Auckland ICT Graduate School

Internship Final Report

31/10/2022

CSMAX596-22B - Computer Science Internship

Student: Nicholas Jones

Supervisor: Jessica Turner

Internship Host: Radford Software Ltd

Industry Mentor: David Draffin

Declaration of Originality

This report is my own unaided work and was not copied from nor written in collaboration with any other person.

Name: Nicholas Jones

Abstract—**In recent years many enterprise software systems have shifted away from monolithic and towards microservice based architecture for the benefits that it can provide larger software systems. The API Gateway can be seen as an essential component of microservice architecture. This academic report documents the process, methodology and result of a project described by Radford Software Ltd that aims to incorporate some aspects of microservice based architecture. The internship project has provided an opportunity to work on a real-world problem, in an industry setting. The learning and reflection of this experience is provided within the discussion section of this report. The project results section outlines the deliverables that were achieved during the internship. It is hoped that the work completed during the internship will be of use to Radford Software Ltd and provide valuable insight that aids in the integration of a software solution in the future.**

I. Introduction

The Master of Information Technology (MInfoTech) programme was developed as part of the Auckland Information and Communications Technology (ICT) Graduate School, a collaborative effort between the University of Waikato and the University of Auckland to train industry-ready ICT professionals. The goal of the programme is to meet the growing demand for highly skilled network and data professionals, programmers, system architects, web developers, amongst others.

The main component of the programme is the industry internship paper which provides full-time industry experience for ten weeks with the host organisation. During the internship, the intern is mentored and supported by the host organisation and a University supervisor as the work for a real-world problem is achieved in an industry team setting [1].

This report is the result of the internship experience for the intern at Radford Software Limited (Radfords). The report covers a comprehensive outline of what was achieved, including the deliverables and the internship experience as a whole.

In Section II of this report the company background and the work environment are described. The background section provides the company history and information about Radfords. The work environment describes how the Agile methodology is incorporated into Radfords business practices. The work environment is further elaborated through discussing the company values and Radfords mission statement.

In Section III the analysis of the internship project is outlined and includes the process taken to fully describe how the project goals and project objectives are realised. This process includes requirements gathering, where meetings with the Radfords team take place, which leads next to an analysis of software architecture. A review of the technology stack leads into the development of a series of prototypes used to further understand some of the functions and programming needed for the project. Lastly for this section, research into software architecture patterns is conducted, which provides a basis for the the architecture to follow while developing the internship project.

The project goals are introduced in Section IV. These are defined by bringing together the findings from the project analysis in section III. The description of the project goals also includes the technical specifications of the technologies that will be used for the internship project.

Section V describes the project objectives for the internship project. The objectives describe the purpose of the software solutions and more detail is supplied for the Application Programming Interface (API) communications, including the endpoints that need to be developed to provide the functionality.

Section VI provides information around the project milestones including project management software and methodology for project management. A breakdown of tasks undertaken is supplied through a timeline.

The literature review provided in Section VII describes research and findings relevant to the development cycles of the internship project. The literature review starts by looking at the latest trends in the software industry to move towards cloud computing and microservice architecture solutions. Next the API Gateway architecture is described and the benefits associated with having an API Gateway are discussed. History and terminology of the REpresentational State Transfer (REST) API is provided which describes the REST request and response structure.

Section VIII provides the project results and deliverables for the internship project. These results include the Data Flow Diagram used to describe the system and a description of the functionality of the work completed, along with an overview of the three solutions developed; the API Gateway, the Hardware Integration API and the Transmission Control Protocol/Internet Protocol (TCP/IP) listener.

The API Gateway solution is described in Section IX. This section includes what was achieved during the development cycles for the API Gateway. A detailed look at how the reverse proxy functionality is implemented occurs within this section and how the reverse proxy can be configured programmatically and through an external configuration file.

Section X describes the Hardware Integration API solution which discusses the technical aspects of the functionality. A description of the source code files is supplied with information about the function of each of the source files.

Section XI describes the TCP/IP listener solution for the internship project and how this component fits into and its function with the other components of the solution.

The discussion in Section XII covers what further work is required of the next development cycles. Lessons that have been learnt and reflection is provided in this section along with examples of knowledge and skills that have been acquired through undertaking the internship project. The professional attributes are also discussed in this section which provide a few examples of how the attributes discussed have been applied for the duration of the internship project.

II. Company

A. Background

Radfords started thirty-two years ago when Phil Radford began collaborating with kiwifruit pack-houses in the Bay of

Plenty. Today the business has grown to over fifty passionate staff, supporting multiple fresh food clients in over seven countries across seven time zones - including New Zealand, Australia, France, Italy, Japan, Korea and USA. Radfords mission statement is: *To enable our customers to grow the world's food basket* which helps meet the global consumer demands by providing effective, professional software solutions that gives complete control – from soil to supermarket for fresh produce inventory management and traceability software.

B. Work Environment

Radfords operate from a head office in Tauranga, which provides a combination of open plan workspace and separate offices. The organisation hierarchy arranges the staff into teams relating to different strategic goals. The software development teams are divided into areas of focus such as the kiwifruit team, the apples team and the multi team (Maha), which is focused on a variety of different produce.

Radfords operate with an agile approach through their wider business, this brings more clarity between project teams with clearer communication and collaboration. The Agile methodology at Radfords is based around the Agile Manifesto of Software Development, which was formally launched in 2001 [2]. The four core values of Agile software development include:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation; and.
- Responding to change over following a plan.

