THE UNIVERSITY OF AUCKLAND

NEW ZEALAND

Te Whare Wānanga o Tāmaki Makaurau

Bridgestone NZ Ltd: Fleet App - Electronic Job Card

BTech Project Final Report

Scott Patterson

10/23/2012

ID: 1304962

UPI: spat239

Department of Computer Science

University of Auckland

Auckland, New Zealand

Academic Supervisor:
Dr S. Manoharan
Senior Lecturer - University of Auckland

Industrial Supervisors:
Robert A. Lee
Senior Business Analyst - Bridgestone NZ

Philip Low
Business Analyst - Bridgestone NZ

Candy He (Start of Project)
Business Analyst - Bridgestone NZ

Akmal Akmaloni (End of Project)
Business Analyst - Bridgestone NZ

CONTENTS

Abstract ..... Page 02
1. Introduction ..... Page 03
    1.1. Overview ..... Page 03
    1.2. The Company ..... Page 03
    1.3. The Project Motivation ..... Page 04
    1.4. The Project Goals ..... Page 06
2. Related Work ..... Page 08
    2.1. Vehicle Inspection Reporting App ..... Page 08
3. Planning ..... Page 08
    3.1. Electronic Forms ..... Page 08
    3.2. Development Strategy ..... Page 10
    3.3. Systems in Place ..... Page 12
    3.4. Initial Application ..... Page 13
    3.5. One App vs. Two Apps ..... Page 16
4. Research ..... Page 17
    4.1. End Users ..... Page 17
    4.2. Possible Devices ..... Page 19
    4.3. Web Service Design ..... Page 24
    4.4. Database Options ..... Page 26
    4.5. Deployment ..... Page 28
5. Modelling and Architecture ..... Page 28
    5.1. Technologies Used ..... Page 28
    5.2. Work Flow ..... Page 29
    5.3. Software Architecture ..... Page 32
    5.4. Class Diagram ..... Page 33
    5.5. Use Case ..... Page 36
    5.6. Data Flow Diagram ..... Page 37
6. Problems and Solutions ..... Page 38
    6.1. Non-Technical ..... Page 38
    6.2. Technical ..... Page 41
7. Implementation ..... Page 43
    7.1. Designs ..... Page 43
    7.2. Usability ..... Page 45
    7.3. Key Screens ..... Page 46
8. Future Work ..... Page 52
    8.1. Synchronisation ..... Page 52
    8.2. Drag Functionality ..... Page 54
    8.3. GPS Implementation ..... Page 54
    8.4. E-Mails ..... Page 55
    8.5. Comprehensive Validation ..... Page 55
    8.6. Visual Additions ..... Page 56
    8.7. Analysis of Data ..... Page 57
9. Conclusion ..... Page 58
    9.1. Summary ..... Page 58
    9.2. Research Implementation ..... Page 59
    9.3. Achievements vs. Goals ..... Page 60
Acknowledgement ..... Page 61
References ..... Page 62

Abstract-**With mobile devices being integrated more and more into the everyday lives of today's people, the area of tablet app design and development is one of the more innovative and quickly advancing parts in the current I.T. industry. Particularly in recent years with the release of Apple's iPad and the new Microsoft Windows 8 Operating System being developed for use in not only normal PCs but also for tablets. The people at Bridgestone New Zealand Limited saw the potential in these devices and were looking for an application to be created for them in order to help their Fleet Technicians complete some portions of their jobs in an efficient manner.**

The purpose of this report is to detail in depth a project carried out by Scott Patterson as part of a fourth year Bachelor of Technology degree at the University of Auckland, New Zealand. This project was concerned with creating one such tablet app for Bridgestone New Zealand in order to mainly replicate the current paper process of writing up a Job Card, with hopes of reducing redundancy and making the whole process more efficient than it is in the present. This project deals with queries to databases for current data, and pushing information back to them when new data has been gathered as well as allowing for manipulation of this data in a simple and natural feeling way for the user. There is also some room for additional functionality that is relevant to the project scope in terms of helping the Bridgestone Fleet Technicians complete their jobs in a more effective way.

This report is separated into nine main sections; an introduction to the project and the company it is for, giving a look at the motivation and goals behind the project as well as what potential this app is able to bring the company; a related work section to briefly talk about a second project being developed alongside the functions outlined in this report; a planning section giving some insight into how and why the app was developed in the way it was; a research section looking into seeing how the work process is from an end user perspective, as well as understanding the pros and cons of multiple mobile device operating systems, methods of web service design, choices made when working with databases and other minor research; a modelling and architecture section showing some models and diagrams created in the process of developing this app; problems and solutions which details some technical and non-technical problems that came up while working on this project; an implementation section which details the application developed and how it works; future work which details the potential for further development on the application which may further improve efficiency and some research into how these could be accomplished; and finally a conclusion which is primarily concerned with how well the project has been able to meet the goals set at the beginning.

1. INTRODUCTION

1.1. OVERVIEW

One of the major thriving industries in today's world is that of the handheld system. Today's society is full of busy people always on the move, it is imperative, especially in a business setting, that activities can be carried out quickly and whenever they need to be in a convenient manner. In social lives too more and more people are turning to their phones or tablets to handle tasks which would have required them to stop at a PC only a handful of years ago. Handheld phones and tablets are constantly being given additional functionality and are becoming a competitive market with a number of participants. iPhones and iPads were one of the frontrunners leading this revolution in 2007 [1], and more recently Android phones have been seeing a rise in popularity after their 2008 release [2], being named the leading platform in Q4 2010 [3]. Seeing the potential in this market Microsoft came out with Windows phones in 2010 [4] and has been developing their Windows 8 platform with touch screens in mind as a central focus, allowing it to be used on Windows phones and tablets in addition to normal desktop PCs and laptops. A couple of the biggest draws of these devices are that they are easily portable, being able to be used almost anywhere, and that the applications that they support are very versatile, being able to be used for almost anything. With these two main draws in mind, Bridgestone wants to get a mobile application created for them through The University of Auckland Bachelor of Technology programme in order to streamline their Job Card process and improve the efficiency of a Fleet Technicians daily work flow.

The goal of this report is to detail the final state of this project, giving introductions to Bridgestone as a company, the project and the motivation behind it, and the goals the project hopes to achieve. This report will also detail all planning put in to the project hoping to explain why the application was developed as it was and how the current systems in place were used for the final product. Research done both inside Bridgestone with the end users and outside of it for what hardware and software would be best to develop on and in what ways, such as what device to build for, how to handle data transfers and choices made when dealing with databases. A variety of models of the application will be shown in order to demonstrate how it works in theory as an abstracted view of the application and how it functions in reality. I will talk about problems and solutions which came up during the project will be detailed and explained in order to give an idea of some of the problem solving that came about throughout the work process. Finally the implementation of the final application will also be described, showing the key functions and explaining why some choices in design and functionality were done in the way that they were. After describing what has been done it is natural to mention future room for development, outlining where the application may fall short or where there is room for it to even better make a Fleet Technician's work process more efficient.

1.2. THE COMPANY

The Bridgestone Corporation was founded in 1931 in Japan by a man named Shojiro Ishibashi. It has since become the world's leading tyre manufacturer for cars, trucks and busses among others [5]. On the path to becoming the well-known corporation it is today Bridgestone bought out and merged with Firestone, one of its leading competitors in the market in 1988 which only helped the two to grow further. The New Zealand section known as Bridgestone New Zealand Limited was formed ten years later in September 1998. The size of the corporation alone is not all that has led to their success, as they are also renowned for producing quality tyres around the world, in many parts of Europe, the United States, Asia and Oceania. Being allowed to work as a part of the corporation this year for the duration of this project for attempting to promote efficiency in their work is truly an honour considering their position as a major global corporation.

Bridgestone is also a strong presence in motorsport and develops race tyres, for a multitude of professional races including formula racing and motorcycling. In addition to this Bridgestone also has a hand in bicycles in Japan and some golf products in parts of the world.

From inside the Auckland offices it is clear to see that the Bridgestone Group stays strong to its roots and bases its philosophy on four goals given by the initial Japanese company:

Seijitsu-Kyocho [Integrity and Teamwork]
Shinshu-Dokuso [Creative Pioneering]
Genbutsu-Genba [Decision-Making Based on Verified, On-Site Observations]
Jukuryo-Danko [Decisive Action after Thorough Planning]

These four goals can be seen in multiple places inside the offices as a constant reminder of what the focuses of the corporation should be. In addition to this Bridgestone New Zealand Ltd is known to be conscious of the environment, and their Ecopia range of tyres holds the internationally recognised carboNZero^{CertTM} certification.

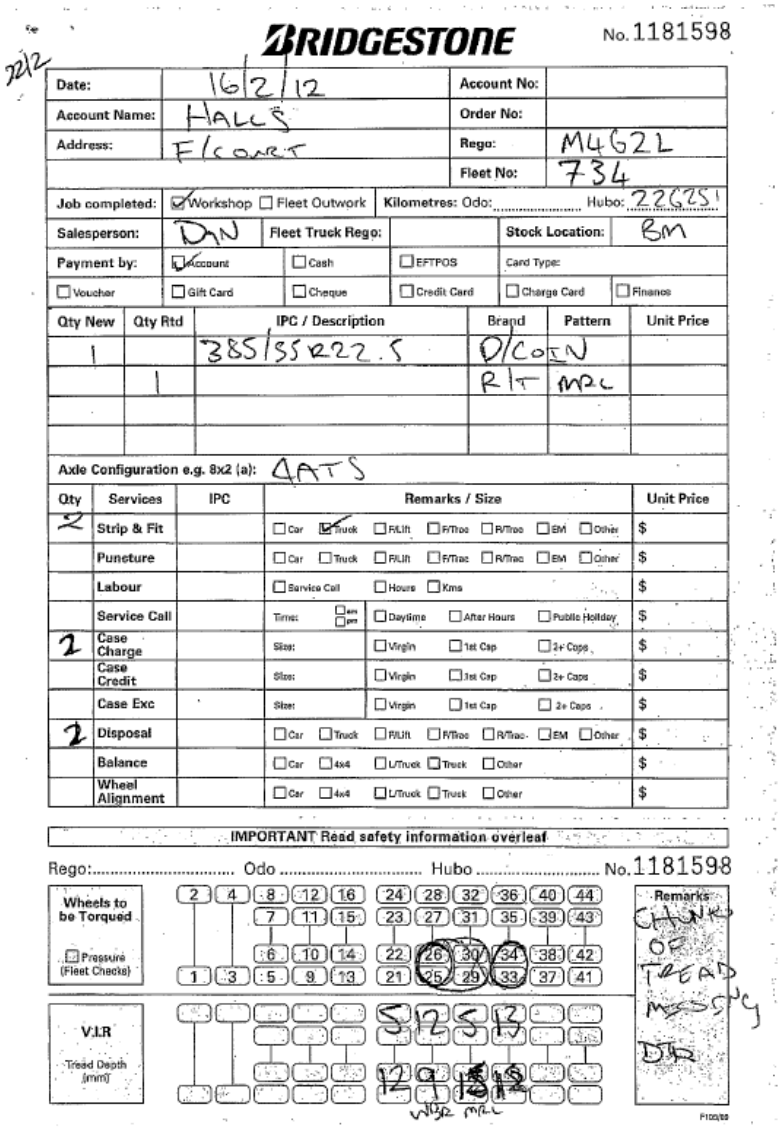
1.3. THE PROJECT MOTIVATION

The official project description as given by Bridgestone was as follows: *"A mobile application needs to be developed for their electronic job card. This is to emulate the manual recording of information on a job card to a Mobile device with a view linking ultimately to the AdvanceRetail POS system."* This gives a fairly general but easy to understand view of the project. The primary motivation behind this project is that of efficiency, attempting to speed up the process of Fleet Technicians working on fleet vehicles as well as streamline the process of producing invoices based on work done. In the current process there is manual recording of Job Card information on paper documents like that shown in figure 1. It is a Fleet Technician's job to fill out this kind of form when performing work on a vehicle in order to indicate what has been done. These forms are collected and checked over by one person at the shop to make sure there are no clear errors, and then they are later inputted into the AdvanceRetail Point of Sales (POS) system by someone else who will have to look at each sheet individually and logically convert the information written into services and product codes (IPCs) in the creation of an invoice. Even at first glance there is potential here for improving efficiency in reducing redundancy of needing to input data twice, and in having to manually convert written data through logic into items on an invoice. The application should replicate this work flow in a way that is simple and easy for Fleet Technicians to use, but still link with the POS system so that invoices can be produced for each job done with the data sent without the need for so much redundancy. The difficulty here is that the application needs to be both simple and familiar to the Fleet Technicians, as the current form has separate sections for the actual services performed and item IPCs used in the jobs carried out; so I try to integrate these two together into one flow adding the items for each service at the same time as the service. The current paper form also does not distinguish between services carried out for the vehicle and services that relate to a single wheel on the vehicle and I try to separate these two in order to allow the user more interaction with the wheels in a visual way. Putting this sort of application into real use would drastically help reduce the redundancy and promote efficiency in the whole work flow, the data need only be inputted once into a mobile application which can then have that data be sent to databases from which an invoice could later be produced with no need to re-enter this information.

This would prove to be an end-to-end process from the user selecting the items for the job they wish to carry out as they currently do on paper, to producing an invoice for the work being done. It is important that the user does not have the ability to directly create invoices when they have finished filling out the form as it is still important that these be checked over before becoming invoices, however it is possible to write them as quotes in a way that allows for them to be converted into an invoice easily once it has been checked.

I also hope to include other functionality outside of simply reproducing the paper system in order to help the jobs be carried out more easily, mainly the ability to look up stock and find the closest location to a Fleet Technician, so they will be able to pick up stock more quickly instead of calling different places and waiting for the person on the line to find out what stock is available. This fits in with a Fleet Technicians current work flow as they may need to find specific stock for the job they want to carry out. Currently if a Fleet Technician does not have the stock they need they will have to phone other branches or Fleet Technicians in order to gather this information but it would be much more efficient if they were able to access this themselves, especially considering that this information is indeed stored in a database and can be searched, just not by Fleet Technicians out on a fleet site working on trucks currently.

Another function, somewhat separate from the rest of the fleet work that was identified as an area that could be made more efficient was that of reporting Incidents and Injuries to the Health and Safety department. Currently there are two separate paper forms for Incident and Injury notification forms as shown in figures 2 and 3. An Incident form must be completed and handed in to the Health and Safety department any time an Incident occurs, and for every Incident form an Injury notification form must be filled out for every person Injured. While this is a pretty simple process of form filling and does not currently require any databases there is still room to make the process more efficient as it can be tedious dealing with multiple forms, especially considering that one Incident form may require a number of Injury forms. It would also be beneficial to link these Injuries directly to the Incident for which they are a result of, as currently they do not have any unique identifiers that enable them to do this and they must be kept together in paper manually.


BRIDGESTONE No. 1181598

Date:	16/2/12	Account No:	
Account Name:	HALLS	Order No:	
Address:	FICOMT	Rego:	M462L
		Fleet No:	734

Job completed: ☑ Workshop ☐ Fleet Outwork | Kilometres: Odo: ..... Hubo: 226251

Salesperson: DAN | Fleet Truck Rego: | Stock Location: BM

Payment by: ☑ Account ☐ Cash ☐ EFTPOS | Card Type:

☐ Voucher ☐ Gift Card ☐ Cheque ☐ Credit Card ☐ Charge Card ☐ Finance

Qty New	Qty Rtd	IPC / Description	Brand	Pattern	Unit Price
1		385/55R22.5	D/COIN		
	1		R/T	MRL	

Axle Configuration e.g. 8x2 (a): 4ATS

Qty	Services	IPC	Remarks / Size	Unit Price
2	Strip & Fit		☐ Car ☑ Truck ☐ F/Lift ☐ F/Trac ☐ R/Trac ☐ EM ☐ Other	$
	Puncture		☐ Car ☐ Truck ☐ F/Lift ☐ F/Trac ☐ R/Trac ☐ EM ☐ Other	$
	Labour		☐ Service Call ☐ Hours ☐ Kms	$
	Service Call		Time: ☐ am ☐ pm ☐ Daytime ☐ After Hours ☐ Public Holiday	$
2	Case Charge		Size: ☐ Virgin ☐ 1st Cap ☐ 2+ Caps	$
	Case Credit		Size: ☐ Virgin ☐ 1st Cap ☐ 2+ Caps	$
	Case Exc		Size: ☐ Virgin ☐ 1st Cap ☐ 2+ Caps	$
2	Disposal		☐ Car ☐ Truck ☐ F/Lift ☐ F/Trac ☐ R/Trac ☐ EM ☐ Other	$
	Balance		☐ Car ☐ 4x4 ☐ L/Truck ☐ Truck ☐ Other	$
	Wheel Alignment		☐ Car ☐ 4x4 ☐ L/Truck ☐ Truck ☐ Other	$

IMPORTANT Read safety information overleaf

Rego: ..... Odo ..... Hubo ..... No. 1181598

Wheels to be Torqued ☐ Pressure (Fleet Checks)

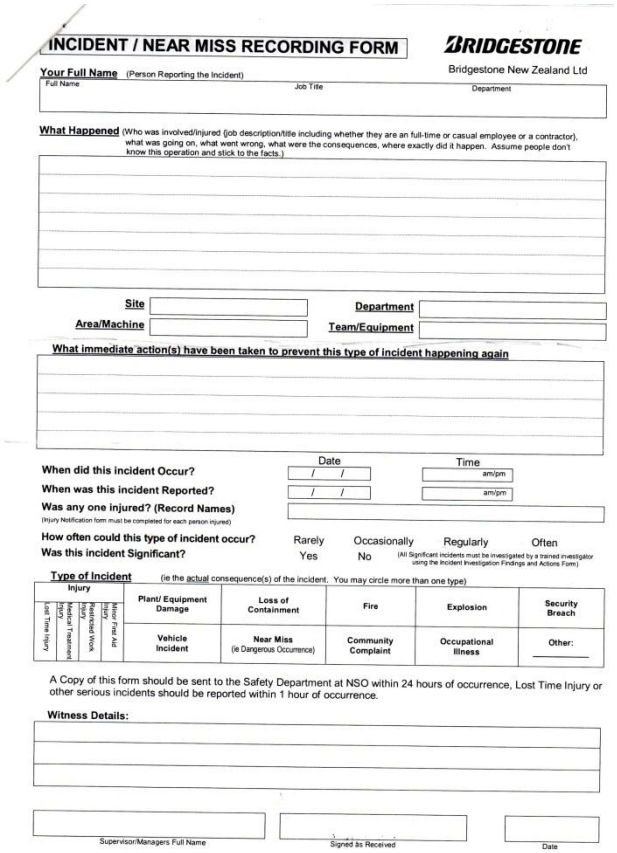
2 4 8 12 16 24 28 32 36 40 44
7 11 15 23 27 31 35 39 43
6 10 14 22 26 30 34 38 42
1 3 5 9 13 21 25 29 33 37 41

V.I.R Tread Depth (mm)

5 12 5 13
12 9 14 12

Remarks: CHANGE OF TREAD MISSING DR

Fig. 1. This is the currently used Job Card in its paper form. This specific Job Card is already completed so you can see the customer information, invoice number, job information and vehicle information all written, as well as some remarks.



INCIDENT / NEAR MISS RECORDING FORM

BRIDGESTONE

Bridgestone New Zealand Ltd

Your Full Name (Person Reporting the Incident)

Full Name | Job Title | Department

What Happened (Who was involved/injured (job description/title including whether they are an full-time or casual employee or a contractor), what was going on, what went wrong, what were the consequences, where exactly did it happen. Assume people don't know this operation and stick to the facts.)

Site | **Department**

Area/Machine | **Team/Equipment**

What immediate action(s) have been taken to prevent this type of incident happening again

When did this incident Occur? Date / / Time am/pm

When was this incident Reported? / / am/pm

Was any one injured? (Record Names)
(Injury Notification form must be completed for each person injured)

How often could this type of incident occur? Rarely Occasionally Regularly Often

Was this incident Significant? Yes No (All Significant incidents must be investigated by a trained investigator using the Incident Investigation Findings and Actions Form)

Type of Incident (ie the actual consequence(s) of the incident. You may circle more than one type)

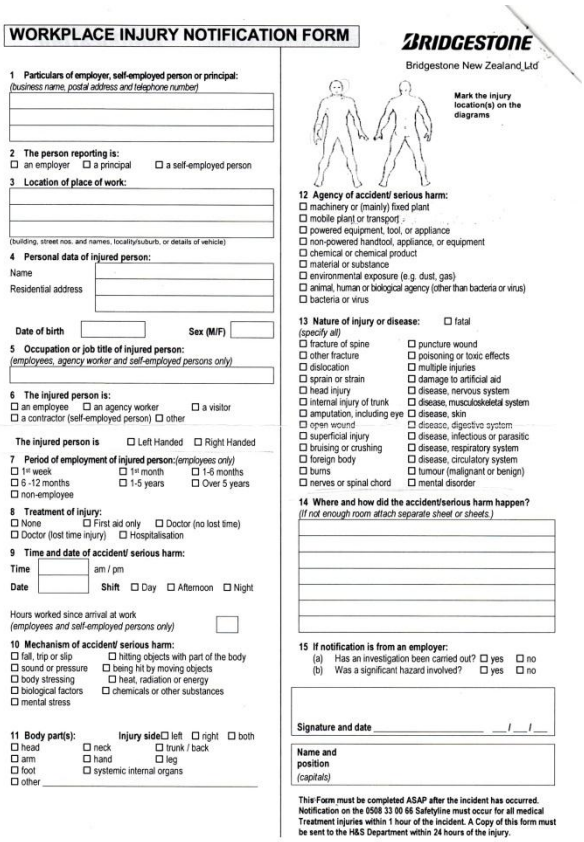
Injury						
Lost Time Injury	Medical Treatment Injury	Restricted Work Injury	Minor First Aid Injury	Plant/ Equipment Damage	Loss of Containment	Fire	Explosion	Security Breach
				Vehicle Incident	Near Miss (ie Dangerous Occurrence)	Community Complaint	Occupational Illness	Other:

A Copy of this form should be sent to the Safety Department at NSO within 24 hours of occurrence, Lost Time Injury or other serious incidents should be reported within 1 hour of occurrence.

Witness Details:

Supervisor/Managers Full Name | Signed as Received | Date

Fig. 2. This is the currently used Incident form in its paper format. On this empty form you can see the basic areas which require descriptions of the Incident to be written out.



WORKPLACE INJURY NOTIFICATION FORM

BRIDGESTONE

Bridgestone New Zealand Ltd

1 Particulars of employer, self-employed person or principal:
(business name, postal address and telephone number)

2 The person reporting is:
☐ an employer ☐ a principal ☐ a self-employed person

3 Location of place of work:
(building, street nos. and names, locality/suburb, or details of vehicle)

4 Personal data of injured person:
Name
Residential address
Date of birth Sex (M/F)

5 Occupation or job title of injured person:
(employees, agency worker and self-employed persons only)

6 The injured person is:
☐ an employee ☐ an agency worker ☐ a visitor
☐ a contractor (self-employed person) ☐ other
The injured person is ☐ Left Handed ☐ Right Handed

7 Period of employment of injured person:(employees only)
☐ 1st week ☐ 1st month ☐ 1-6 months
☐ 6 -12 months ☐ 1-5 years ☐ Over 5 years
☐ non-employee

8 Treatment of injury:
☐ None ☐ First aid only ☐ Doctor (no lost time)
☐ Doctor (lost time injury) ☐ Hospitalisation

9 Time and date of accident/ serious harm:
Time am / pm
Date Shift ☐ Day ☐ Afternoon ☐ Night
Hours worked since arrival at work
(employees and self-employed persons only)

10 Mechanism of accident/ serious harm:
☐ fall, trip or slip ☐ hitting objects with part of the body
☐ sound or pressure ☐ being hit by moving objects
☐ body stressing ☐ heat, radiation or energy
☐ biological factors ☐ chemicals or other substances
☐ mental stress

11 Body part(s): Injury side ☐ left ☐ right ☐ both
☐ head ☐ neck ☐ trunk / back
☐ arm ☐ hand ☐ leg
☐ foot ☐ systemic internal organs
☐ other

Mark the injury location(s) on the diagrams

12 Agency of accident/ serious harm:
☐ machinery or (mainly) fixed plant
☐ mobile plant or transport
☐ powered equipment, tool, or appliance
☐ non-powered handtool, appliance, or equipment
☐ chemical or chemical product
☐ material or substance
☐ environmental exposure (e.g. dust, gas)
☐ animal, human or biological agency (other than bacteria or virus)
☐ bacteria or virus

13 Nature of injury or disease: ☐ fatal
(specify all)
☐ fracture of spine ☐ puncture wound
☐ other fracture ☐ poisoning or toxic effects
☐ dislocation ☐ multiple injuries
☐ sprain or strain ☐ damage to artificial aid
☐ head injury ☐ disease, nervous system
☐ internal injury of trunk ☐ disease, musculoskeletal system
☐ amputation, including eye ☐ disease, skin
☐ open wound ☐ disease, digestive system
☐ superficial injury ☐ disease, infectious or parasitic
☐ bruising or crushing ☐ disease, respiratory system
☐ foreign body ☐ disease, circulatory system
☐ burns ☐ tumour (malignant or benign)
☐ nerves or spinal chord ☐ mental disorder

14 Where and how did the accident/serious harm happen?
(If not enough room attach separate sheet or sheets.)

15 If notification is from an employer:
(a) Has an investigation been carried out? ☐ yes ☐ no
(b) Was a significant hazard involved? ☐ yes ☐ no

Signature and date ___/___/___

Name and position
(capitals)

This Form must be completed ASAP after the incident has occurred. Notification on the 0508 33 00 66 Safetyline must occur for all medical Treatment injuries within 1 hour of the incident. A Copy of this form must be sent to the H&S Department within 24 hours of the injury.

Fig. 3. This is the currently used Injury Notification form in its paper format. On this empty form you can see many areas where the user needs to make selections instead of write out descriptions.

1.4. THE PROJECT GOALS

To successfully complete this project a number of goals were identified early on, though the application primarily needs to be easy to use and must mimic their current processes shown in figure 1 on a smaller and smooth looking interface. The goals of this project can be separated into the five main parts of functionality; aesthetics; programming; legal; and skills.

The final application needs to have the fundamental functionality of being able to input Job Card information that can be sent off to a database. The application will need to link closely with Vehicle Inspection Report (VIR) information in order to populate any and all information about the current job that it can, and then after this additional job information will need to be able to be easily entered. The input the users are able to select should be restricted depending on what is possibly relevant to the vehicle in order to limit possible mistakes. In the middle of the process there should be functionality to allow for Stock Lookups by location to find out where they can get certain tyres from. If possible the app should also include information on a company's Service Level Agreements (SLAs) and credit limit and use this sort of information in a helpful way, such as for perform validation against them. Finally after all information has been entered it should be able to be sent back to Bridgestone's databases so that an invoice can be produced without needing to re-enter any information, but should still be able to be looked over and checked before this invoice is produced. At the end of the Job Card process the application should be able to update the vehicle information with new information based on the work done, for example if pressure on a wheel was low and then a puncture was dealt with, the

vehicle information should be updated to show the state of the wheel after it has had its pressure fixed instead of being left showing the low pressure. Other goals include the Job Card replication needing to contain only information that needs to be entered into these Job Cards, no unnecessary extra information which overcomplicate the application and add clutter to the small screen. Any additional functionality should be on separate screens as separate parts of the application so as to not complicate the main Job Card portion. It was identified that the application should auto-fill any information possible in order to minimize redundancy and help the process, so it should be able to store and display the date based on the mobile clock, account and customer information based on the VIR the job is for, and vehicle information based on the vehicle the user is working with. The appropriate configuration diagram for each vehicle being worked on should be displayed from which job information is to be written into. If an optional field is left out the application should be able to progress without issues, and functions like dropdowns, radio buttons and checkboxes should be used when appropriate. The application should function like a wizard allowing the user to move forward and back between multiple screens as opposed to vertical scrolling, and the pages should be cached so information is not lost while moving. There should be an Incident and Injury reporting portion of the app which should have simple functionality that enables the user to input all necessary information from the relevant forms, minimising text input where possible and allowing for some parts to be automatically filled in such as the date or company address if they are a Bridgestone employee. It should also be able to bind many Injury forms to a single Incident form. The app needs to be usable outside of an office in a common fleet site where a Fleet Technician works and should simply replace their current paper process in a more efficient way without disrupting their work flow.

In terms of aesthetics there are multiple goals that I hope to achieve with the design of this application. I hope to make the content easy to read and easy to view for the target audience. Content should be sized appropriately and be visually recognizable to the users so they can distinguish it from other non-Bridgestone applications. All images and visual content should not only be appropriate but should also be smooth and should fit well onto the screen size of the designated device. As mobile devices have smaller resolutions than PCs and laptops it is a goal to avoid scrolling where possible, so content should be divided between multiple pages that are easy to navigate between naturally instead of simply having one screen replicating the paper form that needs to be scrolled and zoomed to use. In areas of the application that are not simple to present in a natural way there should be instructions to guide the user to make the correct choices, however the instructions should not be too complicated themselves. Visually the application should be familiar to the users using colours fitting for Bridgestone and stand out as a modern tablet app.

From a programming perspective I hope to make the application run in a way so it does not function too slowly, as this would be somewhat inefficient which contradicts the main purpose behind the application. In order to achieve this I will need to look at ways to create mobile applications properly and how to handle wireless data transfer using web services. I should also be conscious of how I code and indent everything properly with appropriate comments, in case the application needs to be understood and modified by someone else at a later date as there is room for future development if the company chooses to proceed using this application.

