BTech 451 2015

Final Report

CHEP Pallet Management System

Avish Kapadia
akap171 | 2657196
COCA-COLA AMATIL NEW ZEALAND

Contents

Abstract

I undertook a project for Coca-Cola Amatil (“CCA”) and created a solution to improve their current Commonwealth Handling Equipment Pool (“CHEP”) pallet management process. Pallet liabilities were manually entered into the system by CCA Distribution Centre employees and as a result certain scenarios existed in-which pallet transfers did not occur. Therefore, CCA were liable for these ‘unaccounted’ pallets. The main goal of my project was to automate and correctly determine the pallet liability transfer process of pallets.

The first half of my report focusses on the business issue and possible alternative solutions that already exist within the marketplace. I also proposed changes for CCA’s current system and business processes that included the use an intermediary system. In the second half of my report I focus on the steps I took to designing and developing an intermediary solution also known as my Python application to address CCA and CHEP’s requirements. My developed intermediary solution significantly improves business processes and also avoids the governance that is set in place by Coca-Cola Amatil Australia.

Disclaimer

People

Author

Avish Kapadia

Academic Supervisor

Dr. Giovanni Russello – University of Auckland

Industry Mentors

Jet Wu – Coca-Cola Amatil New Zealand

Darren Ruston – Coca-Cola Amatil New Zealand

BTech. Information Technology Coordinator

Dr. Sathiamoorthy Manoharan – University of Auckland

Acknowledgements

I would like to thank the following people for their tremendous continual support whilst I undertake my Bachelor of Technology Degree and BTech 451 Project.

My parents: Anil & Darshana Kapadia.
Academic Supervisor: Dr. Giovanni Russello.
Industry Mentors: Jet Wu and Darren Ruston.
BTech. Information Technology Coordinator: (Mano) Dr. Sathiamoorthy Manoharan.

The Company

Coca-Cola Amatil New Zealand are the licenced bottlers of the Coca-Cola Company in New Zealand. CCA produce an extensive range of well-known products such as L&P, Coca-Cola, Coca-Cola Zero, Coca-Cola Diet, Powerade and

Keri Juice. CCA has been in operation since the early 1900's and currently employs well over 1000 employees in New Zealand alone. Within New Zealand CCA has five production facilities located across Auckland, Putaruru and Christchurch (Coca-Cola Amatil, n.d.).

CCA have a significant impact on the local economy. Recently CCA spent 50 million NZD upgrading two CCA facilities within Auckland and Christchurch to support the blow-fill technology. In 2008 CCA spent 80 million NZD to create their new and advanced distribution centre in Auckland (Coca-Cola Amatil, n.d.). With these investments and upgrades, customers are provided with a high quality product within in timely manner.

Motivations

The goal of my project was to improve the efficiencies around CCA's current CHEP pallet movement. CHEP pallets are distinct blue coloured pallets that have standardised sizes for distribution within New Zealand. CHEP pallets are a key requirement when supplying goods to supermarkets and major organisations. These pallets meet rigorous standards and undergo compliance audits (CHEP, n.d.). The current CHEP pooling model is designed for pallets to be repaired and reused. CHEP reuses damaged pallets in an effort to reduce landfill. CHEP pallets provide many benefits, such as:

- Compatibility with existing supply chain infrastructure (CHEP, n.d.)
- Greater efficiencies during transport and storage (CHEP, n.d.)
- High pallet quality and consistency via CHEP's worldwide network (CHEP, n.d.)
- Load stability and reduced product damage (CHEP, n.d.)

The current CHEP pallet reconciliation process is extremely time consuming and takes approximately four days to complete. In addition to the four days required, the pallet reconciliation process requires the assistance of 3 additional internal systems to balance CHEP pallets nationwide. Thus, the logistics team have requested me to complete the process improvement to include customer pallet movements.

Furthermore, the CHEP pallet management process is complex and manual and these processes leave the business open to potential losses with pallets being given to customers without receipts. CHEP holds CCA accountable for these pallets and for inaccuracies in pallet inventory balances, which CCA are financially accountable for. Pallets can be dispatched to distribution centres throughout New Zealand, customer's nationwide or shipping companies. Shipping companies / Vendors such as Toll also distribute CHEP pallets, in this scenario the shipping companies take ownership of the accountabilities associated with the pallets.

At present, all information is manually keyed into CHEP Portfolio Plus, which is CHEP's pallet accounting system that tracks pallet movement and manages transfers of pallet liability. The issue within the business process that I had identified was the lack of accountability and the data integrity factors not being met. Information can be simply forgotten about, which then results in the pallets not being transferred to a customer. In these mentioned scenarios, CCA is left accountable for these pallets which have not been recorded but have already been transferred to a customer/shipping agent. This issue at present costs CCA hundreds of thousands of dollars per year.

My Project in a Nutshell

To identify and create a solution for CCA, I had to initially investigate and address the current issue at CCA. This required a thorough understanding of CCA's current business processes and the ability to identify areas of weakness. Once I understood the extremely complicated business process in regards to the inwards and outwards goods process, I was able to determine areas that could be significantly improved. When I initially began my project all shipping information was manually entered into CHEP Portfolio Plus software which tracks the pallet movement and manages transfers.

The solution I have designed and created is an application that will transfer and modify set fields of shipment information from CCA's current SAP system into Portfolio Plus. In addition to the aforementioned tasks, my Python application will also be mapping customers to shipments and making changes to the information as required. Though there are possible alternative methods that do not require the use of an intermediary system, I am unable to proceed with them due to the resourcing and governance that have been set by Coca-Cola Amatil Australia. Coca-Cola Amatil Australia have put strict conditions in place which prohibit me from making multiple changes to the existing software and business workplace infrastructure.

Furthermore, the application created removes these barriers as the application allows for greater flexibility and control. Flexibility is achieved in the sense that the application can be altered by myself or my nominated CCA representative at any time. I do not require clearance from CCA Australia to make adjustments to the application. As I created this solution for CCA, I had to ensure that I met CCA's goals of achieving data integrity of information and zero discrepancies between actual stocktake count of pallets and reported amount of pallets on hand.

Feasibility Analysis

Programming Languages

As part of the design process and research required for my BTech project, I researched and evaluated the programming languages that would be suitable for the creation and implementation of my solution for CCA. The primary purpose of this application was to have information parsed from CCA's SAP system which is based on the SAP ABAP language – extracted from my created query within SAP and parsed into my Pallet management application for manipulation. The extracted information is then manipulated in accordance with the requirements set by CHEP. In addition to the requirements set by CHEP I ensured that CCA's business processes are met and pallet transfers are completed successfully. They were not to be performed in a manner which meant that CCA's customers were transferred pallets that they were not meant to be held accountable for.

Several fields of extracted information has been adjusted prior to being forwarded onto CHEP's Portfolio Plus application which accounts for a key requirement of my project. The transfer was done via the use of CHEP's EDI system. My Python application transfers all the shipment orders and related information to the CHEP EDI system which then feeds the information into the

Portfolio Plus software that the CCA logistics team can then view and modify if required.

Therefore, bearing the requirements of the application in mind, my research below covers three major programming platforms. These languages are SAP Advanced Business Application Programming (ABAP), Python and SQLite. After concluding my assessment study on these languages, I chose the most essential languages that I must know sufficiently in order to be able to complete the design and creation phase of my Python application.

Due to the requirements of my project, I was required to use different platforms to complete my project to a high quality standard. Bearing in mind the quality demanded of this project, I ensured that I met the functionality requirements that align with CCA's business requirements. Below is a description of each language researched, which allowed me to create my solution for CCA.

SAP ABAP

SAP Advanced Business Application Programming (ABAP) is a programming language that is used by large organisations worldwide. It essentially runs their critical SAP business systems. SAP is the world's largest inter-enterprise software company and the world's fourth-largest independent software supplier (Rose, 2013). The SAP applications provide capabilities to manage finances, asset accounting, cost accounting, production operations and materials, personnel, plants and archived documents (Rose, 2013). This programming language also includes the use of logical databases, which I will use thoroughly within this project.

The use of SAP was required for the first phase of my project as the majority of the required information that was needed for CHEP Portfolio Plus was contained within CCA's SAP system. Throughout the design and creation of my Python application I was required to educate myself in order to have the required skill level to program and create extraction reports of the required shipment, customer and vendor information. Since the project with CCA begun, I strived to study the basics of SAP and learn how to use the required SAP transactions.

I had then learnt the required basics from my industry mentor Jet and I had also used an online SAP ABAP Programming for Beginners Udemy course (Moxon, 2014). Currently within the New Zealand and Australia CCA sites, SAP is used heavily for the critical business systems such as orders, shipping and inventory management. Therefore the use of SAP is a significantly large factor within my project.

The importance of knowing how SAP operated was crucial to this project as the EDI system I used also connected onto CHEP's SAP system. The information from CHEP's SAP system as previously mentioned was then parsed on automatically via the back-end of CHEP onto their respective Portfolio Plus software of which was then allowed for authorised CHEP and CCA employees to access the data and transactions that had been recorded into the system.

Python3

Python is a relatively new language which was released back in February 1991 by Guido Van Rossum who was the primary designer of the language (The Python Programming Language, 1997).
Many of Python's original/initial features originated from the ABC language. ABC had certain problems that Guido Van Rossum wanted to correct. Moreover, certain features were kept and taken from ABC. One of Guido's primary goals was to create a scripting language that was generally extensible (The Python Programming Language, 1997).
The following are the significant features included with Python:

- Object-oriented programming language
- High level dynamic data types and classes
- Combination of remarkable power with clear syntax
- Several interfaces to system calls and libraries
- Extensible in C and C++

Python has been noted as highly suitable for rapid prototyping of complex applications and this is of significant advantage to me. It is important to note that this enabled me to show progress to my stakeholders at CCA and to my university supervisor. Python is a language that is used to connect multiple

sources of information together and allow for the automation of shipment transactions to be transferred from CCA to CHEP.
Python is the main programming language that I used within my solution as I was required to develop a solution to address my brief within a timely manner and also to address the requirements of future modifiability. Additionally, I chose Python as I believe it outweighs Java's advantages for the purpose of my solution. As of July 2014, the Python programming language is the most popular language for teaching introductory computer science at the top-ranked U.S. departments (Guo, 2014).

Python allowed me to program an application that is light-weight and runs efficiently with the use of the several libraries that are included within Python. Thus, I decided to perform my initial research on Python rather than Java. Python was significantly more efficient and a simpler language to understand and create an application in, in comparison to Java. An equally important factor to note is that the Python application works heavily with .csv formats and at present Python has an excellent .csv module built-in which has allowed me to extract and modify the information at ease (Python.org, 2015). The use of the .csv module has been closely tied in with the SQLite module, which manages the SQLite database in Python (Python.org, 2015).

Another essential module that I used within Python is the email.generator module that generates MIME documents (Python.org, 2015). This module has been used within my solution towards the final stage of the process.

Concluding with my Python research, I believe that the Python programming language has provided me with the right tools and modules to complete the main coding and aspect of the project for CCA New Zealand.

SQLite

It is defined as an in-process library that implements a self-contained, server-less, zero-configuration transactional SQL database engine (SQLite.org, n.d.). An important aspect to be noted with SQLite is that it is listed within the public domain and is free for use including the commercial sector. This was an important aspect for me to consider as once my application is used commercially

I believe that it would be bound automatically against such clauses. Therefore it being allowed to be used within the commercial sector for free was also a key point I kept in mind. SQLite is the most widely developed database in the world (SQLite.org, n.d.). Organisations that currently use SQLite are:

- Adobe
- Airbus
- Apple
- Dropbox
- Mozilla
- Google
- Microsoft
- Python
- Skype

All the above listed organisations including many other high profile organisations use SQLite in some way. SQLite is an embedded SQL database engine that does not use a separate server process which enables the application to be significantly 'lighter'.

Ensuring that my Python application was a 'light' application allowed me to compute somewhat high-resource tasks without too much of load bearing on the CPU. SQLite was a great choice for my project as a complete SQL database with multiple tables, indices, triggers and views can be saved into a single disk file (SQLite.org, n.d.). The database within SQLite is a cross-platform which ensures that if in the future CCA wish to copy the database between different systems, such as migrating from a 32-bit to 64-bit system, then it is therefore a possibility.

The library for SQLite at present is significantly compact, the library with all the features enabled results in a size of less than 500 kilobytes. It is important to note that the size of the library would be dependent on the platform and compiler organization settings (SQLite.org, n.d.). The size of this library alone demonstrates the agility and compactness of such a platform and is therefore a popular database engine choice amongst many individuals and organisations around the world. SQLite runs faster as more memory is allocated to it.

According to sqlite.org, SQLite is carefully tested prior to each individual release and has a reputation for having a high level of reliability (SQLite.org, n.d.). This was an extremely important concern for my project as I must ensure that the database does not crash or have any software related issues. Although my application will initially run within the TEST_DAR environment at CCA, I had to ensure that my Python application did not have any bugs before it is released into the Production Environment. Devastating effects can occur for the Production and Distribution plant for CCA New Zealand if the application was to go offline.

Chosen Programming Languages

As a result of my research into SAP ABAP, Python3 and SQLite (Version 3) I had concluded that I was to use a combination of all three languages to create an optimized and efficient solution for CCA. Each individual component within my solution was be created to the highest of standards. Robert Martin considered software as rotting design when the following five symptoms where present (Martin, 2000):

- Rigidity – In this symptom, software would be extremely difficult to change when required. A single change within a system would result in several undesired changes in other linked modules. When a software has such a symptom, many companies are hesitant to allow their software engineers to work on the program to make changes.

- Fragility – This symptom is closely linked with rigidity. With this rotting design symptom, the software is easily broken in multiple places whilst a change is being performed within another area of the software. Many companies also then become hesitant to allow changes to the software. Robert Martin stated "As the fragility becomes worse, the probability of breakage increases with time" (Martin, 2000).

- Immobility – This relates to the inability of being able to use software from other projects or parts within the same project. Software engineers may attempt to reuse software from other projects but due to the complexity and extra ‘baggage’ the engineers may decide to re-write the module instead of reusing the module.

- Viscosity of the Design - This is defined as while making a change, many ways can preserve the design of software and others do not preserve the design. Essentially, they can be classified as hacks to the software. It is easier to do the wrong thing than do the right thing (Martin, 2000).

- Viscosity of the Environment – The issue with this symptom is where the development environment is slow and inefficient (Martin, 2000). Compile time may take an excessive amount of time and software engineers may be required to change parts of code that do not require the compiling of the entire code.

I have tried to ensure that the five symptoms mentioned above have been avoided at all costs to ensure that my Python application is of the highest quality of standards and runs efficiently. Whilst I programmed my application, I ensured that I had sufficient in-line notation to allow future developers to make changes as and if required by CCA.