The Agile core values are incorporated at Radfords through business practices such as:

- Daily Stand-up Meetings.
- Sprint reviews.
- Weekly developers catch-ups.
- Kanban board and project management software.

Radfords offers a flexible working environment which follows a hybrid working from home model if this option is the preference. At Radfords it is encouraged to come to the office on Mondays and Fridays to help facilitate communication within the company. Radfords have a strong culture of inclusion with the value statement *Team Rad. We win as a Team*. In addition to this value statement, Radfords values also include:

- Respect, Integrity and Commitment.
- Simplicity.
- Being Passionate.
- Having Fun.

III. Project Analysis

The initial description for the project details were supplied by Radfords (Appendix C). The project description provided was used as a baseline for understanding the project, it was only with a thorough evaluation that included requirements gathering, an analysis of the current system, technology stack consideration and research into architecture patterns that the understanding of the project goals and objectives could be described in Section IV and Section V.

A. Requirements Gathering

An initial discussion with the industry mentor was held. From this initial discussion it was understood that the real-life use case for this project would be to perform communications between third-party hardware, this would include the robotic graders (Appendix D) that sort the produce in a pack-house and communications between the Radfords software.

The network communication will be achieved with HyperText Transfer Protocol (HTTP) by building a REST API. The purpose of this communication is to provide a way to supply information required to print out pack labels through the printers. The labels are either required to be printed directly from the Radfords software or alternatively back through the API to the grader to be printed at the hardware end.

A second internal meeting was held during the first week of the internship, with a second development team here at Radfords. At this meeting a developer demonstrated how he designed a similar API integration that communicates with graders for their specific use case. The solution that the team had come up with was through a TCP connection using sockets. This was later found to be a necessary component for the Radfords Hardware Integration API and was included as an additional feature to be implemented in the Radfords Hardware Integration API project.

B. Analysis of Current Software Architecture

The Radfords software suite would best fit being described as a monolithic architecture system, rather than a more modern microservices architecture. Due to the code-base being quite mature the feature set has evolved over time and currently the core and majority of the system is built with legacy code using VB.NET.

As per the project description, *historically API integrations have been added on an ad-hoc basis* and as such have more flexibility around the development language and framework. The Radfords Core library code-base known as *Radfords.Core* is built on the .NET Framework and has the option of being used with other programming languages that support the .NET Framework.

Radfords.Core supports the older .NET 4.7.2 Framework (but not yet .NET Core or later implementations such as .NET 6). From the framework support it is possible to build the Hardware Integration API on the software side through a more modern programming language such as C# targeting the .NET 4.7.2 Framework.

The API to provide the common endpoint functionality does not require the use of any Radfords libraries, and from this it is possible to build on a later and more modern Microsoft framework, such as .NET 6.

C. Technology Stack Considerations

It is important to have a understanding of the technology stack for the Radfords Hardware Integration API Project. To achieve this a series of tasks were carried out to prototype some of the functions that will be needed for programming the software solutions. These tasks included:

- **Debugging**. Which was used by observing the data flow of the Create, Read, Update and Delete (CRUD) operations within the Radfords Software. This was achieved by setting breakpoints within the Integrated Development Environment (IDE) and watching the results through the running software. The purpose of this task was to gain some understanding of the business logic within the Radfords Software and how data is stored within the Structured Query Language (SQL) database.
- **A prototype API** referred to as *DevRadAPI* was built that used the Microsoft Entity framework software library. This gave understanding of how database connectivity can be achieved, but more importantly how it can be used to store data for later development purposes, as each request can be stored as a record in a SQL database table.
- **An API client** using Windows Forms (WinForms) was built. This application has similar functionality to other API testing tools such as Postman, and was built as a refresher into building a WinForms application and a client that can be used to send POST requests and responses asynchronously. This tool proved useful for development later to load test the network communications. The asynchronous functionality was particularly useful as this feature is not supported in a testing tool such as Postman.

D. Software Architecture Patterns

Research into software architecture patterns was conducted through web searches. It was found that the requirements discussed for the internship project fall very closely within the API Gateway architecture. An API Gateway provides a single-entry into a system and can act as a reverse proxy to provide common endpoint functionality. The architecture and concept for an API Gateway was suggested to the industry mentor, and it was agreed that this is a good architecture to work towards following. The API Gateway architecture pattern is described in more detail in Section VII.

IV. Project Goals

The primary goal of the project is to develop an API Gateway that can provide a single point of entry into the system. The API Gateway will achieve this by using reverse proxy functionality that will give common communication endpoints for the hardware API integration. The API Gateway will have the ability to have the endpoints and routes defined through an external eXtensible Markup Language (XML) file, which can be modified as needed.

The hardware API integration for the Radfords software is achieved through a REST API that provides networking with the HTTP protocol. The data will be provided by sending and receiving JavaScript Object Notation (JSON) that will allow for the communication between graders and the Radfords software services such as *FreshPackMulti* software.

The grader hardware requires the ability to send and receive data streams through a TCP/IP connection. To allow this functionality, a separate TCP/IP listener has been created that will interface with the API Gateway.