Avoiding copyrights is an important societal - legal factor, if copyrights are ignored and information or images are used without the permission of their creators then it becomes a legal issue. I must make sure all content is used with proper rights and references, as well as make sure that no sensitive information is leaked outside of the company about their business and how it operates. I should also consider the possibility of unauthorized users being able to modify data incorrectly in ways that could disrupt the proper business flow.

There is also a skills factor in that the creator of the application will need to develop the skills necessary to meet the requirements of the project to create a product that is both properly functional and aesthetically pleasing in ways that are appropriate for the end use. As I will be using a lot of new technologies in this project that I have never used before I will need to develop skills in all of these technologies, both hardware and software, as well as programming knowledge in the chosen languages so that it can be coded in an efficient manner.

2. RELATED WORK

2.1. VEHICLE INSPECTION REPORTING APP

Closely related to the main Job Card portion of this application is the Vehicle Inspection Reporting (VIR) portion being developed alongside the Job Card. In a Fleet Technicians standard work flow they do not simply go out to a fleet and do any work they see or are requested to do, they must carefully inspect each vehicle they are in charge of and determine where work needs to be done and where one vehicle may not meet requirements to consider it fit for work. This sort of inspection is, just like the Job Card, written down on a paper form. The primary purpose of this form is to record the tread depths and possibly the tyre pressures of each wheel on a vehicle as these are the foremost indicators of when a job needs to be carried out on a vehicle. Bridgestone New Zealand was not only looking for their Job Card to be replicated electronically, but also their VIR forms, which they passed on to the University of Auckland as two separate projects they would like done. The University assigned two people to these projects, myself to the electronic Job Card and another student Vishal Naidu to the VIR form. It was identified very early on in the process that these two applications would heavily interact in order to avoid redundancy, as the same customer and vehicle information is required inside both apps, and more importantly the results of a VIR are what gives rise to a Job Card in the first place. As the VIR is earlier on in the work flow, before a Job Card is done there is potential for reading necessary information from the electronic VIR form for use in the electronic Job Card.

As this VIR form is to be built at the same time as the electronic Job Card I cannot simply plan my app based on it and what information it should have, and will have to instead plan alongside it and discuss with the student in charge what kind of information he will be using, what information I am able to obtain from his form, as well as some design choices should the two portions be consistent. As both portions are heavily centred around a vehicle type and the user will do different things to each specific wheel (fill in tread and pressure information or add job card items for work on a position) it would be beneficial to allow the user to visually select each position, as both the VIR and Job Card forms currently have a visual configuration diagram allowing them to do just that, so there is potential for re-using the same diagram inside both electronic forms both for consistency and to help spread the workload between projects.

3. PLANNING

3.1. ELECTRONIC FORMS

This project had been outlined from the start as an electronic form filling replacement of a current paper form process for the purpose of handling Job Cards to create invoices through POS. Bridgestone saw the potential in electronic forms, however I felt it was necessary to understand and plan for the distinct advantages, or plan around the potential disadvantages this electronic form may have in comparison to the current paper version. By analysing the differences and noting the areas where an electronic form is able to surpass a paper one I would be able to proceed to plan and design this application catering to these benefits, and possibly create it keeping any drawbacks in mind to minimise them.

Some of the pros of electronic forms are fairly obvious, such as the potential to save time and improve efficiency as stated in the project motivation. Using an electronic device instead of paper forms allows for the filled out forms to be created in a data format capable of being instantly stored in databases without any further logic or reading to be performed. The time saved in not requiring translating what is written on each individual filled out paper form for use in a more long term data storage is immense. In addition to this there is also potential for forms to be sent to databases as they are filled in with no time lag, allowing for databases to always be up to date with the more current and applicable data possible. This sort of thing is just not possible with paper forms as they have to be collected, given to someone who handles data entry, and then entered into a database in bulk later on. This is hardly efficient in time and results in databases holding potentially outdated information for prolonged periods of time. Another potential benefit of electronic forms over paper ones comes into light with this in that the actual process of transcribing hand written forms into a database can result in potential human errors with misread information. Electronic forms can help discourage any incorrect data being entered with functionality such as validation or only allowing users to input information into relevant fields which can

be dynamically handled in real time as the form is being filled out, with no need to even allow the user to submit a form with these mistakes, allowing them to correct any errors while they are still working on it. This can be done by programming a variety of different fields in a variety of different ways, allowing for forced input, able to be modified, or invisible to specific users [6]. Electronic devices do not only have functions that enable less errors to be made, but can also reduce the information needed to be written into the forms in the first place using functionality such as the date of the local device leaving no need for the user to write out the current date for each form. There may also be potential to use more advanced device functions with cameras or touch screens allowing for different kinds of inputs seamlessly that a paper form cannot accurately handle. There is also an environmental and storage benefit of not needing to produce and keep track of a mass of paper forms for every instance of one being filled out.

There has been significant testing done with user response to using electronic forms for many years with the steady rise of consumer computers, both looking at how users feel about actually inputting information into the forms or at the ability to handle the data written by them. In general it is perceived that most end users are likely to either prefer an electronic interface, or have no preference but it is unlikely for them to rather use paper, while from the viewpoint of those dealing with the data of the forms an electronic version universally favoured [7], due to the simplicity of data consistency and the lack of need for data entry of completed forms. Generally user response for using electronic forms in favour of paper forms is positive, with interfaces being able to easily be more visual and smoother, in addition to the severe impact it can have on saving time and improving efficiency.

The primary concerns with using an electronic form when a paper one is currently in place are those of disrupting the work flow and security [7]. Disrupting the work flow is a valid initial concern due to the fact that users are currently familiar with how they go about their jobs and how they are to fill out their forms, introducing them to a foreign electronic version of the form on a tablet device would feel strange at first and there may be some initial training required in order to bring users up to speed on how they can perform all the actions they are used to on the paper form. This downside should be minimised by making the form link somewhat closely to the paper one, and also providing either visual cues or instructions on how to carry out specific tasks. Security on the other hand is more of a concern; when using paper forms it is difficult for information to be modified by unauthorised individuals, and even if a form is lost and filled in by an outsider there is virtually no way that this can end up entered back into the work flow in a way that may cause issues. When it comes to using an electronic application that links directly to databases however there is much more of an immediate need for security with a large number of current fraud risks in electronic systems [8]. In the case that the device is compromised there is a real issue of unauthorised rows being inserted into databases unless appropriate access controls or user accounts are in place. There are many ways to combat these potential security issues in order to make an application like this safe to use, for example security user logins with sufficient password or other such secret information required for use, or to make the application unable to write to anywhere in such a way that may cause damage, possibly allowing written data to be checked by a separate person before the work flow is completed.

In order to create an electronic version of a current paper form the paper form must be extensively understood [9]. This is mainly due to the fact that the logic used in understanding how a form is filled out needs to be reflected in an electronic form in the backend, converting specific inputs to some form of data that can be understood by a database. However there may still be issues in the conversion of a paper form into an electronic format, potentially with being able to gain appropriate information and compliance from those who are currently familiar with the paper version, and with creating an electronic form that is clear and natural enough for use of untrained persons [9]. It is important that users are involved with the development of the application, giving constant feedback on the work flow and the application functions in order to develop an understandable form that is recognised and accepted by everyone involved.

3.2. DEVELOPMENT STRATEGY

When planning a major full year project such as this it can be imperative to determine a development strategy to undertake throughout the year. A development strategy can be used in order to help plan for how to meet goals, giving an overall view of the development process, noting when analysis, designing, implementation and evaluations should take place. There are a number of different approaches to this each with their own perceived costs and benefits, some common methodologies used today are Waterfall Development, Incremental Development, Spiral Development, Rapid Application Development and Agile Software Development [10].

To briefly outline these approaches, Waterfall Development is a linear approach with sequential phases [10]. These phases are carried out one after another, from the initial investigation, defining requirements, designing the system, coding and testing, implementation, and finally orientation and support. This kind of flow is meant to promote a straightforward approach where each step is clearly defined and separate from the other steps, and earlier steps in the process should be completed before moving on to the following ones, this promotes well defined documentation, however it is fairly inflexible and has little room for backtracking. Due to this linear process this would ne be ideal for the project, as it better caters to professionals with a lot of experience that are unlikely to need to backtrack.

Incremental Development is partially linear and partially iterative [10]. It can be thought of as multiple waterfalls one after the other. The main purpose of this is to further analyse what has been done and re-think previous stages of development, allowing for modifications of early stages to adapt to a growing project. This approach holds more benefit for very large projects that have a large number of functions to develop over time, meaning it may not be the most appropriate for a medium-sides project such as this one.

Spiral Development is again partially linear and partially iterative with four main areas [10]. These are analysis of options, evaluation of alternatives, development and then planning for future phases. This cycle is repeated many times for every phase of development in a kind of ‘spiral’, hence the name. The main benefit of this strategy is that it is very flexible, allowing for other methodologies to be incorporated into this framework where possible. This methodology was potentially usable for this project however it is suggested for highly skilled and experienced project managers and may be difficult to manage for a University project.

Rapid Application Development (RAD) is an iterative methodology [10]. The goal behind RAD is to develop a quick quality system with minimal investment. In order to meet this goal the development has heavy user involvement for as much feedback as possible, and also uses development tools wherever possible for building things like interfaces, creating databases, and generating code. The priority of this is development and delivery of a final product, adjusting the scope based on restrictive time constraints, with only necessary documentation for future work sufficiently defined. This allows for the software to be running at earlier stages of development as early versions with partial functionality, as opposed to throwaway prototypes, and also promotes software to be developed very quickly resulting in low costs. As this project is supposed to last one year with a balanced level of planning and development RAD is definitely not a strategy that warrants consideration, especially considering the fact that Bridgestone does not need to pay me as a developer for the project anyway, therefore cost is not an issue even over longer periods of time.

Agile Software Development was suggested by one of the Bridgestone supervisors as a good development method for use with teams. This seemed to apply to this project due to the related work of the VIR sections of the application being handled by another student, effectively another team which I needed to interact with and plan around. While there were only two ‘teams’ involved in the creation of this application this is in fact beneficial to simplifying the development process, as the fewer members involved the fewer social network connections required for communication between groups, meaning progress is much smoother with less groups [11]. Agile development is iterative and using adaptive methods there is a focus on measuring progress as development is undertaken and adjusting plans based on how it progresses [12]. This dynamic planning enables for goals and milestones to be appropriately adjusted depending on the current success of the project; if progress is behind schedule goals are relaxed in order to easier catch up, while if the application is ahead of schedule additional goals or functionality can be planned for. These planned estimates for delivered items are supposed to get more accurate as the teams get more familiar with the project and can better predict progress. Progress is measures in ‘commit-to-progress’ ratios, measuring the ratio of planned work and achieved work at each iteration. A higher ratio is beneficial as this is an indication of keeping closer to the projected goals. Keeping

track of this ratio helps better determine just how many adjustments need to be made to the current milestones, keeping the project plan dynamic and flexible based on what is feasible at any given time. Agile development is more concerned with showing working software as opposed to planning documents [13], therefore most of the work done in Bridgestone offices was on the software, and this was what was presented to the supervisors. It is also important for users to be involved in the development of the application allowing for feedback on progress based on the working software presented in meetings [13].

The benefits of using agile development can be seen clearly in figure 4. Due to constant interaction for feedback from users the visibility of the development process remains high with small spikes at each demonstration given. The Value of the application rises early on in development as constant interaction with users allows for primary functions and usability issues to be taken into account and dealt with early on in development, levelling off as time progresses the development heads towards the features identified as having lower priorities. The adaptability of the application remains fairly high throughout due to it being an iterative process, it slowly decreases over time as more portions of the application are completed and no longer worked on, yet even at the end there is still room to adapt. Finally the risks involved start out high as the developers involved are initially unfamiliar with the requirements of the project, however the constant interaction with users ensures this gets minimised early on and risk remains low for most of the development stages. There is also low risk involved due to a constant pace being carried out over development of the application, making sure that there are no risks that arise from quick development in a restricted period of time.

This kind of agile development was followed throughout the development of the application, taking advantage of the team based approaches with constant discussions between the two ‘teams’ to share ideas and better clarify what points of development we were at. This has been shown to increase empowerment and promote higher moral with interacting groups as opposed to solo development [13]. During the development process multiple lists of prioritised milestones were created, each time adjusting them to better reflect what has been done and what could be done within the time frame. At some points in the project extra milestones were added when we were ahead of schedule and new ideas were obtained through various meetings and discussions. We also had to adjust in the other direction a couple of times pushing some low priority goals back when other more important functions were taking too much time. Having another ‘team’ to work alongside with, plan with and share ideas with allowed for more opinions to be gathered for how to handle certain areas of the project and these sorts of discussions allowed for a more dynamic and comprehensive application with far less problems than would have been apparent otherwise.

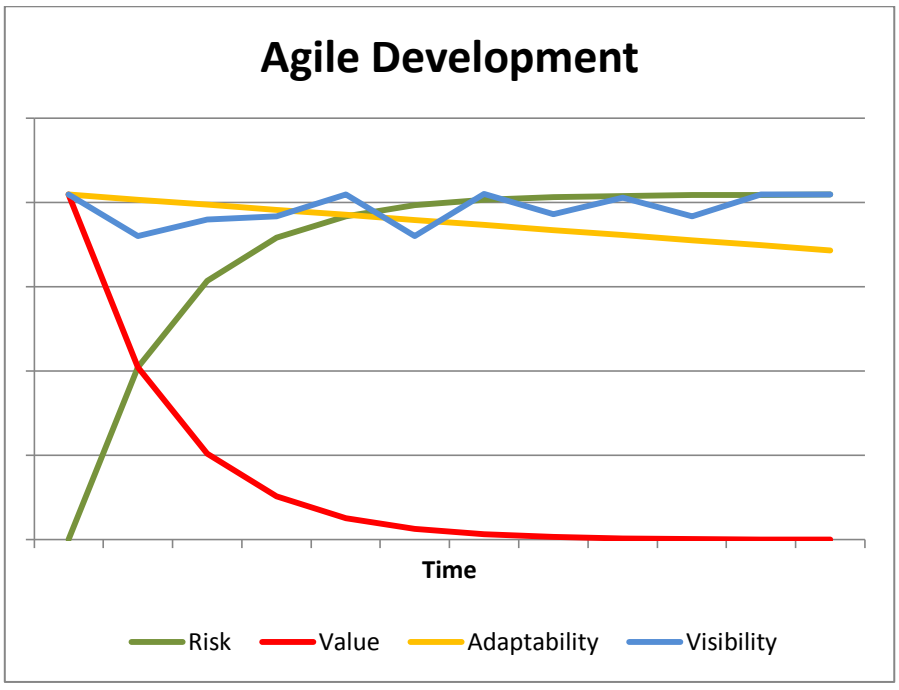


Fig. 4. Common trends seen in Agile Development created from the information in [14].

3.3. SYSTEMS IN PLACE

Bridgestone being a well-established corporation has many current systems in place to handle their data in various ways (figure 5). The data warehouse is where most data is stored eventually after coming from various locations; Wherescape RED builds this data warehouse and its fact tables. Figure 5 gives a closer look at where the customer, transaction and vehicle data comes from (indicated by the red circle in figure 5) before it gets sent to a data warehouse, and how it interacts with SAP PI. The data stores circled in this figure are the ones relevant to this project in order to gather information of customers, their vehicles, and transactions. BPCS deals primarily with transaction information. CRM deals mainly with customer details but also the POS system information which is then sent to the data warehouse via SQL. These systems interact with the SAP PI which has been recently introduced into Bridgestone New Zealand Limited. Vehicle information comes from a vehicle repository where a third party web service allows a registration number to be looked up in order to return comprehensive information on a certain vehicle. The customer and fleet information, including this vehicle information is stored in the data warehouse via overnight processing.

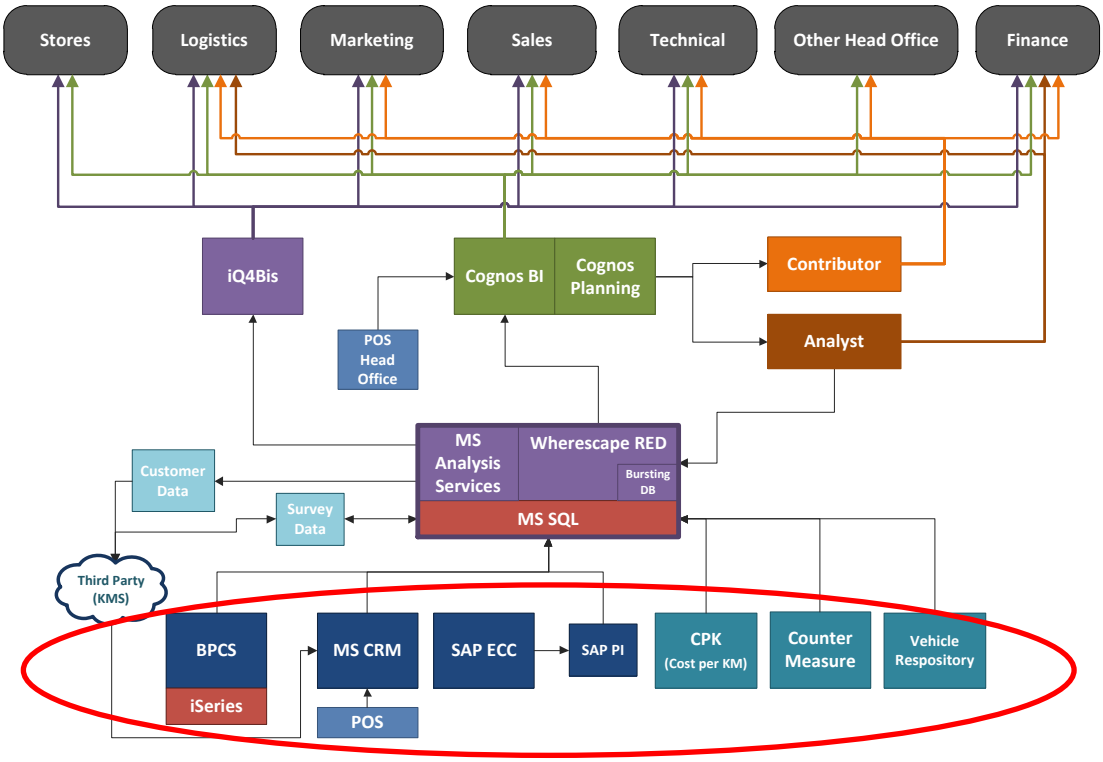


Fig. 5. An overall look at the systems and links in place inside Bridgestone New Zealand Ltd.

The Electronic Job Card will mainly be pulling data from just the VIR application as opposed to from the Wherescape RED data warehouse. However it will be dealing with the customer, transaction and vehicle information that the VIR application has. Customer information will be able to give my application data like contact information. For a specific customer I will be able to find previous job information to give outputs like previous transactions to give an overview of what kind of jobs their fleet has been having and how frequently. I will also need information about the vehicles in these fleets and their tyres as these are what the jobs will be carried out on, tyre information can be used to enforce specific rules such as same tread type across an axle. The Job Card will also need stock information from the data warehouse in order to match products chosen by the user to their distinct IPCs, as well as to give information on the prices of products chosen and some other information which is necessary in writing an invoice to POS for the items in a Job Card.

Stock Lookups will also require the stock information from these databases in order to relate a product description, IPC and location information to a user. The location given to the user should be as detailed as

possible giving not only the address of where stock is stored but also a phone number and an indication of how much stock might be available.

As the current Incident and Injury forms are not stored in databases this sort of information cannot be pulled and filled out, and also after being recorded has no place in Bridgestone's current systems to be stored.

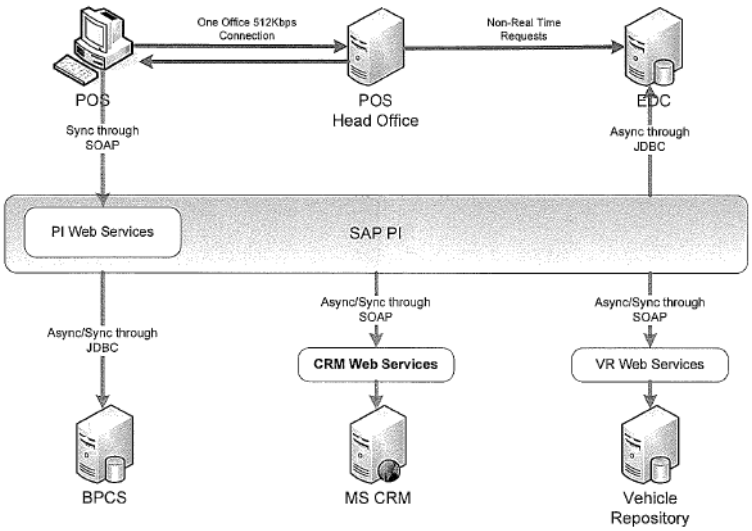


Fig. 6. A closer look at BPCS (transactional data), CRM (customer data) and the Vehicle Repository (vehicle data) and how they link with POS through SAP PI.

3.4. INITIAL APPLICATION

Primarily as a test of our skills as programmers, and to prove our understanding of Bridgestone's databases we were given a small project to create and consume a web service which allowed a user to enter a registration number and then returned various information about the vehicle and owner of that vehicle. We took this application quite seriously as it provided functionality that would be included in our final projects and went so far as developing a scope document, diagrams and conceptual designs of the user interface that was to consume the web service. In addition to this as we were unfamiliar with how to go about creating a web service and how they worked so we needed to carry out sufficient research to boost our understanding of the project. Figure 7 shows the use case diagram we created for the very simple application, only containing two self-explanatory use cases.

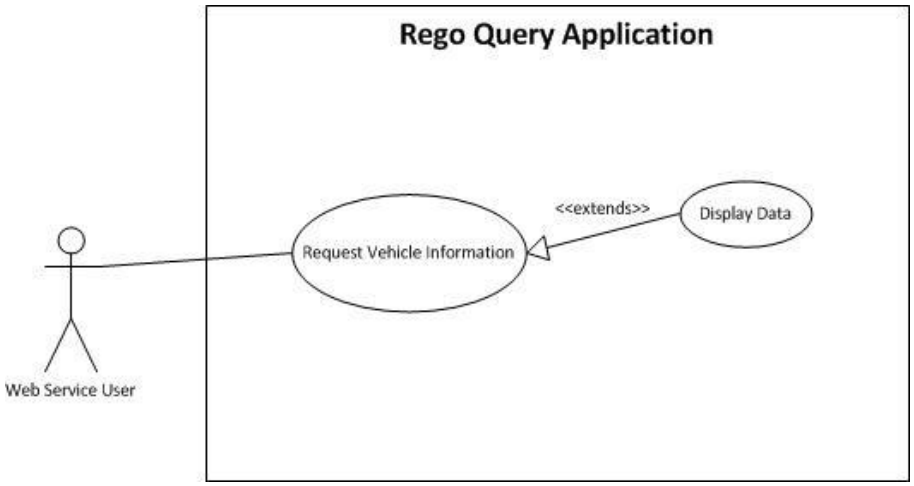


Fig. 7. A simple use case for the initial application

As you can see this is a fairly simple application with only one function a user is able to perform, that of requesting vehicle information with an inputted registration number. Simple as it was, it is still important to understand how web services could be created and consumed, as this was a major part of the final implementation.

For interface design we started with concepts on paper, as you can see in figure 8, and then after deciding on a concept to go with we created the interface as a WPF application in Microsoft Expression Blend as shown in figure 9. For this Blend interface we included multiple pages with navigation, XAML customised hover and click effects on buttons, and a colour scheme which we felt represented Bridgestone.

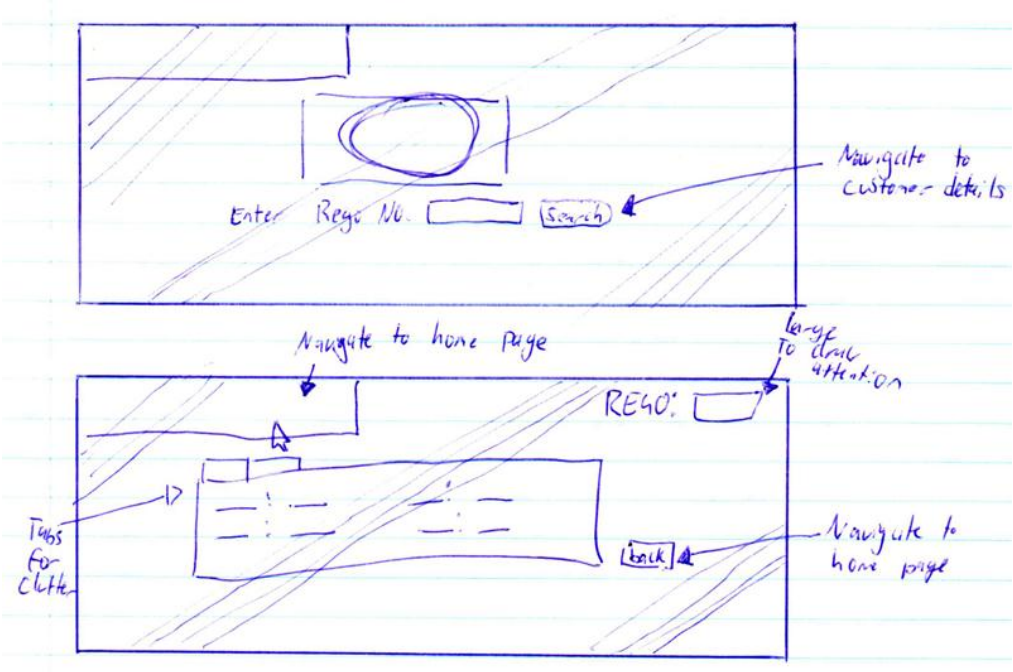


Fig. 8. One of the initial conceptual design sketches of the interface for the initial application

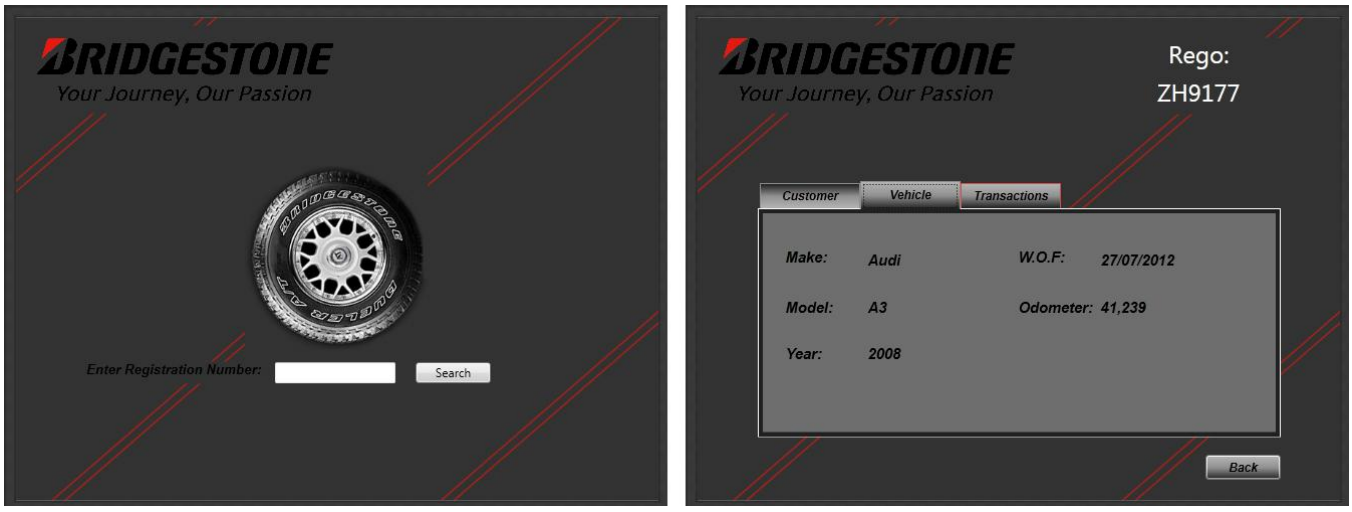


Fig. 9. The completed Blend interface for the initial application

You can fairly clearly see from these that the chosen design translated almost exactly into the final Blend interface. This is mainly due to the fact that Microsoft Expression Blend is a WYSIWYG interface design tool that gives users a fair amount of control over how they wish to customize their interfaces. Blend also allowed you to view your applications in XAML in order to further customize them through programming. We took advantage of this to customize tabs and buttons for specific gradient effects, as well as hover and click

animations. Figure 10 shows a small section of this code where dynamic resource styles were created and applied to a button for specific gradient effects. This needed to be done using the XAML section of Blend otherwise these specific brushes could not be properly applied.

```
202     <TabItem Header="Transactions" Background="{DynamicResource Gradient Tab Brush}" Style="{DynamicRe
203         <Grid Background="#FF6E6E6E"/>
204     </TabItem>
205 </TabControl>
206 <Button Content="Back" HorizontalAlignment="Right" Margin="0,0,59,20.04" VerticalAlignment="Bottom" Wi
207     Background="{DynamicResource Gradient Tab Brush Invert}" Style="{DynamicResource ButtonStyle1 Edit
208     <i:Interaction.Triggers>
209         <i:EventTrigger EventName="Click">
210             <pi:NavigateToScreenAction TargetScreen="WpfPrototype1Screens.Screen_1_1"/>
211         </i:EventTrigger>
212     </i:Interaction.Triggers>
213 </Button>
214 id>
```

Fig. 10. Some customized XAML in Microsoft Expression Blend

The front end design process was carried out thoroughly to reach the application shown above, and alongside it the backend was created in Visual Studio as a WCF web service. This kind of process is exactly how we plan to produce the final Windows 8 product for Bridgestone. First we had to create a web service that accessed the Bridgestone data warehouse. This required research on how to achieve this, after which we applied the knowledge we learnt to implement a web service using Bridgestone's database. We then had to connect to this web service and consume it. The web service was created with the default method which was SOAP. The code shown in figure 11 is a section that was the main hurdle in connecting to our web service and consuming it, showing the connection string and SQL commands.

```
            string regono = InputTextBox.Text;

            regono = regono.ToUpper();

            ServiceReferenceRego.RegoLookupClient c = new
ServiceReferenceRego.RegoLookupClient();

            Label1.Text = Label1.Text + c.RegoSearch(regono);

            SqlConnection db = new SqlConnection("Data
Source=baksql2 ;Initial Catalog=datawarehouse_UAT;Integrated
Security=SSPI; persist security info=False;");
            try
            {
                db.Open();
            }
            catch (Exception exc)
            {

                Console.WriteLine("Problem opening
connection..");
            }

            SqlCommand command1 = new SqlCommand("select
plate, make, model, year_of_manufacture,
expiry_date_of_last_successful_wof, latest_odometer_reading
from dim_vehicle where plate = '" + regono + "';", db);
            SqlDataReader myReader = command1.ExecuteReader();


            if (myReader.Read())
            {
                OutputLabel.Text = myReader.GetString(0);
                OutputLabel0.Text = myReader.GetString(1);
                OutputLabel1.Text = myReader.GetString(2);
                //OutputLabel2.Text = myReader.Item
(3).ToString;
                //OutputLabel3.Text = (string)
myReader.GetString(4);

            }

            myReader.Close();
            db.Close();
        }
```

Fig. 11. A section of the backend coding on consuming a created web service.

