Requirements Specification for

Durham Food Bank

Prepared By:

Patrick Coglan

Jerzy Foss

Ethan Kho

Nuowa Lora Liu

Wangxiuzi Peng

Aidan Williamson

1 – Introduction

1.1 - Overview and Justification

The client is Durham Foodbank or rather the Durham Christian Partnership Foundation (herein referred to as the DCP). The client acts as a warehouse for local food banks, storing goods ranging from canned foods to nappies and then distributing the goods to nearby food banks as orders come in.

Currently, the clients warehouse consists of four different areas, referred to as “zones”. Each zone is a long stretch of many ‘bays’, these bays are labelled with respect to the alphabet so A, B, C and so on… Let's take bay ‘A’ as an example, this bay will have 5 separate shelves (vertically) each shelf having space for 12 trays, 4 trays wide and 3 trays stacked, with the exception of the very top shelf having space for 16 trays, 4 trays wide and 4 trays stacked. Each individual tray has 3 labels on it: what goods the tray contains, the general expiry date of its goods, and the weight of the goods within that tray. The weights must be collected so that the client can fulfill their required annual report to the Trussell Trust (a national charity). The records containing the contents of each shelf are currently stored in Excel and are updated once a week. There is also a 5th storage area, consisting of unsorted goods, on the floor of the warehouse.

Problems currently faced by the client include a supply of goods that exceeds the maximum storage they have available (~1200 trays) and a lack of a stable method of categorising goods which causes difficulty in looking up specific items. The proposed system will help the client manage their stock and efficiently search for specific items while also getting an accurate picture of their current stock.

The rest of this document contains a more detailed look at the goals we aim to achieve with this project and exactly how much the client values each aspect of the project. It starts by outlining what the project itself will be and the processes within the physical warehouse that it will cover before moving on to highlight specific requirements and how these will be achieved.

1.2 - Project Scope

The project is a warehouse stock management system that has functionalities for searching within and editing the database. The project should improve the client’s workflow and introduce a more efficient culture within the client’s workspace while also keeping as close as possible to the client’s specifications, including their existing categorisation system.

The project system can be roughly divided into three interconnected but distinct parts. The database, the search and the editing. For the database part, the data should be stored in such a way that it gives a reasonably current (to an accuracy of around 1 day) representation of how the goods are physically stored in the DCP warehouse. That is, each item should list the zone, the shelf, the level it is in and its expiry date. It may also be beneficial to have each zone have a

corresponding colour for ease of differentiation. The database should be big enough to contain information for all the goods in the warehouse with room for scalability (to about 10000 trays). The database should also contain the information of unsorted items and items not stored on the shelves.

As for the search system, users should be able to enter the name of a specified item and have the system return the data for a tray of that item in the warehouse. The project system is expected to return the information quickly and accurately.

Lastly, the editing system itself can be divided into three parts; inserting new items, deleting item and moving items. Users should be able to insert new items into specific parts of the database with ease, move items from one shelf to another and delete items without causing inconsistencies in how the data is stored. For deletion and insertion, the client requires the system to facilitate the ability to add/delete multiple items at a time (to stock or remove an entire shelf of trays in a column). The system should also log changes to the database to ensure integrity of data.

Each of the systems should be intuitive to use while also providing enough functionality to substantially increase the rate at which the client and update their records.

As additional functions, the system should be able to make a simple but easy to read report on the total amount of food processed by the DCP and be portable to different systems for other food banks.

1.3 - System description

The system does not need to be able to store and manage a detailed inventory with granularity to individual items. Instead, it needs to be able to store stock information to a granularity of “trays” of an item. The food bank needs to know the tray quantity, location and expiry date for the various food items it processes. The food bank would like to use the system to minimise the stock re-shuffling requirements, use warehouse space more efficiently and be able to tell how many trays of an item they have at any given time which also helps them ensure that items do not expire. In addition, the system must be able to provide a report of weights and tray quantities to provide to the Trussell Trust. The food bank has no control over its food intake and as such, volumes and the nature of stored goods can fluctuate rapidly. This is especially true during busy periods such as the Harvest Festival and Christmas where several times the usual amount of stock arrives. As such, the food bank needs to allocate space in an ad-hoc manner, unlike a traditional warehouse where stock control usually means goods in the warehouse do not move much.

The current approach the food bank is using is to manually tally up stock on paper once a week. This is then transferred to an Excel document which is used to generate reports. This is a time-

consuming approach and is prone to data transcription errors, so double entry is used. In addition, this manual system quickly runs out of sync with the real situation in the warehouse. The food bank has since created its own system in Excel using VBA (on a Microsoft Surface Go Tablet) with the aim of increasing data collection speed. However, this system has not been rolled out properly since, in practice, it has fallen short in several key areas. It is not platform-independent meaning a device with specific software must be used and in addition, the system is quite slow. The user is often left waiting a couple of seconds for the sheets to redraw between views. It is also not as simple to start up as the food bank would like, so only specific members of staff know how to run it. There is a lack of proper file storage and data archiving/backup functionality which may be problematic in the long term. One of the major issues is that there is a lack of customisation options so the VBA must be edited to make any changes to the warehouse layout or to allow dates more than a couple of years in the future to work. Currently only the member of staff who created the system can make these changes. The food bank would also like the system to have export functionality and be able to create printed reports, features which are not available on their system.