To ensure I achieved my goal of developing a cost-effective, responsive, efficient and well-designed application, I followed these key design practices (Microsoft, 2009):

- Keep design patterns consistent within each layer
- Do not duplicate functionality within an application
- Prefer composition to inheritance
- Establish a coding style and naming convention
- Maintain system quality
- Consider the operation of the application

Related Work

WiseTrack Asset Tracking

Currently there are similar technologies that are used within the world in relation to tracking assets. From the research that I conducted, I was unable to find exact solutions that would be fit for the requirements of CCA. Current solutions include general asset tracking that require the use of RFID Tags. This option was not suitable for a company such as CCA as the pallets that require tracking are not the property of CCA, thus physical changes cannot be implemented to the pallets to allow for tracking.

In addition, inserting a physical RFID tag per pallet was clearly not feasible, on average 40,000 pallets are sent out of the Auckland Distribution Centre per month. Each one of these pallets will not be returned to CCA as these pallets are transferred to the customers and shipping companies accordingly. Once again this proves that adding a chip to each one of these pallets is not a feasible option.

Another point to consider with RFID tracking of each pallet is the practicality concept. Each individual customer of CCA would not install and upgrade their technologies to suit the tracking pallet requirements for CCA. This responsibility does not come down to the customers of CCA. An option to somewhat solve this current issue is WiseTrack's solution which is a company that provides RFID tracking of assets (WiseTrack, n.d.).

WiseTrack was built using the Microsoft Structure and uses Microsoft SQL. The WiseTrack software combined with the use of WiseTrack's Antennas and equipment help deliver a solution to their clients and customers.

This option provided by WiseTrack was once again not feasible for CCA due to the costs required and large diverse range of CCA customers within New Zealand. Although the more cost-effective solution of Barcode Labelling is available for customers of WiseTrack, I deemed this unsuitable due to the following reasons:

- Not Practical

- Not Cost-effective
- Not Viable for all CCA Customers
- Prone to sticker being damaged during transit

After analysing WiseTrack's solution in-depth, I had concluded that the solution available by WiseTrack is not a practical and cost-effective solution for CCA. It is important to note that their developed solution is primarily for companies who ship internally within multiple distribution and shipping centres.

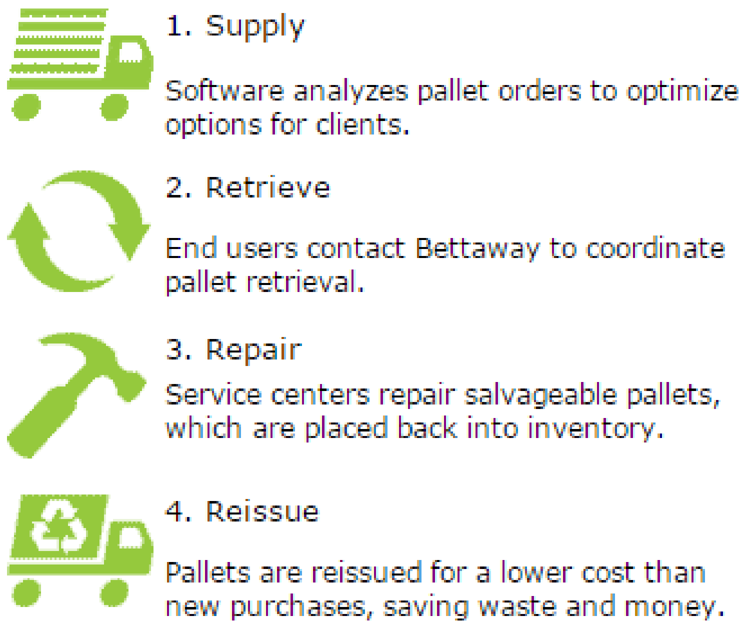
Although CCA ship within multiple warehouses within New Zealand, their major consignments are sent to customers and via the use of shipping containers. As a result of this, CCA is unable to resolve their issue by integrating a pallet tracking system within their warehouses only.

Bettaway Pallet Systems

This pallet inventory management system known as the Bettaway Pallet System was similar to the technology I was implementing for CCA. The current pallet management in accordance with CCA business processes are managed by CHEP – the manufacturer and supplier of the CHEP pallets. All pallet orders and requests are sent to CHEP who allocate the requested number of pallets and deliver them to CCA's delivery location. The delivery location of the ordered pallets are one of five nationwide distribution centres.

Bettaway have developed a solution that is currently deployed within the United States and Canada, the solution provides pallets to customers from one of 200 pallet yards (Bettaway, n.d.). Bettaway's solution is primarily a dispatch and management system that looks to improve current pallet distribution within the United States and Canada. Bettaway state that customers can save up to 35% whilst using their platform and service. This is compared to the traditional methods of purchase and discard methods of pallets (Bettaway, n.d.).

The use of Bettaway leans significantly more towards the supply-chain process where end-users must contact Bettaway to organise pallet retrieval. Whereas with CHEP, pallets are transferred to the customer's organisation who can then use those pallets to ship inventory to other vendors or back to CCA. Bettaway also manages repair and reissue of pallets, which, although it is similar, is not the same as CHEP. CHEP will not dispatch faulty pallets to their clients and if a client is to damage a CHEP pallet, they are liable to pay for a total replacement of that pallet. The following diagram represents the Bettaway current pallet management process.



1. Supply

Software analyzes pallet orders to optimize options for clients.

2. Retrieve

End users contact Bettaway to coordinate pallet retrieval.

3. Repair

Service centers repair salvageable pallets, which are placed back into inventory.

4. Reissue

Pallets are reissued for a lower cost than new purchases, saving waste and money.

Image retrieved from: https://www.bettaway.com/baw/palletsServices

Given the Bettaway process, I was unable to implement this solution for CCA due to the fact this solution is based on general wooden pallets which are not CHEP branded. These pallets do not undergo the same level of testing and compliance tests as compared to CHEP pallets. Additionally this solution is also restricted to clients based in the United States and Canada. I found that this potential solution is unfeasible for the current business issues I face due to the following reasons:

- Limited to United States and Canada
- Not adherent to requirement of CHEP pallets
- No consistent quality and strength of pallets
- Not feasible to implement solution for all New Zealand CCA customers

TrackIt Asset Tracking

TrackIt asset tracking is a New Zealand based company who specialise in asset tracking with the use of RFID and GPS technology combined. Their software is a web-based solution that enables customers to track, manage and monitor their assets. This solution is similar to the previously mentioned WiseTrack solution, which uses RFID tags and barcodes to enable a client to track their assets in real-time (TrackIt, n.d.). The TrackIt solution allows a user to track their assets and inventory in real time on any device or computer worldwide.

A collection of technologies are used such as RFID, barcodes and numerous other methods. The management platform allows their clients to track the use of their fixed assets. As mentioned previously, this form of solution was also not feasible and practical for CCA as the TrackIt solution is primarily used for tracking fixed assets. Each individual pallet cannot have such technology implemented within them, the costs involved to do such a task would heavily outweigh the proposed cost-savings of my solution.

Subsequently, this form of solution is suitable for a company of who transfer their goods between multiple sites and will not dispose of the asset within one shipment such as the transfer of a CHEP pallet. TrackIt provide a great solution for clients who may require features such as the Real-Time tracking screen (TrackIt, n.d.), but it is important to note that CCA did not require these extensive features in regards to tracking a pallet.

TrackIt also provide site auditing, which is a critical feature that as a result of my solution will allow CCA to audit their pallets much more accurately. My implementation of this concept allows for a more accurate and quicker monthly reconciliation process of the total number of pallets dispatched. After further consideration and discussions with my CCA supervisors I was informed that they wanted me to primarily focus on the automation of the shipment transactions and that the auditing can be done on the Portfolio Plus end. In addition to the current issue of pallet count discrepancies, CCA also faced the issue where pallet stocktake is an extremely time-consuming and tedious task to complete. I used the knowledge that I have learned from current similar technologies to develop an optimal solution for CCA New Zealand whereby accurate information is fed through to Portfolio Plus that allows for a better and more efficient manner in which monthly audits can be undertaken.

Solution Design

My solution for CCA required the use of several platforms and technologies to create a cost-effective and efficient solution, which can be heavily used on a day-to-day basis within the CCA production environment. As with any project, changes to the project continue to occur as the project progresses. Initially, I had planned for my solution to be a sub-interface of CCA's SAP system that would collect and aggregate all the required fields of information. I have learnt that, within a large business organisation, several requirements and constraints are always present.

Due to resourcing and governance in-place by Coca-Cola Amatil Australia, I was restricted to making environment changes to the production and development SAP environments. The latest revision of my solution required me to create an intermediary application that would receive the majority of information from the current CCA SAP system. I was initially required to use five main tables and I had designed the table join structure that had best matched CCA and CHEP's requirements. Unfortunately due to a misunderstanding of the manner in which CHEP pallets were transferred, I had to then completely change the processing logic of my Python application. Subsequently this meant I was required to pull additional information through CCA's SAP System. Thus, I then added a new table which is the LFA1: Vendor Master. The 6 main tables I used to extract information from CCA's SAP system are as follows:

- VTTK: Shipment Header
- VTTP: Shipment Item
- VBFA: Sales Document Flow
- VBAK: Sales Document: Header Data
- KNA1: General Data in Customer Master
- LFA1: Vendor Master (General Section)

I chose the above tables as this allowed me to join the necessary tables together in order to link all the required fields of information together. Now that all the required tables from which information had to be extracted from have been chosen, the next step in the design process was to come up with a solution to view and extract that information.

Subsequently, I had to create the table joins in SAP QuickViewer also known as SQVI transaction code. Within this transaction of SAP, I created the required layout of the information that was to be extracted from CCA's SAP system. Furthermore, I designed and generated a SQVI report, which included the majority of fields. The report contents was parsed into my Python application which handles all the incoming fields and appends set fields of information. As I previously mentioned, certain fields was modified by my Python application to suit the level of quality, accuracy and data integrity of information required by CHEP's Portfolio Plus application and CCA's accounting requirements.

This technological layout and design enabled me to create the most efficient system for CCA, within the resourcing and governance that had been implied by CCA Australia. Once the report template was completed with the final layout design. I was meant to be given the opportunity to convert the SQVI extract into SE93 - Maintain Transaction Code, which would allow me to view the transaction code and modify it further to allow for automatic batch extract movements every 5-7 minutes. But unfortunately, I had put in a request numerous times to access the SE93 transaction and was subsequently denied due to the administrative rights in place. I will mention which changes need to be implemented within the future work section of this report.

The automatic scheduled run time of the batch job will be discussed in-depth with CCA and CHEP to ensure that all information is present on CHEP's Portfolio Plus at any given time or day.

The scheduled batch job from SE93 with the set fields of information was required to run as a scheduled task. In the latest scenario the output file can still be extracted as a .csv file and parsed through to my Python Application as one of the primary sources of input. I aimed to parse the output file from SE93 by uploading it to a set location that my Python application can constantly read from. This location will be located within the internal network of CCA. This file will be imported by my Python application and will be used according to the required input specifications by CHEP. Furthermore, my application makes changes to the shipment information that is received from CCA's SAP system and in-turn makes the subsequent changes as required by CHEP.

The Python application also displays the total number of dispatched pallets at during each runtime of the application. This information can be is printed onto the command line screen of the application. This application will allow CCA for a much greater, simpler and more efficient pallet transfer process. This in-turn should allow for CCA to perform more accurate monthly reconciliations of all pallets and shipments that have been dispatched from the New Zealand Distribution Centres and Warehouses. Furthermore, CCA Distribution Centre employees will not be required to manually enter in each shipment record into the CHEP Portfolio Plus system as this will now be automated by my Python application.

The Python application was initially required to match customer addresses from SAP to a CHEP-recognised customer ID. This however was then changed to a more complicated approach due to the way pallets were allocated to customers and vendors, thus my application was required to match two different sets of customers and vendors according to their Customer ID and Vendor ID respectively including other fields of relevant information. I will talk about this in detail shortly.

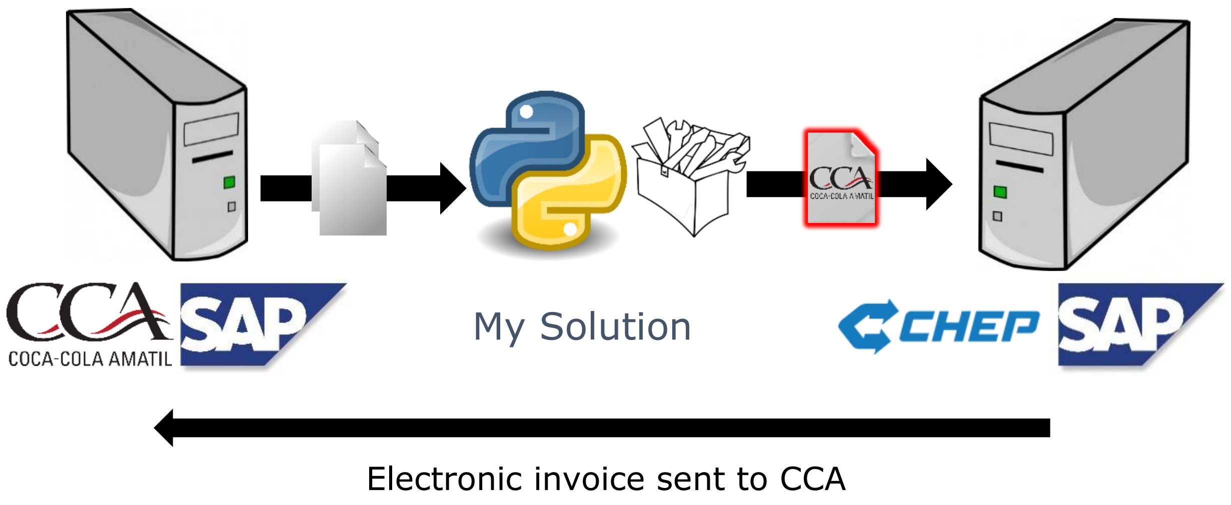
I had initially decided that this information will be accessed via the use of a data dictionary. However I decided to go down another more feasible path of which I will also describe shortly. The Python application has been designed to match any other values required to ensure the aspect of data integrity and credibility is met at all times. To maintain table data within Python I used SQLite to handle all the tabular data. SQLite enabled me to also create tables whereby the information from them was then extracted with the correct header and field records.

Once my Python application completed performing all operations on the received data from CCA's SAP system, the Python application performed the required modifications of which I will also go into detail shortly. The EDI – Electronic Data Interchange process is used finally to parse the information onto CHEP's SAP system which in-turn is parsed through to CHEP's Portfolio Plus software. An EDI is defined as computer-to-computer exchange of business documents in a standard electronic format between business partners (EDI Basics, n.d.). The business partners in this instance are Coca-Cola Amatil New Zealand and CHEP New Zealand.