At the top of figure 11 you can see some code for handling different inputs that should all be read the same, as some users would be likely to input a registration number using upper case while others may use lower case, so after being placed into a variable for use in the SQL query all registration numbers were forced to upper case.

The connection string posed a large problem as we had not created a web service like this before and we were unsure of how to structure a connection to a database, let alone to Bridgestone's specific database. However we got it working in the end and were able to execute SQL queries based on an inputted registration number, and then display the results as was required by the application. This whole initial project was a very useful stepping stone in the project for creating the final application as it allowed us to create many sections that our final applications will use, but on a smaller scale. It allowed us to test both Microsoft Expression Blend and Microsoft Visual Studio in parallel to develop the backend and frontend separately to give a feel as to what this course of implementation would feel like and allowed us to determine how easy it would be to use these programs on a larger scale. It also allowed us to research and develop an understanding of how web services work, how they are created and how they are consumed, and more importantly taught us how to create and use these to interact with databases that Bridgestone owns.

3.5. ONE APP VS. TWO APPS

During the initial stages of planning the Job Card functionality and the VIR functionality described in section 2 as related work were considered two separate applications. This was mainly due to these projects being presented to the University as two things they would like done, and subsequently being given to two separate students. In terms of distribution of work these two projects were fair both between themselves and in comparison to other projects, however once planning and research was underway it seemed more and more that the application would be able to much better meet its goals should the two planned applications be condensed into one with different functional sections. Having two applications which require heavy sharing of data; both data created by each of these applications and data drawn from external sources would be quite difficult to handle efficiently. It would need to be handled in a way that did not repeat web service calls to remote databases, yet still allowed for the sharing of the results from these calls for some things such as vehicle and customer information. This could be done by writing every result from these calls to local databases which could be accessed by both applications; however this seems like an unnecessary use of both time and space. Local database tables would also be needed to be accessed for the passing of vehicle inspection data to the Job Card application and could not be simply passed between pages.

Another issue with using two separate applications would have been the need to run both applications at the same time to allow for a dynamic work flow between vehicle inspections and Job Cards. This would be easily possible however it seems like an unnecessary use of system memory to keep both open, where the user would have to constantly switch from one application to the other application and back again, with instances of both open. Centralising all functionality into one application would be far more efficient from both a user and a programming perspective for the handling of data and reducing the need to deal with multiple things at once. As one of the primary motivations behind this project in the first place is in fact to promote efficiency in the Fleet Technicians work flow we feared that having to adapt their work flows to deal with two applications may in fact be a step backwards away from this goal. One application would be able to better demonstrate flexible flows in a way that does not irritate the user and require the work flow to be disrupted by switching applications, as well as for passing the necessary information internally between pages for the two portions of the application.

One other benefit of having one centralised Bridgestone application was that it would enable us to also include some other functionality to it such as the Stock Lookup and the Incident and Injury forms that were outlined as potential goals. We were able to allow the user to use all of these functionalities seamlessly from one application as a much cleaner and simpler solution from a user's perspective, and permission was granted from the academic supervisor for both projects to ensure this was allowed. Bridgestone immensely preferred this approach to their projects and was in full support of developing one application, as they had been visualising it this way prior to the development starting.

4. RESEARCH

4.1. END USERS

This project is a fairly practical project with only a few research possibilities when compared to some other research-heavy projects, however as many decisions needed to be made for planning of the application and deciding how it will be carried out the alternatives of these were researched in order to make informed choices as the project moved forward.

On the 30th of April we went on a 'field trip' of sorts to meet the end users of the application. In this case the end users are Fleet Technicians, workers who go out to various locations to monitor and work on fleets of trucks. These locations can be both dirty and dangerous, and are sometimes in locations out of the way where phones are out of range. The purpose of this trip was to meet end users and see the current work process in action, allowing us to ask questions and identify areas where there is potential for our applications to improve the efficiency of their job. After talking to the Fleet Technician I was assigned to and asking questions about different situations I was able to come up with the following figure 12 to show their usual work process, however they stressed that it is not a constant thing as there are many different things that can happen, therefore the final application needs to be appropriately flexible to deal with these situations.

It was interesting to see that there were a large number of different fleets that Bridgestone handles, some very well-known over New Zealand. Examples of these were Transrail, Foodstuffs, Komatsu, KiwiRail, Fulton Hogan (a recently acquired customer), Mainfreight, and TR Group (a rental company so there are always different vehicles at the site, but if they are at the site they usually remain there for a while between rentals).

Usually it takes about 30 minutes between start to finish for a Job Card, but it can take a lot longer depending on the job, such as if difficulties arise or things like changing the larger tyres which are quite expensive and require at least 2 people, 3 or more ideally, and can take about an hour and a half each. Each tyre gets the related Job Card number written on them, so it is quite important that this information be easily visible on the tablet application, possibly on every page in a corner.

The most important pieces of information on the Job Card as perceived by the users are the customer name and phone number, these are the things they make sure to have for every job so that they can contact a customer. Each Job Card is double checked before it is sent off to make sure all required information is present and correct so it may be important to provide functionality to check over jobs done so that the inputs can be double checked, however with some validation I hope to reduce the human errors that can be caused. Different workers do things differently; some do all the work first then write the paper documents later to get the work out of the way; others check out the vehicle first and note everything down before doing any work, so it is important to allow for the application to be opened and used at different parts of the work flow as opposed to being required to be always before or always after a job is carried out.

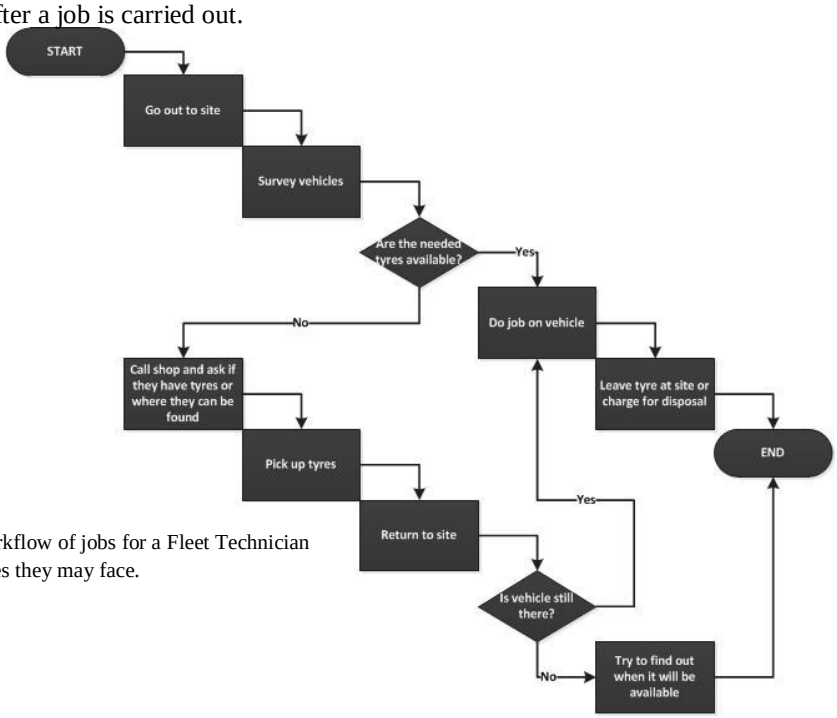


Fig. 12. A common workflow of jobs for a Fleet Technician and some kinds of issues they may face.

Fleeties often have locations they go to first thing where they stay most of the morning checking out the different fleets for issues. This takes pressure off of the customers with the fleetie going to them instead of needing every vehicle to come to their workplace. Fleeties usually have set hours for locations they check. When checking out a fleet first they survey all the vehicles there which can take 15 minutes to half an hour depending on fleet size, looking for things like punctures, bare tread, or mismatching tread on the same axle (this is not allowed due to balance issues). They then get authorization to work on these. The fleetie I was with often wrote notes about the vehicles and work to be done on his hand and then re-wrote it in a notebook later when he could. There is definitely potential here for reduction in redundancy with portable tablets.

If the fleetie does not have the correct tyres for the jobs they will need to pick these up. This can entirely depend on the vehicle they are driving and how many tyres it can hold. The fleetie I was with was working in a smaller vehicle and could only carry around 8 tyres at a time, therefore he went to make note of all work to be done, and then returned to the store pick up the needed tyres. Sometimes these tyres need to be picked up from different places than their usual workshop. If a tyre is indeed replaced the old case is usually left with the company and not taken back to the workshop, though sometimes Bridgestone will buy these cases off the companies and then re-tread them to resell later.

Before going back with tyres the fleetie will usually call the company they are going to in order to double check that the vehicles they wish to work on are still there as often they can be in for only a short time before going out again, it can be difficult to catch them. It depends on the company, for example a rental company will usually have vehicles sitting for a while, but other companies with working trucks only have some vehicles sitting at a location for minutes before they are back out, sometimes work needs to be scheduled in advance for these.

One person a week takes the after hour calls, this means they must respond to all calls outside of business hours at various locations which includes both late night and early morning calls, often working all night without sleep. The Fleet Technicians have a week where they work after hours then at least 3 weeks off on a rotation with other technicians. After they have worked for 16 hours they are required to get at least a 9 hour break for sleep. This is particularly stressful with them lacking staff right now and it putting a large workload on these people during these times.

The busiest times for a Fleet Technician are usually very early morning before people start work before 7am, or late afternoon when work is finished after 5pm. This is due to the fact that during these times fleet vehicles are not as likely to be on the road. During normal work hours is usually the slower time for fleeties and there is more catching up of vehicles and doing whatever work they can do. There are still things to do constantly but there is more moving required as vehicles are not bunched up together in one location as they are at the start and end of a work day.

In addition to going out to fleets, the workshop is open to whoever comes in. This almost always involves people with an existing account but they do get the occasional cash sale. Customers usually have an idea of what needs to be done or a specific request. When the Fleet Technicians are not at the workshop there is an in house GPS system which exists to see where all fleeties are in real time. This is used quite frequently throughout a day. There is a lot of communication and phone calls between the fleeties and the workshop, as well as between the fleeties and the clients, so it is useful to know which fleetie is where.

When taking stock the in store stock-taker must be informed and it must be noted, it is important to keep track of all stock at all times and all changes in stock must go through this worker. Stock has three states being: in the workshop, on a work truck, or used after a job. If a tyre is damaged by a worker it needs to be brought up with the stock-taker and replaced, the worker is charged for the tyre as it is their responsibility. There is a lot of knowledge required for damage reports in terms of identifying cause and effect of damage, they must observe, identify and note this based on what they already know and this knowledge cannot easily be replicated by simple logic in a computer system. After re-treading the new tread does not necessarily match the case, and thus the tread type noted on the case is not always right; workers must be careful. Customers do not usually know this and can misinform the workers if they are talking through a phone call and the worker cannot see the tread.

Often different fleets are located right next to each other, as different companies own spaces that are in close proximity. This allows fleeties to do multiple sets of jobs back to back assuming there is work to do and vehicles are available which is quite efficient. It is also possible for fleeties to be called out by known customers with large vehicles to work on the customer's personal car. However most of their tools cannot be used on them

as they are designed for stronger, larger wheels. For example the fleetie I was with was called out to pump a wheel that blew out on a customer's personal car so that they could drive somewhere where they could get it replaced or repaired.

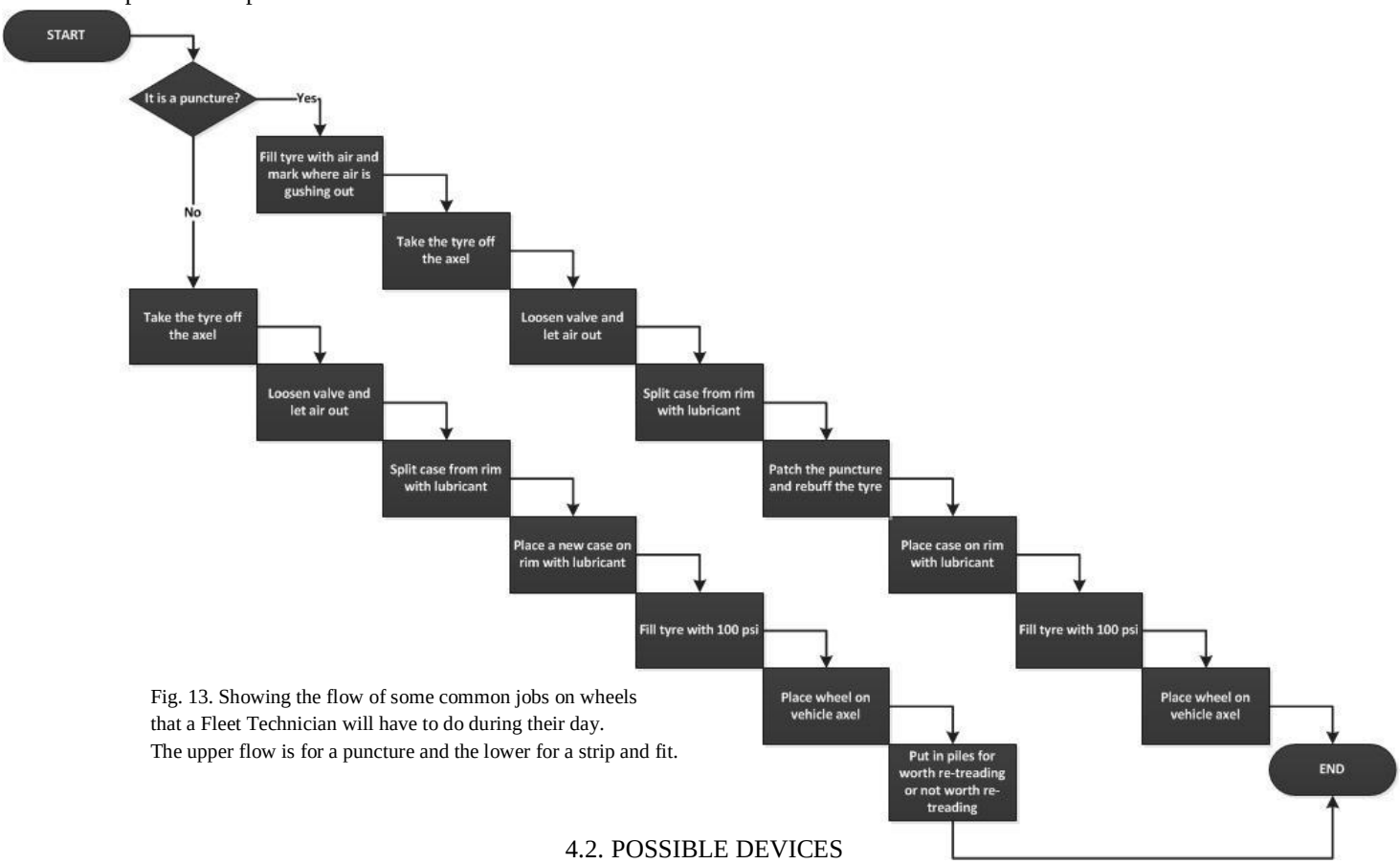


Fig. 13. Showing the flow of some common jobs on wheels that a Fleet Technician will have to do during their day. The upper flow is for a puncture and the lower for a strip and fit.

4.2. POSSIBLE DEVICES

In today's market when developing for mobile devices there are really two main competitors that come to mind, those being iOS and Android, with Windows attempting to break into the industry in a big way more recently. All three of these OSs have different pros and cons, and different rules and restrictions for them, therefore when deciding exactly which device to build the Electronic Job Card application for I first researched the alternatives to gain a better overall understanding of the options. I have researched iOS development by looking at various sources and finding relevant information. Due to the requirements of an apple device for the use of development for this there would have been many difficulties in attempting to obtain one for work both inside and outside of Bridgestone with Bridgestone currently running Windows on all of their computers and my only personal means of using an apple OS currently is by means of using University computers. Android research was a little more thorough with some testing to get familiar with developing for Android as it can be developed on a variety of operating systems and seems to be fairly flexible in both development and deployment. However when it came to researching Windows 8 devices there was not as much information available as they are still new, however on the 22nd and 23rd of March I attended a windows camp, with a 9-5 lecture on the first day and a 2-4 interactive lab on the second. There are also current Windows Phones running an earlier version of the OS that can be looked at.

First looking at Apple, iOS is known for being very smooth and consistent for their interfaces on both the iPhone and iPad, including inside of apps. The Apple Store that is the source of applications distributed on iPhones and iPads is the largest app store currently, and it very well maintained. However in order to keep the reputation Apple has with these things, applications must be relatively consistent throughout. This means there are some rules put in place and there are restrictions on flexibility for developing apps, especially if you wish for them to be distributed.

An example of this is in interface design, where Apple has a list of 'Interface Guidelines' [15] that influence all applications that run on their devices. While this can be quite useful in terms of design tips, it is also somewhat restrictive for those that wish to have more freedom with what they develop. These guidelines include such things as the characteristics of the platform and how to best integrate these for a successful app; interface principles which define how well an applications appearance integrates with its function as well as if the app is consistent with iOS standards; strategies for designing apps based on iOS UI paradigms, and guidelines to elevate user experience based on using iOS. Overall while they have the users in mind when giving these guidelines and standards and the end result is a very consistent and clean platform, it can also be restrictive.

One thing that separated iOS devices from others is that the range of hardware that supports iOS is very limited. Official Apple made iPhones and iPads are the only devices which will run Apple's iOS and therefore users have little choice in their purchase and device specifications, instead they only have updated hardware versions such as iPhone 4 and iPhone 5. Again this is somewhat restrictive however the hardware is trustworthy and secure, also while smartphones in general are known for having bad battery life Apple devices are not known to be better or worse than their competitors, they are just a solid and trustworthy middle ground.

Originally in New Zealand iPhones were only available for use with Vodafone as the carrier, however they have since become open for use on Telecom networks as well, and are compatible with 2 degrees micro-sims though they are not official carriers. Therefore the use of networks for Bridgestone will not be an issue here.

In order to develop for iOS there are specific hardware and software requirements that must be met. Xcode and the iOS SDK are required to develop apps, while free, are only usable on Apple operating systems [16]. In addition to this the programming for these apps must be done in Objective-C, which is a superset of C. This means that being unfamiliar with this language, I would need to learn its conventions in order to program for Apple devices. This would not be too difficult as Objective-C is a simple object-oriented language with easy to learn syntax, but I would need to get familiar with them none-the-less.

Using Xcode and the iOS SDK developers are able to create and simulate their apps, however they are not able to distribute them through the App Store or as in-house iOS apps. For this the appropriate developer programs must be purchased. As the Electronic Job Card is to be created as an in-house app the iOS Developer Enterprise Program would be needed [17]. This would cost Bridgestone $299 USD per annum, which using June 2012 market rates would be an estimated $395 NZD per annum. Without this program we would not be able to test the app on a device and would only be able to simulate it with the simulation tools provided by Xcode and the iOS SDK.

Looking at the same areas for Android devices we can see a clear increase in the flexibility of both the devices and the applications used on them. Android has been developed by the Open Handset Alliance led by Google to be an open source smartphone and tablet operating system, allowing users more visibility and innovation with what they are doing when they create an application.

Because of this flexibility there are less restrictive guidelines on how an application must be created or standards which must be met for the interface than we saw with Apple. Android lists guidelines for creating icons and widgets for the created applications so these may look somewhat clean and consistent [18], however there are less restrictions placed on the look inside the applications themselves. They do provide design principles for user interfaces, but they give more freedom to the users than the iOS principles.

Continuing with the flexible theme there are a great variety of devices that support Android as an operating system. A number of different companies are able to create hardware for Android, leading to a more competitive smartphone market, allowing users to pick a device that has specifications that more suit their needs. Of the companies that create these devices Samsung stands out as one of the leading manufacturers and are known for providing quality products. There are other devices created by other companies that are also more economical options providing the core functionality of a smartphone if this is what a user is looking for. Because of this variety in hardware there is clear variance in the device specifications, including that of battery life. It is difficult to say if Android is more or less battery intensive than iOS due to the fact that different Android devices will be using different batteries that handle stress differently, you can see some Android devices with poor battery lives well below that of an iOS device, while you can also see some with greater battery lives [19].

Android devices have been open for use on the primary New Zealand networks, being Vodafone, Telecom and 2 Degrees so again Bridgestone should encounter no issues with this.

As far as coding for Android goes, it is quite simple to set up the Android SDK on most PCs. One such way of doing so if through the Android Development Tool for Eclipse, allowing for simple programming of Android apps inside Eclipse. Programming for Android with the SDK is primarily done with Java, which is a language I am somewhat familiar with from my studies at the University of Auckland. However with the Android NDK companion tool it is also possible to code in different languages such as C or C++ [20]. While I am less familiar with these languages the flexibility is nice for developers to have, allowing them more freedom to develop in a way that feels more comfortable to them, however some developers are not aware that this NDK is available as Java is the primary programming language for Android.

In order to publish an Android application through Google Play, a store similar to Apple's App Store, you must create a developer profile and pay a registration fee, similar to the developer programs Apple has. However this is not the only way of distributing Android apps as Android also allows for apps to be e-mailed or made downloadable on developers own websites outside of Google Play, and therefore no fee is required [21].

It is slightly more difficult to analyse all these areas for Windows 8 as the Windows 8 platform is still in development, however a lot of information can be obtained from the current Windows Phones running the Windows Phone 7 OS. In addition to this research a lot of information was gathered from attending the Windows Camp held in Auckland. Windows Phone is not open source like Android is, and therefore does have its restrictions. It seems to have a strong focus on consistency aiming for an approach similar to what Apple has done, being solid and consistent as opposed to flexible, so that they are able to promote stability.

The new Windows systems have been putting emphasis on what they call 'metro style apps' for interface design. They have created the OS with touch screens in mind and have attempted to provide developers with all sorts of tools necessary to create consistent apps that take full advantage of their OS [22]. Windows likes to explain their metro style apps and has guidelines on how to develop them. This restricts users similar to how the Apple apps are handled however it has worked quite well for Apple so it is quite possible that metro style apps will take off with users after the full release of Windows 8 on both PCs and other devices.

The standard Windows 8 operating system is designed for use with both normal PCs and tablet interfaces therefore there is expected to be a growing list of possible tablet devices available that support this operating system. There are already multiple different models of phone that run the current Windows Phone operating system such as Nokia devices. So while Windows places restrictions on how their apps are created like Apple does in order to promote consistency, they allow for a variety of devices with different specifications to meet their users' needs like Android. This means there will be options for buyers of different budgets and differing needs in a competitive market. This also means that the battery life of Windows Phones will be varied depending on the device the operating system is placed on as opposed to a constant across one device.

It can also be confirmed based on the current Windows Phones in the market that Windows Phones will be able to connect to the networks all three leading carriers in New Zealand, those being Vodafone, Telecom and 2 Degrees. Current Windows Phones such as the Nokia Lumia 800 support all networks [23], therefore again Bridgestone should have no connectivity issues with these devices.

The current development tools for Windows 8 are still being worked on and therefore the development toolkits are available free for use until their release. The windows 8 labs held in Auckland focused on using Expression Blend for interface design and Visual Studio 2012 for back end programming for the Windows Phone labs, and Visual Studio 11 for both front and back end programming in the more general Windows 8 labs. Both of these approaches are possibilities for the creation of a final product. These tools include the Windows 8 SDK. Microsoft also hopes to provide developers with the flexibility of deciding how they wish to program their apps and provided guides for their labs at the Windows Camp in both HTML with JavaScript or in XAML with C#, VB or C++, they also allow metro style apps to be created with DirectX and C++ [24]. With all these options available to developers from the start Microsoft are allowing developers to feel comfortable with developing apps for their OS, and unlike Android they do not need to find out about and download the extra NDK.

In order to distribute applications developed for Windows 8 operating systems a Windows Store developer account is required, and is not optional, similar to how Apple handles this. Different prices are required depending on if the registration is for an individual or for a company, and differs by location. For New Zealand the annual company registration fee is expected to be $140 NZD [25].

After these looks at each operating system in theory research was done in comparing the development possibilities, mainly Windows 8 and Android systems. The main reason further research was not done into iOS was the difficulty that would be encountered with consistency with Bridgestone systems, currently they use Microsoft operating systems in their business and requesting an Apple computer for use in developing this application would not be a reasonable cost. One of the project supervisors did offer to lend his Apple Laptop for the use of this project however this was the only device that was identified as usable for this development inside Bridgestone, and in addition to potential issues in borrowing it for the whole year there was also a problem due to the VIR portion of the application needing to be developed side by side, a single computer would not allow for such development. It was looked into as an option anyway in order to better inform both myself and Bridgestone of the options however it was not identified as a primary candidate for development.

For experimentation with Windows 8 and Android devices a simple application implementing a simple web service was attempted to be created on both systems, with varying levels of success. As I had already created the initial registration lookup application detailed in section 3.4 before doing these tests I was already somewhat familiar with Visual Studio and using web services inside this, however not for Windows 8. I created a simple application on a PC running Windows 8 in order to get an idea of how development in this environment would feel. This application seen in figure 14 used simple web service calls to read data based on an input. This was programmed using a C# backend and a XAML front end, using Visual Studio's UI designer to create and modify the XAML interface; I was comfortable using C# for this as it is what most of my more recent assignments at the University of Auckland were programmed in. I felt that using this for interface creation went much smoother than I expected with simple drag and drops of new elements that needed to be placed on the screen and the property editor for each tool giving the ability to modify how it functions. While modifying the UI in this way felt very natural I also appreciated being able to use a split view to edit the XAML while viewing the effects on the interface for more specific changes, such as correct values for the alignment of elements and configuring properties that I had already used without the need to browse the properties box to find what I was looking for. Programming the C# back end was also not overly difficult, and I found myself using Visual Studio to autocomplete many of my references as well as using its object explorer for easy visualisations of the services I was using and what methods and overloads were available for the calls I needed. One issue I faced when using Visual Studio 2012 was that because it was using .NET 4.5 framework some things I tried to reference were not able to be used in the same way I expected, however Visual Studio did provide me with suggestions on what to include in my page as the correct reference in some of these situations.

I installed and Android Development Tool for Eclipse and attempted to work through the same process in creating an Android version of this simple application. This time I used Java instead of C# as that felt more natural for use with Android, not requiring the NDK to be installed. While I do have experience with Java form previous University courses these were mostly done earlier on in my degree so I felt slightly lost in trying to remember some specifics, however this was not a big issue. Eclipse made up for any Java programming difficulties with more efficient building and debugging than Visual Studio, as the application automatically built itself in the background every time changes were saved, and this was done fast enough to be hardly noticeable. This enabled debugging to proceed more efficiently due to not having to re-build the application when it was debugged. This meant I did not have to waste time waiting for the project to build and could quickly get back to coding. Without the same kind of UI design tools that are available for the interface I had more difficulties both visualising and constructing this inside Eclipse. This was my main issue with developing for Android. Some third party add-ons were available that enable some form of interactive interface design however they were not nearly as intuitive or flexible as that in Visual Studio, which did not require the installation of any other tools. Visual Studio also did not require something like the Android SDK to be installed in order to start programming for the tablet, which was not a major issue but still one difficulty with developing for Android. Android is able to be developed on different OSs not being restricted to Windows, however it is not likely that any systems not running Windows will be used throughout the year.

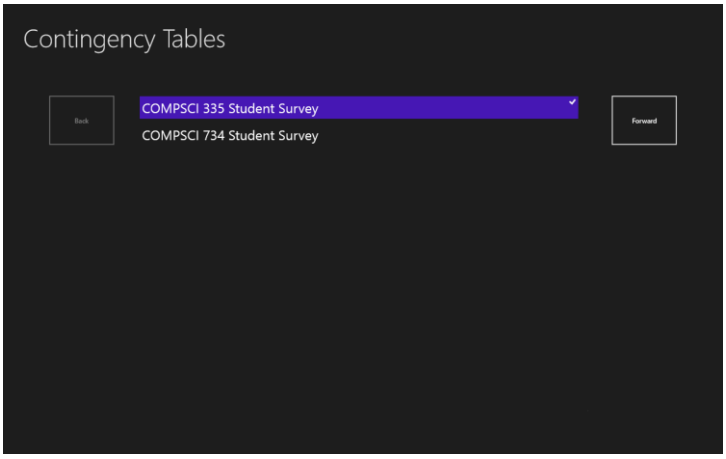


Fig. 14. A simple interface created for the application used in testing Windows 8 and Android development.