V. Project Objectives

The objective of the API communications for the Radfords Hardware Integration API is to allow for the printing of labels that will be printed at either the *FreshPackMulti* software side, or by passing back through to the hardware grader. The communication for label printing at the *FreshPackMulti* side is known as a *printlabelrequest*. The communication for printing at the grader hardware side is known as a *labelrequest*.

The project will be developed with scalability as a consideration, as it should be flexible enough to be modified to take onboard additional hardware or software features in the future. The API Gateway can be expanded to take onboard such things as the addition of middleware, for example authentication services, or load balancing requirements.

VI. Project Milestones

Milestones were supplied through the issue tracking and team collaboration software, Jira. Jira provides tools to work together with an agile and project management approach including tools such as Kanban boards with cards known as issues that can be assigned to an assignee. Through the duration of the project several of these issues were assigned for the internship project. These issues provide smaller targets to achieve within the full scope of the project. A project timeline is included (Appendix A) with a detailed break down of the tasks involved in the internship project.

VII. Literature review

In recent years there has been a shift in the software engineering community towards cloud computing. Many large companies including such companies as Amazon, Microsoft, and International Business Machines Corporation (IBM) are now embracing cloud platforms as the preferred delivery and operating model for modern applications [3]. This shift and the infrastructural changes that occur as a result has lead to architectural styles that better take advantage of the benefits obtained with cloud computing. Microservice Architecture (MSA) is one such technique that has been utilised for this purpose. Where MSA is gaining traction as a new programming paradigm for the benefits it can include many systems are still maintained as Monolithic Architecture (MA). MA was the traditional approach to software development used in the past by companies such as Amazon and Ebay. With MA the functions are encapsulated into a single application.

The prevalence of many systems still using MA has lead to academic research into methods that can be used to extract microservices from within MA. One such example is a research paper *Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems* [4].

The Radfords internship project not only required research into software architecture but also research for the RESTful web API. The REST API paradigm is explored deeper in this literature review and a detailed analysis of how a REST API functions is explored.

A. Monolithic architecture

In a monolithic architecture, all functionality is encapsulated into one single application, so the modules cannot be executed independently. This type of architecture is tightly-coupled, and all the logic for handling a request runs in a single process. This allows for the basic features of the language to divide up the application into classes, functions, and namespaces [5].

B. Microservice architecture

Microservice architecture is made up of a collection of services that may be deployed separately, are tiny, modular, and compassable (composable). Every service operates a distinct process and communicates through a well-defined, lightweight mechanism to serve a business goal. It aligns with the business to deal with changes in an agile manner, matches business changes with agile responses, and delivers solutions decentralised. The API gateway and other elements, in addition to modular services, are essential components of microservice architecture [6].

C. API Gateway

1) Single Entry Point: The API Gateway is essentially a reverse proxy for microservices that serves as a single point of entry into the system as shown in Fig 1. It significantly simplifies and improves the processes of API design, implementation, and management. The API Gateway helps address some of the key concerns, including:

- How to deal with features such as security, throttling, caching and monitoring at one place.
- How to avoid chatty communication between clients and microservices.
- How to satisfy the needs of heterogeneous clients.
- How to route requests to backend microservices.
- How to discover working microservice instances.
- How to discover when a microservice instance is not running.

2) Transformation: On the front end, microservices frequently have to deal with various clients. They have different requirements in terms of protocol (Simple Object Access Protocol (SOAP), REST, JSON and XML) and data. Backend services may understand different protocols as you transition from monolith to microservices (SOAP, REST, Advanced Message Queuing Protocol (AMQP) etc).

The API Gateway serves as a data transformation hub, allowing messages to be translated between backend, API, and app formats and protocols. The gateway serves as a central data transformation point for all traffic, translating it for:

- Requested payload transformations.
- Header transformations.
- Protocol transformations.

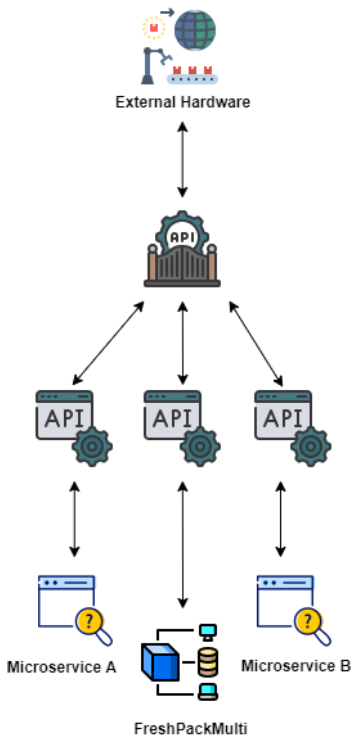


Fig. 1. Diagram demonstrating the single point of entry with an API Gateway.

3) Monitoring: Because the API Gateway is a single point of entry into the system, all traffic in and out of the system passes through it, so monitoring the gateway is critical. This allows for the capture of data flow information, which becomes an input for IT administration and IT policies.

4) Load Balancing and Scaling: The analysis of traffic and data aids in understanding and estimating the load on the system. As a result, the gateway and underlying services can be scaled appropriately. The gateway can scale horizontally as well as vertically. Load balancing an API Gateway uses the same configuration to virtualize the same APIs and executes the same policies. Load balancing should be done across groups if multiple API Gateway groups are deployed.