I used the EDI process to provide CCA with the following benefits:

- Increased volume of CCA Transactions
- Time savings
- Monetary savings
- Greater flexibility for employees
- Increased employee productivity
- Enables CCA information transfer directly to CHEP systems
- Data accuracy
- Data integrity
- Eliminates several paperwork processes
- Business streamlining
- Better reporting information

Electronic Data Interchange Solution for CCA:



Furthermore, my Python application has been designed in a manner where the output which needs to be sent to CHEP meets their approved document format. This format includes the set fields of information as requested by CHEP. These fields are required as CCA are responsible for reporting the movements of the CHEP pallets. The pallets are transferred to Customers and Vendors of CCA. With this reported information CHEP determines whom or which company the pallet liability must be transferred to. For accurate pallet movement, each field must be accurately reported to CHEP else CCA may be responsible for those dispatched pallets. Seven movement record fields are required to successfully log each movement. The fields are listed below with their respective representation:

- Docket Number – Unique docket number
- Sender ID – CCA's sender ID
- Receiver ID – CCA's customers CHEP ID
- Dates – Date of Dispatch (DOD), Date or Receipt (DOR) and Effective Date (EFD)
- References – Sales Order, Purchase Order and Consignment Note
- Material – CHEP Equipment ID
- Quantity – Number of pallets

Once my Python application had composed the table with all the relevant fields according to the criteria set by CCA and CHEP, the next stage in the process was to compose the document in the approved format and layout for CHEP's EDI. Many conditions had to be met in order to successfully log a pallet movement. These conditions are that the correct header records and movement records must be recorded within each exported file. The following header records are also required to successfully log a pallet movement:

- Header Indicator – Indicated by 'H'
- Recordnum – Running total of the record/row
- RecordCount – Total records in file including header record
- SendDate – Date record sent
- ProgramName – Indicated by CDKTF
- ProgramVersion – CHEP Standard EDI Docket format version number

- InformerGLID – CHEP Assigned Customer GLID, this indicates CCA location
- CountryCode – Indicated by 64, thus New Zealand
- CustomerFileRef – First 4 characters of the CCA allocated docket prefix with an additional sequential counter. This is unique for each file that is sent to CHEP.

The next step in the solution process was to parse the fully composed document with the correct header records and correct field records into the CHEP EDI which essentially parses the information through CHEP's EDI and into Portfolio Plus. I have discussed this in-depth with CCA and CHEP and the most cost-effective and easiest solution will be to securely e-mail the file on an automatically scheduled basis.

Bearing in mind confidentiality and sensitivity of the data that will be sent on a daily basis, I had to consider the key security requirements whilst transmitting the data over to CHEP. I have requested for a secure email account from CCA which will be used primarily for data transmission to CHEP.

Ideally, the account requires Transport Layer Security (TLS) to encrypt all outgoing mail. TLS is the most widely used and developed protocol for sending e-mails and web traffic content (Rescorla, 2012). In the instance of my application the focus was on ensuring the email was sent securely. This was a high priority when considering security and confidentiality of information. The security of incoming mail is negligible as this e-mail account will not be used to receive any e-mails. Any e-mails received into this account will be automatically deleted, this condition will be set by default by CCA mail administrators.

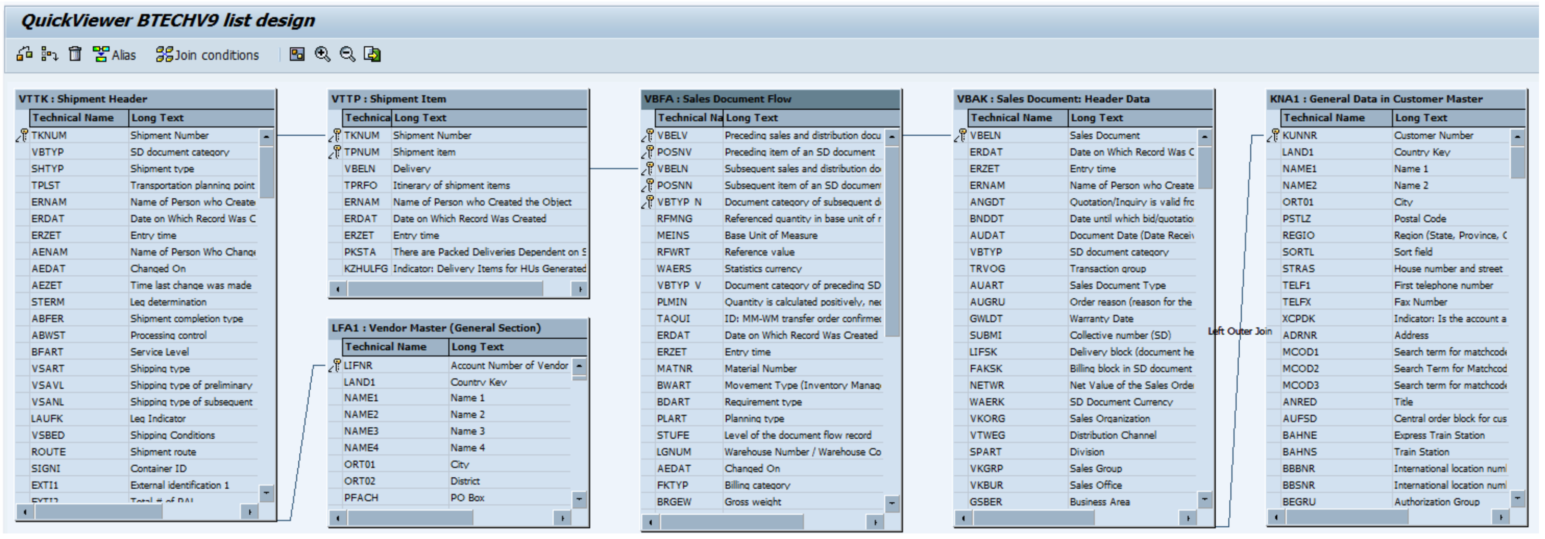
Project Development

A significant amount of time had been spent completing the initial and preliminary stages of this project due to the fact that a large amount of investigation work into the current business process had been required. To begin with I was required to understand the current shipping process at CCA and attempt to identify an area that needed to be improved. I found that this process was extremely complicated. CCA is an extremely large business organisation and understanding their entire business operations process was a difficult task.

My project required me to meet with several stakeholders within the CCA business to understand their requirements and the limitations within the business and their relative business processes. I was in contact with CHEP and their Senior Manager, who managed their Business Solutions and Assets. My progress consisted of me conducting interviews with Distribution Centre employees and Outwards Goods employees. As part of my continual learning and development process within CCA, I had created the initial stage of my project within two main SAP environments:

- SAP ECC Production - PAR
- SAP ECC Development - DAR

Within these two environments I created and developed an extract report, with the use of the SAP Tables described within the Solution Design chapter of this report. The following screenshots represent the latest and final revision of my progress in the Client-specific TEST_DAR environment within SAP. All the following screenshots as displayed below have been updated due to the change of my solution design and requirements of my Python Application.



QuickViewer BTECHV9 list design

VTTK : Shipment Header

Technical Name	Long Text
TKNUM	Shipment Number
VBTYP	SD document category
SHTYP	Shipment type
TPLST	Transportation planning point
ERNAM	Name of Person who Create
ERDAT	Date on Which Record Was C
ERZET	Entry time
AENAM	Name of Person Who Change
AEDAT	Changed On
AEZET	Time last change was made
STERM	Leg determination
ABFER	Shipment completion type
ABWST	Processing control
BFART	Service Level
VSART	Shipping type
VSAVL	Shipping type of preliminary
VSANL	Shipping type of subsequent
LAUFK	Leg Indicator
VSBED	Shipping Conditions
ROUTE	Shipment route
SIGNI	Container ID
EXTI1	External identification 1

VTTP : Shipment Item

Technica	Long Text
TKNUM	Shipment Number
TPNUM	Shipment item
VBELN	Delivery
TPRFO	Itinerary of shipment items
ERNAM	Name of Person who Created the Object
ERDAT	Date on Which Record Was Created
ERZET	Entry time
PKSTA	There are Packed Deliveries Dependent on S
KZHULFG	Indicator: Delivery Items for HUs Generated

LFA1 : Vendor Master (General Section)

Technical Name	Long Text
LIFNR	Account Number of Vendor
LAND1	Country Key
NAME1	Name 1
NAME2	Name 2
NAME3	Name 3
NAME4	Name 4
ORT01	City
ORT02	District
PFACH	PO Box

VBFA : Sales Document Flow

Technical Na	Long Text
VBELV	Preceding sales and distribution docu
POSNV	Preceding item of an SD document
VBELN	Subsequent sales and distribution do
POSNN	Subsequent item of an SD document
VBTYP_N	Document category of subsequent d
RFMNG	Referenced quantity in base unit of r
MEINS	Base Unit of Measure
RFWRT	Reference value
WAERS	Statistics currency
VBTYP_V	Document category of preceding SD
PLMIN	Quantity is calculated positively, ne
TAQUI	ID: MM-WM transfer order confirmed
ERDAT	Date on Which Record Was Created
ERZET	Entry time
MATNR	Material Number
BWART	Movement Type (Inventory Manag
BDART	Requirement type
PLART	Planning type
STUFE	Level of the document flow record
LGNUM	Warehouse Number / Warehouse Co
AEDAT	Changed On
FKTYP	Billing category
BRGEW	Gross weight

VBAK : Sales Document: Header Data

Technical Name	Long Text
VBELN	Sales Document
ERDAT	Date on Which Record Was C
ERZET	Entry time
ERNAM	Name of Person who Create
ANGDT	Quotation/Inquiry is valid fro
BNDDT	Date until which bid/quotatio
AUDAT	Document Date (Date Receiv
VBTYP	SD document category
TRVOG	Transaction group
AUART	Sales Document Type
AUGRU	Order reason (reason for the
GWLDT	Warranty Date
SUBMI	Collective number (SD)
LIFSK	Delivery block (document he
FAKSK	Billing block in SD document
NETWR	Net Value of the Sales Order
WAERK	SD Document Currency
VKORG	Sales Organization
VTWEG	Distribution Channel
SPART	Division
VKGRP	Sales Group
VKBUR	Sales Office
GSBER	Business Area

KNA1 : General Data in Customer Master

Technical Name	Long Text
KUNNR	Customer Number
LAND1	Country Key
NAME1	Name 1
NAME2	Name 2
ORT01	City
PSTLZ	Postal Code
REGIO	Region (State, Province, C
SORTL	Sort field
STRAS	House number and street
TELF1	First telephone number
TELFX	Fax Number
XCPDK	Indicator: Is the account a
ADRNR	Address
MCOD1	Search term for matchcode
MCOD2	Search Term for Matchcod
MCOD3	Search term for matchcode
ANRED	Title
AUFSD	Central order block for cus
BAHNE	Express Train Station
BAHNS	Train Station
BBBNR	International location num
BBSNR	International location num
BEGRU	Authorization Group

SQVI Report Extraction Table Joins

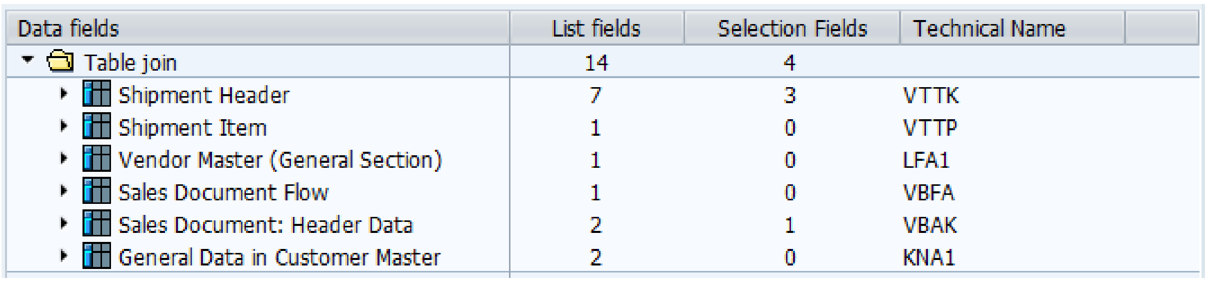
Whilst creating the extract report interface, I was initially required to link 5 tables to create a relationship, which was then changed to 6 tables and resulted in the required information becoming available for extraction. The above screenshot demonstrates this respectively.

These fields were linked to create the required table relationship they are listed as follows:

- TKNUM: Shipment Number = VTTK – VTTP
- VBELN: Delivery / Subsequent sales and distribution document = VTTP – VBFA
- VBELN: Sales Document = VBFA – VBAK
- KUNNR: Sold-to-party / Customer Number = VBAK – KNA1
- TDLNR: Number of Forwarding Agent / Account Number of Vendor = VTTK – LFA1

The next step in the process required me to research and learn about the required fields I would need to use within the SAP extraction report and my Python application. This was done in order to gather all the correct information as required by CHEP's EDI and Portfolio Plus. This process took me a significant amount of time as there are hundreds of fields within the 6 chosen tables of which all were not used or required for this stage of development. The fields that I was initially required to use to extract information are as follows:

- Driver 1
- Vehicle
- Actual date of check-in
- Total # of PAL
- Shipment Number
- Subsequent item of an SD document
- Customer Purchase order number
- Name 1
- House number and street



Data fields	List fields	Selection Fields	Technical Name
Table join	14	4	
Shipment Header	7	3	VTTK
Shipment Item	1	0	VTTP
Vendor Master (General Section)	1	0	LFA1
Sales Document Flow	1	0	VBFA
Sales Document: Header Data	2	1	VBAK
General Data in Customer Master	2	0	KNA1

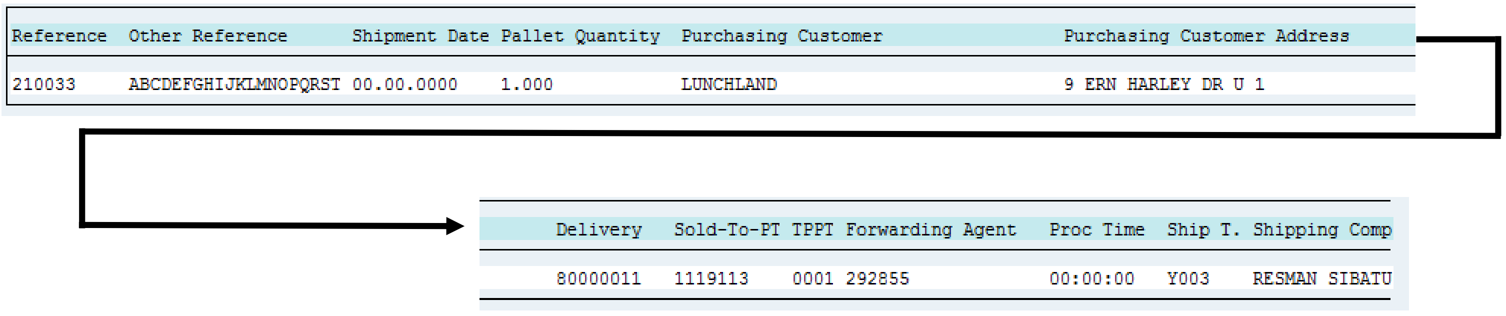
The fields listed above have been retrieved from the 6 listed tables within SAP

I have been required to adjust the tables and fields required as per my above screenshot. This is due to the differences that lay with my thinking and the actual information that was required. I had initially programmed my Python application to map each shipment that had been received from CCA's SAP system to the customer of which the goods were being delivered to. I.E. University of Auckland, I would then retrieve the CHEP ID for the University and assign the pallets to the customer.