In my personal experience with developing for both I found development for Windows 8 to be somewhat easier, predominantly due to the Visual Studio editor being easier to work with than Eclipse used for Android. The actual programming language of C# or Java was not much of an issue as I only have very specific experience in both from University courses so the project would be a learning experience regardless of the language. One other worry that I found with Android was that it was brought to my attention that a lot of work was currently being put in to fixing security issues with Android applications [26, 27]. This is due to Android not having native fine-grained security on the device, either allowing installed applications permissions to access any data on the device, or none at all if it is not installed. The device cannot pick and choose which applications can gain access to which pieces of information without the use of extra security extensions such as those seen in [26] and [27]. These research papers go into detail about methods of adding the extra needed security to Android for use in businesses where security should be a primary concern, looking at methods such as giving permissions based on context, where context switching based on the current location, time or other criteria can either tighten or loosen the security functions of the device.

After discussing the options with my supervisors at Bridgestone they definitely preferred to work with Windows 8, which matches up with my personal experience with the devices. One Bridgestone employee had already been looking at some Windows 8 tablets that he felt were robust and solid enough for use by Fleet Technicians in a work environment where the possibility of damage to the tablet is quite high. Working with a tablet in such conditions was a concern and this kind of tablet seemed to be a perfect solution, giving further motivation to develop for Windows 8. If development was done on Android there was also the alternative of buying an external case for an android device, as the devices themselves are not very sturdy but cases can be purchased for fairly low prices that give these devices more stability. Bridgestone also informed me that they currently have an enterprise agreement with Microsoft making the licensing of Microsoft based devices a very simple process in their business, not requiring as many hoops to jump through in order to put into practice, allowing for us to start development earlier than would have been possible with something like iOS which would have required permission and acquirement of Apple based devices to be used inside a Bridgestone office work environment. There was also a concern in using non-Microsoft devices due to the currently employed IT staff in Bridgestone having expertise with Microsoft systems and if development was performed for something else there would have been a requirement of additional training for Bridgestone staff to handle the devices and application in the future.

In the event that Bridgestone decides in the future to want to use iOS or the Android OS I have looked into some options for code portability. The primary option that seemed available for this was to use MonoTouch for iOS and MonoDroid for Android [28]. These tools allow for cross platform development with C# and .NET development of Android or iOS applications, allowing for the importing of existing .NET libraries. MonoDroid even offers an add-on for Visual Studio, using a local Android Emulator to test the application. Another option that was available for Android would be to use Windows 8 for Android [29] allowing for an Android device to simply emulate Windows 8 itself. Interestingly enough Android apps are usable on Windows 8 so in the event that Bridgestone creates and builds on this Android version of the application and later decides to return to a Windows 8 device it would be possible [30].

4.3. WEB SERVICE DESIGN

As the Electronic Job Card application will need to interact with the Bridgestone databases I needed to look into creating and consuming web services for these data transfers, therefore I must decide on an approach to building web services. The possibilities for this include SOAP or POX protocols, or the RESTful architecture for web service design. As SOAP and POX are protocols while REST is an architecture they cannot be directly compared, however performance and ease of development between them can be [31]. All of these methods have their own pros and cons and I have researched these ways in order to gain a better understanding of my options.

SOAP has been around since the late 1990s and is still the prevailing standard for web service design. It is known to be safe in terms of error handling and is able to transfer attachments without problems. There are set contracts that SOAP must adhere to such as having a compulsory body section with an optional header, giving it more structure than REST and making it very robust [32]. Due to this SOAP is often considered better for the purpose of consistency as it is more standardised.

POX is a similar protocol to SOAP in that both use only POST when invoking web services [33]. This means the XML must be parsed first in order to figure out the web service name and parameters. Both the advantage and the drawback POX has when compared to SOAP is that it does not use the elements SOAP does, what this means is that while this reduces the size of the sent message, it forgoes quality of service. POX may be used instead of SOAP if a message was required to be small and quality of service did not matter.

REST on the other hand can be considered simpler than SOAP as has less reliance on tools, everything is simply a resource accessed through a URI [32]. REST also does not encapsulate the messages in XML like SOAP and POX do so the data usage is lower than that of POX [33]. The messages sent are very simple (GET, POST, PUT, DELETE) and because of these, it is easy for a program to see what the web service is doing. For example from a security perspective if a REST GET message is sent it can only query data and will always be safe, however if a SOAP POST message is sent it is impossible to tell just from the POST flag if it will be just a query or a more potentially destructive instruction, you must look into the message for this [34]. Another potential advantage given to REST over SOAP and POX due to not encapsulating its messages in XML is that even web intermediaries will be able to read the messages, allowing for the efficiency of caching which does not happen with SOAP and POX messages as they are protocol agnostic. As mentioned REST has simpler commands and does not require an XML wrapper around every request and response like SOAP does; it is more lightweight and uses less bandwidth at the expense of less structure. REST has smaller request messages than both SOAP and POX and also smaller response messages than SOAP, therefore on average REST should have quicker response times than SOAP [33]. The characteristics of SOAP and REST are compared in table I. Some major companies such as Yahoo and Google opt to use REST [32] however there are still some flaws such as it lacking the helpful tools and W3C standards SOAP has and is more difficult to develop sophisticated requirements for.

Table I

Characteristics of SOAP and REST from [31]

	REST	SOAP
Characteristics	Operations are defined in the messages Unique address for every process instance Each object supports the defined (standard) operations Loose coupling of components	Operations are defined as WSDL ports Unique address for every operation Multiple process instances share the same operation Tight coupling of components
Self-Declared Advantages	Late binding is possible Process instances are created explicitly Client needs no routing information beyond the initial process factory URI Client can have one generic listener interface for notifications	Debugging is possible Complex operations can be hidden behind façade Wrapping existing APIs is straightforward Increased privacy
Possible Disadvantages	Large number of objects Managing the URI namespace can become cumbersome	Client needs to know operations and their semantics beforehand Client needs dedicated ports for different types of notification Process instances are created implicitly

This table shows some key differences between SOAP and REST

I performed some tests consuming three different calls from a web service with the same end point with both SOAP and REST. For each call I gathered 20 response times, that is 20 response times for one of three methods, for both SOAP and REST, totalling 120 recorded times. I performed this many calls in order to better give an idea of the average response time for these services, also it is important to note than when collecting this data I ignored the first readings of these calls, as this was always far longer than subsequent calls, this would be due to needing to establish a connection to the endpoint to make the first call which subsequent calls would not have to do as long as the connection remains open throughout this test. These calls were done in a controlled environment as a sort of unit test, testing different ways of implementing web service calls external to the application with results that would help inform my decision when developing the finished product. The three calls I used all provided slightly different response messages between them, but the 20 calls done for each method were kept consistent; this means that the response times between the three different methods should not be compared, however comparison can be done three times for the three methods for SOAP and REST. The results of these tests can be seen in figure 15 below. Note that XML was used for the tests with REST not JSON.

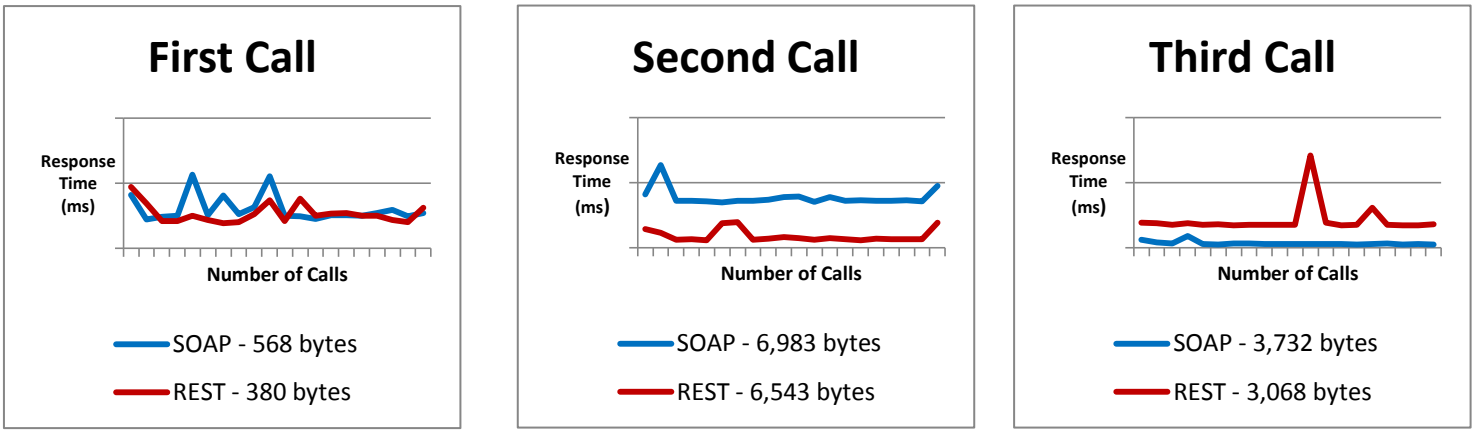


Fig. 15. Response times for three SOAP and REST calls based on tests performed.

As you can see on all three calls the number of bytes the call was for was slightly lower for REST than it was for SOAP, with a 188 byte difference in the first call, a 440 byte difference for the second call and a 664 byte difference for the third call. Ignoring the graph for the third call for now and only looking at the first two we can see a definite trend in REST having lower response times than SOAP, as you would expect with the lower bytes transferred. There are values in both of these that are further away from the trend line than others, however for the most part the lines are fairly straightforward with consistent values, especially in the second call for the larger amount of data. In the first call the average response time for SOAP was 60.35 ms, while for REST it was 53.2 ms, so even though these values are similar REST was still faster on average by around 7 ms. In the second call where the lines were more distinct the average response times for SOAP and REST were 386.4 ms and 90.45 ms respectively. There is a much greater difference here, REST saving on average around 300 ms. All of this is entirely consistent with the theory of REST and SOAP as discussed above as researched from the academic papers referenced. The third call produced some very strange results, with REST consistently requiring more time to get a response than SOAP and even having one outlier where a response time of 1,419 was measured while the other values averaged 370.42 ms. After further testing and analysis of both the SOAP and REST service calls and the collected data I have continued to obtain similar results. I have been unable to account for the disparity between the data collected for this method and the theory, especially taking into consideration that the REST response is still 664 bytes lower than that of the SOAP one. The request made is slightly more complex than the other two, requiring two values to be sent and having to dive deeper to pull back information, however this is the only difference I have been able to see and I cannot explain why this would cause these results. In my decision making based on this data I have chosen to ignore the third call entirely due to this.

The tests carried out may not have been ideal due to not taking into consideration the extra time needed to establish a connection, as all values were gathered while a connection with the endpoint was still open, it may have been more beneficial to gather data after the connection had been closed for each test in order to better give

an idea of what the web service calls would be like in a real environment where they are not handled in quick succession, however for these tests I felt this would not only take more time but it may also cause the results to not be consistent due to external factors potentially changing the results, for example changes in other network traffic over time.

While REST was indeed identified as being faster in theory and in most tests I personally carried out, as well as consistently requiring less bytes to be transferred, over the course of developing this application the focus of the calls drifted further and further away from efficiency and closer to functionality. In addition to this the current web services in place in Bridgestone for interacting with their CRM and Vehicle Repository both implement SOAP as shown in section 3.3, figure 6. Due to this the current employees at Bridgestone were more familiar with using SOAP and would potentially require training for REST should that be implemented with this application. The current CRM and Vehicle Repository services would also need to be modified in order to be compatible with REST should it be implemented for those web service calls. Taking these into account as well as the potential other benefits of SOAP such as the consistent structure and standards it uses we decided to continue to use this method consuming web services for this application. It was identified that while it may be slower it should not be considerably so, as in the tests I performed it was less than a second quicker for the largest difference. This difference is thought to be acceptable for the application, and large data transfers for things such as potential synchronisation these could be performed via overnight processing similar to how Bridgestone currently stores vehicle data into their data warehouse, making time not an issue for these larger tasks.

4.4. DATABASE OPTIONS

As this project needed to deal with the storage and access of data for populating and recording various forms the need for databases was one of the primary requirements. Bridgestone has several of its own databases as seen in section 3.3, however local databases on the tablet also needed to be created and used in order to promote offline functionality of the application. There are a number of options for database management systems (DBMS) in the market today, some common ones that I identified as options being MySQL, SQLite and Microsoft SQL Server. MySQL is the most popular open source database due to its cheap cost, ability to run on multiple platforms, and ease of use [35]. It can be thought of as similar to Android in this way, in that being open source and available for a variety of operating systems the use of MySQL is very flexible. The MySQL Workbench is available as a GUI tool for the creation and manipulation of tables in a MySQL server. Looking at SQLite is somewhat different in that it is what is known as an embedded database engine [36] for client side storage integrated with applications. This means that it does not run as a separate process and instead is linked in to be integrated as part of the application itself, making it more lightweight than most others. Microsoft SQL Server is a Microsoft based product that is presently in use by Bridgestone for management of their current databases. It is a fairly robust management system with a variety of different versions all targeted at different audiences. Another Microsoft product SQL Server Management Studio is also used by Bridgestone which is a simple GUI tool that provides simple visual managing of database tables, allowing for manipulation of data in a view that is more consistent with users mental models of data relationships. This software again is very robust and can not only read Microsoft SQL Server databases but also has functionality to link to other servers such as a MySQL Server [37].

In terms of functionality required for this project all three of these options among others would all be viable, but a comprehensive comparison of these three can be seen in [38]. Due to most of the functionality concerned in this comparison being unnecessary to account for in this application and taking into consideration the choices that have been made earlier on in this section to create a Windows 8 application I felt that Microsoft SQL Server would be the simplest DBMS to use in this project. Making this choice also means that we were able to get basic training from employees in Bridgestone who are familiar with Microsoft SQL Server databases and manipulating them via Microsoft SQL Server Management Studio.

The version used at Bridgestone while developing and testing this application was SQL Server Standard, which is a paid application with complete core functionality for these SQL databases, however there are other versions available as well. Two of these other versions that were researched were SQL Server Express and SQL Server Compact, both of which are free, and both of which are able to be managed through Microsoft SQL

Server Management Studio. Both of these provide less functionality than the Standard edition of SQL Server, however for the use in this project they have been identified as appropriate for use. SQL Server Express is the most common free SQL Server edition due to being simply a scaled down version of the core functionality [39]. SQL Server Compact on the other hand is a specialised edition that is designed primarily for use in mobile devices such as in this project. Due to this SQL Server Compact has built in synchronization capabilities using the MS Sync framework which has potential use in this project for synchronising local and remote databases for offline functionality. The databases created with SQL Server Compact can also be encrypted and given password protection as this may be an issue for data storage on mobile devices that are not always kept where Bridgestone is able to monitor them [40]. I would suggest SQL Server Compact as the best option for use on multiple tablets if this application is to be deployed to a number of devices due to these functions being readily available, and also from a cost perspective as unlike SQL Server Standard, Compact is free to download and use with all necessary functionality.

Stored Procedures were looked into as an option for writing to the local databases, and Microsoft SQL Server Standard as was used in testing this application does indeed provide the ability to use these. Stored procedures are subroutines stored inside a database for easy and quick access. Using these would promote efficiency of the frequent used calls to databases, such as the frequent inserts being used when writing Job Cards of Incident and Injury reports to the databases. The one reason we did not implement these into the final application was due to the fact that SQL Server Compact does not support them. As SQL Server Compact was identified as having potential in future implementations of this application on tablet devices it was decided that SQL statements would instead be handled through the application, making service calls using appropriate connection strings and hard coded SQL statements, this means that if Bridgestone does end up using SQL Server Compact in the future they should not be met with any issues with the application accessing and modifying the local databases.

During the development process of this project a number of database tables were created for use by the application, each inside a central database specific to the app. These can be seen in figure 16. The ConfigurationData, Wheel and Damage tables seen are primarily used by the VIR portion of the application for writing and reading VIR data created by the application. The main Job Card portion of my project deals with the JobCardTable and the ItemTable. The JobCardTable for data concerning each Job Card and the ItemTable for each item that is attached to this Job Card as work to be completed. The IncidentTable and InjuryTable hold data about Incident and Injuries from the forms filled out by the application, with a one-to-many relationship between a single row in the IncidentTable and multiple rows in the InjuryTable, for each Injury form filled out for a single Incident form. All other tables seen here are snapshots of some of Bridgestone's current databases to help the application function in an offline environment.

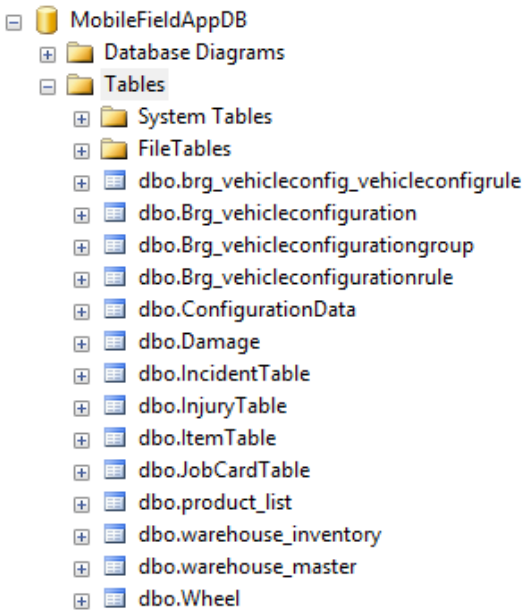


Fig. 16. The local database and tables that were used by this application.

4.5. DEPLOYMENT

One concern that was had at the beginning of development of a Windows 8 app was if there would be the ability to deploy the app outside of the Windows Store. The primarily advertised method of app deployment for consumers is via the Windows Store, which both an individual or a business can obtain accounts to do so, however for businesses it is likely that software would rather be kept internal. Windows offers the ability to sideload applications [41] which enables them to be deployed without being certified or installed via the Windows Store. In order to sideload applications the app still needs to be converted into an app package using the Windows Store, but this app package can be installed on sideloading-enabled devices. Enabling side-loading on devices is a very simple process of simply changing one group policy option on a device that is domain joined. I tested this option on a Windows 8 device and had no problems toggling the group policy to allow trusted apps to install. Once the device has been appropriately set up the application can be installed either at runtime or with a windows image allowing for the app to be installed for every user of that Windows installation when they log on [42]. The one problem with sideloading in this way is that instillations must be manually done on a per-device basis for each device to be used, whereas if the Windows Store was used the devices would be able to install the application themselves and obtain updates with a connection to the store. While instillation may seem tedious for sideloading, especially in the event of a simple update which again must be handled on a per-device basis, this should not be too large of a problem in the near future as the application should not need to be deployed to too large a number of devices.

5. MODELLING AND ARCHITECTURE

5.1. TECHNOLOGIES USED

After becoming familiar with the systems in place at Bridgestone and both planning and research had been done as in sections 3 and 4, it comes to the point where specific technologies on which to use in building the application must be identified. As it was chosen that this application would be created for a Windows 8 tablet the technologies which were available to us were somewhat restricted due to needing to be compatible with a Windows 8 tablet, however this did not prove to be an issue. A number of technologies were necessary, the first of which is a code editor with which to work on.

As Windows 8 development is being advertised to primarily be carried out using Visual Studio 2012 it seemed natural to precede with that software. Visual Studio 2012 Developer Preview was free for download with a MSND subscription alongside a beta version of Windows 8, however a Microsoft Developer's licence was required in order to debug and run the applications created using this software in its beta state. Bridgestone was able to obtain a Developer's license for use in this project allowing the use of Visual Studio 2012 on a PC running Windows 8. Using this code editor the two main options of languages that the application could be coded in as identified at the Windows Camp tutorials were C# and XAML or JavaScript and HTML, however as most of the more recent university courses that I had performed programming in were using C# it was decided that this route would be the best to take for development. Visual Studio 2012 being somewhat new did not have full functionality and there were some minor options missing that I noticed throughout the project, however for the most part these were not necessary for progressing with the application or an alternative was available. One primary example of this was with Date Pickers not being part of the toolbox for XAML interfaces, while they were indeed available for HTML interfaces. As C# and XAML were to be used this was a problem, as Microsoft had not yet given this tool for use in our interface, though likely will be in the future as indicated by the existence of an HTML tool. Microsoft Expression Blend was also a piece of software identified for potential use for the design of the front end interface of the application as used in the initial application in section 3.4, and the Developer Preview of Expression Blend for Visual Studio 2012 was made available alongside the Windows 8 beta and Visual Studio 2012 Developer Preview. However the process of creating the interface in Blend and combining this with the backend from Visual Studio as experimented with in the initial application seemed to have a steep learning curve and was a difficult process to initially understand. In addition to this after attempting some simple interface design in Visual Studio in an attempt to make a Metro Style App it was determined that Blend would not be required as Visual Studio was more than robust enough for the job of creating the kind of interface that was needed with the right skills.

SOAP UI was a piece of software commonly used by the Bridgestone employees that supervised the project, as it provided an interface to very easily visualise and experiment with web services, allowing for quick and easy identification of the Requests and Responses and the information included within them. This piece of software was introduced to me early on in the project and was used frequently throughout the year for the testing of Bridgestone's currently in place web services in order to ensure that the data being sent and received was indeed what it should have been. SOAP UI was primarily designed to deal with the sending and receiving of SOAP envelopes however its functionality also extends to using REST, allowing for the testing of those should they be used in the future. SOAP UI also gives feedback on the data size of the responses and the response time of the request, providing useful analytical insight.

For the use of database table creation and manipulation SQL Server Management Studio was used, offering a simple graphical interface for interacting with and managing database tables within Microsoft SQL server. Again this software was already in use inside Bridgestone New Zealand and was introduced to me early on in the project, I was able to quickly learn how to create tables, modify them, perform simple SQL commands as well as backup the states of tables to ensure I had something to return to in the case that tables or table rows were modified incorrectly. This software provided very easy construction of SQL statements for selecting rows to display and to create tables should they need to be re-created, as there were methods of producing the SQL statements for these without having to type out the statements in full every time, which was immensely useful.

5.2. WORK FLOW

After having gone to meet the end users and seeing how they carry out their job, I was able to put together a well-defined proposed work process. The work flow diagram shown in figure 17 can be thought of as showing the scope of the application giving an understanding of what functionality must be implemented and how these will be used. To go over the steps in figure 17 the start of a job will always be after a vehicle inspection has been done identifying work needed, be it a complete VIR or not. Therefore the first thing that needs to happen is to read information from the VIR portion of the application; this will mainly be customer and vehicle information. This relevant information is readily viewable in the new portion.

Initially when starting a new Job Card the user is able to either start a new Job Card based on a previously recorded VIR, or they are able to load a Job Card that they were working on previously, to either check and modify information or to continue where they left off if it was incomplete. Based on the data received by the VIR application the appropriate vehicle configuration diagram should be displayed for each vehicle and this should be able to be changed or entered if necessary. The Job Card should visually prompt the user where work may be required on what wheels based on data from the VIR however the specifics as to what is used to do this job will need to be entered into the Electronic Job Card application. After the user can see what kind of job they will be potentially carrying out on a vehicle at this time they will be able to input the Job Card info. This will be the specifics of the job detailing what was done on what wheels with what products. This is separated into two main portions of Vehicle Items or Wheel Items, with as many of each able to be added as necessary. The user must be able to add information about the current job, so fields must be properly specified as optional or required allowing the user freedom for any kinds of jobs they may come across, but with only relevant options available in order to minimise potential mistakes. The application will use dropdowns and some equivalent of radio buttons and checkboxes where appropriate if there are predetermined options for a specific input, this enables the user to both understand what is being done and help reduce the time spent entering information.

After having the information about the job to be carried out completed we will be able to identify what stock is required for the successful completion of the job, and from this the user will know if they have the stock with them or not. If they do not have the required stock I will provide functionality to perform a Stock Lookup, checking databases of stocks by location to show where the needed stock may be available to be obtained from, this is in hopes of making the process of locating stock more efficient. This Stock Lookup will not be compulsory and if the user desires they will be able to find stock on their own by phoning other branches or Fleet Technicians.

After the Job Card is ready for a job to be carried out the user should be reminded that they are to gain permission to carry out the jobs on some vehicles, and not to continue unless they are sure they have this permission. If they need to obtain this permission the option of storing the Job Card to continue again later after permission has been granted should be available, otherwise they should be free to complete the Job Card. At this point ideally the actual physical job will be carried out, this step is not done in the application at all and is just mentioned for where it should be handled in an ideal work flow.

After the job is completed all information used in the job should be saved to relevant databases for Job Card and item information for the items used in the job. Transactional data should be saved in a way that allows for the POS system to produce an invoice for each job handled in this way, but not force the invoice to be created. In the event that there is no connectivity to send this data live the application needs to provide functionality to save a Job Card and send the information at a later time without the need to re-input data concerning the jobs.

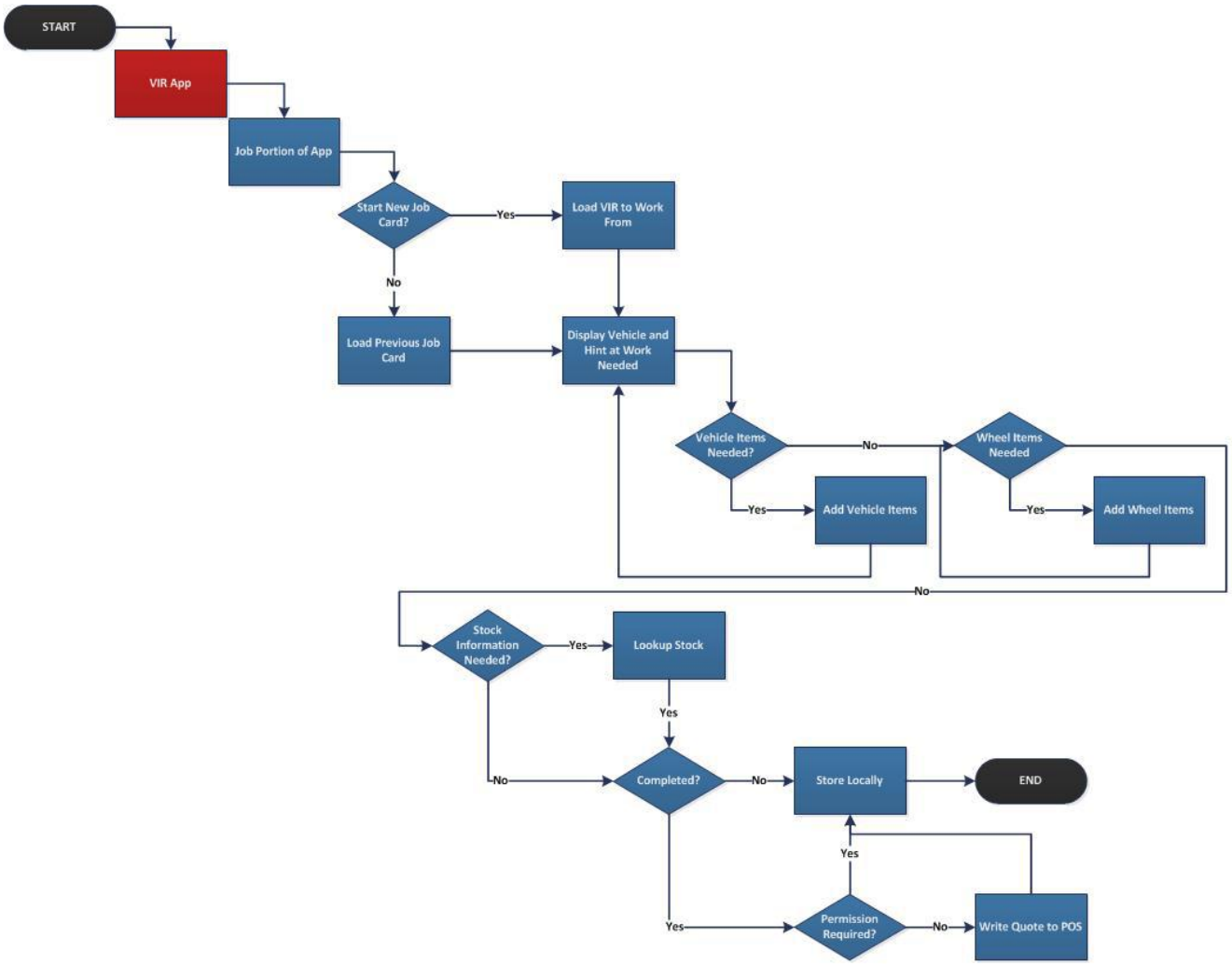


Fig. 17. The proposed work flow of the Electronic Job Card system and the functionality it provides.

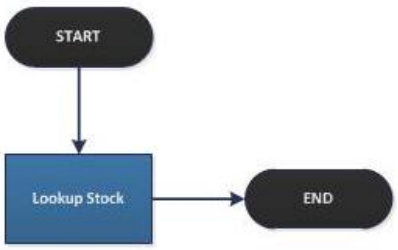


Fig. 18. The proposed work flow of the Stock Lookup functionality.

The Stock Lookup does not have to be accessed as part of the workflow during a Job Card, and can also act as a standalone work flow in case the technician wishes to look up stock while not in a current Job Card, for example between Job Cards or after storing one if they forgot the location. This screen should be exactly identical to the one accessed inside the Job Card as to avoid confusion and ideally should be the same screen accessed from two different places, that can go back to where it was accessed from.

Fig. 19. The proposed work flow of the Incident and Injury reporting functionality.

As far as the Incident and Injury forms are concerned it is quite a straightforward workflow starting at opening the Incident portion of the app. Following this the user must input relevant Incident information, with the type of Incident being compulsory as one of the keys for use in the database. The date should be automatically recorded based off of the tablets clock, forming another key for an Incident form, and the number of people Injured is necessary for the application to know how many Injury forms that need to be filled out. If no people were Injured and no Injury forms are needed then all the Incident information can then be stored and the workflow will finish. However if people were Injured more information needs to be input for each person Injured. The same kind of Injury form needs to be completed for each person, but still be attached to the single Incident form. The user should not be able to record more Injuries than they identified happened, and should not be able to fill in less, so every Injury identified needs a compulsory Injury form.