D. RESTful web API

REST is an abbreviation for Representational State Transfer, an architectural style for developing web services that communicate using the HTTP protocol. An example of communication using REST is shown in Fig 2. The principles of REST were developed by computer scientist Roy Fielding in 2000 and quickly gained popularity as a scalable and flexible alternative to older methods of machine-to-machine communication. It is still considered the gold standard for public APIs [7].

The key elements of the REST API paradigm are:

- A client or software that runs on a user's computer or hardware and initiates communication;
- A server that offers an API as a means of access to its data or features; and

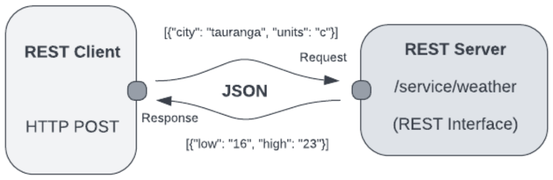


Fig. 2. Demonstrates the data flow, from client to server through an HTTP POST request and responses.

- A resource, which is any piece of content that the server can provide to the client (for example, a video or a JSON file).

1) REST request structure: Any REST request includes four essential parts: an HTTP method, an endpoint, headers, and a body. An HTTP method describes what is to be done with a resource. There are four basic methods also named CRUD operations:

- POST to Create a resource.
- GET to Retrieve a resource.
- PUT to Update a resource.
- DELETE to Delete a resource.

An endpoint contains a Uniform Resource Identifier (URI) which identifies where and how to find a resource on the Internet. A Unique Resource Location (URL), which serves as a complete web address, is the most common type of URI.

Headers contain information that is relevant to both the client and the server. Headers primarily provide authentication data, such as an API key, the name or Internet Protocol (IP) address of the computer on which the server is installed, and response format information.

A body is used to send additional data to the server. For example, you might want to add or replace some data.

2) REST response structure: In response, the server sends its representation — a machine-readable description of the resource's current state — rather than the requested resource itself. The same resource can be represented in a variety of formats, the most common of which are XML and JSON.

When appropriate, a server includes hyperlinks or hypermedia in the response that links to other related resources. In this manner, the server instructs the client on what to do next and what additional requests it may make.

E. Initial Findings

The initial findings included:

- API Gateway architecture is a good software engineering approach to follow to achieve the common endpoint functionality.
- The API Gateway can be built using a newer .NET Framework (such as .NET 6) as there are no requirements needed to use the Radfords software libraries for it.
- The Hardware API which provides integration within the Radfords Software System needs to be integrated within the current codebase and needs to be able to use the Radfords software libraries. The Hardware API must be built using the older .NET Framework version 4.7.2.
- Techniques exist that can help in the transition between monolithic and microservice architecture.

VIII. Project Results

A. Data Flow Diagram

The development of a Data Flow Diagram (Appendix B) was produced as a way to gain a high level overview of the components for the software solution. The diagram was produced as a way to visualise how the different components interact within the full system.

B. Overview of Work Completed

- **API Gateway**. Development of a functional REST API Gateway that provides reverse proxy and route controllers for three specified endpoints:
  - Print Label Request. The endpoint that receives a POST request in the JSON format and initiates the print sequence for a pack label through the Radfords Software.
  - Label Request. The endpoint that receives a POST request in the JSON format and returns a created pack label as a JSON response.
  - Istari Label Request. The endpoint that receives a UTF-8 text string and returns a JSON response.
- **Hardware Integration API**. Development of a functional API integration within the Radfords software that receives HTTP communications from the API Gateway.
- **TCP/IP listener**. Development of a TCP/IP listener that allows sending and receiving of data streams in the UTF-8 format for the purpose of communications back to the connected client.

IX. API Gateway

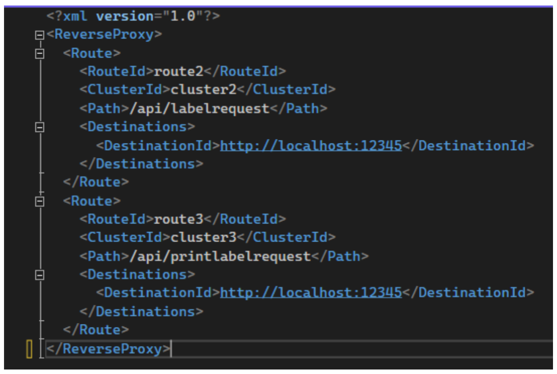
The development of the API Gateway started as a C# .NET Web API project targeting the .NET 5 Framework. This was later updated to target the .NET 6 Framework. The development of the API Gateway went through three distinct development cycles.

- The first cycle produced the Web API and reverse proxy with the Yet Another Reverse Proxy (YARP) nuget package.
- The second cycle added support for XML file configurations and the controller to receive the label response.
- The third cycle added support for the istari label request controller, and logging with the log4net nuget package.

1) Reverse Proxy Consideration: Research into reverse proxies for an API Gateway was conducted and it was found two open-source packages that remain popular choices for developing an reverse proxy with .NET. Ocelot and YARP were both considered candidates for the solution. Ocelot is a more mature package and feature rich. Ocelot is currently quite popular and provides a large feature set for use with larger scale microservices.

YARP began as an open-source reverse proxy that has recently started being maintained by Microsoft and has become the official reverse proxy from Microsoft. YARP has had more git commits on Github and active development over the last year, while development for Ocelot appears less frequently. YARP has recently released a version 1 of the software and is now out of beta development. From the Research findings it was decided that YARP would be a better fit for Radfords for the purpose of this project. The documentation provided by YARP and a tutorial guide were followed for how to set-up a YARP install [8].