Unfortunately, as I had later realised after talking to my industry mentor I had unfortunately programmed the application with my understanding of the business process which was slightly incorrect. As I was using dummy data I was unaware that each CCA customer was not assigned a CHEP Global ID and each customer was not liable for the CHEP pallets.

However the responsibility and ownership of the pallets being delivered to small-medium size customers came down to the logistics company that would deliver the pallets. I.E. University of Auckland may not have a Global CHEP ID but goods from CCA are delivered to the University via the logistics company Tolls, the ownership of pallets are theoretically meant to be transferred to Tolls and not the University of Auckland. I came across this finding when I had one of my regular meetings with my industry mentor Jet and Outwards Good Distribution manager Storm, who manages the Outwards goods process for CCA.

As part of my initial development process I had created the layout and set the required fields into an information extract report template. The fields as listed in the following screenshot do not have the same names as listed by SAP due to the requirement input from CHEP's EDI. The fields have been modified and by me to accommodate CHEP's requirements.



Reference	Other Reference	Shipment Date	Pallet Quantity	Purchasing Customer	Purchasing Customer Address
210033	ABCDEFGHIJKLMNOPQRST	00.00.0000	1.000	LUNCHLAND	9 ERN HARLEY DR U 1

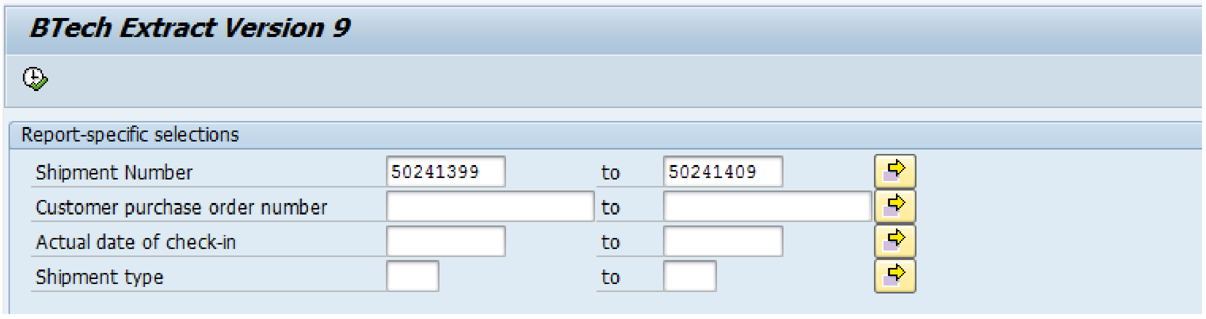
Delivery	Sold-To-PT	TPPT	Forwarding Agent	Proc Time	Ship T.	Shipping Comp
80000011	1119113	0001	292855	00:00:00	Y003	RESMAN SIBATU

SAP Extraction Report Headers

The above screenshot demonstrates the latest and final revision of my SAP extraction headers, the SAP extraction report has been updated significantly when compared to the initial development stages of this project.

To limit the duplication of shipments and to ensure that I met the requirement of data integrity and validity, I have created and set a default filter on the subsequent item field. With the filter on as default, data is not duplicated and is not listed multiple times within the results. This modification was required as entries for multiple types of items were listed with the same quantity of pallets.

The total number of pallets field represents the pallet quantity for the entire order. The following screenshot displays the interface created that can be run in an automated extraction of results every 5 minutes. I will discuss briefly other options as an alternative to regular batch jobs of 5 minutes.



BTech Extract Version 9

Report-specific selections

Shipment Number	50241399	to	50241409
Customer purchase order number		to	
Actual date of check-in		to	
Shipment type		to	

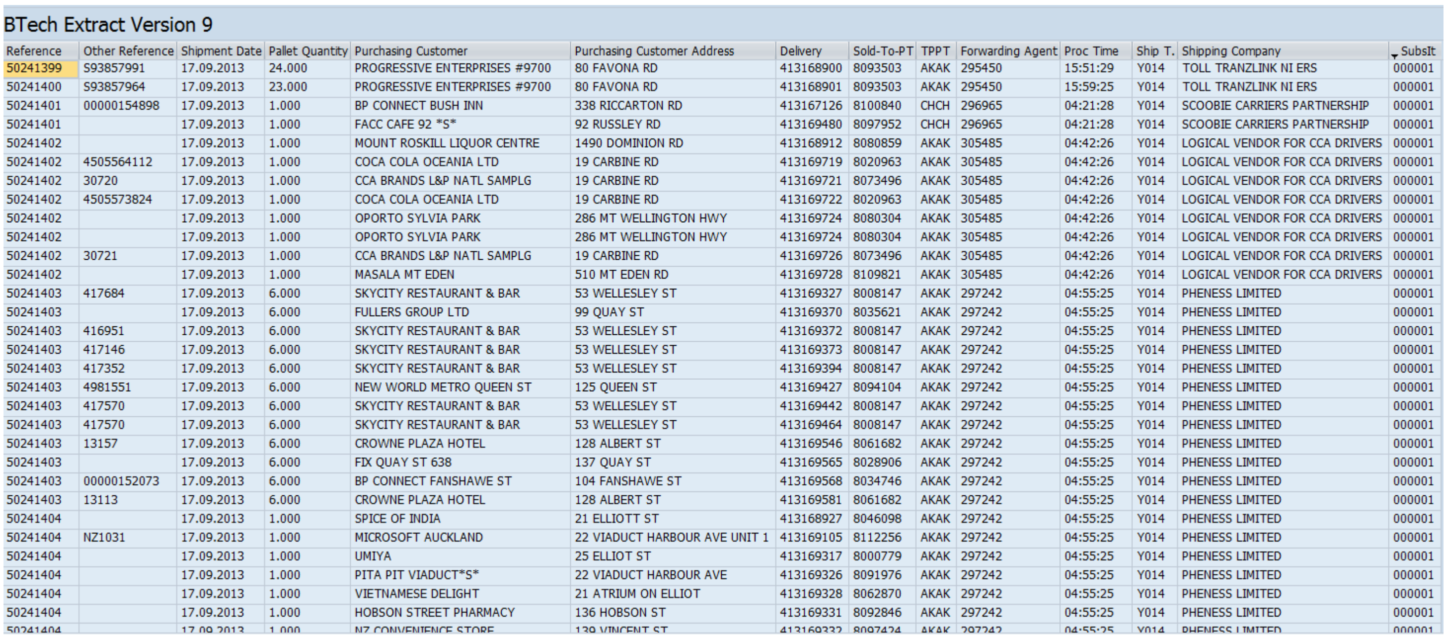
Shipment Numbers 50241399 – 50241409 to be selected

As an example, I created an extraction report for 10 individual shipments. Shipment numbers 50241399 – 50241409 were entered into a select query.

Whilst I designed and created my Python application which serves as the solution to CCA's current business problem, I had been using data that was extremely dense. The data that was being used as the import file for my Python application consisted of over 27,000 rows of information also known as shipments. The amount of information within that data file represents a typical day of shipments and orders for Coca-Cola Amatil NZ.

It is important to note that the extraction query can be configured to run automatically on a daily basis or hourly basis. Essentially, it is up to CCA and CHEP's discretion as to how often they would like to process the shipment information. Additionally, the longer CCA wait to process the transfer of shipments it will subsequently result in CCA being liable for those pallets for a longer period of time.

Furthermore, as a result of the above Shipment Number conditions that were entered into the shipment number range, the following screenshot demonstrates the results that were produced by my SAP extract report interface.


BTech Extract Version 9

Reference	Other Reference	Shipment Date	Pallet Quantity	Purchasing Customer	Purchasing Customer Address	Delivery	Sold-To-PT	TPPT	Forwarding Agent	Proc Time	Ship T.	Shipping Company	SubsIt
50241399	S93857991	17.09.2013	24.000	PROGRESSIVE ENTERPRISES #9700	80 FAVONA RD	413168900	8093503	AKAK	295450	15:51:29	Y014	TOLL TRANZLINK NI ERS	000001
50241400	S93857964	17.09.2013	23.000	PROGRESSIVE ENTERPRISES #9700	80 FAVONA RD	413168901	8093503	AKAK	295450	15:59:25	Y014	TOLL TRANZLINK NI ERS	000001
50241401	00000154898	17.09.2013	1.000	BP CONNECT BUSH INN	338 RICCARTON RD	413167126	8100840	CHCH	296965	04:21:28	Y014	SCOOBIE CARRIERS PARTNERSHIP	000001
50241401		17.09.2013	1.000	FACC CAFE 92 *S*	92 RUSSLEY RD	413169480	8097952	CHCH	296965	04:21:28	Y014	SCOOBIE CARRIERS PARTNERSHIP	000001
50241402		17.09.2013	1.000	MOUNT ROSKILL LIQUOR CENTRE	1490 DOMINION RD	413168912	8080859	AKAK	305485	04:42:26	Y014	LOGICAL VENDOR FOR CCA DRIVERS	000001
50241402	4505564112	17.09.2013	1.000	COCA COLA OCEANIA LTD	19 CARBINE RD	413169719	8020963	AKAK	305485	04:42:26	Y014	LOGICAL VENDOR FOR CCA DRIVERS	000001
50241402	30720	17.09.2013	1.000	CCA BRANDS L&P NATL SAMPLG	19 CARBINE RD	413169721	8073496	AKAK	305485	04:42:26	Y014	LOGICAL VENDOR FOR CCA DRIVERS	000001
50241402	4505573824	17.09.2013	1.000	COCA COLA OCEANIA LTD	19 CARBINE RD	413169722	8020963	AKAK	305485	04:42:26	Y014	LOGICAL VENDOR FOR CCA DRIVERS	000001
50241402		17.09.2013	1.000	OPORTO SYLVIA PARK	286 MT WELLINGTON HWY	413169724	8080304	AKAK	305485	04:42:26	Y014	LOGICAL VENDOR FOR CCA DRIVERS	000001
50241402		17.09.2013	1.000	OPORTO SYLVIA PARK	286 MT WELLINGTON HWY	413169724	8080304	AKAK	305485	04:42:26	Y014	LOGICAL VENDOR FOR CCA DRIVERS	000001
50241402	30721	17.09.2013	1.000	CCA BRANDS L&P NATL SAMPLG	19 CARBINE RD	413169726	8073496	AKAK	305485	04:42:26	Y014	LOGICAL VENDOR FOR CCA DRIVERS	000001
50241402		17.09.2013	1.000	MASALA MT EDEN	510 MT EDEN RD	413169728	8109821	AKAK	305485	04:42:26	Y014	LOGICAL VENDOR FOR CCA DRIVERS	000001
50241403	417684	17.09.2013	6.000	SKYCITY RESTAURANT & BAR	53 WELLESLEY ST	413169327	8008147	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241403		17.09.2013	6.000	FULLERS GROUP LTD	99 QUAY ST	413169370	8035621	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241403	416951	17.09.2013	6.000	SKYCITY RESTAURANT & BAR	53 WELLESLEY ST	413169372	8008147	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241403	417146	17.09.2013	6.000	SKYCITY RESTAURANT & BAR	53 WELLESLEY ST	413169373	8008147	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241403	417352	17.09.2013	6.000	SKYCITY RESTAURANT & BAR	53 WELLESLEY ST	413169394	8008147	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241403	4981551	17.09.2013	6.000	NEW WORLD METRO QUEEN ST	125 QUEEN ST	413169427	8094104	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241403	417570	17.09.2013	6.000	SKYCITY RESTAURANT & BAR	53 WELLESLEY ST	413169442	8008147	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241403	417570	17.09.2013	6.000	SKYCITY RESTAURANT & BAR	53 WELLESLEY ST	413169464	8008147	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241403	13157	17.09.2013	6.000	CROWNE PLAZA HOTEL	128 ALBERT ST	413169546	8061682	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241403		17.09.2013	6.000	FIX QUAY ST 638	137 QUAY ST	413169565	8028906	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241403	00000152073	17.09.2013	6.000	BP CONNECT FANSHAWE ST	104 FANSHAWE ST	413169568	8034746	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241403	13113	17.09.2013	6.000	CROWNE PLAZA HOTEL	128 ALBERT ST	413169581	8061682	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241404		17.09.2013	1.000	SPICE OF INDIA	21 ELLIOTT ST	413168927	8046098	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241404	NZ1031	17.09.2013	1.000	MICROSOFT AUCKLAND	22 VIADUCT HARBOUR AVE UNIT 1	413169105	8112256	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241404		17.09.2013	1.000	UMIYA	25 ELLIOT ST	413169317	8000779	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241404		17.09.2013	1.000	PITA PIT VIADUCT*S*	22 VIADUCT HARBOUR AVE	413169326	8091976	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241404		17.09.2013	1.000	VIETNAMESE DELIGHT	21 ATRIUM ON ELLIOT	413169328	8062870	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241404		17.09.2013	1.000	HOBSON STREET PHARMACY	136 HOBSON ST	413169331	8092846	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001
50241404		17.09.2013	1.000	NZ CONVENIENCE STORE	139 VINCENT ST	413169332	8097424	AKAK	297242	04:55:25	Y014	PHENESS LIMITED	000001