The data used by this application can be separated into four sections, data that it obtains from the VIR application and data that is unique to just the Job Card and Incident portions as shown in Table II.

Table II
Data available to the Electronic Job Card application

Data From VIR Application	Data Unique to Job Card	Data For Stock	Data for Incidents
VIR Number	Invoice/Job Number (using blocks where needed)	Item Description	Date (from tablet clock)
Salesperson	Date (from tablet clock)	Item IPC	Type of Incident
Customer/Account Name	Work Done (Qty, Service, Remarks/Size, Unit Price can be auto calculated)	Item Location (address, phone number and quantity are the important aspects)	Number of People Injured
Customer Address	Equipment Used (Qty New/Rtd, IPC, Brand, Pattern, Unit Price can be auto calculated)		Incident Details
Vehicle Registration Number			Injury Details (For each person)
Vehicle/Fleet Number			
Make/Model			
Odo/Hubo			
Vehicle Configuration (indicated by diagram)			
Tread Depth (for each tyre)			
Account, Customer and Vehicle GUIDs			

This table shows the general idea of different pieces of data used by the Electronic Job Card application

Most of the generic information about the customer and vehicle is given to the Job Card portion from the VIR portion, while most transaction information is specific to the Electronic Job Card, this is because a VIR does not result in a transaction. Of the data unique to the Job Card the Invoice Number has been identified as being particularly important as it is physically written on wheels that have had jobs done on them for identification purposes and should therefore be visible at all times.

The primary job information will be the equipment used (what kinds of tyres were used on the job, how many and how much they cost) and the work done (what jobs were carried out on the vehicle, how many of each job and other information such as the cost of the job). These are the pieces of data that will be showing up on an invoice and are taken from the central area of figure 1.

5.3. SOFTWARE ARCHITECTURE

In an attempt to give a clear illustration of the technologies used and how some things interact with the application I created a Software Architecture diagram. This diagram shows how the tablet interacts with different software in different locations and what kind of ways these interactions are done. The two main areas that the tablet interacts with are the local Microsoft SQL Server tables on the tablet, where data is stored locally for quick and easy reading or writing of data without the need for online connectivity, and the Bridgestone environment interacting with their databases with an internet connection for writing quotes into Bridgestone's POS system at the end of the workflow, ready to be turned into invoices after being checked. The diagram shows how the device interacts with each kind of table, showing that it can read from the Stock tables in order to gather data for the user to use or view to speed up their process, and that the device can both read and write to the Incident, VIR and Job Card tables, writing completed reports and reading these back in for viewing. The writes back into the VIR tables are to update the vehicle information with the new information based on work done. The web service call to the Bridgestone Environment is just a write into the POS system, no information is read back.

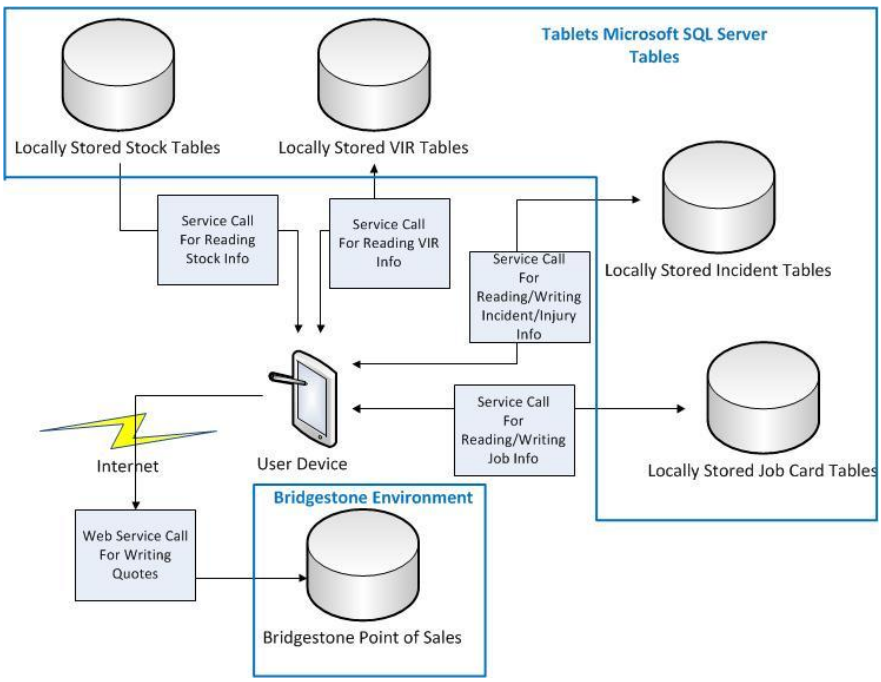


Fig. 20. A Software Architecture diagram representing the device and how it acts with various database tables with different software and in what ways

5.4. CLASS DIAGRAM

Based on the above information the following is a class diagram to show how data is handled as objects inside the Job Card application. Do note that the data names and method names in this diagram are directly from the final product, the diagram has been produced from the final application and it accurate for the final version. The classes directly connected to the LayoutAwarePage are the classes for each individual page of the Job Card, Stock Lookup and Incident Reporting portions of the app. The green service is the primary service for interaction with local tables, with various methods for different functions dealing with different tables. And the disconnected classes are used for collecting information for each portion of the app together into instances of classes that can be easily passed between pages to add to or read from at each page. Note that this class diagram is quite large so it has been separated into three figures, 21, 22 and 23. In addition to this some classes (Job_Card, and Injury) were too large to fit onto single pages on this document so have been sliced in order to save space.

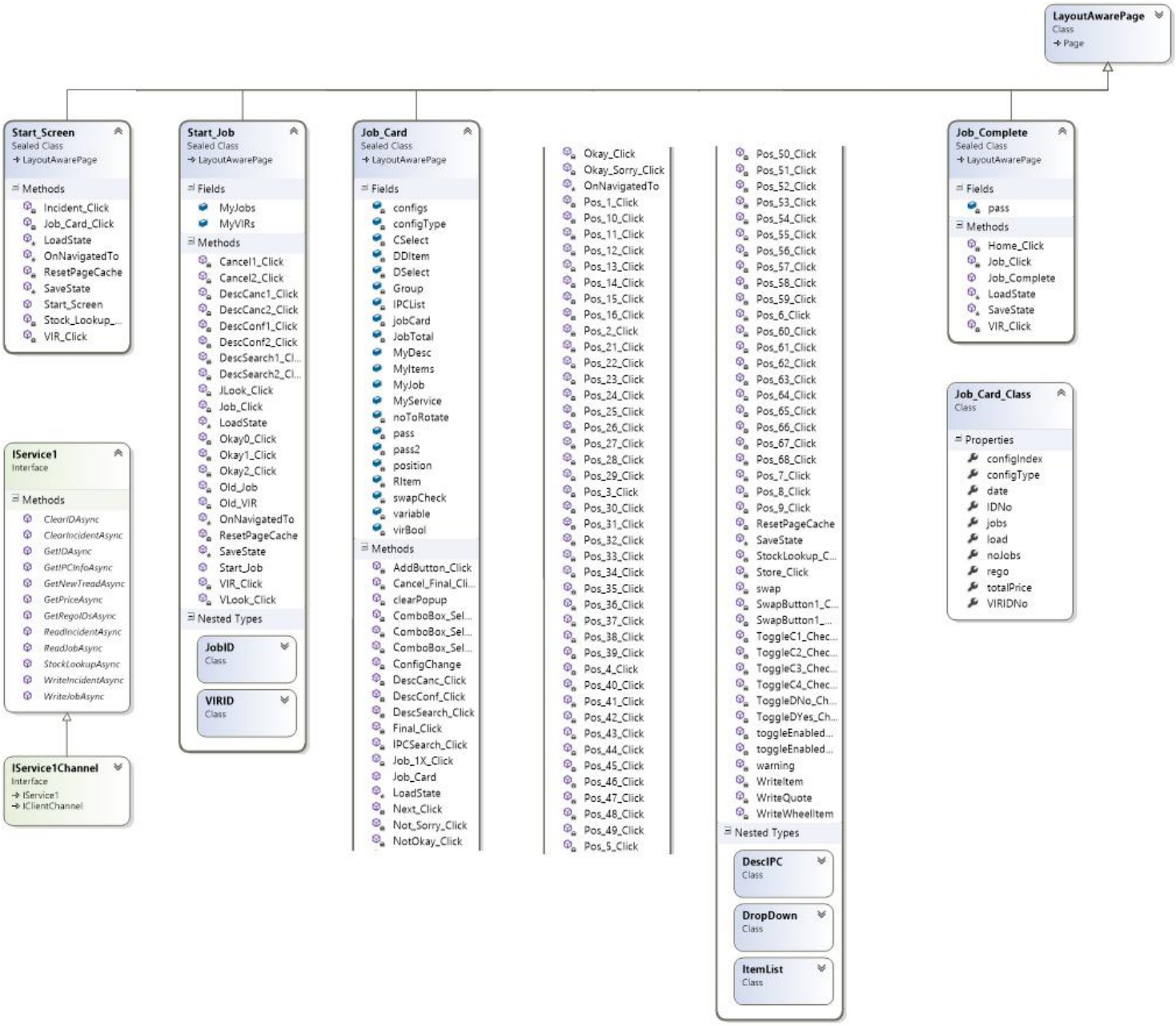


Fig. 21. Part of the applications class diagram dealing with the classes relevant to the Job Card portion.

These pages primarily deal with the Job Card, having the Start Job, Job Card and Job Complete pages. The nested classes on the Start Job page are used to fill observable collections for binding to ListViews in order to populate them with information based on search results for VIRs or Job Cards currently stored locally. The Job Card page is the largest page in the application mostly due to using logic to convert what the user has input into different IPCs to be added to a Job Cart. There are also somewhat decent amounts of logic put into functionality such as the swapping wheel positions, looking up an IPC for a wheel, and changing the layout of a popup depending on users' choices. There are a large number of methods for every button corresponding to a wheel position that the user may click, however each of these button click events are quite small. The nested classes on this page are again for populating collections for a ListView looking up the IPCs of wheels, populating dropdown lists of applicable jobs depending on the kind of vehicle loaded, as different vehicle types produce different IPCs with different corresponding costs. An instance of the Job_Card_Class is used to gather all the Job Card information into one class that covers all data used by and inputted into the Job Card pages.

In the lower left of this section is the service used for dealing with Job information, Incident information and Stock information. It has methods that include database connections to local databases, handling these as a service allows the application to re-use these same SQL statements whenever it needs to from within any page. These database statements were also handled this way due to the lack of System.Data in the normal Metro Style App pages, as this was not available, it could only be accessed after passing relevant information into a service and then accessing System.Data inside that service for use with database connections and statements. Most of the methods in this service deal with either reading local tables based on inputted keys, writing to local tables based on whole classes, or clearing rows from local tables based on inputted keys.

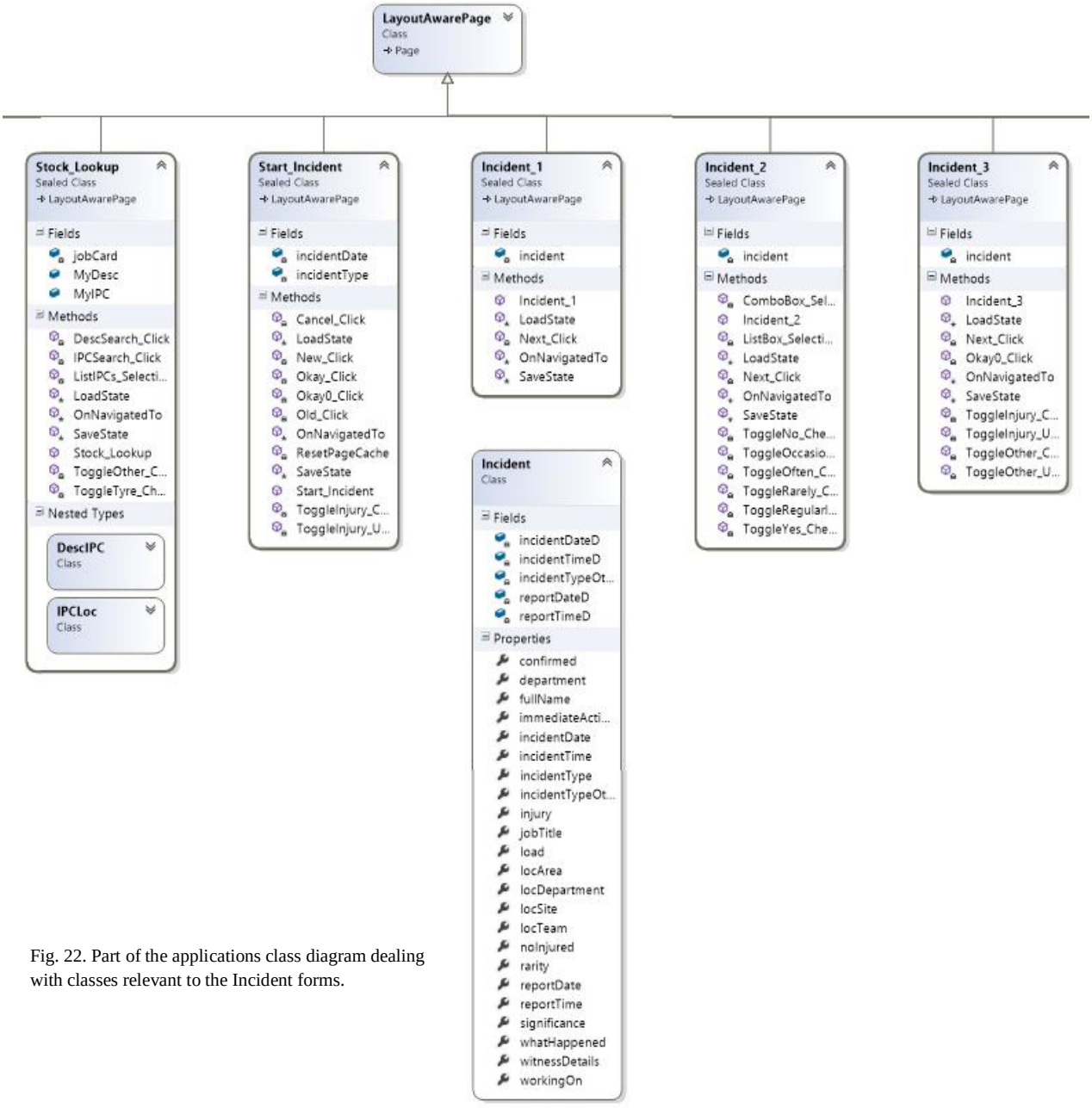


Fig. 22. Part of the applications class diagram dealing with classes relevant to the Incident forms.

The left part of the above Class Diagram shows the Stock Lookup page and two nested classes inside it for populating results into ListViews to firstly show IPC information based on a description, and then show location information based on an IPC. The Incident pages and class are fairly simple, recording information on each page into an instance of the Incident class. The toggle methods in the pages are in order to make certain groups of toggle buttons act as radio buttons in that only one is able to be selected at one time.

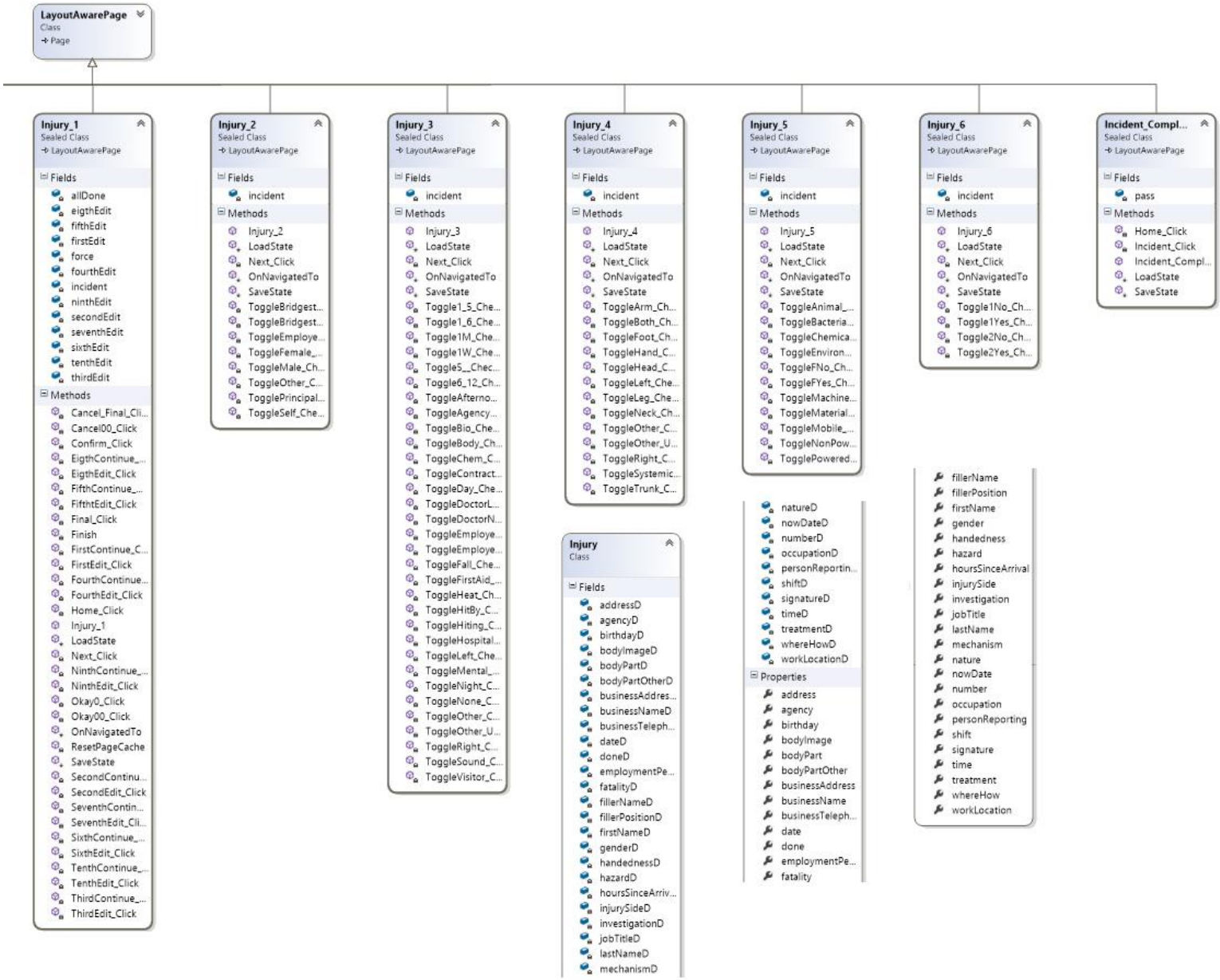


Fig. 23. Part of the applications class diagram dealing with classes relevant to the Injury forms.

The final part of the class Diagram shows the Injury pages and class. There are more Injury pages than any of the other form pages as there was a number of different information required, almost all of these are selected based on ToggleButtons though with toggle methods for groups of toggle buttons making it so only one button in that group can be selected at one time. The Injury_1 page also has methods for checking how many people were identified as Injured and allows for that many Injury reports to be created and filled in.

5.5. USE CASE

The use case shown in figure 24 was created to give a look into the functionality of the application from a user's perspective. Uses cases show what an 'actor' can actually do to an application, the basic functions they have available to them and how these functions may link to database actors.

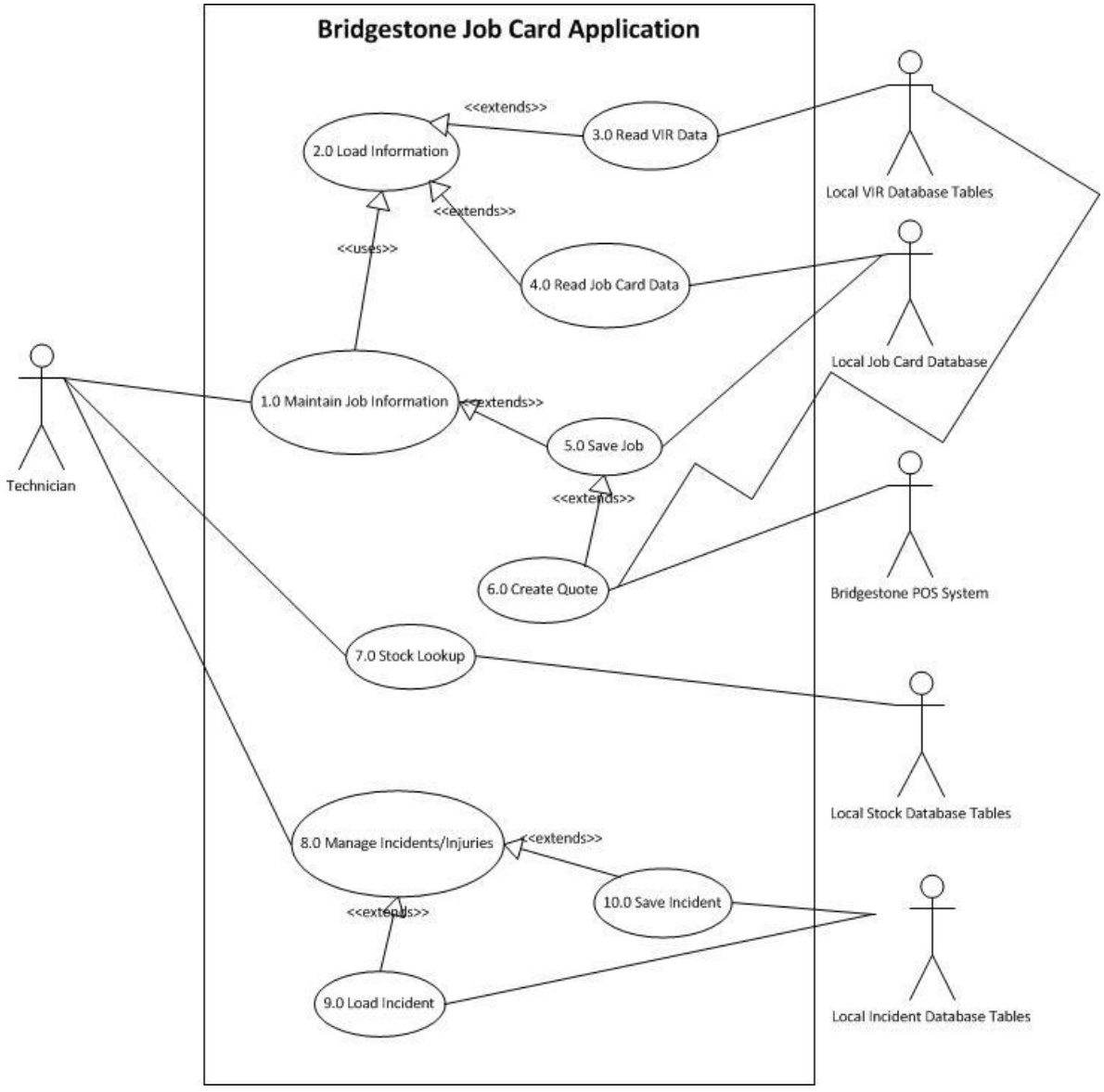


Fig. 24. A Use Case diagram for the proposed Electronic Job Card application.

We can see that the Fleet Technician actor on the left can interact with my parts of the application in three ways, managing Job Cards, Managing Incident and Injury forms, and looking up stock. These can interact with a number of databases seen on the right outside of the system boundary, both local and remote.

Going over the use cases shown in this diagram use case 1.0 is managing Job Card information, this means being able to edit a Job Card with user input. This use case uses the compulsory use case 2.0, loading information, which can extend either use case 3.0 or 4.0 depending on if the user is starting a new Job Card by loading a previous VIR, or simply loading a previous Job Card. Depending on which of these two loading use cases is used it will read information from different tables, be it the tables dealing with the VIR or the Job Card. Use case 5.0 is extended from managing the job information if the user choses to save their current Job Card to the database, and this will write relevant information into the local Job Card tables. This use case may also extend use case 6.0 for Creating a Quote should the user chose to complete the Job Card as being finished, which will write a quote directly into Bridgestone's POS system. This also connects with the VIR tables as it writes a completion report showing the state of the vehicle after work has been done on it.

The second way the user can interact with the system is with Use Case 7.0 for Stock Lookup, this is simply a read to stock information which is displayed to the user. This can be thought of as a lone use case that does not interact with any others, nothing us used by or extends from this and nothing is written to any tables. This use case simply links in with stock databases outside the system boundary.

The final way the user can interact with the system is with use case 8.0 to manage Incidents and Injuries. The user will have the option to load a previous Incident and corresponding Injury forms from the local database tables outside the system boundary shown by the extended use case 9.0, and they will have the option to save these reports into these local database tables should they wish to save the Incident and Injury reports they created or updated, this is the extended use case 10.0.

5.6. DATA FLOW DIAGRAM

A data flow diagram is used to give an overview of where what kinds of data are flowing through the system. External entities are given in square boxes, processes in circles, and data stores in ovals (often they are over and underlined instead). Data flow diagrams often have multiple levels of abstraction, with the top level being a 'context diagram' showing the system as a single process, and the external forces that work on it. As this context diagram is extremely simplistic and can be inferred from the diagram of the next level down, I will not be including it here. Below in figure 25 is the level 0 data flow diagram showing the main processes inside the application and where data will be flowing.

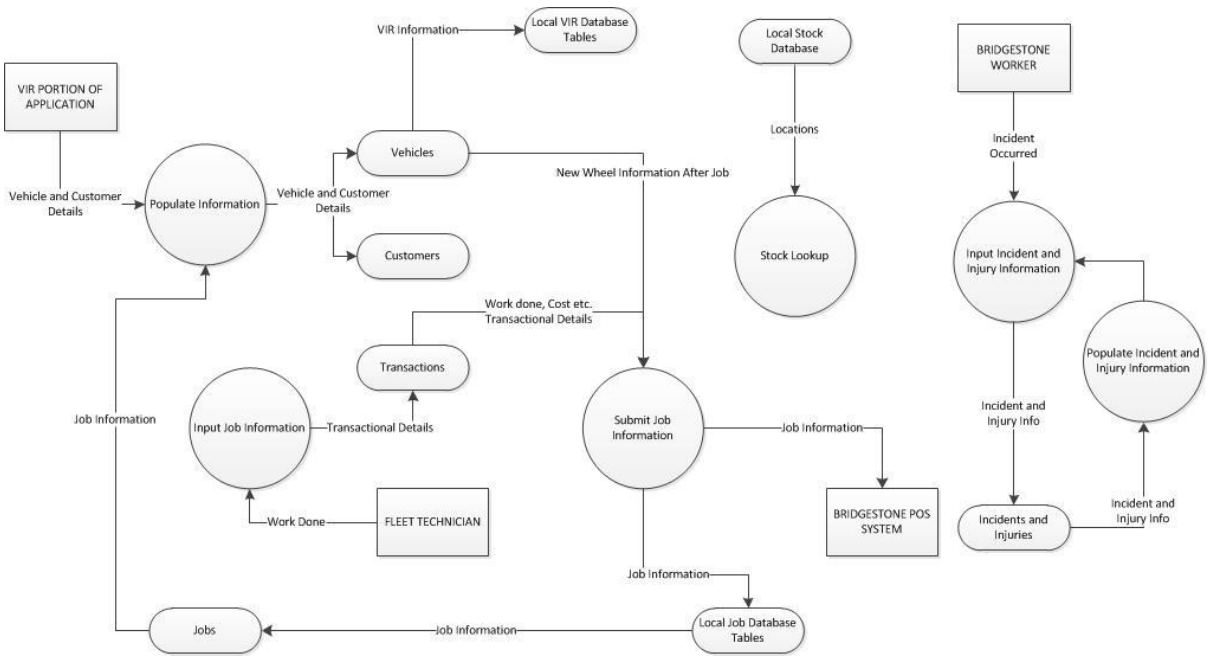


Fig. 25. A level 0 data flow diagram showing how data will move through the application to/from external sources.

From this level 0 data flow diagram you can see that there is one main flow of data in the central application, and then again similar to the last few diagrams, Stock Lookups and Incident/Injury forms are handled to the side separately. The data flowing for the Stock Lookup is simply the location from the stock databases being fed through to the tablet based on SQL queries. The queries will filter the locations based on a product IPC however no data is actually being given to the stock databases so this is only a one way flow.

Starting from the top left the VIR portion of the app will be considered an external source of data, syncing with my Job Card portion in order to give it customer and vehicle data that the VIR application obtained either from Bridgestone databases or through vehicle inspection. This information is populated into my application through a process, and then stored as Customer and Vehicle details inside the application as variables. The customer information will not be changed at all by my application, however it is still required. Therefore after it is stored it does not need to flow anywhere from that point. The vehicle data however may change depending on what work is done on the vehicle and because of this after a job is completed the updated vehicle information will flow down into the process to submit job information.

The Fleet Technician will be inputting various pieces of transaction information depending on what work is done, and is therefore an external source of job data. Through the input job information process transactional information is entered and stored into a class for the Job Card dealing with transactional information. After a job has been completed this transactional information will need to be sent to a database so the data will flow towards the submit job information process.

After the process for submitting job information has complete data from the vehicle and transaction storage, the job information will be pushed towards the external databases at Bridgestone as a quote is written. In addition to this, this is written into the local data storage tables as job information which can later be read back into the application when population information.

The data flow for Incident reporting to the side is very simple, the external source of a Bridgestone worker will need to input data into the application concerning details of the Incident and corresponding Injuries. This information is then stored into a local data store for Incidents and Injuries. The data from this data store can be used as the source of data for this portion of the app in the process of populating information when loading old reports.