Our proposed solution is to create a web application. This fulfills the platform-independent criterion since it will be able to run on any modern device and this means that the food bank will be able to buy cheaper devices or have staff use their own. This helps fulfill one of their goals of bringing about wider adoption of this sort of system between food banks. The monthly cost of paying for a cloud server to host the database and handle data integrity and redundancy will be less than that of buying specialist devices in the short to medium term at this sort of scale and will far outweigh the operational cost should a locally stored database become lost or corrupted. The web application will have a responsive front-end for cross-platform use. We will use NodeJS as a back-end to allow for rapid prototyping and to make future changes simple. This will help us to push out an MVP to the food bank faster than we would otherwise be able to. The food bank wants flexibility with shelf structure and adding additional storage space (currently a corner of their floor) to the system. For this reason, we intend to use a schema-less database which will easily provide the warehouse layout customisation options required without complex relations. MongoDB is a good candidate here since it has great NodeJS support, is quick to build databases on, and can be high performance due to its B Tree indexes. In addition, should the food bank reach a scale whereby they have a significant increase in transactions/second or total size of data stored, a MongoDB database easily supports sharding across storage and process nodes. This will cover the core requirements of the food bank but as an extension we would also intend to offer them a way to speed up their data entry into the system via a tray coding system and OCR.

The food bank has evaluated many pre-built warehouse management systems, but none have matched their requirements in terms of data collection speed and, more importantly, the ability to move items. Additionally, the food bank has tried several data input systems such as programmable keypads and barcode scanners but decided that data input via a tablet was the most efficient solution which also avoided accidental input.

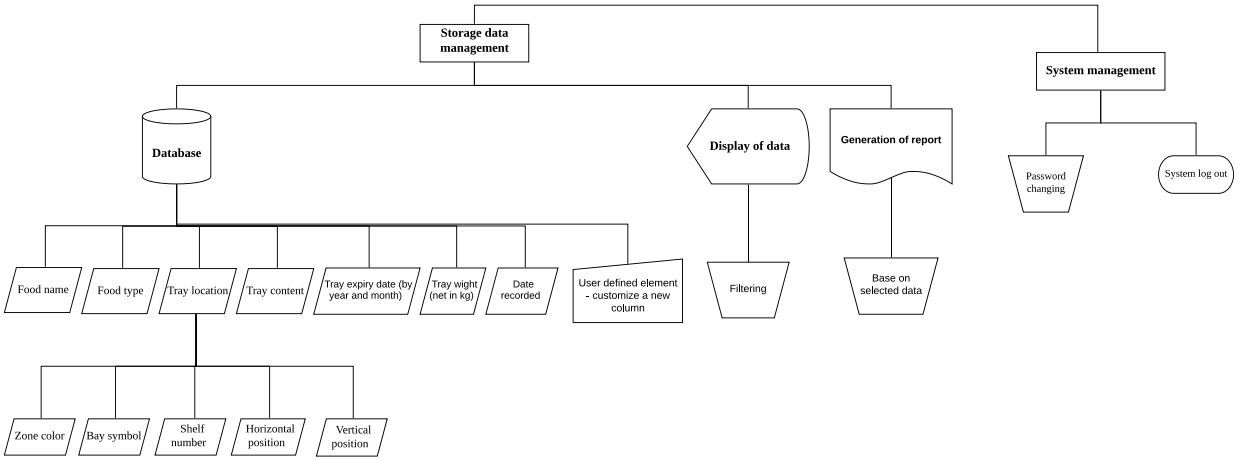
We have also investigated some warehouse management systems such as the one built into Zoho. This gives control on a per item basis which could be mapped to a tray representation for

the food bank and comes complete with stock level warnings which is something the food bank would like. It allows logging transfer of items between warehouses, which could probably be mapped to represent shelf locations instead of individual buildings but this is not an intended use and as such the experience for the food bank would be far inferior (no shelf layout, slow data entry etc) to a custom solution. It is even a feature marketed for inter-warehouse not intra-warehouse transfers. Many other products suffer from this same limitation.

2 – Solution Requirements

2.1 - Functional requirements

Below is a rough diagram of what the system may look like based on the specifications in this section.



ID	Title	Priority	MuShCo	Dependencies
FR_1	Data elements	High	Must	N/A
Description	The system must capture and retain the following data elements: 1. Tray Location 2. Tray Contents 3. Tray Expiry date 4. Tray Weight (Net) 5. One user definable element 6. Date recorded			

Expected Results	These six data elements should comprise the full dataset required by the system. Whether they are stored as a flat file or in a relational database is not material to functionality unless it adversely affects performance.
Exception Handling	No free-text input will be allowed to fulfil the need for data input validation.