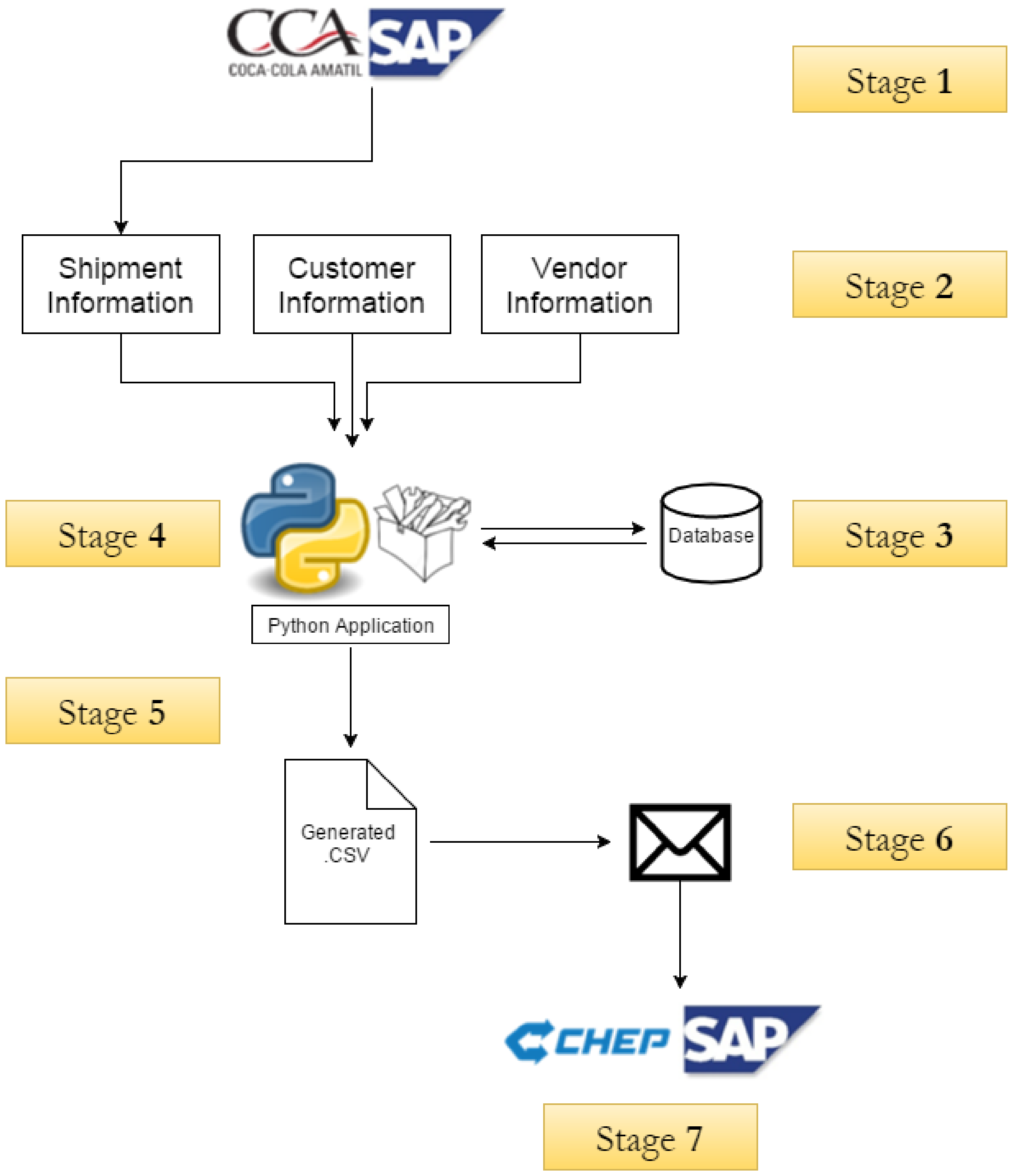
Shipment Numbers 50241399 – 50241409 extraction results

The above screenshot of information can, for instance, be automatically created and extracted by a batch job approximately every 5 minutes. This time can be set when the transaction SE93 is used to modify my current SQVI Report. With changes made to the processing of the report, the report can be extracted automatically which can serve as an update for the newest shipments that have been dispatched.

Additionally, this extraction can also be done without a timer, whereby the batch job is run only when new shipments are detected and that information is then parsed to my Python application. Although, with parameters as aforementioned, several changes will be required to be made to my extraction report query which is run on CCA's SAP system.

The information from my SAP report is exported and is then read by my Python application. As previously mentioned my Python application has been programmed to work with the customers and vendors details that have been provided to me by CCA. These details are then used in conjunction with the extracted SAP CSV file.

Python Application Flow



Python Application Flow Diagram

The above diagram demonstrates the operation stages of my Python Application at an abstract level. The initial design of this Python Application was different in the sense that there were only two input sources whereas there are now three input sources. The input sources - Shipment Information, Customer Information and Vendor Information now replaces my previous design which did not account for the Vendors. My Python Application operates in a manner which can be described as a process that has seven main stages.

Stage 1

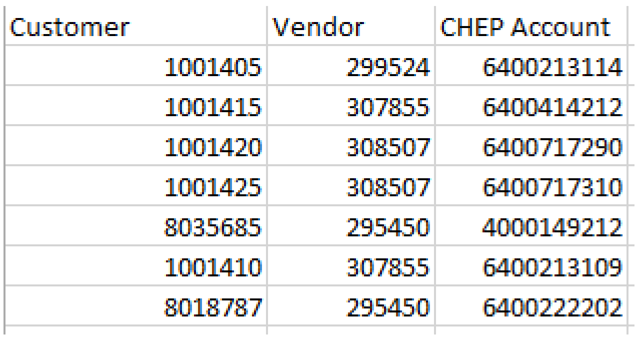
In this stage of the process the majority of required information is exported from CCA's SAP system. The information exported from the SAP system is defined by my extraction query I had completed in the earlier stages of this project. An overview of the SAP development section of this project can be found within the Solution Design section of this report. The shipment information that had been extracted from the CCA SAP system accounts for one third of the information required in stage 2. The extracted query from SAP is imported into my Python Application as a .CSV file and the fields of information extracted via SAP from stage 1 are as follows with their uses for this project:

- Reference – CCA's reference that acts as a Primary Key.
- Other Reference – Additional information that is often provided by customers such as Progressive Enterprises. This information is generally a Purchase Order number.
- Shipment Date – Date shipment was dispatched from CCA Distribution Centre.
- Pallet Quantity – Number of pallets that were dispatched. The liability of these pallets are then transferred to either a Customer or Vendor (Shipping Company).
- Purchasing Customer – Customer that will pay for the goods supplied by CCA. I.e. Progressive Enterprises.
- Purchasing Customer Address – Customer's delivery address.
- Delivery – Unique delivery number is allocated to each shipment.
- Sold-To-Party – CCA's unique identifier for each Customer.
- Transportation Planning Point – Route/Area that the goods will be delivered to.
- Forwarding Agent – CCA's unique identifier for each Vendor (Shipping Company).
- Processing Time – Processing time for each shipment.
- Shipment Type – Type of shipment, this was a very important field as there were multiple shipment types and only there are only a few that are applicable to my project.

- Shipping Company – Company that delivers the goods.
- Subsequent Item – Required field to limit duplicate results.

Stage 2

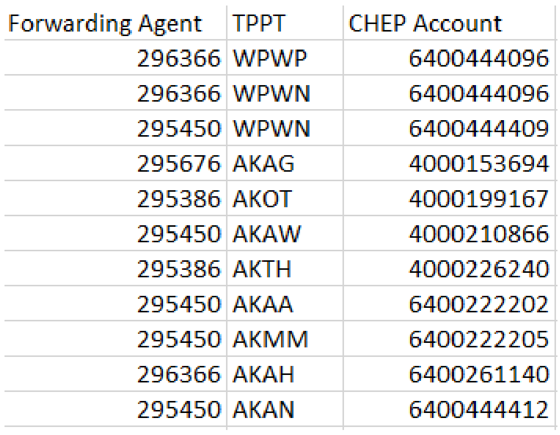
As described in stage 1 the information that had been extracted from SAP via the use of my extraction query did not account for all the information required. The next set of information that was required was the Customer Information. This set of information contains data such as CCA's unique identifier for each Customer, CCA's unique identifier for each Vendor and the Customers Global CHEP ID as displayed in the Customer Information table below. The Customer Information table is imported into my Python application as a .CSV file which allows CCA to modify the contents of the file at any time given the authorisation. For example, if new customers are added or deleted, the .CSV file must be updated to reflect those changes.



Customer	Vendor	CHEP Account
1001405	299524	6400213114
1001415	307855	6400414212
1001420	308507	6400717290
1001425	308507	6400717310
8035685	295450	4000149212
1001410	307855	6400213109
8018787	295450	6400222202

Customer Information file

Another set of information was also required which was the Forwarding Agent/Vendor/Shipping company information along with the Transportation Planning Point and Global CHEP ID. As with the Customer Information table the Vendor Information table is also imported into my Python application as a .CSV file. The information from these two tables allow me to link shipments from stage 1's import and modify the information in a manner which allows me to vadiate the data correctly for CCA and CHEP. The above screenshot represents the typical layout of a Customer Information file.



Forwarding Agent	TPPT	CHEP Account
296366	WPWP	6400444096
296366	WPWN	6400444096
295450	WPWN	6400444409
295676	AKAG	4000153694
295386	AKOT	4000199167
295450	AKAW	4000210866
295386	AKTH	4000226240
295450	AKAA	6400222202
295450	AKMM	6400222205
296366	AKAH	6400261140
295450	AKAN	6400444412

Vendor Information file

The above screenshot represents the typical layout of the Vendor Information file. To conclude stage 2, all the information as aforementioned is imported into the Python Application. The next step within the process will be described in the next stage.

Stage 3

The flow of information in this stage is simplified and comes down to the use of multiple CSV file reader instances and the SQLite database. Within this stage of the process my Python application reads all the information as aforementioned in the previous stages and then inserts the information into the respective tables within the SQLite database. Multiple instances of the CSV reader were used to do so which then allowed my application to import the Shipment Information, Customer Information and Vendor Information into the CCA.db database file and into their respective tables.

Stage 4

This stage proved to be the most time-consuming and crucial stage in regards to the design and development aspect of the project. Details in regards to the development of this stage will be discussed within the next section of this report. In this section I will give an overview of the information flow at this stage of the flow diagram. In this stage my decision-making logic process occurs, this process can be thought of a set of ‘questions’. These ‘questions’ allow my Python application to essentially decide how the shipment records should be manipulated. The data is then manipulated at this stage in accordance with the results from the decision-making logic process. In order to meet CCA and CHEP’s requirements to effectively make-use of the EDI solution, I had to ensure my logic process adhered strictly to their requirements and guidelines.

Furthermore, shipment information that was previously imported and inserted into the SQLite database and tables is now modified and deleted according to set conditions, this will also be discussed in-depth in the next section of this report. Shipments within this stage are linked to Customers, Vendors, Transport Routes and Shipment Types which then allows the program to determine with who must accept liability / take ownership of the pallets.

Stage 5

Within this stage of the process, the export of manipulated data from my Python application occurs. The data is read from my CCA.db database and is then written to a file "CHEP_EXPORT.csv" which adheres to CHEP's requirements.

Stage 6

Once the file has been created from Stage 5, my Python application then emails the file using multiples modules, I will discuss this further within the next section of this report. The file "CHEP_EXPORT.csv" is emailed to a pre-configured email address provided by CHEP that monitors all incoming files from my Python application.

Stage 7

Within this stage of the process it is CHEP's liability to ensure the file received from stage 6 has been processed correctly. The exported file from my Python application is imported into CHEP's SAP system which then allows the data to be parsed through to the CHEP's Portfolio Plus software. This Portfolio Plus software allows CCA and CHEP representatives to view and make changes to the shipment data if necessary.

Python Application Development

Now I move onto the major programming aspect of my project. Extending from the previous section, this is whereby the main development and main logic operations occur. Firstly I began programming my Python application with the use of the following Python modules:

- SQLITE3 – SQLite 3 Library that implements the SQL database engine (SQLite.org, n.d.)

- CSV – Comma Separated Values Module (Python.org, 2015)
- XLRD – Library to extract data from Microsoft Excel spreadsheet files (Python.org, 2015)
- OS – Miscellaneous operating system interfaces (Python.org, 2015)
- SYS – System-specific parameters and functions (Python.org, 2015)

These aforementioned modules allowed me to begin programming in respect to creating the database and tables required for the operation of my Python Application. It is important to note that a large variety of modules were used, this can be seen within the source code files. To adhere to the requirements of my design and the requirements set in-place for the project I created the following tables:

- CCASHIPMENTS
- CCACUSTOMERS
- CCAOUTLETS
- CHEPINFO
- CHEPEXPORT
- TEMP

The following table shows the fields that are allocated to each SQL Table.

SQL Table	Allocated Fields
CCASHIPMENTS	• REFERENCE • OTHER_REFERENCE • SHIPMENT_DATE • PALLET_COUNT • CUST • CUST_ADDRESS • DELIVERY_NUM • SOLD_TO • TPPT • FWDING_AGENT • PROC_TIME • SHIP_TYPE • SHIP_COMP

	• SUBNO
CCACUSTOMERS	• SAP_FWDING • TPPT • CUSTCHEP_ID
CCAOUTLETS	• OUTLET • SAP_FWDING • OUTLETCHEP_ID
CHEPINFO	• LINENO • REFERENCE • OTHER_REFERENCE • SHIPMENT_DATE • PALLET_COUNT • CUST • CUSTOMER_SAP • CUSTCHEP_ID • PROC_TIME
CHEPEXPORT	• LINE_TYPE • RECORD_NUM • DOCKET_NUMBER • SENDER • RECEIVER • DOD • DOR • EFD • REF1 • REF2 • MATERIAL • QUANTITY
TEMP	• LINE_TYPE • DOCKET_NUMBER • DATE • SENDER • MATERIAL

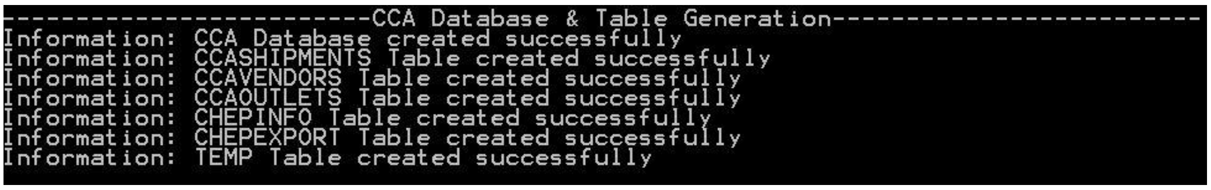
These fields as shown in the table above were required for me to design my Python application whereby the logic processing occurred effectively. I utilised the input file from my SAP generated report, CCA's Customer and Vendor files and inserted the data into the SQL Tables and the fields associated with each field. Whilst designing the structure of my Python application I decided to keep the main functions separate. Thus, I developed my Python Application in the following manner:

- Main.py
- generateDatabase.py
- pythonEmailer.py

Therefore the generation of the database in the instance of my application is imported by main.py and as a result 'CCA.db' and the SQL tables are all generated within the generateDatabase.py script of my application. A snippet of code demonstrating the CCASHIPMENTS table creation is shown below.

```
conn.execute (''''' CREATE TABLE CCASHIPMENTS
    (REFERENCE LONG NOT NULL,
    OTHER_REFERENCE TEXT,
    SHIPMENT_DATE TEXT NOT NULL,
    PALLET_COUNT INT NOT NULL,
    CUST CHAR(50) NOT NULL,
    CUST_ADDRESS CHAR(50) NOT NULL,
    DELIVERY_NUM NOT NULL,
    SOLD_TO NUM NOT NULL,
      TPPT NOT NULL,
      FWDING_AGENT NOT NULL,
      PROC_TIME NOT NULL,
      SHIP_TYPE NOT NULL,
      SHIP_COMP NOT NULL,
      SUBSNO INT NOT NULL);''')
```

Moreover, the generation of the SQL Tables and the SQL database was only one part of the application. Once I had the tables and database generation running correctly, I had the following information displayed as per my design requirements.


```
-------------------------CCA Database & Table Generation------------------------
Information: CCA Database created successfully
Information: CCASHIPMENTS Table created successfully
Information: CCAVENDORS Table created successfully
Information: CCAOUTLETS Table created successfully
Information: CHEPINFO Table created successfully
Information: CHEPEXPORT Table created successfully
Information: TEMP Table created successfully
```

The above screen will be presented every time a user runs the application, this is due to the fact that I was unable to get permission to run the SAP SE93 transaction and automate the report extraction from SAP every 5 minutes. I will discuss this further in the Future Work section of this report. It is important to note that with one simple line of code the application can be modified to store all the fields of data without having to clear the tables when the program runs.