6. PROBLEMS AND SOLUTIONS

6.1. NON-TECHNICAL

During the course of the year while working on the project it is natural to run into a number of problems that must be overcome, with a variety of potential solutions. These sorts of problems can come about in the creation of the app while programming and be more technical, or during planning or design and be non-technical. It is imperative to the project progress that when such problems are encountered they are met and solutions are quickly found and carried out, even if they are not the best possible solutions providing alternatives within the time frame is quite important, this section will detail this very idea and give examples of problem solving throughout this project.

The first major problem that we faced was which platform to develop the application for, be it an Apple device with iOS, an Android device or a Windows Phone. This was researched in detail as shown in section 4.2. After thorough research and being influenced by the Windows Camp it was decided to go ahead with developing for a Windows 8 platform. This decision was also supported by those at Bridgestone as most of the I.T. infrastructure from top to bottom in Bridgestone New Zealand and Australia is Microsoft based and running on Microsoft systems, and the I.T. staff are both familiar with and comfortable with Microsoft products. In addition to this one of the workers had his eyes on a robust tablet designed to use the Windows 8 OS and had previously tested it, feeling good about using it. Developing for Windows 8 also means I will be developing for something that is new in the industry, allowing me to stay at the forefront of new technologies, looking towards the future as opposed to devices currently in use that are soon to be made obsolete by new versions.

After it was determined that a Windows 8 Application would be the primary solution given to Bridgestone, an environment that allows for the development of such an app was required. In order to program for and test Windows 8 Metro Style Apps not only do I have to be running Windows 8 on a PC or tablet, but I also required access to appropriate development tools such as Visual Studio 2012. While Windows 8 Beta and Visual Studio 2012 Developer Preview are both available for free download for MSDN subscribers, in order to debug and test

applications created in Visual Studio 2012 a developer's licence is also required [43], and this comes with an annual monetary cost as described in section 4. One solution to this problem would have been to install Windows 8 and the development tools necessary onto my personal laptop and to pay the fee for a developer's licence for the purpose of this project, however this would not have been ideal for a number of reasons. My personal laptop endures heavy use and is two years old, therefore at this point it is prone to overheating and is usually used along with a cooling pad, I fear that installing Windows 8 on a separate partition may put more stress on my laptop than I would like, and attempting to run Windows 8 in a virtual environment even more so, especially considering that success with Windows 8 running virtually have been mixed, and would not be sufficient for the development of an entire application. Instead it was requested of Bridgestone to set up Windows 8 on the machines used in their offices, with developer licenses also provided by Bridgestone for use in producing their application. This was far more favourable as it ensured that the computers used for developing this app would not be cluttered with other software, and also reduced the need for maintaining the system running the OS on me as a student. The one drawback of this solution was that this meant all programming of the application needed to be carried out at the Bridgestone offices, however this did help promote consistent work to be done weekly with frequent hours put in to coming into the offices every week, which pleased my supervisor, in addition to helping give a better experience of working in a real environment with real hours.

Another problem that arose due to working on developing a Metro Style App for Windows 8 was that seeing as how Windows 8 is still new and developers have only recently begun experimenting with it, there is drastically less help and support to be found on the web in attempting to solve minor programming issues. This led to needing far more time than was usual to solve some problems as relevant answers and examples were few and far between. More often than not the solutions to minor problems were documented in depth in the web for previous .NET versions, however when attempting to implement them in .NET 4.5 for Visual Studio 2012 a large amount of methods were different and often times the entire solution needed to be changed. This along with the problem mentioned above caused me to spend time at home while not at the office researching methods on how to implement functionality that I wished to include, gathering a number of solutions, and then trying them all out when I was next in the office with the Windows 8 development environment. I was able to gather some answers by posting questions on MSDN forums and other such help websites.

When running the application it was noticed that it was not properly adjusting to the screen resolution, only taking up space in the upper left part of the screen. This was identified as a major design problem due to the fact that there was a decent amount of unused space around the lower and bottom sides. With some research it was found that in order to make the application stretch to various screen resolutions when designing an application for larger screens it was possible to use a scale-to-fit approach placing content in a ViewBox to stretch the application to various screen sizes without disrupting the position of any interface elements or changing the amount of content viewed on screen [44]. It was also possible to use an adaptive layout that would display more of an application (for example more fitting on a page for an application that scrolls), however I thought it would be preferable to keep the content shown in the application consistent on various monitors.

It was also identified that it was important to the end users that the application be very simple. In order to solve this I planned to use as many visual cues as possible such as representing a vehicle as a configuration diagram like that shown in figure 26, where wheels that jobs will be done on can be naturally selected. I also planned to handle the application as a sort of wizard that flowed seamlessly forward and backwards between steps.

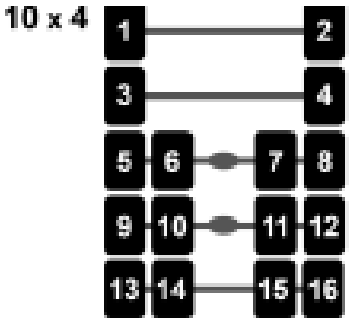


Fig. 26. A simple vehicle configuration diagram that I hope to use as a focus in my interface.

In the current work process carried out by Bridgestone's Fleet Technicians the paper Job Cards are collected by an individual in the store and checked over to ensure all information is correct and mistakes are kept to a minimum. This part of the work process did not seem to initially fit into the work flow of the application I was creating as it seemed tedious to have to reload every Job Card carried out during that day back into the app, and this time allowing them to be written to the POS systems. I would have also had to distinguish what was allowed to be done by Fleet Technicians while performing work and what was allowed to be done in a store after checking a Job Cards as one user would be able to write to POS while the other should not, which seemed to unnecessarily complicate things. The solution found was to write Quotes into POS instead of directly writing invoices, this allowed for all relevant information to be sent to POS just as it would need to be for invoice creation, however it is still a step prior to an invoice being created. What this meant for the work flow was that a Fleet Technician would no longer have to worry about what they were and were not allowed to do and are able to write to POS, where the checking of paper forms would be simply replaced with using the POS system to check the quotes written by the application and then select for an invoice to be created. This also provided an extra benefit of security as it meant that no matter what mistakes a user of the app creates, no invoice can be directly created unless someone checks over it in the POS system and makes sure it is valid, this effectively makes the writing portion of the application 'child-safe' as no real damage could be done from the application alone. As the same system is used for Quotes and Invoices with the same information it is very simple to create an invoice after a quote has been written. This also enabled me to handle all the logic that went into converting what was written on the paper form to put into the POS systems with the application itself, so the information sent to POS would already be appropriate for POS to understand without having to attempt and translate ambiguous service descriptions to chargeable IPCs. The final benefit of this solution was that POS systems also allow for invoices to be printed, which was outlined as a potential goal for the application. We briefly considered the possibility of a small printer attachment that could be used; however we ended up not pursuing this option and instead figured it much simpler and consistent to print from the POS system after the job information has been sent to the database. Thanks to this there should be no issues in the end-to-end process remaining unchanged as far as invoices are concerned, as the already in place POS systems are used where possible.

On the Injury form pages when attempting to mimic the paper form I was faced with the problem of replicating an image where someone was able to circle parts of a human body to indicate where on a body the Injury was. My first thought in solving this was to arrange buttons in a human-like format and allow the user to toggle select on or off depending on of the area represented by each button was relevant or not. This solution was easily put in place however I also wished to use images inside these buttons to better represent a human. I figured out the coding required to attach an image to the foreground of a button in the backend C# coding such that it could be modified once clicked to indicate it has been selected, however I overestimated my ability to create images in Photoshop and was unable to create clear and smooth enough images that did not disrupt the cleanness of the application. The alternative of relying simply on the button shape and placement was used instead.

As development of the app was nearing its final stages it became more and more necessary to obtain a real tablet running Windows 8 that would be able to be used for testing the application. This was to ensure that the application ran correctly on a tablet, was able to run on a small screen without visibility issues, and also to test some tablet-based functionality such as using a camera for the VIR portion of the application. It was made sure that Bridgestone was aware of the need for this tablet and with very few reminders we were able to obtain a tablet to test on. This tablet was borrowed by Bridgestone and provided for use for this project. Having access to this tablet allowed for more relevant tests to be carried out on a device that is not unlike that which would be used in the final work flow, and also being a portable device allowed for additional work to be carried out outside of the Bridgestone offices for the final weeks of the project.

6.2. TECHNICAL

Every Job Card is required to have a unique identifier. These numbers are provided in blocks and the application needed to be able to be provided a block to use, which could be updated in the future with new blocks when necessary. The numbers to be used as IDs are always increasing and there will never be a need to start using a block of numbers below a previous block value. With these in mind the solution was fairly simple, using one insert statement into the local Job Card table with the starting number of the block as the key creating a near blank row. From this the application is able to perform a select statement on the table and retrieve the last highest ID, increment it by one, and use that value for the ID of the new Job Card. The way I programmed this makes the application write a row for the ID to be used as soon as it is obtained, and then when the Job Card is completed and written to the database properly that row is cleared allowing the completed Job Card to be written. I coded it this way in order to prevent overlapping of Job Card IDs if in the future these are stored remotely as opposed to locally. This would mean that as soon as a tablet reads for an ID number it will create a row for it, effectively locking in its ID and ensuring no other tablet attempts to use the same ID.

A problem I encountered when dealing with the clearing and re-entering of a database row for the saved IDs mentioned above was that part of the time, approximately one in four times, the application would simply delete the row and not write anything. I determined this was due to the calls to the database being done asynchronously, sometimes causing the application to attempt the SQL insert statement before the delete statement has finished, causing the insert statement to detect that the row already exists and thus failing to write. This was solved by forcing the write statement to wait until the clear statement was completed before executing.

Synchronisation of databases was identified as a potential problem early on in the project, and this came into play in two main parts of the application in development. The first of these being the process of writing a Job Card through to POS as a Quote, as this deals with tables foreign to the application. It would be ideal for the application to be able to be used without internet connectivity, and instead synchronise all relevant databases together at set times or whenever possible. This would allow for the application to be used at any location regardless of connectivity or a signal, but still allow for the work flow to precede unhindered, worrying about writing through to proper databases later. It would be ideal to have a local version of the POS database synchronised with the remote tables whenever possible, allowing for the quotes to be written locally at the end of the Job Card work flow, and then push these quotes through to the proper POS system in a synchronization of these tables. Currently the offered solution is to be connected online either directly to Bridgestone's network or through the use of Bridgestone's VPN, allowing for access to the POS system, though synchronisation would allow for an offline work flow to be done.

The second place where there was a problem that synchronisation could be a potential solution for was in the Stock Lookup. The stock information that needed to be read by the application in order to give the locations of various stock items in different warehouses is currently stored in Bridgestone's databases, and it needs to be accessed by the application. The current solution in place for these stock locations is to have a snapshot of this data stored locally on the application; this allows for Stock Lookups to be handled offline however this also means that the results would only be as relevant as the last time the database snapshot was written into the local database. This is not a major problem as the results are just an indication, and they do provide telephone numbers allowing the technician to check and reserve stock before committing, however the purpose of the Stock Lookup function is to minimize this. Local tables are in place and manually updating them with a series of SORT->JOIN->INSERT SQL statements is simple, however tedious. It would be far more efficient to synchronise them with the remote databases in order to ensure they are up to date whenever possible, not requiring manual updates.

One of the primary uses of a tablet aside from portability is the use of an interactive touch screen. Because of this it seems natural to implement touch screen functionality wherever possible, using swipes, pinching, and dragging. As one of the services carried out on a vehicle was that of swapping wheel positions it seemed like the perfect opportunity to implement natural feeling touch screen functionality with dragging, allowing one wheel position to be dragged over to another, dropped, and have them swap places. A decent amount of time was spent researching and testing this however the tools in Visual Studio that have inbuilt drag and drop functions, ListViews and GridViews were quite strict in how they handle the items inside them, being restricted to a simple list of items or a grid with restrictive formatting. Using a GridView was looked into as an option however the drag and drop functionality provided was only able to reorder the elements, as if it were an ordered list, so

dragging one element further back just caused all elements in front of its new position to move forward. This, in addition to the work put into creating the configuration diagrams in a way that displays a great number of different vehicle configurations, each with a well-defined diagram, it was determined unfeasible to use a GridView for this. Instead an alternative was used that unfortunately forwent the use of dragging, and now instead a button is used that puts the user in a sort of 'swap' state, allowing two wheel positions to be selected and then swapped.

During various parts of the application when making service calls to database it is often possible for the service to return a list of results. This can happen when searching for a VIR or Job Card to load back into the application, or when searching for stock. In order to display results properly I needed a way of adding each result to a dynamic control in such a way that would enable me to display various pieces of information about each result in a clear and formatted manner. Initially I attempted to place a large number of TextBlocks inside a ScrollViewer that would allow for scrolling if the number of TextBlocks set to visible exceeded the space on screen. This was a highly tedious approach as a text block for every new item needed to be created and placed in the front end XAML, meaning it would only be scalable up to the number of items created and set to invisible. It also meant that a lot of backend C# coding was necessary in order to add new items adjusting the visibility of these blocks, but more so in removing them allowing for removals to be done on any item, adjusting the positions of all subsequent TextBlocks to move up. As this approach was feasible but was not scalable at all I researched a new approach and found data binding to ListViews [45]. This enabled me to set the content of a ListView to a collection of objects stored in the backend, enabling me to simply sort each item into a list to display, and also gave functionality for customisation of which variables these objects held to display. I created nested classes to hold the results of SQL queries, having the class hold the columns of the data returned such as three variables for IPC, Description and Price, and then created a collection of this nested class to be binded to the ListView. After doing this the ListView was entirely scalable to any number of items and I did not have to make any extra interface elements.

When loading the Job Card page I wished to display to the user an indication as to how likely it is that a job will need to be carried out on a specific wheel. I did this by colouring wheels green, orange or red depending on how low the tread depth was for that wheel when inspected in the corresponding VIR. The problem here was determining what to classify as 'low' enough to warrant colouring each wheel. The best solution in doing this would be to consider the specific tread depth of a new tyre and base the low values on a percentage of this, this way it would be dynamic depending on the specific IPC of tyre currently attached to that position. This information is stored in Bridgestone's databases however it is not read into the application along with the other vehicle information, and while it would be possible to read this information in on the Job Card it seems more beneficial to allow the Job Card to function as much as possible offline except for the final write to POS for creating a quote. Therefore an alternative was looked at in hardcoding the values that are considered low to be valid for the majority of vehicles, or at least be close enough to still give a hint to the user that work may need to be done without forcing them to do so. These hard coded values were tested against the recorded values from the VIR and the wheels were coloured where appropriate.

It is sometimes necessary for a Fleet Technician to gain appropriate permission in order to perform work on fleet vehicles. Currently it is simply up to the technician to ensure they have proper permissions before continuing however it would be more efficient if the application could help the user in some way. The problem here was that there was no clear cut way to determine if a specific Job required permissions or not, and in addition to this the most efficient method of gaining permission would be to call as opposed to e-mail. I looked in to allowing the user to select if they required permission and send off an e-mail to the appropriate customer if this were the case, because customer information was already stored inside the app at this point in the work flow, however there were complications in handing e-mails in .NET 4.5 and after trying a large number of methods to get this working I was unable to complete it in a limited amount of time. The solution I went with instead was to remind the user that permission may be required before they complete a Job Card and allow them to store the card should they require this permission, allowing them to manually gain permission and then load the Job Card back up. This was not ideal but it has been identified as a viable immediate solution that works.

Currently in Visual Studio 2012 there is no DatePicker tool available for XAML interfaces, the tool is only available for HTML. It was necessary to have the user select dates in certain places where the date is not necessarily the day the application was being used, the date of the tablet could not be used, therefore some kind

of alternative needed to be found. The most simplistic alternative would have been to give the user a text box to input the date, or perhaps three boxes for day, month, year, and then concatenate them into a DateTime object, however this seems needlessly complicated for the user and also requires text input which I wanted to try and minimize. After some searching I found Telerik dev. tools [46], a free native toolset for building Windows 8 apps. This add-on to Visual Studio 2012 once installed added additional tools to be used inside the application with ensured smooth functionality, and among them were a DatePicker and TimePicker. This toolset was installed and tested on the Windows 8 machine with great results, allowing for the user to select the day, month and year with a mouse or finger in a scrolling-like fashion, which felt very natural. It was also amazingly simple to read values from, and write values to these DatePickers in the backend C# code. I was very happy with the toolset and incorporated its use for DatePickers throughout the application wherever it was needed. However the one drawback of this solution was that the app could not be worked on and debugged without this Telerik toolkit installed, therefore it had to be installed on every computer and device where development was done. This was only a minor inconvenience as there were only three devices used, including the tablet in the last weeks of the project, and though the installation required an account and an internet connection it was free and easy, automatically incorporating itself into Visual Studio 2012 once installed.

During the replication of the Incident and Injury forms a problem was encountered when attempting to deal with options the user was able to select where they were able to select as many options in a question as needed. The implementation of this in the user interface was simple with simply giving the user toggle buttons for every option, however the problem was encountered when it came to storing this in a way that would be easily viewed in a database table. One possible option was to have a large number of rows for every option, indication whether yes this option was selected or no it was not, however this did not feel right as the options would not be grouped together when stored as well as the fact that some of these groups had up to near 30 options. The solution that was found was a very logical one in that the groups of buttons were handled in a ‘binary’ way. What this means is each option was thought of as a bit that could be on or off, and depending on these options it was stored as a single integer, for example if the buttons corresponding to the two least significant bits were both selected, these would be considered ‘on’ and added to the int with values of 2 and 1, as the two least significant bits of a binary number. This enabled it to be stored as a single value as well as in an efficient way that saves space. It could also be easily read back in and simple checked as if the program were converting the decimal integer to a binary number and checking each bit.

One very simple problem that was faced when attempting to use the application on a new device was connections to the databases. I had been hardcoding the database data source as the computer I was working on, therefore when on a different device it no longer recognised this data source. The solution was to simply change these to always use localhost for local tables, allowing the application to be used on any device without having to modify the services used.

7. IMPLEMENTATION

7.1. DESIGNS

Early on in the development process after the flow of the application had been decided I developed some simple hand drawn designs for some of the more prominent pages to be in my sections of the applications, these can be seen in figures 27 and 28. These designs were mainly in order to get an idea of the screens on paper as a starting point in the visual creation of these pages. I identified the central pieces of information needed on these screens and laid out the pages in a way that I felt was a good representation of these. Both of these pages were supposed to conform to a set template that I was going to use for every page, having a bar at the top of the page for the company logo and the page title, then the major content section being shown below this. Figure 27 is the initial design I created for the Job Card page, initially there was a decent amount of room allocated to the Job Cart and each item in the cart was also fairly large. The Job Card number was placed in the upper right of the page as very large text in order to draw focus, as this number was identified by the end users as being of significant importance when I met with them. A dropdown is made available to select items to add to the job cart for this vehicle that does not relate to any specific wheel such as a callout service. The left part of the screen was left fairly clear in order to account for different kinds of configuration diagrams to be able to be represented in this space with as much area around them as possible so the user does not mistakenly select the wrong ones on the touch screen. Selecting one of these wheels would open the popup shown below the screen giving the

user an indication of what position they had selected (more feedback), and enabling them to select an item to add to the Job Card for that wheel such as a strip and fit

In Figure 28 you can see the initial design of the Stock Lookup page. I designed this under the assumption that a Fleet Technician would not be familiar with the unique product IPCs for every item they may wish to look up, however they would be likely to know the more natural descriptions of them. Therefore I split the screen into two sections, the left side for entering a description and finding any IPCs that relate to the search, and the right side allowing finding the location of a specific IPC after it has been identified. The screen was split in this way as I did not want to prevent the user from directly searching for an IPC should they know one, perhaps due to frequently working with it. I also did not want to directly return the locations for every piece of stock based on a description as there may have been many results from a description search, many of which would be irrelevant and could potentially cause confusion to the user, I felt it would be preferable to only display the locations for the exact item they want after they have identified it.

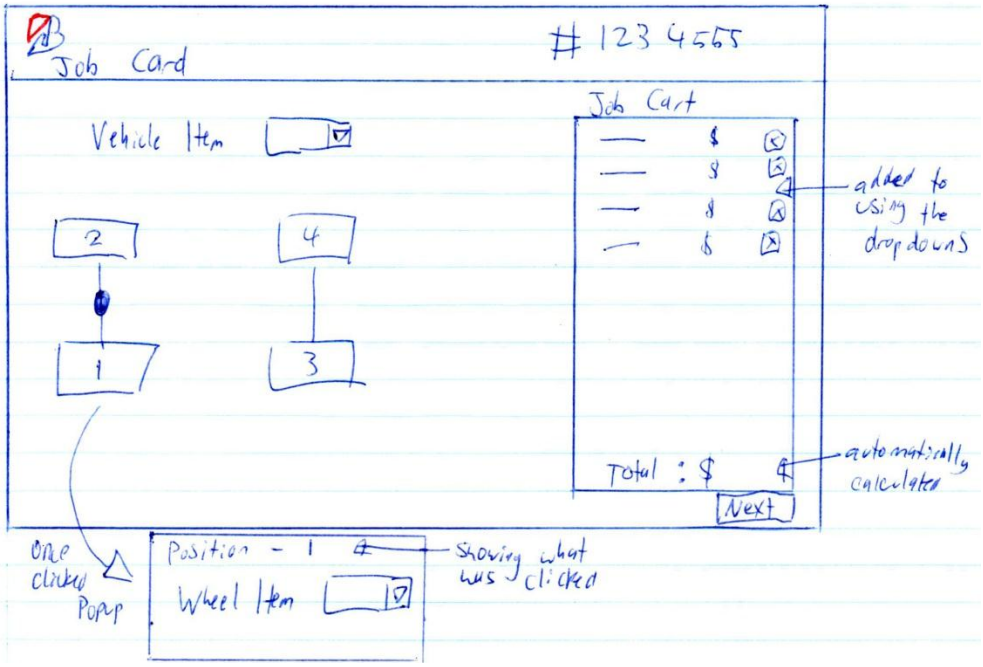


Fig. 27. One of the initial conceptual design sketches of the Job Card interface for the application.

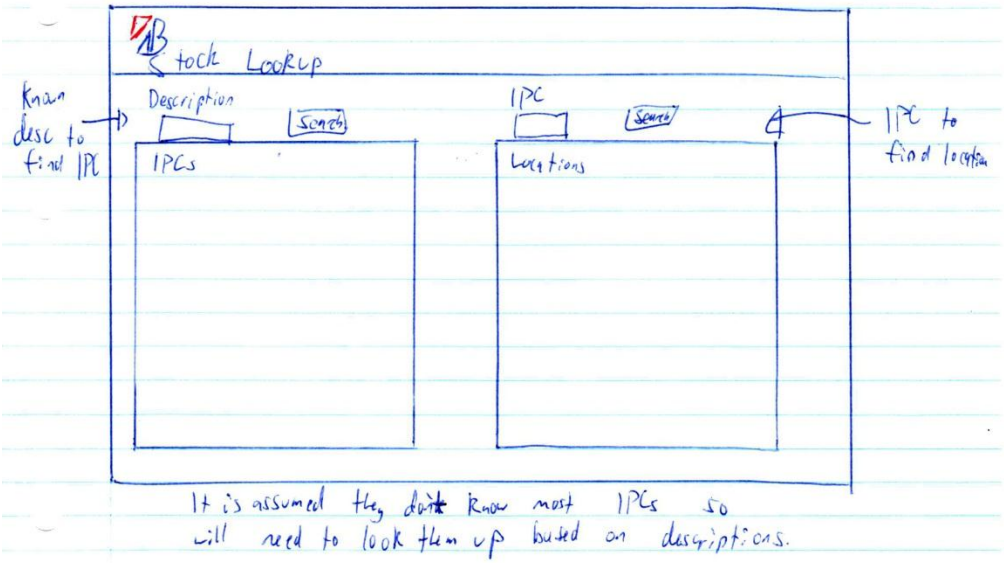


Fig. 28. One of the initial conceptual design sketches of the Stock Lookup interface for the application.

7.2. USABILITY

During the iterative project development described in section 3.2 one of the Bridgestone supervisors frequently checked in with the progress of the application. Every time this happened the current working version of the application was demonstrated and described, and feedback was obtained and noted for changes. Most of these were dealing with usability and interface options, but occasionally this also called for minor functionality adjustments in order to make the app feel more natural and usable from a user's perspective. Another unexpected benefit of these demonstrations was that some minor bugs were observed that were not encountered during testing, which was immensely helpful in providing a fully functional application. This kind of constant feedback of the application and running through proper workflow scenarios with a Bridgestone supervisor acted as usability testing for the project. If some wording had been incorrectly stated in the application it was pointed out and able to be fixed, and the same can be said for text types, sizes, colours, and the placement of buttons or fields. Some examples of feedback that was beneficial to building a usable app were that throughout many parts of the Job Card portion of the application I had initially incorrectly used the word 'job' to refer to an item that is to be added to the Job Card, this was noticed and I was informed that the 'job' would be all items together as a single Job Card, and each individual IPC that is charged should instead be referred to as an item or line item. For the Stock Lookup I initially had it set out to describe to the user the format for a wheel description, being 'Width' / 'Ratio' R 'Rim' with these three measurements to be entered into one box, for example '225/70R15'. However it was pointed out that this kind of instruction was somewhat confusing and probably unnecessarily so, therefore I was able to modify the screen to use three text boxes instead, one for each measurement and allow the user to input one, two, or all three measurements and return a list of all relevant IPCs. This approach was far easier to understand and was less prone to human error, however the problem with this was that the '/' and the 'R' needed to be hard coded, forcing the user to only be able to search up wheel items. During another demonstration the demonstrator pointed out this as a usability flaw and requested the option of searching up item descriptions that may not be wheels, for example tubes, so this was added. It was also noticed during these demonstrations that there was no real information given to the user on the Job Card concerning the results of the VIR, the Job Card functionality was fine however it would be more usable if there was a suggestion given to the user where areas of work may be based on the inspection that was completed, therefore the colours of positions were changed depending on the tread depth recorded to give an easy visual cue to the user where work may be required.

Feedback was also gained on colour schemes, a number of different choices were prepared and shown to a number of our Bridgestone supervisors, from these screens the supervisors were asked to determine which screens they liked over other ones and were told to explain why. They were asked which portions of the screens had the most impact on their decisions as to which screens they liked or did not like, and why they focus on these. We were sure to inform the supervisors that any modifications to the colour scheme were entirely possible, being able to mix and match different parts of different schemes we created and that we were open to entirely new ideas. When designing the different colour schemes to show the people at Bridgestone we attempted to keep the colours consistent with those that are associated with their company, using the shades of black, white and red that can be seen in their logo and their website. The intention of this was not only to associate the application with Bridgestone but also because the colours used felt visually suited to something like a tablet app. After looking at the screens provided as seen in figure 29 some feedback was obtained from the supervisors. They did not like the screens that lacked red at all, as these were dull in comparison. It was also identified by people was that the grey buttons on the red-black background, while they did stand out they did not look very professional and did not fit in with the colour scheme. It was also mentioned that the red text on black buttons may be difficult to read due to lack of contrast. The main point that was mentioned by those at Bridgestone was that the solid red buttons with white text inside commanded more attention than the other buttons which seemed to have a problem of either fading into or being outshined by the gradient background. In the Bridgestone logo the small amount of red draws viewers' attention, we attempted to mimic this kind of draw in the buttons making them a solid red as a central focus that the user knows they should interact with instead of them being lost to the background. It was an option to have a solid single colour background and white may have been appropriate for this as it would be a better representation of their logo colours, however this feels like it would be more suited for a website or legacy system and we wanted the tablet interface to look more dynamic and fresh. A colour scheme using the black, white and red with a solid white background was in fact used for my project website where I felt it was better suited.

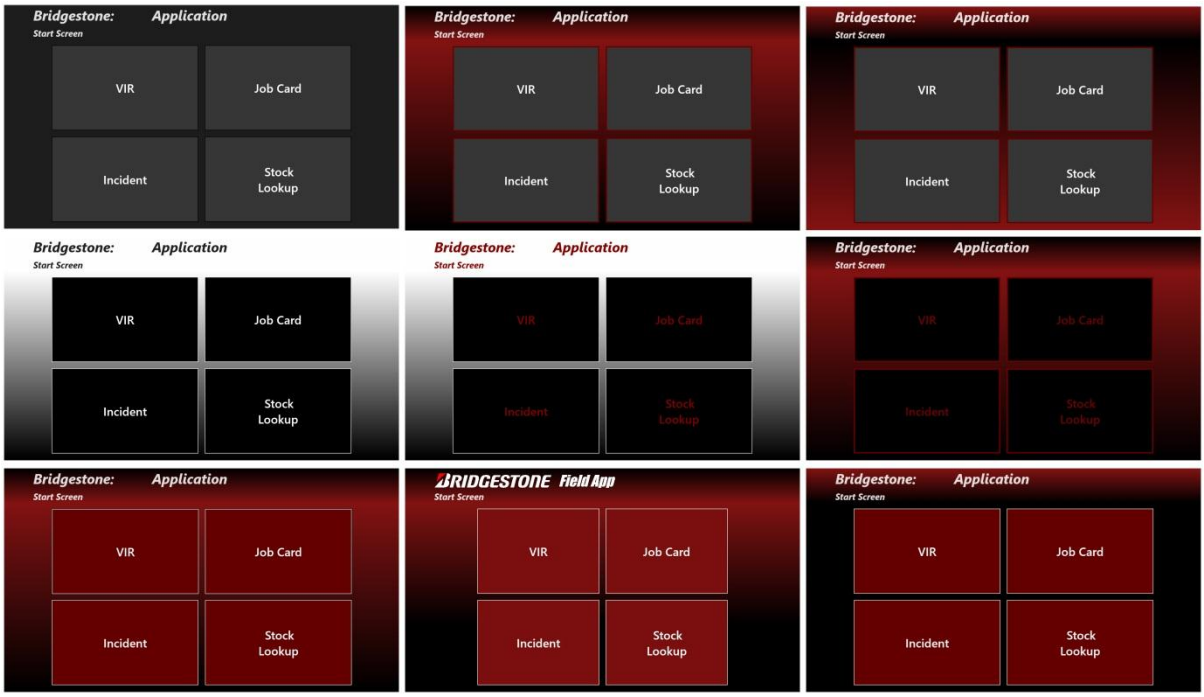


Fig. 29. Some colour scheme ideas presented to stakeholders as part of creating a usable and likable interface

7.3. KEY SCREENS

In order to demonstrate exactly what has been created throughout this project I will give an overview of the core functionality, showing some key screens of the application and showing some of the various solutions to problems that were mentioned in section 6. At the start of the application the user is greeted with the option of accessing multiple parts of the application as seen in the screens in figure 29 above, the VIR and Job Card screens are the main functionality given by this application, with the Incident and Stock Lookup functions being much more shallow in comparison, but still usable in the application.

