference Series*, vol. 1018, no. 1, 2018.
- [13] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures: Doctoral dissertation," University of California, Irvine, 2000.
- [14] National Data Resource, NHS Wales, "NDR Data Strategy (For Wider Distribution) Version 1.0," February 2022. [Online]. Available: <https://dhcw.nhs.wales/files/national-data-resource1/national-data-resource-data-strategy/>. [Accessed 23 03 2023].
- [15] Firebird Project, "Firebird," Firebird Foundation Inc., 27 03 2023. [Online]. Available: <https://firebirdsql.org/>. [Accessed 01 04 2023].
- [16] Docker, "docker stats," Docker, 01 2023. [Online]. Available: <https://docs.docker.com/engine/reference/commandline/stats/>. [Accessed 25 03 2023].
- [17] aws, "Amazon CloudWatch," Amazon, 2023. [Online]. Available: <https://aws.amazon.com/cloudwatch/>. [Accessed 25 03 2023].
- [18] D. J. K. M. e. a. A. Noor, "A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environments," *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 156-163, 2019.
- [19] J. G. A. Jose M. Alcaraz Calero, "Comparative analysis of architectures for monitoring cloud computing infrastructures," *Future Generation Computer Systems*, vol. 47, pp. 16-30, 2015.

- [20] S. K. T. W. F. D. T. & B. V. Jerico Moeyersons, "Towards cloud-based unobtrusive monitoring in remote multi-vendor environments," *Software: Practice and Experience*, vol. 52, pp. 427-442, 2022.
- [21] W. Richter, "Agentless Cloud-wide Monitoring of Virtual Disk State (Doctoral Thesis)," *School of Computer Science, Carnegie Mellon University, Pittsburgh, PA*, 2015.
- [22] J. G. R. C. M. M. B. J. H. S. M. A. v. B. A. H. K. S. A. C. R. J. J. S. W. C. J. K. D. T. M. F. B. M. Kent Beck, "Manifesto for Agile Software Development," Agile Alliance, 2001. [Online]. Available: <http://agilemanifesto.org/>. [Accessed 26 03 2023].
- [23] J. Martin, *Rapid Application Development*, New York: Macmillan Publishing Company, 2001.
- [24] C. C. P. B.-D. & D. T. Hugh Mackay, "Reconfiguring the User: Using Rapid Application Development," *Social Studies of Science*, vol. 30, no. 5, pp. 2737-757, 2000.
- [25] A. v. d. M. & R. A. Aurona Gerber, "Practical Implications of Rapid Development," in *Computer Science and IT Education Conference*, Mauritius, November 2007.
- [26] Embarcadero Inc., "25 Years of Delphi and History of Pascal," Embarcadero Inc., 2020. [Online]. Available: <https://delphi.embarcadero.com/25-years-of-delphi-and-history-of-pascal/>. [Accessed 15 03 2023].
- [27] TIOBE Index, "The Delphi/Object Pascal Programming Language," TIOBE, 01 04 2023. [Online]. Available: <https://www.tiobe.com/tiobe-index/delphi-object-pascal/>. [Accessed 01 04 2023].
- [28] TIOBE Index, "TIOBE Index for April 2023," TIOBE, 01 04 2023. [Online]. Available: <https://www.tiobe.com/tiobe-index/>. [Accessed 01 04 2023].
- [29] Embarcadero Inc., "Delphi 11.3," Embarcadero Inc., 2023. [Online]. Available: <https://www.embarcadero.com/products/delphi>. [Accessed 12 03 2023].

- [30] M. Shapel, “.NET vs. Node.js: What to Choose in 2023,” SaM Solutions, 28 03 2023.
[Online]. Available: <https://www.sam-solutions.com/blog/net-vs-node-js/>. [Accessed 06 04 2023].
- [31] V8, “What is V8?,” V8, 2023. [Online]. Available: <https://v8.dev/>. [Accessed 12 04 2023].
- [32] node.js, “The V8 JavaScript Engine,” OpenJS Foundation, 2023. [Online]. Available: <https://nodejs.dev/en/learn/the-v8-javascript-engine/>. [Accessed 12 04 2023].
- [33] node.js, “Don't Block the Event Loop (or the Worker Pool),” OpenJS Foundation, 2023.
[Online]. Available: <https://nodejs.org/en/docs/guides/dont-block-the-event-loop>.
[Accessed 11 04 2023].
- [34] F. P.-W. J. C. M. K. D. M. Richard A. Bloomfield Jr, “Opening the Duke electronic health record to apps: Implementing SMART on FHIR,” *International Journal of Medical Informatics*, vol. 99, pp. 1-10, 2017.
- [35] npm, Inc, “Pure JavaScript and Asynchronous Firebird client for Node.js,” npm, Inc, 02 03 2023. [Online]. Available: <https://www.npmjs.com/package/node-firebird>. [Accessed 01 04 2023].
- [36] J. Lake, “Encryption, hashing, salting – what’s the difference?,” Comparitech, 21 12 2018.
[Online]. Available: <https://www.comparitech.com/blog/information-security/encryption-hashing-salting/>. [Accessed 01 04 2023].
- [37] npm, Inc, “express,” npm, Inc, 02 10 2022. [Online]. Available: <https://www.npmjs.com/package/express>. [Accessed 01 04 2023].
- [38] ExpressJS, “Express,” OpenJS Foundation, 2017. [Online]. Available: <https://expressjs.com/>. [Accessed 01 04 2023].
- [39] npm, Inc, “axios,” npm, Inc, 2023. [Online]. Available: <https://www.npmjs.com/package/axios>. [Accessed 01 04 2023].

- [40] npm, Inc, “ajax,” npm, Inc, 2014. [Online]. Available:
<https://www.npmjs.com/package/ajax>. [Accessed 01 04 2023].
- [41] npm, Inc, “jQuery,” npm, Inc, 02 03 2023. [Online]. Available:
<https://www.npmjs.com/package/jquery>. [Accessed 01 04 2023].
- [42] npm, Inc, “Nodemailer,” npm, Inc, 02 01 2023. [Online]. Available:
<https://www.npmjs.com/package/nodemailer>. [Accessed 01 04 2023].
- [43] wireframe.cc, “wireframe.cc,” wireframe.cc, 2023. [Online]. Available:
<https://wireframe.cc/>. [Accessed 23 03 2023].
- [44] S. G. & A. Misra, “Automation of Incident Management Processes and Benefits of Hosting Servers on Cloud,” in *017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*, Mysore, India, 2017.
- [45] Microsoft, “Should I install on Windows or Windows Subsystem for Linux?,” Microsoft, 22 02 2022. [Online]. Available: <https://learn.microsoft.com/en-us/windows/dev-environment/javascript/windows-or-wsl>. [Accessed 02 04 2023].
- [46] M. Rouse, “Showstopper Bug,” techopedia, 02 02 2017. [Online]. Available:
<https://www.techopedia.com/definition/22054/showstopper-bug>. [Accessed 12 04 2023].
- [47] Vue.js, “Comparison with Other Frameworks,” Vue.js, 2023. [Online]. Available:
<https://v2.vuejs.org/v2/guide/comparison.html?redirect=true>. [Accessed 14 04 2023].
- [48] K. B. & C. Andres, *Extreme Programming Explained, Second Edition: Embrace Change*, Upper Saddle River, NJ: Pearson Education, 2004.