The next step in my development process was to write additional code to allow for the reading of Shipment, Customer and Vendor Information which would be subsequently inserted into my SQLite Database and tables. I did this by using the CSV Reader (Python.org, 2015), the following code demonstrates that there are three instances of the CSV reader. Each instance of the CSV reader reads each file respectively from a defined location which should be stored securely.

```python
data =csv.reader(open(r'C:\Users\Avish\Desktop\CCA Python 1\Python INPUT\shipments9.csv', 'r'))
data2 =csv.reader(open(r'C:\Users\Avish\Desktop\CCA Python 1\Python INPUT\vendors.csv', 'r'))
data3 =csv.reader(open(r'C:\Users\Avish\Desktop\CCA Python 1\Python INPUT\customers.csv', 'r'))
```

Once my Python application is deployed into the production environment at the CCA worksite, a network location will then be accessible to only my Python application. Therefore the above directories can then be changed to a secure network location to where these files are stored and maintained by an authorised CCA employee. This would most likely be an employee who works within the Distribution Centre for CCA.

Once I had tested that the files were being read correctly I was then required to insert all the imported information from the Shipments, Vendors and Customers CSV files. Therefore as per my Python application design I defined the "import sqlite3" module at the top of my Main.py class and subsequently used the "sqlite3.connect" method to connect to my CCA.db database. I deemed that the database connection was successful as I did not encounter any issues once I had written the code for the connection. The manner in which a connection is established was defined on the DB-API 2.0 interface webpage (Python.org, 2015). As I had followed the guidelines as defined on the webpage I had to ensure I did not develop code in a manner which would be considered as insecure.

Moreover, as I went further into the development process, I had to extend my knowledge in Python and as a result I was experimenting and testing my code simultaneously. Thus, before I could continue developing the rest of my Python application, I was required to understand how the data would be inserted into my database. This data as previously mentioned was read by the multiple instances of CSV reader, I then had to learn how the SQLite's 'executemany' function operated within Python (Python.org, 2015).

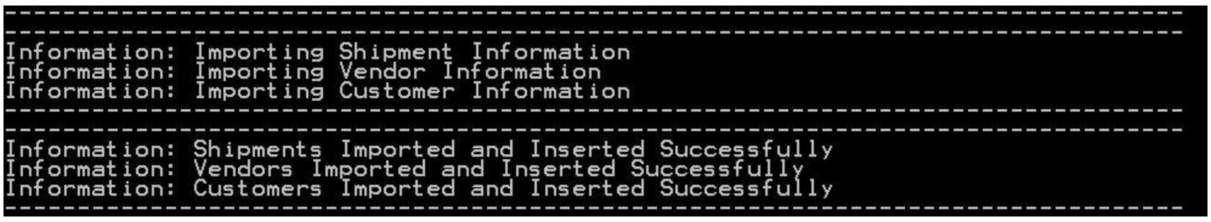
Once I had referred to the online documentation, I was then aware of how to code my Python application to insert the information from the CSV reader instances and into my SQLite tables. The data insertion for the shipments is shown below with a snippet of code, this demonstrates that the table CCASHIPMENTS is used to store the shipments which were extracted from the earlier development of my SAP extraction report. I also ensured that I followed good naming convention throughout the development process.

```
conn.executemany("INSERT INTO CCASHIPMENTS(REFERENCE, OTHER_REFERENCE, SHIPMENT_DATE, PALLET_COUNT, CUST, CUST_ADDRESS, DELIVERY_NUM, SOLD_TO, TPPT, FWDING_AGENT, PROC_TIME, SHIP_TYPE, SHIP_COMP, SUBSNO) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", data)
```

Subsequently within my development phase, I also used the SQLite 'select' function to select all the data and also used the 'cursor' object which allowed me to print each row of data that was read by the CSV Reader.

Once all the shipment information extracted from SAP, Customer and Vendor Information provided to me by CCA was inserted into the respective tables in my SQLite database. The next step within my development process was to start the manipulation process of the imported data.

The following screen is displayed once the Shipment, Vendor and Customer information has been imported and inserted successfully into the SQLite database and tables.



```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Information: Importing Shipment Information
Information: Importing Vendor Information
Information: Importing Customer Information
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Information: Shipments Imported and Inserted Successfully
Information: Vendors Imported and Inserted Successfully
Information: Customers Imported and Inserted Successfully
--------------------------------------------------------------------------------
```

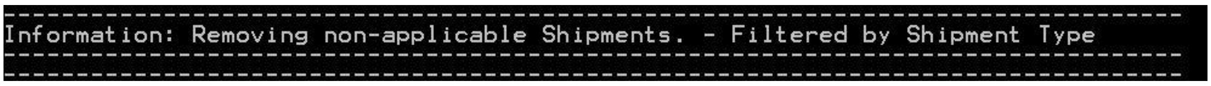
The first set of data that required manipulation was from the CCASHIPMENTS table, this data had to be filtered down to certain Shipment Types. The irrelevant shipment types had to be deleted from the database. Thus as part of my business process learning phase at CCA I determined which shipment types were not required for the Auckland Distribution Centre. Shipment types that were to be deleted from the dataset are as follows:

- Y009
- Y012
- Y013
- Y017

These aforementioned shipment types had to be deleted from the database as I learnt they were not applicable for the Auckland Distribution centre. The result of keeping those shipments could be detrimental to the CCA pallet transfer process. The following snippet of code demonstrates the manner in which I deleted the aforementioned shipment types.

```
cursor = conn.execute("DELETE FROM CCASHIPMENTS where SHIP_TYPE =='Y
009' OR SHIP_TYPE =='Y012' OR SHIP_TYPE =='Y013' OR SHIP_TYPE =='Y01
7'")
```

The following screenshot demonstrates the screen which is shown to the user when the irrelevant shipment types are deleted.



```
--------------------------------------------------------------------------------
Information: Removing non-applicable Shipments. - Filtered by Shipment Type
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

I had initially gone down the wrong path as my understanding of the business process and CHEP pallet transfer process was incorrect. My understanding of the process inferred that each CCA Customer was assigned a CHEP ID which unfortunately was not the case. To avoid large programming catastrophes I tried to ensure I met regularly with the Distribution Centre team, they would advise me if my development progress at the time was heading in the right direction or not. Therefore as part of my learning process I then came to the understanding that my Python application logic in which the Shipments were linked to Customers was incorrect.

More importantly, I was then required to further my learning within the CCA's business environment and understand the manner in which shipment data should be linked to each Customer/Outlet and the newly introduced concept of a Vendor. I was unaware at the time that Vendors also known as the shipping agents would take liability of the pallets when goods were dispatched to smaller customers of CCA such as a local dairy.

I will now go in-depth into the logic that was then required to link and manipulate shipment data for customers and vendors of CCA. The data manipulation within this phase was continually adjusted as my understanding of the pallet transfer process grew.

```
cursor = conn.execute("SELECT ship.REFERENCE, ship.OTHER_REFERENCE,
ship.SHIPMENT_DATE, ship.PALLET_COUNT,ship.CUST, ship.CUST_ADDRESS,
vend.CUSTCHEP_ID, ship.PROC_TIME from CCASHIPMENTS ship, CCAVENDORS
vend where (ship.TPPT = vend.TPPT) AND (ship.FWDING_AGENT = vend.SAP
_FWDING) group by REFERENCE")
```

The above snippet of code demonstrates the manner in which the CCA Shipments are linked to CCA Vendors. Fields taken from CCASHIPMENTS are joined to fields from CCAVENDORS only in the instance where the Transportation Planning Point from CCASHIPMENTS matches the Transportation Planning Point from the CCAVENDORS table and where the Forwarding Agent (Vendor) from CCASHIPMENTS matches the Forwarding Agent from CCAVENDORS. These results are then grouped by Reference.

The first instance of linking and manipulating Vendor information to the shipments can be best explained with an example. For example if CCA dispatch products to "Company A" via the Forwarding agent (Vendor) "Tolls" this snippet of code allows the Python application to determine the shipments in which the liability of pallets are directly transferred to "Tolls" instead of the CCA

Customer/Outlet. The liability of these pallets is determined by the Forwarding agent (Vendor) and the Transportation Planning Point that is extracted from SAP. Therefore when both conditions in the snippet of code are met the liability of pallets for the respective shipments are transferred to "Tolls" the Forwarding Agent/Vendor.

Once this query had been executed I used the cursor object to insert all the applicable shipments into the CHEPINFO table. This is demonstrated as per below in the snippet of code.

```
conn.executemany("INSERT INTO CHEPINFO(REFERENCE, OTHER_REFERENCE, SHIPMENT_DATE, PALLET_COUNT, CUST, CUSTOMER_SAP, CUSTCHEP_ID, PROC_TIME) VALUES (?, ?, ?, ?, ?, ?, ?, ?)", cursor)
```

The use of the CHEPINFO table was required as I needed to use the defined Autoincrement function (SQLite.org, n.d.). This function was required as I needed a line number for each shipment when the information was transferred to the CHEPEXPORT table. The use of Autoincrement would automatically store a line number for each row of information inserted into the table, the line number would also automatically increment when another row of information was added. I did not directly use the Autoincrement function within the CHEPEXPORT table as the line numbers needed to be adjusted.

This therefore was a better option for me to use instead of defining and creating a method that calculates line numbers. Although due to the use of the Autoincrement function the CPU usage was increased but I believed this was a fair trade off as the number of rows that would be inserted into CHEPINFO would not be extremely large (SQLite.org, n.d.).

The next stage in the development process required me to perform the second instance of linking shipments and manipulating data for Outlets also known as CCA's Customers. This stage also included the insertion into the CHEPINFO table. The following snippet of code demonstrated the linking and manipulation process that was required.

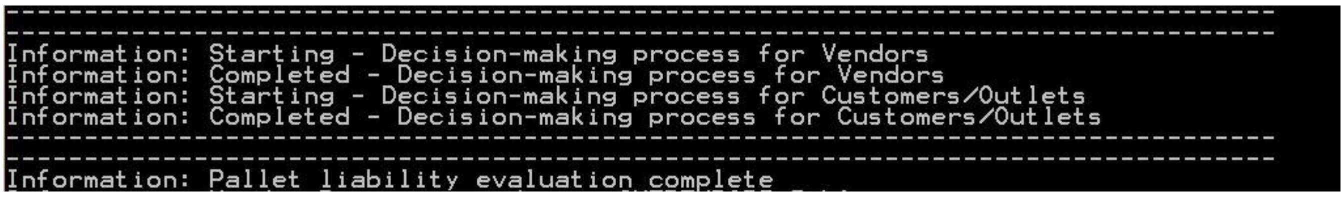
```
1. cursor2 = conn.execute("SELECT ship.REFERENCE, ship.OTHER_REFERENCE, ship.SHIPMENT_DATE, ship.PALLET_COUNT, ship.CUST, ship.CUST_ADDRESS, outlet.OUTLETCHEP_ID, ship.PROC_TIME from CCASHIPMENTS ship, CCAOUTLETS outlet where (ship.FWDING_AGENT = outlet.SAP_FWDING) AND (ship.SOLD_TO = outlet.OUTLET) group by REFERENCE")

2. conn.executemany("INSERT INTO CHEPINFO(REFERENCE, OTHER_REFERENCE, SHIPMENT_DATE, PALLET_COUNT, CUST, CUSTOMER_SAP, CUSTCHEP_ID, PROC_TIME) VALUES (?, ?, ?, ?, ?, ?, ?, ?)", cursor2)
```

The above snippet of code demonstrated the manner in which the CCA shipments are linked to CCA Outlets/Customers. Fields that are obtained from CCASHIPMENTS are joined to fields from CCAOUTLETS only in the instance where the Forwarding Agent from CCASHIPMENTS matches the Forwarding Agent from CCAOUTLETS and also where the Sold-To-Party from CCASHIPMENTS matches the Outlet/Customer from CCAOUTLETS. These results are then also grouped by Reference to limit the possibility of duplicate shipments. Line 2 of the above snippet of code demonstrates the manner in which the selected shipments from line 1 are inserted into CHEPINFO via the use of object cursor2.

As shown above with the second instance of linking and data manipulation for CCA Outlets/Customers, this was done in an effort to ensure the pallet liability was transferred to the customer instead of the forwarding agent listed within the shipment record.

The second instance of linking Customer/Outlet Information to the Shipments can also be best explained with an example. For example if CCA dispatch products to "Company B" via the Forwarding agent (Vendor) "Mainfreight" the liability of pallets in this instance will not be transferred to the Forwarding Agent as the Outlet also known as the CCA Customer is a large enough company that takes liability of the pallets and therefore the pallets are then transferred to them.

Once the decision-making process had been executed in regards to the shipment linking and data manipulation for both the Customers and Vendors, the following screen is displayed accordingly.