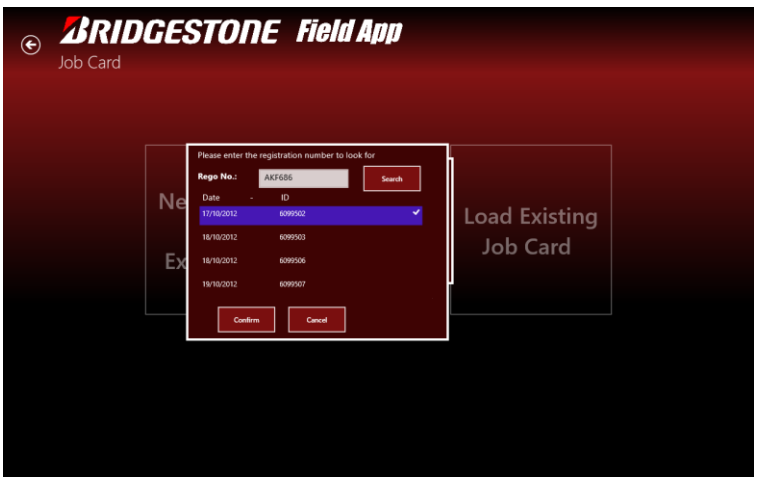


Fig. 30. A screenshot of loading a VIR for use in the Job Card portion of the application

The initial page when starting a Job Card instructs the user to either start a ‘New Job Card’ based on an existing VIR, or to load an existing Job Card. This can be seen in the above figure 30, where a user has selected to load a VIR and then searched for a list of stored VIRs to load based on a registration number. This kind of popup is the same when they chose to load an existing Job Card into the application, allowing a user to see all current data held for a vehicle, the IDs of each VIR or Job Card, and I also decided to display

the date to the user in order to give more insight into what they will be loading. When loading these, a Fleet Technician will be able to use mainly the registration number they input and the date to determine which of the results is the one they were looking for. The IDs for each VIR and Job Card are unique, however it is doubtful that the Fleet Technician will remember which of these relates to which forms, it is displayer here regardless due to the fact that Fleet Technicians still felt it was crucial data as it is used frequently in their work flow to distinguish between jobs. As these IDs are unique they are also used as keys when storing Job Cards into the database as a JobCardNo as seen in figure 31 below. There are two tables that handle Job Card information the first one being general information about the job (left) while the second table (right) is related to this with a one-to-many relationship, having a row for every item added to the Job Card. The keys for this second table are the Job Card ID of the job it is related to and the IPC of the item it is for, with additional columns for item information such as the description or price.

Column Name	Data Type	Allow Nulls
JobCardNo	int	☐
VIRFromNo	int	☑
Date	datetime	☑
Rego	nvarchar(50)	☑
Config	nvarchar(50)	☑
[Number of Items]	int	☑
[Total Price]	nvarchar(50)	☑

Column Name	Data Type	Allow Nulls
JobCardNo	int	☐
ItemType	nvarchar(50)	☑
[Desc]	nvarchar(50)	☑
IPC	nvarchar(50)	☐
Price	nvarchar(50)	☑
Reason	nvarchar(50)	☑

Fig. 31. A screenshot showing the tables used for storing Job Cards and their keys

Once the user has entered the main Job Card screen they will be greeted with something similar to what can be seen below in figure 32. This screen can be compared to the initial drawn design from section 7.1. We see that just like in the design the central focus is on the very configuration diagram, however more space was allocated to this in this real implementation due to some vehicles having a great number of positions. Due to this the Job Cart on the right side was not able to take up as much room as I would have liked and the items listed in it are smaller than initially designed, however there were no readability issues with this when testing or demonstrating this app. You can also see the Job Card Number was moved from the upper right to now act as somewhat of a sub-title, this was done to keep consistency between screens which already had these sub-titles, as well as for space issues. Three new things that were not on the initial design are the ability to proceed to the Stock Lookup from here on the bottom right in order to allow for more flexible work flows, the indication of the work flow process with a progress bar down the bottom, and the key for colour indications on the left.

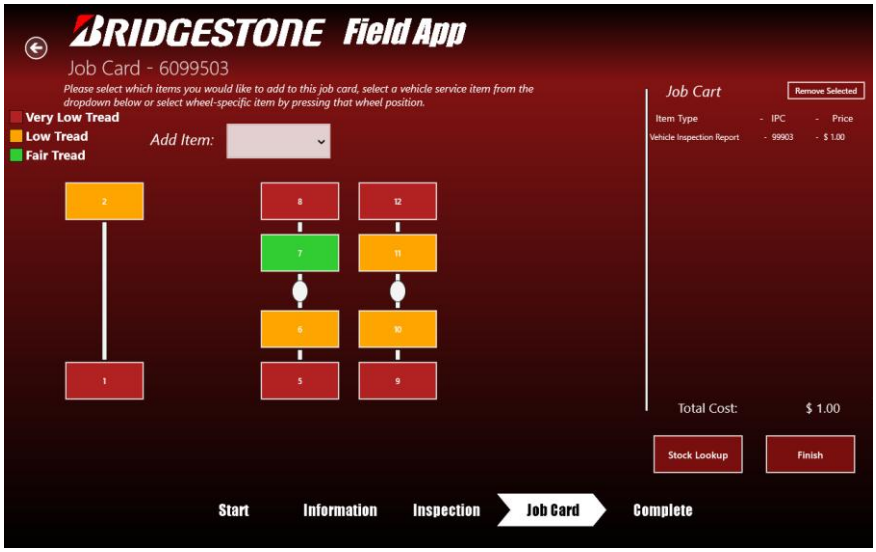


Fig. 32. A screenshot of the main Job Card page showing what the user will be using.

The backend code for this page in particular is in fact greater than any other page within the application, handling methods for every possible wheel position, dynamically populating the drop down boxes with items to add depending on the type of configuration loaded, adding and removing items to the job cart, as well as some neat functionality with things like the colouring of wheels and the swapping of positions. Swapping and colouring needed to be possible to be carried out for every possible wheel, but in a dynamic way therefore I created methods for these kinds of actions as you can see below in figure 33 with a snippet of the colouring method for warning of low treads. I simply pass in the position that needs to be coloured and the colour for which it should be coloured and then am able to have if statements to colour each position button with the colour given. This was somewhat tedious due to the fact that each button on the interface needed to be accounted for individually with their unique names, however it was made far simpler having a method like this.

```
 private void warning(int CPosition, SolidColorBrush colour)
        {
            if (CPosition == 1)
            {

                Pos_1.Background = colour;

            }
            else if (CPosition == 2)
            {

                Pos_2.Background = colour;
            }
.. .. .. ..
```

Fig. 33. A code snippet demonstrating how methods were used for efficiency.

On selecting a wheel position the user is able to add items to the job cart concerning that wheel, however I wanted to be able to make these options dynamic based on what the user is currently doing, gathering information as required. This was accomplished by using a popup as shown in figure 34, which initially has a dropdown for the user to select what kind of item they wish to add for this wheel. When a complex item such as a Strip and Fit is selected the popup grows and shows the user a variety of new options such as if the stripped tyre needs to be disposed of, and a search for an IPC for a new tyre to be fitted. This kind of dynamic popup is one of the beneficial aspects of an electronic form giving users the ability to fill in certain fields only if required, and not displaying them to the user otherwise.

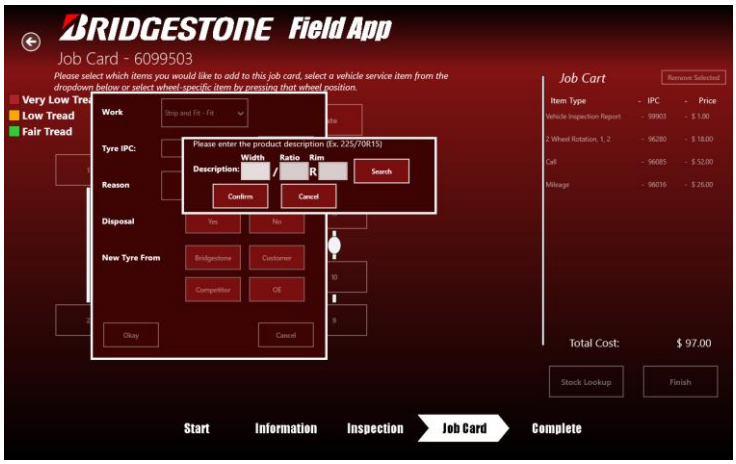


Fig. 34. A screenshot of the dynamic popup for a wheel in a Job Card where wheel items are added.

When completing a job the application will use services in order to interact with the current local databases, as well as with POS for writing a quote. The created services contain a number of methods such as clearing a row, reading entries from tables or writing entries into tables as seen in the class diagram in figure 21 in section

5.4. This means in the C# code for the Job Card page there is code like the section below for creating an instance of a service and calling its service methods. When writing a completed Job Card multiple calls have to be made as shown, a service for interacting with the Job Card tables is initialized, the current row in the database for this job is cleared enabling it to be updated, then after forcing the code to wait for this asynchronous method to be completed the application will write the Job Card back into this row. In a similar process the VIR tables must also be accessed in order to write a completion report, being a report of the final state of the vehicle (tread depths and tyre pressures) as automatically calculated based on the work identified as being done in the Job Card. With successful entries into the local databases the application will proceed to write a quote into the POS databases with a WriteQuote method before proceeding to the next page.

```
.. .. .. ..
           ServiceReference1.Service1Client service = new
        ServiceReference1.Service1Client();
           service.ClearIDAsync(jobCard.IDNo).Wait(500);
           service.WriteJobAsync(serviceCard);
           testService.Service1Client service2 = new
testService.Service1Client();
           service2.ClearIDAsync(pass2.VIRID).Wait(500);
           service2.setOdomAsync(pass2.appOdomDetail, "Completion Report");

            //Try to write a quote into POS
            try
            {
                WriteQuote();
                ResetPageCache();
                this.Frame.Navigate(typeof(Job_Complete));
            }
.. .. .. ..
```

Fig. 35. A code snipped demonstrating the use of asynchronous methods to services

After a Job has been completed the screen seen below is shown, which is a very simple screen giving an indication of the work flow being completed, however the button options as to where the user is able to go from here are worth looking at. They are given the option to go home which will take them back to the start page where they can start any of the apps functionalities again, however the next two buttons are for starting a 'New VIR' or a 'New Job'. I decided to give the user both of these options due to after meeting the Fleet Technicians it was heavily implied that the work flows are not always constant, sometimes they will do a VIR and then a Job Card for one vehicle before moving on to the next vehicle and doing this again, and other times they will want to complete inspections on a number of vehicles one after another and then after this complete Job Cards for each. These two work flows can be seen in the simple diagram in figure 37, and I feel that allowing the user to have the options of going to either a VIR or a Job Card after one Job Card is completed accounts for both of these possible work flows in a simple way, without the need for going back to the start screen and interrupting the flow.

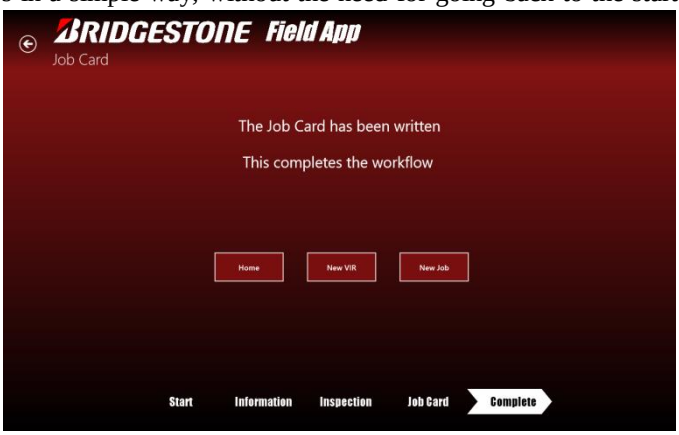


Fig. 36. A screenshot of the completion page for Job Cards demonstrating the possibilities for different work flows.

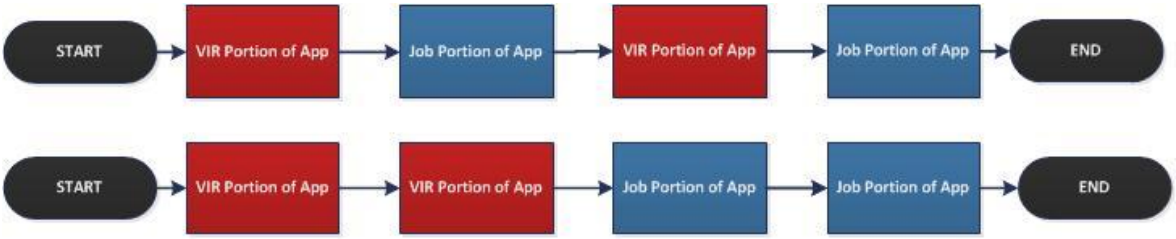


Fig. 37. Diagrams showing different possibilities for a Fleet Technicians work flows.

The Stock Lookup functionality again can be compared to the designs seen in section 7.1, and as you can see from below as this was a simple screen there were not many modifications made during implementation. The left results were made thinner than the location results on the right, due to the fact that the location needed to display long information such as an address and phone number and required more space, but the functionality of searching for an IPC on the left from a description and a location on the right from an IPC remains unchanged. On this screen you can also see that the user has the option to be searching for a specific tyre description allowing them to enter the description in the format 'Width' / 'Ratio' R 'Rim', otherwise there is also an option for a general description to be entered as in the example below, this is the result of adjusting based on usability feedback mentioned in section 7.2.

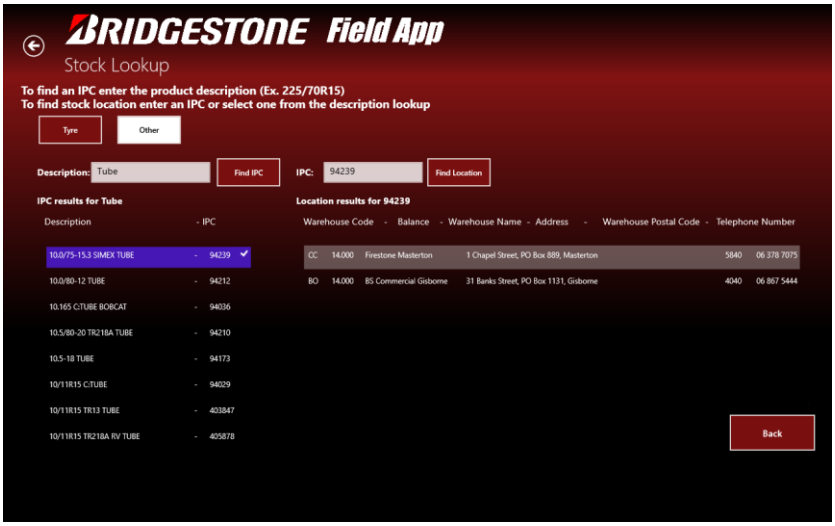


Fig. 38. A screenshot of the Stock Lookup page

In the filling of these result ListViews based on service calls it was necessary to populate a collection of an inner nested class and then to bind this to the view, as described in the problems and solution section 6.2. You can see this occurring in the code snippet below, populating the collection for each result returned from the query and then once this has been completed the DataContext for the list is set to this collection, this is the binding.

```
                for (int i = 0; i < task.Result.IPCs.Count; i++)
                {
                    MyDesc.Add(new
DescIPC(task.Result.Descriptions[i].TrimEnd(' '),
task.Result.IPCs[i].TrimEnd(' ')));
                }

                ListIPCs.DataContext = MyDesc;
```

Fig. 39. A code snippet showing how collections were used and bound to ListViews

During the Incident and Injury reports as there are many toggle buttons that allow for user selection there was a need to account for groups of these that allowed the user to select as many as necessary, as described as a problem in section 6.2. The solution mentioned to this was to handle this as an integer that could be thought of as a binary value, each toggle button relating to a new 'bit' and adding a value to the int accordingly. The implementation of this kind of method is demonstrated below with a series of if statements checking each button and adding a value to the integer accordingly.

```
.. .. .. ..
            if (ToggleNear.IsChecked == true)
            {
                Incident.IncidentType += 8;
            }
            if (ToggleCommunity.IsChecked == true)
            {
                Incident.IncidentType += 4;
            }
            if (ToggleOccupational.IsChecked == true)
            {
                Incident.IncidentType += 2;
            }
            if (ToggleOther.IsChecked == true)
            {
                Incident.IncidentType += 1;
                Incident.IncidentTypeOther = OtherText.Text;
             }
```

Fig. 40. A code snippet demonstrating how an int was treated as a binary number.

The intermediary screen between the Incident and Injury forms required logic to handle creating a number of Injury sub-forms for a single Incident form, which was done by recording the number of people that had been identified as being Injured and working based on that. Names could be entered for the number of people identified and for each of these there was a button in order to proceed to the corresponding form as seen in the screenshot in figure 41 below. As each of these forms were using the exact same Injury pages the switching between Injury forms for different people could not be handled by simple page caching, and was instead handled by testing which form the user was currently in and loading previously written information for that person based on the stored classes for them which worked quite well. One piece of logic dealing with these multiple Injury forms can be seen in figure 42 for testing that every Injury form has indeed been completed before allowing the user to complete the reports, as it was identified as compulsory for these to be filled out.

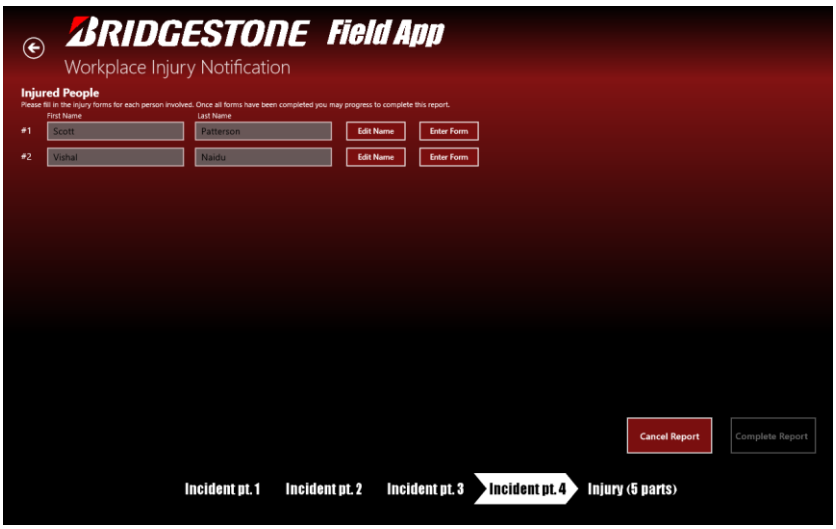


Fig. 41. A screenshot of the intermediary page between Incident forms and Injury forms

```
//Testing for if all Injury reports have been filled out
            allDone = true;
            for (int i = 0; i < Incident.noInjured; i++)
            {
                if (Incident.Injury[i].done != true)
                {
                    allDone = false;
                }
            }
            if (allDone == true)
            {
                Next.IsEnabled = true;
            }
```

Fig. 42. A code snippet demonstrating some logic behind having multiple child Injury forms for an Incident form.

On the completion of these Incident and Injury forms they are saved to the local databases. Visual representations of these can be seen in figure 43 which have been cropped to mainly identify the keys used due to the fact that each of these tables has a fairly large number of columns due to many fields in the forms. There is obviously again a one-to-many relationship between an Incident form and an Injury one, therefore the common keys of Incident Date and Incident Type were used between them. These were identified as keys due to them being well known by those entering the information, compulsory information for the forms, and hold enough variation to ensure these are not repeated in a company like Bridgestone New Zealand. An additional key of Number is used in the Injury tables as an indication of which person this Injury form was for, for example if one Incident form was identified as having two Injured people associated with it, two rows would be created with numbers 1 and 2 in this column. This number was used in favour of something like the Injured individuals name due to the possibility of names being repeated especially when there are separate columns for first and last names, and as number was identified to be a smaller key that is easier for the database to deal with.

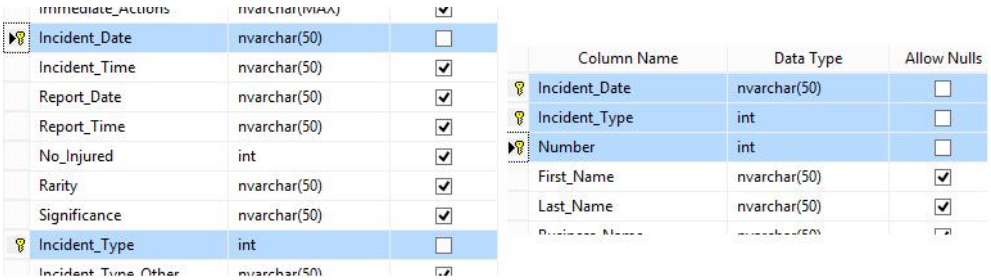


Fig. 43. A cropped screenshot of the tables used for storing Incident and Injury data and their keys.

8. FUTURE WORK

8.1. SYNCHRONISATION

Despite the hefty amount of work put into this application both from myself on the portions described in this report and from the other student working on the VIR portion, there is still a lot of room for future improvements to the application. If in the future this application is built upon to make it more appropriate and robust for use in the field some of the areas in this section should be looked into. Some of these improvements were identified as potential low priority goals at the start of the project and there was not enough time by the end to get around to implementing them, while others were more critical however the best solutions were unable to be accomplished and alternatives were put in place instead.

One such solution with a current less effective alternative in place that should be the initial focus of any future development is that of data synchronisation between local databases on the tablet and remote ones at Bridgestone. This was briefly touched on in section 6.2 when talking about the alternative solution that was found using a connection through Bridgestone's VPN for the writing of Quotes, and a snapshot of the stock data copied to the local database tables. A common fundamental solution to these two problems that would be far

preferable compared to the currently in place alternatives would be to synchronising between the remote databases and the local ones. Having this sort of synchronisation for the interaction with the POS system would allow quotes to be written locally without the need for internet connectivity. This enables a Fleet Technician to complete any job work flows without the need to suspend the job before the end and re-open the application to write quotes later on when an internet connection was available. Synchronisation of stock information would allow the results of the Stock Lookups to be more relevant to the users and more likely to contain the correct stock information, as opposed to an older version depending on when the stock snapshot was last loaded into the tablet. It is possible to manually perform this kind of synchronisation with an internet connection assuming the tables used locally and remotely match appropriately using a series of SQL statements to sort the data, join the local and remote tables, possibly perform a conditional split if any filtering is needed, and then re-inserting back into the tables, however this not only needs to be done manually but also would be sorting and joining entire tables, using a lot of overhead and wasting time.

In the future it would be beneficial to implement automatic synchronisation between these tables allowing the tablet to be used effectively offline and then more accurately while online. As we are using Microsoft SQL Server the most likely choice for data synchronisation would be to use functionality consistent with that, being either Microsoft Sync Framework or simple Merge Replication [47], with their main functionalities described in Table III as I will talk about here. Implementing the Sync Framework would be somewhat more difficult, but provides more flexibility for developers so long as it is implemented properly and sufficiently understood. It is used for a variety of synchronisation services, including synchronisation of file systems and database tables using ADO.NET framework [48]. This is designed primarily for the use of providing offline access to data with caching, exactly what this application requires, but there are also other uses of this in collaborative scenarios. For the scope of this application Merge Replication should be sufficient as the data tables to be synchronised will be in consistent formats and will always be database tables. Merge Replication is still designed for use in synchronising data tables and provides wizards and its own API for configuration, just with more of a focus on the database as opposed to the developer like Microsoft Sync Framework.

Both Merge Replication and Microsoft Sync Framework provide functions to promote efficiency and reduce potential mistakes between databases as seen in Table III. They do these using things like conflict detection in the case that two updates to a synchronised data store overlap. Conflict resolution can be set up and modified in order to determine what should happen to the data rows in this situation, be it the first entry gains priority, or the priority determined by some other weight such as the source device. It is also possible to inform users of such conflicts by flagging them for inspection so they are aware of issues that may arise. They also both allow for the tracking of changed made to the database, allowing only the changed data rows to be synchronised back into the primary data store. This is a massive benefit due to only needing to copy the changed rows as opposed to the whole database for synchronisation, making it much more scalable. This kind of synchronisation uses far less overhead and can be carried out more quickly than normal manual updates to these tables using snapshots.

Implementing this kind of synchronisation could be done in the future with very little modifications to the application itself, as the synchronisation only deals with the database tables themselves. As far as the Stock Lookup is concerned no changes would need to be made as the application currently reads from a local snapshot; if the table that is currently used as this snapshot is synchronised with Bridgestone's databases in order to update frequently then the application will simply be reading the updated values from the same tables. For Quote writing there would need to be minor changes in the application to direct it to write the quote information into synchronised local databases instead of attempting to write directly into POS.

Table III
Features of Merge Replication and Sync Framework from [47]

Key Feature	Merge Replication	Sync Framework
Synchronise by using services	✘	✔
Supports heterogeneous databases	✘	✔
Incremental change tracking	✔	✔
Conflict detection and resolution	✔	✔
Automatically initialize scheme and data	✔	✔
Supports large data sets	✔	✔
Automatically propagate schema changes	✔	✘
Automatically reparation data	✔	✘

This table shows an easy comparison of the key features of Merge Replication and Sync Framework to demonstrate how Merge Replication is capable of everything inside the scope of this project

8.2. DRAG FUNCTIONALITY

As described in section 6.2 drag functionality would be very effective and making the application feel more like a tablet specific application as well as give a better visual representation of wheel swapping as opposed to the instant button pressing in place currently. While the current solution has an end result identical to that of the better solution proposed for this future work it is my belief that there is still a benefit in implementing proper drag functionality for the process of wheel rotations, primarily for ease of use reasons along with making the application as a whole feel more professional and more interactive.

The problems faces when trying to implement drag and drop functionality were outlined in section 6.2, the only alternative as far as dragging was concerned would have been to create a custom control and manually code the handling of dragging and dropping, allowing it to be manipulated based on an X and Y translation and handling manipulation events for positioning with some sort of canvas or Translate Transform. Once this primary function is completed there would be room to implement some form of animation to give an indication as to what buttons were being dragged and what is swapped with what at the end of the process. All of this would have needed to be coded and this was also not feasible given the time frame and skills, however in the future this could be possible with enough research and programming expertise.

One other potential improvement to the current swapping process is that the application does not recognize if the user selects to swap two positions, and then swap them back; it will think two swaps have taken place and will add two items to the job cart for two separate rotations between the same wheels. While this is very easy for a user to correct as it is simple for these items to be removed from the cart it would be more efficient and save the user the trouble if the application detected if a rotation was being performed on wheels that had already been rotated, indicating that the user is just putting them back in their correct places on the application without any rotations taking place in the real world. If the application detected this instead of adding a second redundant and incorrect item to the job cart it should be able to remove the first item instead.

8.3. GPS IMPLIMENTATION

When looking up stock information currently the application allows the user to select a specific kind of stock and then displays all locations that may currently have this stock on hand. While this should be fine there is potential to save the Fleet Technicians time and give an indication as to how close these store locations are to the current user. This could be done using GPS locators inside the tablet to give the application information as to where the user is currently located, and then compare this location with that of the stores in the system. The location of these stores in a way that could be used to calculate the distance they are from the currently location would need to be written for each location, likely in a longitude and latitude format as would be consistent with the GPS location data given for the current location. Based on this the application should be able to order the

results of the stock search based on closest location. Not only this, but there is further work that could be done to better present this information to the user instead of simply ordering results; displaying locations on a map. It is far easier to understand where a location is after seeing it visually on a map, and even so much as a route to a store could be planned based on this. The easiest method for doing this would be to use Google Maps which has published APIs for the use in the development of applications like this one. Google has published a free Maps API for use in free, publically available uses; however this does not apply to the scenario in this application. This application would need to use the more advanced and complete Maps API Premier, which is usable for businesses for international deployments, and asset tracking which is relevant to this application [49]. This API Premier is not free and requires a cost of $10,000 USD or more [50]. Depending on the reach of this application this may seem like too large of a cost until it has demonstrated effective enough use as a proof of concept, improving efficiency in the company significantly.

Another restriction of how stock is currently handled is that stock information is recorded into the database indicating the amount of stock at each store, however the information concerning what stock is located on which Fleet Technician vehicle out on a job is not stored and therefore is not represented in the results of the Stock Lookup.