2) Code-based proxy configuration: After demonstrating the first development of the API Gateway to the industry mentor, the ability to configure the routes through an external XML file was requested. The tutorial by Fabian Zankl [9], was followed which demonstrates how proxy configurations can be configured to be loaded programmatically. The class that manages this *CustomProxyConfigProvider.cs* was modified to then read the routes from an XML file. The *System.Xml.Serialization* library was used to create a object that can be used as a reference to the XML file.



```xml
<?xml version="1.0"?>
<ReverseProxy>
  <Route>
    <RouteId>route2</RouteId>
    <ClusterId>cluster2</ClusterId>
    <Path>/api/labelrequest</Path>
    <Destinations>
      <DestinationId>http://localhost:12345</DestinationId>
    </Destinations>
  </Route>
  <Route>
    <RouteId>route3</RouteId>
    <ClusterId>cluster3</ClusterId>
    <Path>/api/printlabelrequest</Path>
    <Destinations>
      <DestinationId>http://localhost:12345</DestinationId>
    </Destinations>
  </Route>
</ReverseProxy>
```

Fig. 3. An example of the XML config file.

3) XML Structure: The XML config file is shown in Fig 3. Each of the endpoints are defined through the route. The <*Path*> element defines the incoming route which is then set through the <*DestinationID*>.

4) Istari Label Request Controller: The ability for the API Gateway to receive HTTP POST requests in the *text/plain* format was required to be added as a feature. The guide by Peter Rasmussen [10], was followed as an example on how to set up a custom formatter to allow this. Next a controller for the *IstariLabelRequest* endpoint was created that can be used to receive HTTP requests in the Istari native format. The controller manages the request and will return a JSON response furnished with the created pack label request.

5) Apache log4net: Log4net provides the ability to output log statements to text files. Radfords have used error logging with log4net in the past. Based on consistency the decision for using this package was continued to the API Gateway and other solutions for the Radfords Hardware API Integration.

X. Hardware Integration API

The hardware integration API resides within the *FreshPackMultiProduceSolution* and has been created as a C# ASP.NET Web Application using the .NET 4.7.2 Framework. The Hardware API contains a *CommunicationController.cs* class which receives HTTP requests and responses. The development cycle involved forming a connection with the database and bringing through the application system arguments (SysArgs) to then create the requested pack label with the HTTP POST request. This was done through the *CommunicationController.cs*. A *LabelRequestHelper.cs* class was created as a way to abstract out the business logic from within the controller class. The *LabelRequestHelper.cs* contains the two methods, *CreatePack* and *PrintPackLabel*. The *CreatePack* method creates a pack and brings through the data to form a *PackLabel*. The *PackLabel* is uses as the POST response, which is sent back as JSON data. Alternatively it is possible to call the *PrintPackLabel* Method which initiates the printing of the Pack Label.

XI. TCP/IP Listener

The hardware graders require the ability to communicate through TCP/IP. To achieve this functionality a new project was created, the TCP/IP listener. The TCP/IP listener acts as a server to receive data streams from the connected client. A TCP client that had been developed in-house previously was used to simulate the data stream from the hardware grader. The TCP listener was created as a WinForms Application. The TCP listener receives and processes the data by sending an HTTP POST to the API Gateway. The API Gateway then manages the request and routes it to the Hardware API. The response returned is processed by the TCP listener and returned back as a data stream by TCP.

XII. Discussion

A. Further Work

At this point in the software development cycle the infrastructure and main communications for the API integration have been developed. This includes the three primary communication solutions, the TCP/IP listener, the API Gateway and the Hardware API within the *FreshPackMulti* Software.

As the software has been developed through Agile principles further development cycles are required during the software testing phase. Only basic testing to check functionality has been completed at this point, and the project requires more vigorous testing before being put into production. While testing has been completed using a software client that simulates the Tomra hardware communications, the testing should also include an implementation through connection with real hardware, before being deployed.

The three endpoints to communicate data flow developed include the *LabelRequest*, *PrintLabelRequest* and the *IstariLabelRequest*. These have been the three endpoints specified to be developed for the internship project. The API Gateway itself has been developed as a solution with expansion in mind, and it is possible to extend the endpoints to include different use cases and communications for multiple purposes.

B. Lessons Learnt

Being able to reflect on work undertaken is a valuable skill for self-improvement. For the duration of the internship project many opportunities arose that allowed for reflection. Some of the lessons that have been learnt throughout the project are described below:

C. Knowledge and Skills Acquired

The importance of note-taking. University students nowadays are provided with excellent technology for the communication age. Technology platforms such as *Moodle* and *Panopto* provide access to lecture slides and video / audio recordings of lectures given. During an on-site internship less information is recorded for access at a later time. During the first week this different learning environment provided a learning opportunity to adapt to this new environment. After the first couple of days on-site, it was quite apparent that the skill of old-fashioned note taking would be a good skill to improve on. The adaption involved a much more detailed recording of notes taken both by hand in a written journal and through digitally recording the work undertaken through study notes.

Agile Methodology in practice. Students are taught the theory and methodology about Agile principles. Being onsite however is an excellent opportunity to view first hand how Agile is put into place in a real-world situation. Situational learning such as this provide a way to gain a more practical approach to what Agile means. The lesson learnt here is that some things are best to be understood and learnt academically while other things require situational or experience to learn fully.