```
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
Information: Starting - Decision-making process for Vendors
Information: Completed - Decision-making process for Vendors
Information: Starting - Decision-making process for Customers/Outlets
Information: Completed - Decision-making process for Customers/Outlets
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
Information: Pallet liability evaluation complete
```

The CHEPINFO table was required as the majority of data as required by CHEP was within this table, but the key difference between CHEPINFO and CHEPEXPORT was that the CHEPINFO data was not formatted to CHEP's requirements. The use of CHEPINFO was required as I needed to perform manual changes to the value of the Autoincrement function mentioned earlier. Once the changes were applied the final version of shipment data was then inserted into CHEPEXPORT. The next stage of the development process for the Python application was to prepare the information which needed to be exported to CHEP according to their requirements. The following tables demonstrate the data layout requirements as per CHEP's EDI solution. I had to conform exactly to these requirements whilst I was writing the code for my Python application. If these requirements were not met CHEP would advise me that the shipment data sent to them would not be accepted.

Header Records:

Number	Field	Example	Mandatory
1	Header Indicator	H	Y
2	Recordnum	5	Y
3	RecordCount	24	Y
4	SendDate	01102015	Y
5	ProgramName	CDKTF	Y
6	ProgramVersion	1.1	Y
7	InformerGLID	6412345678	Y
8	CountryCode	64	Y
9	CustomerFileRef	AAZA956141	Y

Movement Records:

Field #	Field	Example	Mandatory
1	LineType	F/D/C	Y
2	RecordNum	3	Y
3	Docketnumber	AAZZ00000001A	Y
4	Sender	6412345678	Y
5	Receiver	6400987654	Y
6	DOD	01102015	Y
7	DOR	01102015	Y
8	EFD	01102015	Y
9	REF1	Reference 1	Y
10	REF2	Other Reference	Y
11	MATERIAL	16001	Y
12	QUANTITY	10	Y

The Header and Movement records acted as a strict guideline whilst I was preparing the CHEP_EXPORT.csv file. Prior to the generation of the export file I was required to ensure that the headers as requested by CHEP were inserted prior to the export file being emailed. The reason for using a CHEPEXPORT table was due to the fact that were multiple fields that were not required for CHEP and therefore this new CHEPEXPORT table only contained the relevant data for CHEP. All unnecessary data was not imported into this table.

Moreover, as per CHEP's EDI requirements I was required to obtain the relevant and required details from CHEP which identifies CCA NZ within CHEP's SAP environment. Information from the Header Records table were obtained from CHEP, excluding Recordnum and RecordCount. These two fields are determined and filled in by my Python Application. Another reason as to why I was required to have a separate table for CHEP data extraction was because I was required to calculate in advance the number of records that were to be sent to CHEP. This RecordCount value was to be inserted into the header of the CHEP_EXPORT.csv file.

The RecordCount as required by CHEP was calculated by my Python application in the following manner:

```
cursor =conn.execute ("SELECT count(*) from CHEPINFO")
recordCount =cursor.fetchone();
recordCount2 =int(' '.join(map(str,recordCount)))
recordCount2 =recordCount2 + 2;
```

I was required to use the SQL count aggregate function as this allowed me to calculate the number of rows within the CHEPINFO table. The number of rows in this table directly correlated to the number of shipment records that needed to be sent to CHEP. Whilst I was writing the code to perform such a calculation, I was faced with a tuple related error. For example if I ran the following statement:

```
print(recordCount)
```

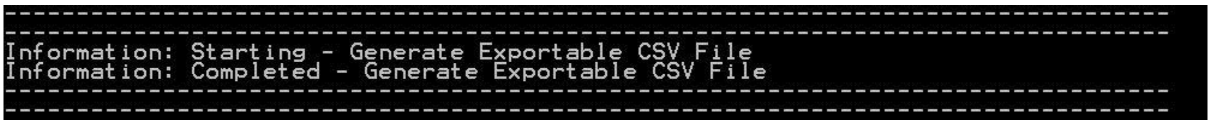
I was presented with a number in this format "(29,)". Due to the tuple being present I was required to remove the brackets and the comma. Essentially I was required to convert the tuple into an integer. The recordCount variable as shown above stores the calculated total number of rows within the CHEPEXPORT table. The second variable recordCount2 was required as I needed to include the 2 header rows that are inserted into each CHEP_EXPORT.csv file. recordCount2 value was therefore inclusive of the two header rows.

Due to several issues I was facing with inserting variables concurrently with information from another table I decided to use a TEMP table to store the variables. The variables stored within this TEMP table were variables such as, lineType, docketNumber, date, informerGLID and materialType. The aforementioned variables were requirements set by CHEP and therefore when I was required to enter all the manipulated shipments into the CHEPEXPORT table I concurrently extracted the information from the TEMP table which included the two header rows.

Moreover, for simplicity I ensured that the CHEPEXPORT table was formatted according to CHEP's requirements. This allowed me to perform a full read of my table to generate the CSV file as required by CHEP. I will now discuss the next major component of my Python application. The following snippet of code demonstrates the manner in which my Python application read the rows of shipment information from the CHEPEXPORT table and subsequently wrote it to the CHEP_EXPORT.csv file.

```python
1. with sqlite3.connect("CCA.db") as connection:
2.     file =open("CHEP_EXPORT.csv", "w", newline='')
3.     csvWriter =csv.writer(file)
4.     connection =cursor.execute("select * from CHEPEXPORT")
5.     info =connection.fetchall()
6.
7. csvWriter.writerows(info)
8. file.close()
```

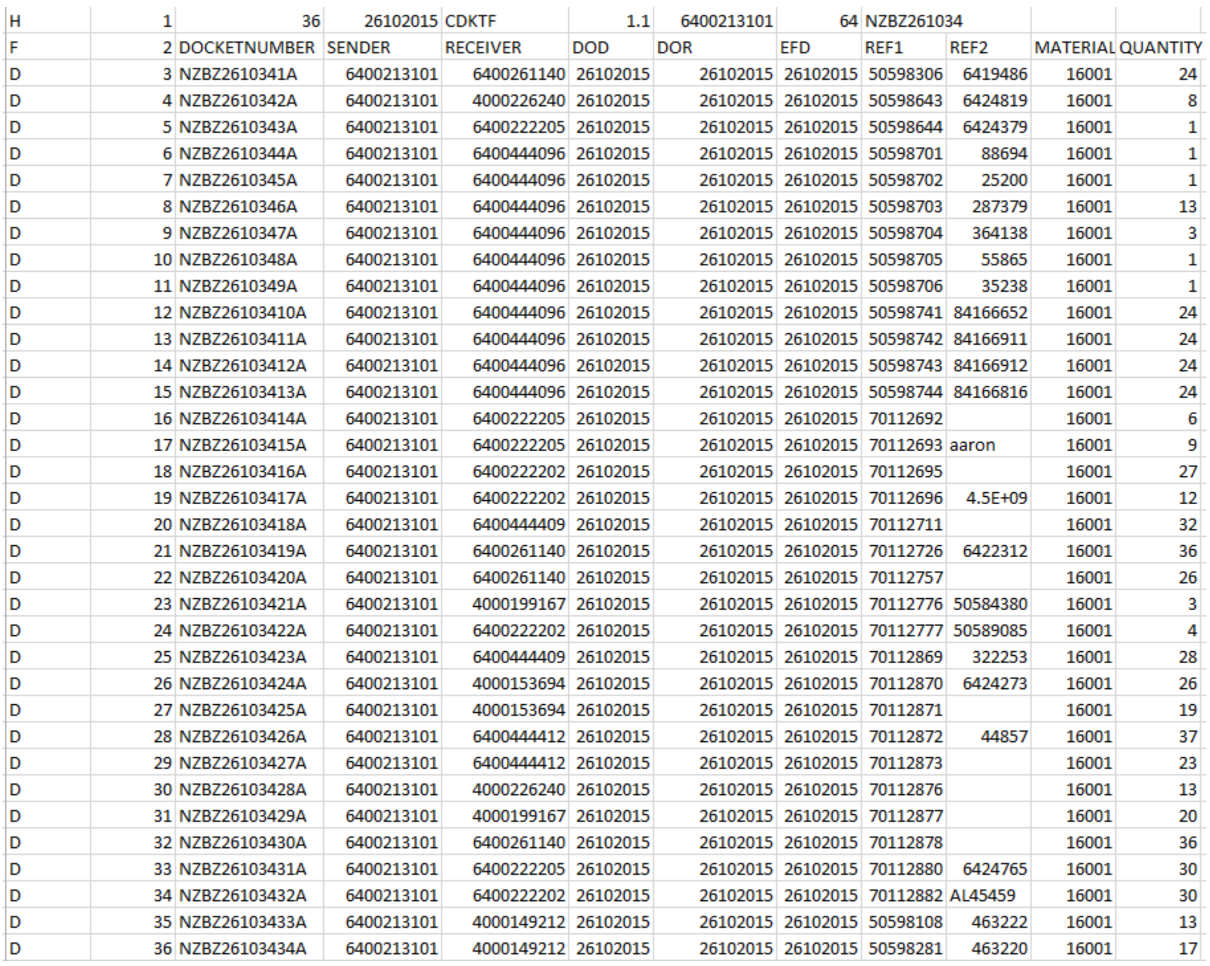
The following screenshot demonstrates the screen in which the user will be shown once the export file has been generated.



```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Information: Starting - Generate Exportable CSV File
Information: Completed - Generate Exportable CSV File
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

The aforementioned snippet of code operates in the following manner. Firstly my Python application was required to use SQLite3 to connect to the database file that was created during the initial stages. This database file was generated within the generateDatabase.py package as 'CCA.db'. Line 2 indicates the file that is to be created which is the CHEP_EXPORT.csv and the parameter "w" indicating 'write'. Line 4 allows the Python application to read the CHEPEXPORT table from the CCA.db and concurrently select all rows of information from the table. The fetchall() function as part of the cursor then allows the Python application to fetch all the rows from the CHEPEXPORT table and of which is then written into the file as shown on line 7.

During the development process I was having several issues whereby the pythonEmailer.py was emailing the CHEP_EXPORT.csv file as a blank file although the file had already been written. After thoroughly reviewing Python documentation online I came to the realization that it was because my Python application still had the CHEP_EXPORT.csv file open and was therefore unable to attach the file with the contents. I required a single piece of code to fix this issue, this is shown on line 8 – file.close(). The following screenshot demonstrates the contents of the extracted CHEP_EXPORT.csv file, this file is sent to CHEP and is processed within the CHEP SAP system of which then parses the information into Portfolio Plus.



H	1	36	26102015	CDKTF	1.1	6400213101	64	NZBZ261034			
F	2	DOCKETNUMBER	SENDER	RECEIVER	DOD	DOR	EFD	REF1	REF2	MATERIAL	QUANTITY
D	3	NZBZ2610341A	6400213101	6400261140	26102015	26102015	26102015	50598306	6419486	16001	24
D	4	NZBZ2610342A	6400213101	4000226240	26102015	26102015	26102015	50598643	6424819	16001	8
D	5	NZBZ2610343A	6400213101	6400222205	26102015	26102015	26102015	50598644	6424379	16001	1
D	6	NZBZ2610344A	6400213101	6400444096	26102015	26102015	26102015	50598701	88694	16001	1
D	7	NZBZ2610345A	6400213101	6400444096	26102015	26102015	26102015	50598702	25200	16001	1
D	8	NZBZ2610346A	6400213101	6400444096	26102015	26102015	26102015	50598703	287379	16001	13
D	9	NZBZ2610347A	6400213101	6400444096	26102015	26102015	26102015	50598704	364138	16001	3
D	10	NZBZ2610348A	6400213101	6400444096	26102015	26102015	26102015	50598705	55865	16001	1
D	11	NZBZ2610349A	6400213101	6400444096	26102015	26102015	26102015	50598706	35238	16001	1
D	12	NZBZ26103410A	6400213101	6400444096	26102015	26102015	26102015	50598741	84166652	16001	24
D	13	NZBZ26103411A	6400213101	6400444096	26102015	26102015	26102015	50598742	84166911	16001	24
D	14	NZBZ26103412A	6400213101	6400444096	26102015	26102015	26102015	50598743	84166912	16001	24
D	15	NZBZ26103413A	6400213101	6400444096	26102015	26102015	26102015	50598744	84166816	16001	24
D	16	NZBZ26103414A	6400213101	6400222205	26102015	26102015	26102015	70112692		16001	6
D	17	NZBZ26103415A	6400213101	6400222205	26102015	26102015	26102015	70112693	aaron	16001	9
D	18	NZBZ26103416A	6400213101	6400222202	26102015	26102015	26102015	70112695		16001	27
D	19	NZBZ26103417A	6400213101	6400222202	26102015	26102015	26102015	70112696	4.5E+09	16001	12
D	20	NZBZ26103418A	6400213101	6400444409	26102015	26102015	26102015	70112711		16001	32
D	21	NZBZ26103419A	6400213101	6400261140	26102015	26102015	26102015	70112726	6422312	16001	36
D	22	NZBZ26103420A	6400213101	6400261140	26102015	26102015	26102015	70112757		16001	26
D	23	NZBZ26103421A	6400213101	4000199167	26102015	26102015	26102015	70112776	50584380	16001	3
D	24	NZBZ26103422A	6400213101	6400222202	26102015	26102015	26102015	70112777	50589085	16001	4
D	25	NZBZ26103423A	6400213101	6400444409	26102015	26102015	26102015	70112869	322253	16001	28
D	26	NZBZ26103424A	6400213101	4000153694	26102015	26102015	26102015	70112870	6424273	16001	26
D	27	NZBZ26103425A	6400213101	4000153694	26102015	26102015	26102015	70112871		16001	19
D	28	NZBZ26103426A	6400213101	6400444412	26102015	26102015	26102015	70112872	44857	16001	37
D	29	NZBZ26103427A	6400213101	6400444412	26102015	26102015	26102015	70112873		16001	23
D	30	NZBZ26103428A	6400213101	4000226240	26102015	26102015	26102015	70112876		16001	13
D	31	NZBZ26103429A	6400213101	4000199167	26102015	26102015	26102015	70112877		16001	20
D	32	NZBZ26103430A	6400213101	6400261140	26102015	26102015	26102015	70112878		16001	36
D	33	NZBZ26103431A	6400213101	6400222205	26102015	26102015	26102015	70112880	6424765	16001	30
D	34	NZBZ26103432A	6400213101	6400222202	26102015	26102015	26102015	70112882	AL45459	16001	30
D	35	NZBZ26103433A	6400213101	4000149212	26102015	26102015	26102015	50598108	463222	16001	13
D	36	NZBZ26103434A	6400213101	4000149212	26102015	26102015	26102015	50598281	463220	16001	17

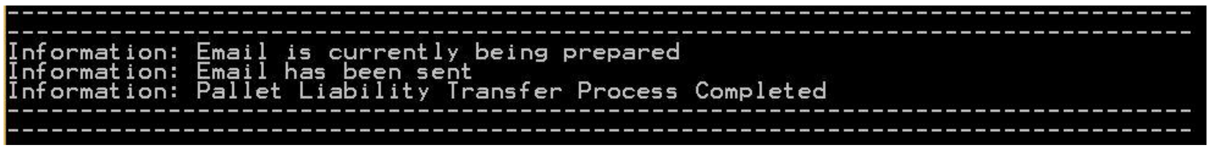
Python Application Output -
CHEP_EXPORT.csv file example

Furthermore, I will now discuss the final aspect of my Python Application which is the pythonEmailer.py. This aspect of the project took a considerable amount of time also as there were several aspects that continuously required adjustment. By the end of the testing and modification phase during development of the pythonEmailer.py I had sent an excess of 250 emails.