Each Fleet Technician vehicle is fitted with a GPS and can be tracked, currently Bridgestone uses a Navman Wireless system (OnlineAVL 2) to track the location of its vehicles in order to ensure they are where they should be and that Fleet Technicians are at the locations they are assigned. This also enables Bridgestone to look up which Fleet Technician is close to a callout location should there be the need to send someone to work on a vehicle elsewhere. This Navman system uses Google maps and already uses the API Premier [51], however this system is only one purchased by Bridgestone and therefore Bridgestone themselves to not manage this or have direct access to the API Premier. Navman systems also have an API for use in software integration, this API also comes with a cost depending on what is hoped to be gained from using it, and requires confirmation that it is possible with a Navman representative and a Mutual Confidentially Agreement to be signed for the protection of confidential information [52]. Using this API however along with the Google Maps API Premier I would be able to display the location of not only nearby stores, but also the current location of other Fleet Technicians' vehicles, and if Bridgestone were to add the stock information for items in transit on these vehicles into their stored data, the user would be able to have even more potential sources of stock should other Fleet Technicians be nearby.

8.4. E-MAILS

As identified in section 6.2 currently the application simply reminds the user that permission may be required for the job when they attempt to complete the Job Card, leaving the user with the decision of storing the Job Card should they need to manually obtain permission through either calling or e-mailing a relevant entity. While this is acceptable currently as this mostly resembles how this is already handled, leaving the Fleet Technician to obtain this themselves, there is definitely room for improvement. If the application were able to determine which jobs required permission and which did not then it could be programmed to use the previously obtained customer information and send off an e-mail to the correct person requesting permission. It may be difficult to prevent the user from writing a quote for this job regardless of permission however, as the application will not be informed once permission has been granted or denied, though this should be acceptable as an incorrect quote being written can be easily ignored in the POS system.

8.5. COMPREHENSIVE VALIDATION

Two of the initial project goals that were talked about but not considered crucial to the work flow of the app were those of performing validation based on Service Level Agreements (SLAs) and Credit Limits for each vehicle in each company. This kind of validation could be performed when attempting to complete a Job Card with logic in place to test the job items listed with what should be allowed on that vehicle. For example if a specific customer had identified in their SLA that the tread depth for wheels on their vehicles should never go below a specific threshold and the Job Card identifies that at the time of a completion report being created that tread depth still remains below this value, the user could be informed of this and prompted to perform additional work. The difficulty in adding this kind of validation to the application was with these SLAs for each customer

currently being available only in written PDF format. It would prove to be quite a hefty task to convert these text SLAs into data logic, and implement them across every vehicle type that could be owned by every customer. This sort of logic engine would definitely be possible in the future but would take time to implement and therefore was not completed during the course of this project.

Similarly with validation if information about a customer's credit limit and current balance with Bridgestone was collected inside the app it would be a simple task to validate the total cost of a job against these values in order to determine that this work is definitely able to be performed. The difficulties encountered with this in development of the app were that customer balance information may be somewhat delicate, and in addition to this the values read into the application would need to be gathered and updated as often as possible to ensure that they are correct and no false positives were thrown.

8.6. VISUAL ADDITIONS

There are a few changes that could be made to what is presented to the user with the application in order to give the whole experience a smoother or more professional feel to it. Two of these changes were mentioned in section 6 as problems where alternative solutions were found, however a fundamental solution would have been more ideal given the time and skills to do so. The first being the colouring of wheel positions based on how likely they are expected to require work based on their corresponding VIR. The specific IPCs of the tyres on each position would be required in order to better customize this recommendation to the user, suggesting that the tread depth is too low when it is under a percentage of the tread depth for a brand new tyre of that type. However the problem explained before was that it was not important enough to give suggestions so accurately at the expense of making new calls to remote databases over the internet, allowing for less offline functionality. In the future it would be possible to gather this information at the start of the VIR portion of the application along with other vehicle data which is loaded, this would allow the data to simply be stored inside the application and VIR database tables and passed along or read into to a Job Card from there, requiring no extra connections to be made, just simply an extra request for data rows in a connection to databases already made earlier on in the work flow. The second change which was mentioned in part 6 was concerning the buttons displayed to the user during the Injury reports attempting to replicate a human diagram on the paper form that allowed for people to circle body parts. I mentioned that the solution to this that the application currently uses is simply arranging buttons in the shape of a human, using the size and placement of these buttons to indicate a body part, however I would like to use images in these buttons to more clearly show this. With my backend programming in place to convert images to the foreground of buttons allowing them to be switched between clicks, representing active or inactive states I attempted to create some images to put into use but with little success at making them look clean and professional. Figure 44 demonstrates the kind of display I hoped to give, showing human images inside the buttons; however it is clear that these are not clean or angled right due to just being scanned right off of the paper form and arranged to fit the buttons I have in place. In the future if these images could be cleaned up, resized and rotated to better fit the buttons in a way that did not look tacky when compared to the consistent smoothness of the interface throughout the rest of the application, this would give a more engaging user experience feeling more natural and assisting the user visually in their filling out of the form.

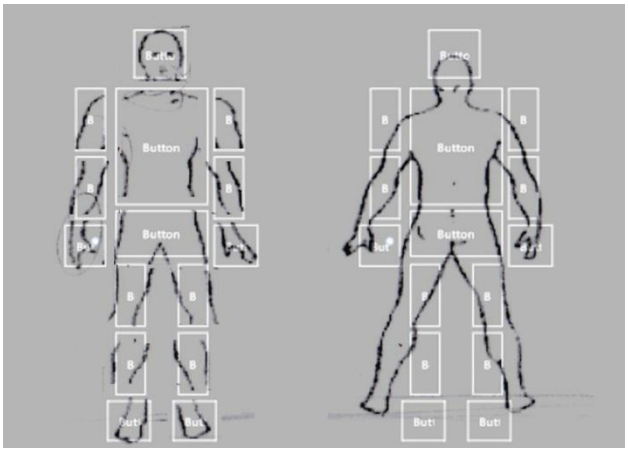


Fig. 44. An image showing a test version of using images as buttons to mimic the Injury form.

There are other areas where there is room for visual improvements to be made to the application, one somewhat related to the colour indications of the wheels mentioned above. While the Job Card indicates which wheels have low tread by colouring them a different colour depending on the remaining tread, the actual numerical values for the tread depths identified on the VIR are no longer visible. Since this data is already stored in the page at this point it would be very simply to display it to the user, however there are issues with displaying it alongside each wheel, for every position, for every type of configuration. There is not enough room to display it in this way so that it could be clearly viewed regardless of the configuration type and therefore this was not done, as further design analysis and potential modifications were necessary. Given the time to make these changes or to implement an alternative that does not negatively affect the user experience I feel that this could be useful in giving the user information that may help them decide what work to do, as opposed to relying on the simple colour indications and their own memory. One other more visual action that could be implemented would be that of allowing users to sign their forms on completion as proof of having completed them, this would promote security as well as make use of the tablets touch screen display. This functionality was attempted incorporating the current touch screens handwriting functionality, however this automatically attempts to recognize what it written and convert it to text. There were methods of reading drawn lines for use as an image using a canvas and Ink Stroke recognition inside the application, however there was not enough time to actually implement these. The process of adding a canvas to the application and attaching the four required input events for different pointer actions inside this canvas does not seem overly complicated. The four events that need to be handled are PointerPressed, PointerMoved, PointerReleased, and PointerExited in order to give the application information in how the user is interacting with the canvas [53]. Converting the strokes drawn to the canvas to an image should also not be a task that required more than a day, there was simply not enough time in available to work on a low priority function such as this during the project.

8.7. ANALYSIS OF DATA

The of the directions that Bridgestone New Zealand wishes to move forward in in the future is that of taking a more proactive approach to their business. Currently the two standard processes for carrying out jobs is for either a customer to enter a Bridgestone store with a known issue and requesting work or an inspection to be done, or for Fleet Technicians to go out to a fleet site and inspect various vehicles assigned to them in order to identify where work may be needed. Both of these scenarios require work to be found before a job is done and depend entirely on either a customer or Fleet Technician to identify these problems. Bridgestone hopes to in the future have some sort of data analysis that is able to calculate the average distance travelled by a certain vehicle in their database, the expected tread wear over time for vehicles travelling this kind of distance, and then compare this to the inspected tread depths of previous VIRs in order to produce a sort of notification as to what vehicles are expected to probably need work based on average and expected trends. The end result of this sort of pro-active data analysis would be for Bridgestone to monitor the expected tread depths of all vehicles they frequently work on, and have some idea as to what the tread depths will probably be like for every position on every vehicle prior to actually going to a fleet site and performing an inspection. The inspection will still be required as the data from this analysis would only be based on expected averages, however it would still allow a Fleet Technician to have some idea of what vehicles they should give their focus and make sure they do not miss certain vehicles, as they will know what vehicles they want to check before going on site.

As the application created in this project deals with reading and updating this sort of tread information through VIRs and completion reports after jobs have been done we feel there is potential to link the data recorded by the application into any future system that performs this sort of data analysis, allowing the application to play a central part in the gathering and filtering of relevant data to be used in not only the current job process, but also in this future analysis. There could be potential for the application to read the tread information based on previously saved inspections as well as the date and odometer reading at these times, allowing logic to be put into the application that calculates the distance travelled between inspections and the average tread depth loss over this distance. All information required for this logic is already recorded by the application and converting it into some form of pro-active logic that suggests vehicles that are expected to have low tread feels almost like a natural process to undertake on this data.

9. CONCLUSION

9.1. SUMMARY

Over the year a lot of time and effort has been given to this project, both by myself as well as my supervisors at Bridgestone New Zealand. Bridgestone first presented the idea for this project to the University of Auckland primarily with the motivation of improving the efficiency of their Fleet Technicians by both promoting a more consistent and manageable workflow, but also by removing the redundancy that occurs with entering Job Card information multiple times before it is finalised in the database. The goals that were identified as best meeting this motivation were to create a smooth tablet application that was easy to use and felt familiar to the users, mimicking the current paper forms that they use. The goals for this applications functionality were to be able to implement a full end to end process starting at the user being able to enter Job Card information about a specific vehicle, and ending with invoice creation through Bridgestone's AdvanceRetail POS system. These introductory pieces of information are detailed during section 1 of this report. Throughout the end to end process another project was also needed, which implements the functionality of inspecting a vehicle and gathering information about the vehicles current state and the customer that owns it, as mentioned in section 2.

Throughout the duration of this project I have comprehensively planned the project as written in section 3, identifying why an electronic form is a good project to meet the motivations that Bridgestone has, as well as planning a development strategy to be used throughout the year as development is done on the application. In order to plan the project effectively I have become very familiar with Bridgestone as a company and how they work. I have been familiarised with their systems as well as how they handle work processes. Seeing these in a professional environment has been very beneficial to me as a student and I have come to feel the weight of what this project means.

Section 4 gives visibility to the extensive research that was carried out over the course of this project to solve problems that needed to be addressed before the designing and development of the Electronic Job Card portion of the application, including meeting the end users and seeing how my application would be used by them in their everyday work flows. I have been able to recognise various areas in which my application should be able to promote efficiency in the Fleet Technicians work and through many discussions with various people inside Bridgestone I have been able to clearly define the scope of the project, which proved to be difficult as it needed to be clearly distinct from but still closely linked to the VIR portions of the application that will be developed alongside the Electronic Job Card parts. Research was also done in order to make informed decisions when deciding what the application would be developed on, how to develop on different operating systems, and the advantages and disadvantages of these options. These options were personally experimented with in order to get a clear idea how it feels developing for these systems and to better inform my decision. Similarly research was done into the options of how to implement web services that were to be used by the application for reading and writing data from databases, outlining the advantages and disadvantages of different methods and performing real tests with them to support my decisions. As the application will be dealing with a lot of data and databases the options available in what kind of DBMS could be use needed to be researched and decided, as well as what kind of functions this database system would be able to give to the application in order to better meet its goals. As the final application needed to be handed over to Bridgestone New Zealand in a way in which they would be able to feasibly use it in a real environment methods of application deployment were also researched.

During the development stages of the application a variety of different models were created in order to give an abstracted view of what the application can do, and how it does it. Section 5 explains these in a way that is supposed to promote understanding of the application for persons external to the application development and enable them to understand various parts of the application in a modelled way. Also during the development of the application a number of problems were faced, both technical and non-technical that needed solutions in order to create an application that is able to meet its goals in a way that keeps the user in mind. Section 6 explains these problems and the process that I undertook to look into and implement solutions for each one.

As the actual implementation of this project is very practical and functionality based the coding of the application took a central part, taking more time and effort than all other aspects combined. The results of this effort are detailed in section 7 which talks about the final implementation of the application and some main functionality that it is capable of, giving some information as to how these were accomplished. Section 8 is

somewhat of an evaluation and a plan, as it details all future work that could be done on this application should it be used in the future. Most of the functionalities described here were not implemented due to time constraints in the coding of them, however research into how these could be done in the future has been mentioned and should be beneficial to Bridgestone should the chose to develop on this further.

9.2. RESEARCH IMPLEMENTATION

As this project has been undertaken with the intention of producing a functional application that is usable in a real life situation it was imperative that the research carried out was done thoroughly, and all options considered. The pros and cons of every option needed to be weighed as well as the feasibilities behind these in order to make an informed decision as possible, enabling me to create a well thought out and proper final application to be handed over to Bridgestone. I felt that I was able to perform extensive research in many areas relevant to this application, either gathering credible academic resources where possible or carrying out my own tests and experiments, all of which were crucial in my decision making process in making choices in how to proceed.

My personal interactions with the potential end users of this application, Bridgestone's Fleet Technicians, allowed me to identify many areas in which there was potential for the application to improve efficiency and promote an easy work flow. I was also able to gain a wide array of information about how things are currently handled by Bridgestone which was key to developing an application that is easy to use and familiar to the user, allowing it to be efficiently implemented in real life without many complications or training required. I definitely put the information gained from this to use in planning the work flow of the application as well as in designing the user interface.

The applications could not be created without first deciding on what operating system to develop for, and therefore research was done on the various systems and their pros and cons. The ease of developing for each of these systems was looked into and I performed experiments to gain real experience with both Windows 8 and Android which were identified as the most likely operating systems to develop for. Through these experiments I was able to make well informed decisions on which was preferable that I could discuss with Bridgestone in the decision of what kind of device to develop for. I used Windows 8 out of these two top choices because I personally had a more positive experience in developing with it, as well as due to Bridgestone being happier in developing for this kind of device.

As pointed out by my academic supervisor when dealing with web services for remote database access there are different methods of implementation that should be considered, primarily those of SOAP and REST. Due to being informed that this would be important I was able to research into academic reports as credible sources for the differences between these implementation as well as the theory behind which should be used in what situations and why. In addition to these readings to understand the theory behind these I also performed tests of my own in order to have personal feedback and experience with both options, understanding the response times involved in the two options. With this research I feel I was able to understand these options in depth and be better informed when I made the decision to use SOAP over rest. Weighing the positives and negatives of both sides I decided that while REST did have definite advantages, they were not significant enough to complicate things and force Bridgestone to adapt to using REST for this application.

Although Bridgestone had already installed and were familiar with some database systems I felt it was necessary to at least look at the alternatives and determine if there were other benefits that should be considered. If there were I would have been able to inform Bridgestone of these and possibly suggest using something else instead, however after researching into the functionalities provided by some major DBMS I was able to determine that none of the different functionalities provided were required and the decision was made to continue using Microsoft SQL Server for this application. However with my research into Microsoft SQL Server Compact it was identified that this would be more ideal for use on tablets should this application be put into use, and therefore this was suggested as the best option to Bridgestone should they take this forward in the future.

All of the research performed had major impacts in the decisions made in progressing with this project with methods that would enable me to produce an application that is able to best meet as many goals as possible in ways that best match up with the initial motivation behind this project.

9.3. ACHIEVEMENT VS. GOALS

Overall the implementation of this project was infinitely larger and more than any project I had ever worked on before, with the main application totalling 11,227 lines of code as seen in figure 45. Services were also created on top of this each with a fair amount of coding logic held within them as well. I feel that the implementation of this project has been handled in a fairly robust way accounting for a variety of possible work flows and giving users the ability to perform all the core functions that should be needed in an efficient way. On top of this the application accounts for a large number of possible exceptions or human errors in input giving the user feedback as to what went wrong and the likely reasons for why, allowing them to correct the mistake themselves without breaking the application and forcing it to close, possibly losing inputted data and time.

Hierarchy	Maintainabil...	Cyclomatic ...	Depth of Inh...	Class Coupli...	Lines of Code
Bridgestone App (Debug)	79	3,612	8	207	11,227
JobService (Debug)	90	217	1	29	570
testService (Debug)	90	297	1	30	631
VehicleRepositoryStandard (Debug)	84	18	1	16	38

Fig. 45. A list of projects inside the application solution with some code statistics

The primary goal for this project, reiterating the motivation was *"A mobile application needs to be developed for their electronic job card. This is to emulate the manual recording of information on a job card to a Mobile device with a view linking ultimately to the AdvanceRetail POS system."* The application needed to mimic a paper form for the input of Job Card information in an efficient way, linking with POS for invoice creation. At the end of this project I have been able to do exactly this, I have created an electronic form that incorporates the functionality given by the paper form, allowing for Job Card information to be entered and electronically stored instantly without disrupting the work flow or needing to be re-entered. In this process I was also able to meet additional goals that are common to electronic forms in making the form dynamic, showing the user only fields that they need to be concerned with and hiding all others, allowing for these to be displayed only when the user makes decisions that would lead up to this. This, along with using tools such as DatePickers and toggle buttons allow the electronic form to outshine its paper counterpart with its ability to reduce the need for manually writing out the inputs for each field, and reduces the possibility of users making mistakes.

I was able to incorporate an interactive configuration diagram based on a link to the VIR portions of the app; reading in relevant vehicle information to display to the user and recommend the user as to where work may potentially be needed, with the colouring in of wheel positions based on the values entered by inspection reports. At the end of the Job Card process I was able to successfully write both a completion report showing the state of the vehicle after work had been done, as well as a Quote to Bridgestone's AdvanceRetail POS system which was given as a primary goal. Proof of this end to end process can be seen in figure 46, showing a quote written into the POS system by the Electronic Job Card which can then be checked for accuracy, and easily turned into an invoice as was needed.

Quote No	Status	Contact Name	Issued Date	Expire Date	Store	Sales Person
96006099502	Open	Default Salesperson	17/10/2012	17/11/2012	BF Commercial Ea...	

Option No	Recommended	Notes	Exp. Purchase Date
1	☑	Quote Option created by electronic job card	17/10/2012

Fig. 46. Proof of a quote being written into Bridgestone's AdvanceRetail POS system from the Job Card

I was able to go above the motivation of the project and implement the additional functionality of Incident and Injury reporting as well as Stock Lookups. The Stock Lookup functionality in particular can be very beneficial in promoting efficiency by saving the time needed finding out which stores have which stock in a way that does not disrupt the work flow.

In meeting all of these goals the final application can perform an end to end process, being completed in a way that allows the application to successfully write quotes to a database to be converted into invoices. This process also meets the goal of being able to be done outside of Bridgestone offices with the use of Bridgestone's VPN. I feel that this project has been a success and the final application that was produced definitely has merit

to the business in promoting efficiency. The project supervisors inside Bridgestone have expressed that they feel the same, and they are excited about getting more familiar with the application and testing its functions more in real scenarios. One of the project supervisors has informed me that he will be demonstrating the application to an important individual from Bridgestone Australia which has recently joined forces with Bridgestone New Zealand to form one strong organization. He feels that this application has gone above being a simple proof of concept and with the incorporation of a successful end to end process has real ability for improving productivity inside Bridgestone, allowing for a significant potential financial gain, easily meeting the company's motivation.

ACKNOWLEDGEMENT

Scott Patterson thanks senior lecturer Dr S. Manoharan for giving the opportunity to take part in this project and for overseeing the BTech degree, as well as for being the supervisor for this project specifically; he has consistently given me feedback on my progress and given me advice on where my focus should be throughout the development of this app. I would also like to thank Bridgestone's senior business analyst Robert A. Lee for looking over my work at Bridgestone and frequently checking up on what I have done to give me advice on where I should be heading, keeping me organized and giving ideas for the app itself. Also Bridgestone's business analysts Candy He for supervising my work while at Bridgestone at the start of the year, and Akmal Akmaloni at the end of the year. Another business analyst Philip Low for constantly giving helpful advice and being there to answer questions and discuss the project with me; he set aside an immense amount of time to teach me of Bridgestone's systems and to provide help when it was needed and I deeply appreciate his understanding of the project and what needed to be done. David Vale, commercial business solutions manager from Bridgestone for being so positive about the project and motivating me to meet his standards. Finally Vishal Naidu who worked on the VIR sections of the application alongside me for being there to bounce ideas off and provide thoughts and advice when I was stuck; having another student in the same position as myself was very reassuring.

REFERENCES

[1] *Apple unveils iPhone | Macworld*. (2007).
Retrieved from http://www.macworld.com/article/1054769/iphone.html
[2] *HTC - Touch Phone, PDA Phone, Smartphone, Mobile Computer*. (2008).
Retrieved from http://web.archive.org/web/20110712230204/http://www.htc.com/www/press.aspx?id=66338
[3] *Google's Android becomes the world's leading smart phone platform.* (2011).
Retrieved from http://www.canalys.com/static/press_release/2011/r2011013.pdf
[4] *Microsoft announces ten Windows Phone 7 handsets for 30 countries*. (2010).
Retrieved from http://www.engadget.com/2010/10/11/microsoft-announces-ten-windows-phone-7-handsets-for-30-countrie/
[5] *Company overview - Bridgestone*. (2008).
Retrieved from http://www.bridgestone.co.nz/corporate/page/company_overview
[6] Gehani, Narain, "The Potential of Forms in Office Automation," *Communications, IEEE Transactions on* , Vol. 30, No.1, pp. 120- 125, Jan. 1982
[7] Christiansen, David; Hosking, James; Dannenberg, Andrew; and Williams, O.Dale, "Computer-assisted data collection in multicenter epidemiologic research: The atherosclerosis risk in communities study," *Controlled Clinical Trials*, Vol. 30, Iss. 2, pp. 101-115, Apr. 1990
[8] Low, Rouhshi "From Paper to Electronic: Exploring the Fraud Risks Stemming From the Use of Technology to Automate the Australian Torrens System," *Bond Law Review*: Vol. 21, Iss. 2, Article 7. 2009
[9] Halonen, Raija, "Digitizing Information Management From paper forms to electronic information system," *Digital Information Management, 2006 1st International Conference on* , pp.351-358, Dec. 2006
[10] Centers for Medicare & Medicaid Services (CMS) Office of Information Service "Selecting a Development Approach," Mar. 2008
[11] Barlow, Jordan; Giboney, Justin; Keith, Mark; Wilson, David; Schuetzler, Ryan; Lowry, Paul; and Vance, Anthony "Overview and Guidance on Agile Development in Large Organizations," *Communications of the Association for Information Systems*: Vol. 29, Article 2. 2011.
[12] Brancewicz, Aleksander "Building Highly Productive Teams Using A Commitment-To-Progress Ratio, Part 1" *Better Software* Vol. 14, Iss 4. 2012
[13] Chasalow, Lewis, "What Makes Agile Development Different?: A Case Study of Agile In Practice," *SAIS 2008 Proceedings.* Paper 2. 2008
[14] *The benefits of agile development | Mixd.* (2011)
Retrieved from http://www.mixd.co.uk/blog/mixd/the_benefits_of_agile_development/
[15] *iOS Human Interface Guidelines: Introduction*. (2012).
Retrieved from
http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/Introduction/Introduction.html
[16] *Developing apps for iPad – Apple Developer*. (2012).
Retrieved from https://developer.apple.com/ipad/sdk/
[17] *iOS Developer Enterprise Program - Apple Developer*. (2012).
Retrieved from https://developer.apple.com/programs/ios/enterprise/
[18] *User Interface Guidelines | Android Developers*. (2012).
Retrieved from http://developer.android.com/guide/practices/ui_guidelines/index.html
[19] *iPhone 4S Battery Life Testes And Compared With Android Smartphones*. (2011).
Retrieved from http://www.redmondpie.com/iphone-4s-battery-life-tested-and-compared-with-android-smartphones-the-result-may-just-surprise-you/
[20] *Android NDK | Android Developers.* (2012).
Retrieved from http://developer.android.com/sdk/ndk/index.html
[21] *Publishing Overview | Android Developers*. (2012).
Retrieved from http://developer.android.com/guide/publishing/publishing_overview.html
[22] *UX guidelines for Metro style app development | BlendInsider*. (2011).
Retrieved from http://blendinsider.com/technical/ux-guidelines-for-metro-style-app-development-2011-10-21/

[23] *Windows Phone NZ : Review: Nokia Lumina 800.* (2012).
Retrieved from http://windowsphonenz.com/wp7/review-nokia-lumia-800.html
[24] *Getting started with Metro style apps*. (2012).
Retrieved from http://msdn.microsoft.com/library/windows/apps/br211386
[25] *Windows Store markets*. (2012).
Retrieved from http://msdn.microsoft.com/library/windows/apps/hh694064
[26] Russello, Giovanni; Conti, Mauro; Crispo, Bruno; and Fernandes, Earlence, "MOSES: Supporting Operation Modes On Smartphones," *SACMAT '12 Proceedings on the 17th ACM symposium on Access Control Models and Technologies*, pp. 3-12. 2012
[27] Russello, Giovanni; Crispo, Bruno; Fernandes, Earlence; and Zhauniarovich, Yuri, "YAASE: Yet Another Android Security Extension," *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)*, pp. 1033-1040. Oct. 2011
[28] *Xamarin* (2012)
Retrieved from https://store.xamarin.com/
[29] *Windows 8 for Android - Android Apps on Google Play* (2012)
Retrieved from https://play.google.com/store/apps/details?id=com.gabelic123.desktop8
[30] *Windows 8 will run Android apps* (2012)
Retrieved from http://digitaljournal.com/article/333860
[31] Muehlen, Michael; Nickerson, Jeffery; and Swenson, Keith, "Developing Web Services Choreography Standards - The Case of REST vs. SOAP" Decision Support Systems - Special Issue: Web services and process management, Vol. 40, Iss 1, pp. 9-29. Jul 2005
[32] Potti, Pavan, "On the Design of Web Services: SOAP vs. REST," *UNF Theses and Dissertations.* Paper 138. 2011
[33] Kennedy, Sean and Molloy, Owen, "A Framework For Transitioning Enterprise Web Services From XML-RPC to REST," *CONF-IRM 2009 Proceedings.* Paper 52. 2009
[34] *Web Services, Part 1: SOAP vs. REST | Ajaxonomy.* (2008).
Retrieved from http://www.ajaxonomy.com/2008/xml/web-services-part-1-soap-vs-rest
[35] *MySQL :: Why MySQL?* (2012)
Retrieved from http://www.mysql.com/why-mysql/
[36] *About SQLite* (2012)
Retrieved from http://www.sqlite.org/about.html
[37] *Creating Linked server to MYSQL from SQL Server* (2009)
Retrieved from http://sql-articles.com/blogs/creating-linked-server-to-mysql-from-sql-server/
[38] *Microsoft SQL Server vs MySQL vs* SQLite (2012)
Retrieved from http://database-management-systems.findthebest.com/compare/26-30-53/Microsoft-SQL-Server-vs-MySQL-vs-SQLite
[39] *Database System | Performance & Stability | SQL Server Express Edition* (2012)
Retrieved from http://www.microsoft.com/sqlserver/en/us/editions/2012-editions/express.aspx
[40] *Data Storage Architecture with SQL Server 2005 Compact Edition* (2007)
Retrieved from http://msdn.microsoft.com/en-us/library/bb380177.aspx
[41] *How to Add and Remove Apps* (2012)
Retrieved from http://technet.microsoft.com/en-us/library/hh852635.aspx
[42] *Deploying Metro style apps to businesses - Windows Store for developers* (2012)
Retrieved from http://blogs.msdn.com/b/windowsstore/archive/2012/04/25/deploying-metro-style-apps-to-businesses.aspx
[43] *Getting a developer license* (2012)
Retrieved from http://msdn.microsoft.com/en-us/library/windows/apps/hh696646%28v=vs.110%29.aspx
[44] *Guidelines for scaling to screens (Windows Store apps)* (2012)
Retrieved from http://msdn.microsoft.com/en-us/library/windows/apps/hh780612.aspx
[45] *Quickstart: Data binding to controls (Windows Store apps using C#/VB/C++ and XAML)* (2012)
Retrieved from http://msdn.microsoft.com/en-us/library/windows/apps/xaml/hh464965.aspx

[46] *Windows 8 Controls, Windows 8 UI Components | Telerik RadControls for Windows 8* (2012)
Retrieved from http://www.telerik.com/products/windows-metro/overview.aspx
[47] *Microsoft Sync Framework > Synchronising Databases > Overview* (2012)
Retrieved from http://msdn.microsoft.com/en-us/library/bb902818%28SQL.110%29.aspx
[48] *Introduction to Sync Framework Database Synchronization* (2012)
Retrieved from http://msdn.microsoft.com/en-us/sync/bb887608
[49] *Google Maps API Premier - Features, Benefits, and FAQ* (2011)
Retrieved from http://www.google.com/enterprise/earthmaps/maps_features.html
[50] *Google Maps API for Business - Google Earth and Maps Enterprise* (2012)
Retrieved from http://www.google.com/enterprise/earthmaps/maps-faq.html
[51] *Navman Wireless New Zealand | GPS Vehicle Tracking & Fleet Management Solutions* (2009)
Retrieved from http://www.navmanwireless.com/about-us/press-releases/2010-press-releases/139-google-maps-to-onlineavl2
[52] Navman Wireless API Brochure (2010)
Retrieved from http://www.onscene-solutions.com/towing-software/navmanbrochure.pdf
[53] *Quickstart: Capturing ink data (Windows Store apps using C#/VB/C++ and XAML)* (2012)
Retrieved from http://msdn.microsoft.com/en-us/library/windows/apps/xaml/Hh974457%28v=win.10%29.aspx