Embrace changing requirements. Expanding further on the point around Agile methodology is the importance of being able to embrace and welcome changing requirements. The development of a TCP/IP listener was not described in the initial internship project and was not included in the preliminary study involved before the internship began. This however was taken as a positive learning opportunity of Agile in practice, as requirements are not always known or understood fully at the beginning of a software development project.

D. Professional Attributes

Being a successful IT Professional requires application of skills and knowledge. Professional Attributes are the application of soft skills, such as effective communication skills, attitude and reliability. The internship at Radfords provided an excellent environment to put into practice development of Professional Attributes. Some examples of how professional attributes have been applied during the internship are listed below:

1) Effective Communication: Being able to communicate effectively is important to being a successful IT Professional. The internship at Radfords provides an excellent platform to keep building and improving communication skills. Communication skills can include verbal communication as a way to convey information effectively and also non-verbal communication, such as body language, dress code and attitude towards others.

Verbal communication is practiced on a daily basis, an example can be described by the Agile stand-up meetings. It is important to be able to present to the team up-to-date progress on the project in an informative and accurate manner. Verbal communication skills are also practiced every week after 4pm on Fridays as the staff often finish early and this provides a good opportunity to practice wider communication with a wider range of professionals.

Non-verbal communication is practiced through a smart and tidy dress-style. An effort is made to wear clean and smart long sleeve shirts and present myself as an open and friendly attitude by smiling and acknowledging staff, especially when arriving and leaving the office for the day.

2) Attendance: Having a desk allocated during the internship on-site provides an excellent work-space to keep a 100% attendance at the office. The benefit of being able to work on-site provides close proximity to colleagues and staff should any assistance be required. In addition to making sure to have great attendance, being punctual and arriving to work ten minutes before being due to start gives an opportunity to prepare and develop reliability, with the computer booted up and ready to go.

3) Ethics: A focus on ethical behaviour during the internship is important. The internship project provides a large amount of freedom and autonomy to work and study without strong supervision or management. It is important to remain focused on the task at hand and not become distracted. An example of how ethics has been applied can be seen through making sure to treat the computer as a tool to achieve the work and not getting distracted with personal browsing or other computer usage during work time. A high level of responsibility has been given by providing administration level access to the computer system. It is important to respect this responsibility and only install software that benefits the company and the internship project. An ethical approach is taken to the programming style, making sure not to take any shortcuts or produce bad code that could become a problem in the future.

Ethics is also shown through always working to the best ability. To be honest and open about lack of knowledge in an area and to make an effort to continuously develop skills through learning and practice. Ethical considerations to other people is important. Treating people fairly, and equally regardless of race, gender or hold any discrimination towards others.

4) Dependability: Dependability can be seen as the quality of being trustworthy and reliable. To be dependable is to always work at your best ability. To have a focus and interest in achieving results that have been set. Dependability has been shown by achieving the milestones and task assigned within the acceptable time frame. Dependability is also important by attending meetings on time and to make sure that the break-time such as lunch is taken at the appropriate time and making sure it is not any longer than it should be.

E. Project development processes

The internship project follows processes and methodology structured in a way that can be seen to follow the System Development Life Cycle (SDLC). The stages of the SDLC include Analysis, Design, Implementation, Maintenance, Planning and are cyclic in nature, providing continuous development. During the internship project, to work towards achieving a successful outcome, application of system development is used for the project. Described below are the first three phases of the SDLC and how they relate to the work achieved on the internship project.

1) Analysis: During the beginning of a project it is important to work towards a clear understanding of how and what needs to be done to achieve success. During the internship project this involved getting a clear understanding of the technology stack used, an understanding of company values and an analysis of the scope of the project through research and requirements gathering. The requirements gathering were achieved through meetings with staff and research into the packhouse machinery graders that will be interfacing with the Radfords Software. Once an understanding of the scope of the project, an analysis of systems and technologies to work with can begin. This included understanding and learning the technology stack, getting familiar with the codebase, databases and other information systems used in the project.

2) Design: During the design phase, research was undertaken that included software engineering considerations. It was found that there existed similar software solutions such as API Gateway architecture pattern that fit the requirements of the project. Conversation was had with key stakeholders to gain approval and make sure the overall programming design should follow these guidelines. In addition to the architecture, prototyping and programming smaller tasks were fleshed out as proof-of-concepts. These were demonstrated to stakeholders for feedback before developing too far into it.

3) Implementation: The implementation was an on-going and continuously improving process. Following Agile principles tasks had been assigned during fortnightly sprints. These tasks provide guidance on how features should be developed and integrated into the project in more easily accessible and managed way. This proves to be an effective way to develop a large scale project by breaking down the development into smaller components.

XIII. Conclusion

This report documents the process and describe the results of an internship at Radfords. The internship undertaken was to develop a Hardware API Integration that allowed communication between a client, such as a grading machine and a server maintained by Radford Software such as the *FreshPackMulti*.

The client and server communications were engineered with a REST API using C# and the .NET 6 Framework. The requirements for the TCP/IP listener were discovered through the Agile development approach, and a working demonstration of the software to achieve this was developed.

The learning outcomes of the internship experience included not only the development of a real-world software as described above but also the experience itself of being part of a larger team in an industry specific setting.