The pythonEmailer initially contained several issues. The main issues I encountered are as follows:

- Failed logon to email server
- No email was being sent
- No attachment within email
- Corrupted attachment when sent

To get past a few of these key issues I spent a considerable amount of time fixing every issue whilst experimenting and using different MIME handling packages. Several issues were related to the MIME types and the encoders I had used. Therefore I continuously refined my pythonEmailer package to suit to my requirements which subsequently then resulted in attaching the CSV file to an email. The following screenshot demonstrates the screen in which the user is shown once the email has been generated and sent.



```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Information: Email is currently being prepared
Information: Email has been sent
Information: Pallet Liability Transfer Process Completed
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Furthermore, I ensured that I met CCA's requirements of this project by successfully demonstrating the operation of my Python application to my industry mentor. I also obtained the CHEP's test EDI email address whereby the email sent from my Python application that contained the CHEP_EXPORT.csv file was classed as approved and therefore meant that my Python application development was a success. I have also ensured that my Python application has met the brief requirements that were initially provided to me by CCA.

Future Work

I believe that this project should be taken further in regards to additional development by a CCA representative or contractor. There are a few aspects which I believe should be changed and unfortunately due to a series of factors such as time, brief requirements, University requirements, CCA requirements and CHEP requirements I was unable to implement certain features. Certain characteristics such as the manner in which the shipment information is transferred to CHEP should be modified. I was required to use CHEP's EDI as they have no alternative solution of parsing data to them. Therefore the security in this sense should be improved from their side. The solution as developed by me allowed my Python application to send extracted shipment information from my database to CHEP. In this section of the report I will discuss improvements in which I believe will further enhance this developed solution.

GUI – Graphical User Interface

To further develop on-top of my existing Python Application I recommend for programmers to develop a Graphical User Interface (GUI). A GUI will allow for CCA Distribution Centre employees to use icons or visual cues to interact with my Python application. A GUI can be considered as a human-computer interface and it essentially allows humans to interact with a computer (Linfo.org, 2004). As my Python application is based off a CLI known as a command line interface (Linfo.org, 2004) a GUI will be a significant addition and should definitely be considered as part of the Future Work.

The implementation of a GUI and additional features can allow CCA Distribution Centre employees to have a pallet transfer count total for set periods. At present a total is given for each instance in which the program runs. For example, a CCA employee can select a custom time period and view the total number of pallets transferred for that period if this additional feature is implemented.

Therefore at the end of month when re-conciliation of pallets occur, an employee can simply select that period and view the total number of pallets that have been transferred to Vendors and CCA Customers. If my Python application is deployed at present the Distribution Centre employees will be able to view the results of the automatic processing and extraction of data via the Portfolio Plus interface which belongs to CHEP.

Furthermore, another additional feature that can be implemented into my Python application is the search function, this function can be added within the GUI which will allow employees to view a Customer/Vendor details and make changes to the data only if the employee is authorised to do so. Additionally, the employee will also be able to view the total number of pallets that have been transferred to that Customer / Vendor over a custom time period.

Over 50% of programming code is now dedicated towards the user interface (Galitz, 2007). Although the use of a command line interface is not a problem in my Python application as the intent for the solution was to purely provide automation of shipment data transfer to CHEP, an implemented GUI can significantly benefit the Python application. I propose that a Direct Manipulation interface be implemented within my Python application. This will allow for the user to interact with the cues and elements displayed on a screen in-front of them (Galitz, 2007).

The advantages that can be provided by the use of a Direct Manipulation GUI are as follows (Galitz, 2007):

- Faster Learning

- Easier Remembering
- Exploits visual/spatial cues
- Easy error recovery
- Provides context
- Immediate feedback

Therefore I would recommend that a GUI is implemented onto the next revision of my Python application. I have advised my industry mentor of my proposed future works and he has assured me that CCA will try and attempt to further develop this application with resources provided by CCA Australia.

SQL Server

In regards to the full scale New Zealand deployment of my Python Application for all CCA Distribution Centres, I believe that it is crucial that the database is upgraded to use SQL Server instead of SQLite. A few main benefits that Microsoft SQL Server provides are as follows (Microsoft, 2015):

- Mission-critical performance
- Faster insights on any data
- Platform for hybrid cloud
- Proven, predictable performance

SQLite is an extremely light cross platform library and I believe that my Python application could process a larger amount of shipments if SQL Server was used in conjunction with Python. If my Python application was deployed to all CCA distribution centres certain changes would be required to be made to my Python application. SQL Server 2014 is much more powerful when compared to SQLite and therefore is deemed to be a more suitable solution when deployed to all NZ distribution centres. SQL Server when compared to SQLite also provides Immediate Consistency this is whereby methods are in place to ensure consistency is met whilst operating in a distributed system. (DB-Engines, 2015)

Furthermore, another benefit that must be considered when comparing SQL Server to SQLite is that Server-side scripts can be deployed (DB-Engines, 2015). The use of Server-side scripts will enable CCA to use stored procedures if required (DB-Engines, 2015). Therefore if there was a critical shipment that was required to be processed immediately, the use of stored procedure could allow this to occur.

Virtual Desktop Infrastructure

The deployment of the Python application can also be done on Virtual Desktop infrastructure (VDI). I propose that as part of the Future Work, CCA should deploy the Python application on a virtual desktop which essentially allows an operating system to operate via the use of server hardware (PCWorld, 2012). Therefore the security in this sense is improved as the use of VDI's enforce strict policies as to who can and cannot access the virtual desktop. The local in-memory storage and processing can all be performed on a virtual desktop. This also in-turn reduces the possibility of a rogue user gaining access and making changes to a local machine. The Python application could possibly be operating at that time which means that pallet transfers could be interrupted and modified in a malicious manner.

Additional Security Measures

I believe that as part of the future work aspect for my Python application additional security measures can be incorporated into my solution at a later date. Preferably before my Python application is deployed to all CCA Distribution Centres. Although the solution I have created meets CCA's and CHEP's requirements, I believe that more can be done in the future in regards to the security aspect of this project. Focussing on the security mechanisms was not a primary concern for me as I was told by my industry mentor to focus on the design and functionality aspect of my Python application.

In this section of the report I will discuss what can be done on the Python application to further increase the application security. We introduce the concept Resin which is a system that is operated within a language runtime such as the Python interpreter (Yip, 2009). Resin provides policy objects, programmers can then use these objects to specify assertion code. Essentially Resin is based off the concept whereby application security can be improved with the use of Data Flow insertions. The runtime that is implemented as part of Resin, checks data flow assertions in the manner whereby policy objects are propagated along with the data (Yip, 2009). Filter objects are then invoked when i.e. my Python application attempts to write the CSV file (Yip, 2009).

The use of Resin can allow future developers of my Python application to ensure the application follows the correct procedure of coding. Although I followed good coding practices, it is equally important that this is checked. Resin has been stated by the authors Alexander et al. that it can be integrated with existing application code. This benefit therefore can subsequently related back to my Python application that has already been developed by me. It is important to note that due to a faulty data flow, vulnerabilities can be severely exploited. In the case of my Python application, a check against the possibility of a SQL Injection and Cross-site scripting vulnerability would be a good area to start in when additional security mechanisms are considered.

Furthermore, the SQL Injection and Cross-site scripting vulnerability has been described in relation to a Web Application, if this Python application was used within a Web application it would be crucial to evaluate my Python application against vulnerabilities that may exist in such an application. SQL injection is when the user's input flows through to the SQL query string (Yip, 2009). The result of this occurring can be detrimental if an attacker managed to hack the files that are read by my application and also successfully use SQL injection to insert malicious data into my SQL tables and database.

Different strategies in regards to the use of Resin are discussed to address the vulnerabilities as aforementioned. The main strategy operates in the following manner:

- Two policy object classes defined. UntrustedData and SQLSanitized (Yip, 2009)
- Untrusted input data is annotated with the use of the UntrustedData policy (Yip, 2009)
- SQLSanitized object is attached to sanitized data (Yip, 2009)
- Policy objects are checked on each SQL query. Therefore if the query as entered by the user contains any characters from the UntrustedData policy filter will throw an exception (Yip, 2009).

Being able to address cross-scripting vulnerabilities operates in a similar manner. HTMLSanitized is therefore used instead of the SQLSanitized policy object.

Additionally, the use of data flow assertions and policy objects can be implemented on-top of my Python application which will hopefully provide a greater sense of security for the application. Moreover, the Python application will also be less susceptible to vulnerabilities such as SQL injection and cross-site scripting.

In addition to the GUI section of the future work, I recommend that a login screen is implemented within my Python application. This will allow CCA to control who has access to the application. Although the system will be implemented within a secure environment, we must take into account the possibility of a malicious user who attacks the network systems within CCA and attempts to change the programming and decision-making logic within my application. The result of this occurring could potentially equal in a loss of monetary terms whereby less pallets are transferred out from CCA's CHEP account. Therefore CCA will then be liable for each individual pallet that has left the Distribution Centre but of which has not been accounted for due to the infection of the Python application.

Furthermore, extending off the idea of a login screen. I also believe that login security should also be implemented in a manner that utilises two-factor authentication. This operates by using two individual procedures to login into a system. The most common method used in relation to this two-factor authentication is via the use of a password and a smart card (Yang, 2008).

The smart card based approach is as stated by Yang et al. one of the most commonly used and most convenient methods of two-factor authentication. Therefore in this scenario where two-factor authentication is deployed, there is a server (S) that is deployed along with a client (A) (Yang, 2008).

Two-factor authentication can be best described in the scenario whereby a CCA representative whom can be represented as client (A), is required to login into the system and view the Customer and Vendor information that is stored within a database. They would be typically required to enter their username and password within the login screen, if these credentials are correct they will then be required to insert their smart-card. Client A has the identity ID_A and will therefore be allowed to login once the server S verifies the users identity.

We are assuming that the smart-card has not been stolen/deactivated. It is important to note that the client also known as the CCA representative will only be allowed to login once both factors of authentication have been verified. This prevents a malicious user from logging in with stolen credentials. An example of a smart-card is the eToken PRO Smart card. This smart-card is a certificate based strong authentication solution (Gemalto, 2015). Smart-cards that have been proven to be secure should be employed in the second factor of a two-factor authentication procedure.

In relation to increasing the security measures indirectly and directly related to my program CCA should also ensure the Customer, Vendor and Shipment information is stored securely and has file access control implemented. This should be done in a manner whereby only the Python application and administrator has access to these files. File access control will allow the control of who can view, modify and delete the data that is vital to the operation of my program.

Overall within the future works section of this report, I have highlighted certain areas that should be looked into, and also areas where resources should be applied to further strengthen and develop the application.

Conclusion

This report has focussed on the business problem, design and development of my BTech Project also known as my Python application and the CHEP pallet management system. I have now completed my findings as discussed within the report. The solution developed by me focuses on the solving the current business issue where shipment and pallet information is manually keyed in into an external CHEP Portfolio Plus system.

Furthermore, as a result of this manual process several shipments were forgotten about or were incorrectly entered. With the use of my Python application this entire process is completely automated and requires no user-intervention. If shipment information was not entered into the CHEP Portfolio Plus software it resulted in CCA being liable for hundreds and thousands of dollars of 'lost' pallets. My developed solution significantly reduces this loss in both monetary terms and physical pallet terms.

Moreover, I have learnt a large variety of skills that I will find extremely valuable in regards to my future career aspects. I have also improved my project and time management skills which I believe is fundamental when developing a solution. During the course of my project I was required to continuously organise meetings and schedule tasks with representatives within CCA. I believe this enabled me to further grow my verbal and written skillset. I was also required to meet constant deadlines which ensured that I had to upskill my knowledge in Python, SAP ABAP and SQLite in a time efficient manner. The knowledge and corporate experience I have gained during the course of this project is deemed invaluable and I am extremely grateful to have been given this opportunity.

Bibliography

Bettaway. (n.d.). *Pallets Network*. Retrieved from Bettaway: https://www.bettaway.com/baw/palletsNetwork

CHEP. (n.d.). *Wooden Pallets*. Retrieved from CHEP: http://www.chep.com/pallets/wooden_pallets/

Coca-Cola Amatil. (n.d.). *About US*. Retrieved from Coca-Cola Amatil: https://ccamatil.co.nz/about-cca/key-facts/

DB-Engines. (2015). *System Properties Comparison Microsoft SQL Server vs. SQLite*. Retrieved from DB-Engines: http://db-engines.com/en/system/Microsoft+SQL+Server%3BSQLite

EDI Basics. (n.d.). *What is EDI?* Retrieved from EDI Basics: http://www.edibasics.com/what-is-edi/

Galitz, W. O. (2007). *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques.* Indonesia: Wiley Publishing.

Gemalto. (2015). *Certificate-based PKI Smart Cards*. Retrieved from Safenet-Inc: http://www.safenet-inc.com/multi-factor-authentication/authenticators/pki-smart-cards/

Guo, P. (2014, July 7). *Python is Now the Most Popular Introuctiory Teaching Language at Top US Universities*. Retrieved from Association for Computing Machinery: http://cacm.acm.org/blogs/blog-cacm/176450-

python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext

Linfo.org. (2004, October). *GUI Definition*. Retrieved from Linfo.org: http://www.linfo.org/gui.html

Martin, R. C. (2000). *Design Principles and Design Patterns.* Robert C. Martin.

Microsoft. (2009). *Software Architecture and Design.* Microsoft.

Microsoft. (2015). *SQL Server 2015*. Retrieved from Microsoft.com: http://www.microsoft.com/en-us/server-cloud/products/sql-server/

Moxon, P. (2014). *SAP ABAP Programming For Beginners*. Retrieved from Udemy: https://www.udemy.com/sap-abap-programming-for-beginners/

PCWorld. (2012). *The Pros and Cons of Using Virtual Desktop Infrastructure*. Retrieved from PCWorld.com: http://www.pcworld.com/article/252314/the_pros_and_cons_of_using_virtual_desktop_infrastructure.html

Python.org. (2015, February 26). *CSV File Reading and Writing*. Retrieved from Python.org: https://docs.python.org/3/library/csv.html

Python.org. (2015, February 26). *DB-API 2.0 interface for SQLite databases*. Retrieved from Python.org: https://docs.python.org/3/library/sqlite3.html

Python.org. (2015, February 26). *Generating MIME documents*. Retrieved from Python.org: https://docs.python.org/3/library/email.generator.html

Python.org. (2015, May 23). *Miscellaneous operating system interfaces*. Retrieved from Python.org: https://docs.python.org/2/library/os.html

Python.org. (2015, May 23). *System-specific parameters and functions*. Retrieved from Python.org: https://docs.python.org/2/library/sys.html

Rescorla, E. a. (2012). Datagram transport layer security version 1.2.

Rose, M. (2013). *What is SAP?* Retrieved from Tech Target: http://searchsap.techtarget.com/definition/SAP

SQLite.org. (n.d.). *About SQLite*. Retrieved from SQLite.org: https://www.sqlite.org/about.html

SQLite.org. (n.d.). *Autoincrement In SQLite*. Retrieved from SQLite.org: https://www.sqlite.org/autoinc.html

The Python Programming Language. (1997, 12 09). Retrieved from University of Michigan: http://groups.engin.umd.umich.edu/CIS/course.des/cis400/python/python.html

TrackIt. (n.d.). *Asset Tracking and Management*. Retrieved from TrackIt: http://www.trackit.co.nz/product/Asset_Management.aspx

WiseTrack. (n.d.). *Asset Tracking Software*. Retrieved from WiseTrack: http://www.wisetrack.com/asset-tracking-software/

Yang, G. a. (2008). Two-factor mutual authentication based on smart cards and passwords. *Journal of Computer and System Sciences*, 1160-1172.

Yip, A. a. (2009). Improving application security with data flow assertions. *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, (pp. 291-304).