The internship gave a valuable opportunity to put into practice many applications of theory and knowledge that have been attained during tertiary studies on computer science and information technology. Overall the project has gone very smoothly and the confidence and experience gained at Radfords will help towards a future career in ICT.

References

[1] University of Waikato, "Master of information technology," https://www.waikato.ac.nz/study/qualifications/master-of-information-technology, accessed: 2022-9-28.

[2] W. Cunningham, "Manifesto for agile software development," https://agilemanifesto.org/, accessed: 2022-10-7.

[3] L. D. Lauretis, "From monolithic architecture to microservices architecture," in *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2019, pp. 93–96.

[4] A. Levcovitz, R. Terra, and M. T. Valente, "Towards a technique for extracting microservices from monolithic enterprise systems," May 2016.

[5] Talend, "Monolithic vs. microservices: a guide to application architecture," https://www.talend.com/resources/monolithic-architecture/, accessed: 2022-10-11.

[6] S. Gadge and V. Kotwani, "Microservice architecture: API gateway considerations," 2017.

[7] AltexSoft, "REST API: Key concepts, best practices, and benefits," https://www.altexsoft.com/blog/rest-api-design/, Mar. 2021, accessed: 2022-9-28.

[8] Microsoft, "Getting started with YARP," https://microsoft.github.io/reverse-proxy/articles/getting-started.html, accessed: 2022-9-28.

[9] F. Zankl, "Building a fast and reliable reverse proxy with YARP," https://medium.com/swlh/building-a-fast-and-reliable-reverse-proxy-with-yarp-4f70daf47300, Jan. 2021, accessed: 2022-9-28.

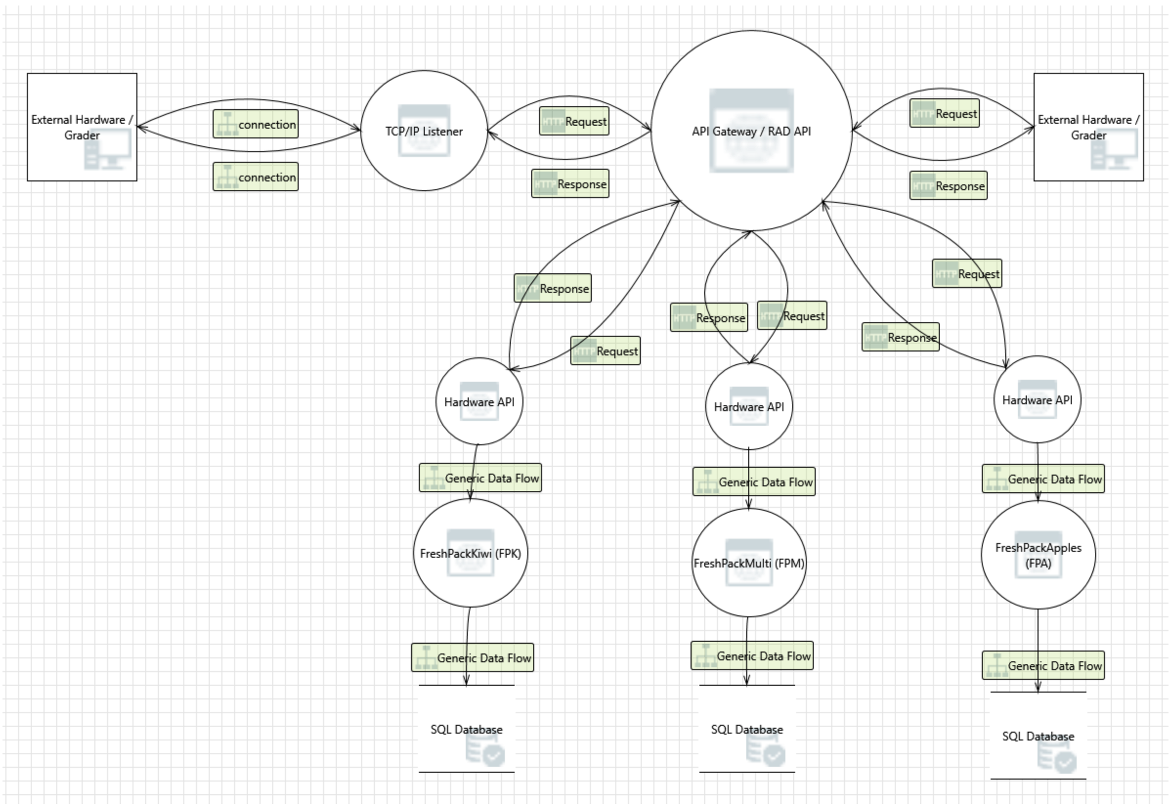
[10] P. D. Rasmussen, "C# httpclient - how to set the content-type header for a request," https://peterdaugaardrasmussen.com/2022/06/26/csharp-how-to-set-the-content-type-header-for-a-httpclient-request/:, accessed: 2022-9-28.

XIV. Appendices

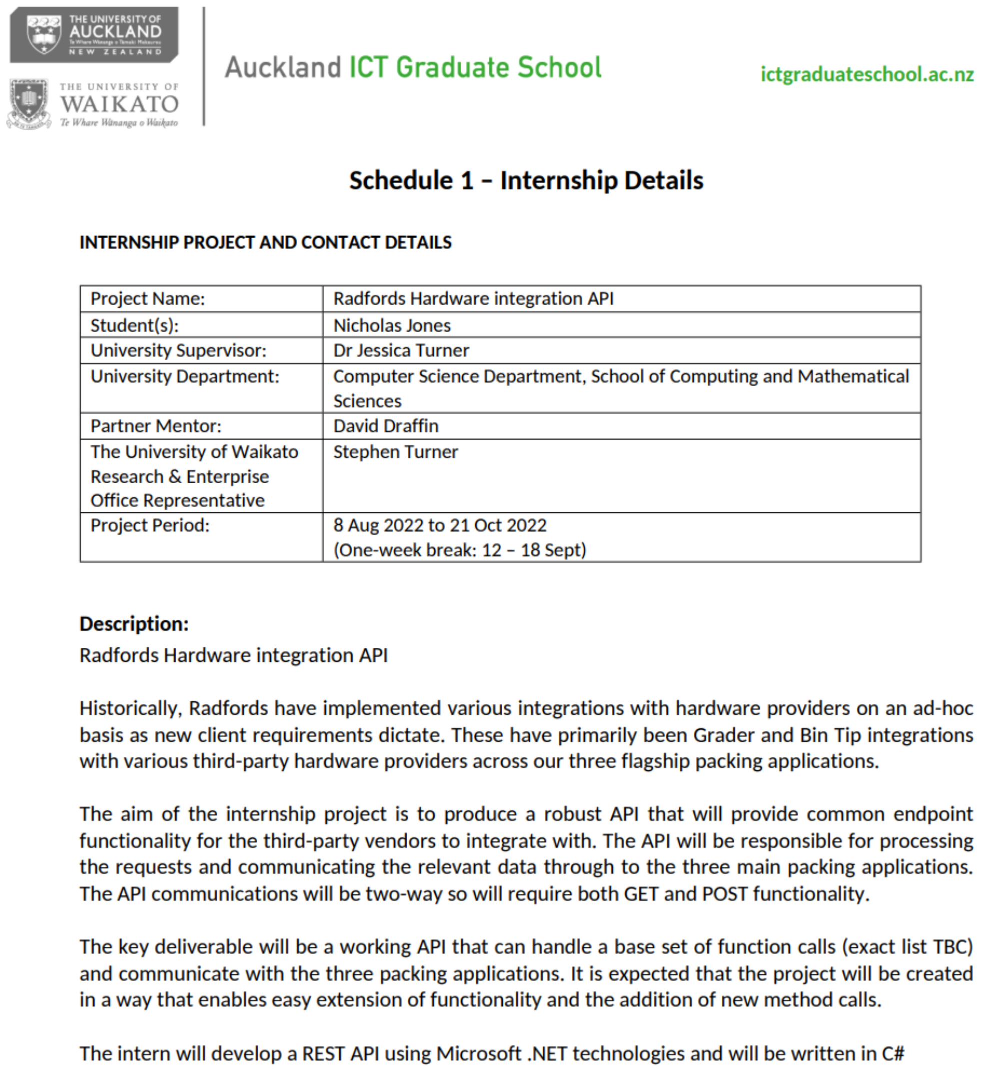
A. Project Timeline

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10
Health & Safety Modules	🟥	🟥								
Familiarisation with IT Systems.	🟪	🟪	🟪							
Alpha development REST API Prototyping.			🟨	🟨						
Research towards API Gateway, design patterns, reverse proxies.		🟦	🟦							
WinForms front-end client to connect to REST API.			🟥							
API Gateway & Reverse Proxy (Part I).			🟪	🟪						
FreshPackMulti.HardwareAPI (Part I).				🟨						
FreshPackMulti.HardwareAPI (Part II).				🟦	🟦					
FreshPackMulti.HardwareAPI (Part III).					🟥	🟥				
TCP/IP Listener (Part I).					🟪	🟪				
TCP/IP Listener (Part II).						🟨				
Report Writing & Presentation.						🟦	🟦	🟦	🟦	🟦

B. Dataflow Diagram



C. Project Description



Auckland ICT Graduate School

ictgraduateschool.ac.nz

Schedule 1 – Internship Details

INTERNSHIP PROJECT AND CONTACT DETAILS

Project Name:	Radfords Hardware integration API
Student(s):	Nicholas Jones
University Supervisor:	Dr Jessica Turner
University Department:	Computer Science Department, School of Computing and Mathematical Sciences
Partner Mentor:	David Draffin
The University of Waikato Research & Enterprise Office Representative	Stephen Turner
Project Period:	8 Aug 2022 to 21 Oct 2022 (One-week break: 12 – 18 Sept)

Description:

Radfords Hardware integration API

Historically, Radfords have implemented various integrations with hardware providers on an ad-hoc basis as new client requirements dictate. These have primarily been Grader and Bin Tip integrations with various third-party hardware providers across our three flagship packing applications.

The aim of the internship project is to produce a robust API that will provide common endpoint functionality for the third-party vendors to integrate with. The API will be responsible for processing the requests and communicating the relevant data through to the three main packing applications. The API communications will be two-way so will require both GET and POST functionality.

The key deliverable will be a working API that can handle a base set of function calls (exact list TBC) and communicate with the three packing applications. It is expected that the project will be created in a way that enables easy extension of functionality and the addition of new method calls.

The intern will develop a REST API using Microsoft .NET technologies and will be written in C#

D. Packhouse Grader