ID	**Title**	**Priority**	**MuShCo**	**Dependencies**
FR_2	Tray Location	High	Must	N/A
Description		The system must uniquely identify every tray location.		
Expected Results		Once the information of an item needs to be put into the system, users should input tray location manually. To identify every tray location, this can be achieved by a concatenation of zone, bay, shelf, horizontal position, and vertical position. (e.g. BLUEA1H4V2. This refers to: Zone-BLUE. Bay-A. Shelf-1. Horizontal Position-4. Vertical Position-2.)		
Exception Handling		No free-text input.		

ID	**Title**	**Priority**	**MuShCo**	**Dependencies**
FR_3	Tray Contents	High	Must	N/A
Description		The system must accurately record and identify the tray contents.		
Expected Results		Once the information of an item needs to be put into the system, users should input the tray content manually. In advance, the food categories should be defined by the food store manager and held in the system configuration tables.		

Exception Handling	Restricting data entry to pre-defined data. No free-text user data entry is allowed within the system.

ID	**Title**	**Priority**	**MuShCo**	**Dependencies**
FR_4.1	Tray Expiry Date	High	Must	N/A
Description	The system must accurately record the tray expiry date as determined by the tray contents.			
Expected Results	Once the information of an item needs to be put into the system, users should input the tray expiry date manually. In addition, it is not necessary but will be better to have a range of applicable date (Years, Months, Quarters etc.) held in the system configuration tables to enable data input validation.			
Exception Handling	No free-text input is allowed.			

ID	**Title**	**Priority**	**MuShCo**	**Dependencies**
FR_4.2	Tray Expiry Date Change	Medium	Should	FB_1.4.1
Description	The input system should automatically provide a sensible set of dates for the next few years of good expiry dates. This should update each year.			
Expected Results	The system can have date changes automatically.			

Exception Handling	Revert to a sensible set of predefined dates.

ID	**Title**	**Priority**	**MuShCo**	**Dependencies**
FR_5	Tray weight	High	Must	N/A
Description	The system must be able to accurately record the weight of each tray.			
Expected Results	'Tray weight' is the sum of items' weight in each tray a column named 'Tray weight' must present in the data table with kilogram(kg) as the unit. The limit of the weight that each tray can have should be set by the user.			
Exception Handling	Error message generated when the sum of the item weights in each tray exceed the limit. Notice generated when the customized tray weight limit exceeds a reasonable limit, e.g. user enter 100kg as the tray limit. The system should be able to generate a notice to ask the user whether to continue with this entered data or need to change the data.			

ID	**Title**	**Priority**	**MuShCo**	**Dependencies**
FR_6	User defined data filed	Medium	Should	N/A
Description	User defined data filed. The system must provide a data field for each tray which allows both numeric data and text selected from a validation tale to be input.			
Expected Results	User defined data field will allow trays to be 'ear-marked' or tagged uniquely. It could also be used as an item count for each tray to facilitate *ad hoc* stock evaluation at a more granular level than 'tray' in the future.			

Exception Handling	Error message generated if any kind of data doesn't match the default limit set by the user. For example, to have 1000 packs of bean in one tray.

ID	**Title**	**Priority**	**MuShCo**	**Dependencies**
FR_7	Date recorded	High	Must	N/A
Description		It is important to record the date of each stock take so that time-based analyses can be performed on the complete dataset. The time is not necessary, but it can be added without adversely affecting the system performance.		
Expected Results		A default data field that cannot be deleted by the users should be built to make sure when every item is stocked, its stock-take date is also added to the system.		
Exception Handling		Error message generated if the date entered is beyond the date when the data is recorded. Notice generated if the user forgets to enter the date.		

ID	**Title**	**Priority**	**MuShCo**	**Dependencies**
FR_8	Report generation	High	Must	N/A
Description		The system must produce clear reports, well formatted to display target information quickly. I.e. The system should have functionality to print out a report includes data chosen by the user, e.g. item_name, stock_taken_date,volume, tray_location, etc		
Expected Results		This can be achieved by adding a filter to the data table and filter out all the unnecessary data.		

Exception Handling	Error message generated if the data for filtering does not exist.

2.2 - Non-functional requirements

Type	UI Requirement
Metrics	The UI must be clear, legible and logically ordered all of the time
Security	N/A
Constraints	The UI must work on devices of varied size with no stylus

Type	UI Requirement
Metrics	The UI should be colour oriented such that it maps the real-world scene
Security	N/A
Constraints	N/A

Type	UI Requirement
Metrics	The reports should be generated such that they are clear and well formatted in order to display target information quickly (no longer than ~ 10 seconds)
Security	N/A
Constraints	The report output must work with small devices.

Type	Usability Requirement
Metrics	The system must be easy to learn and use
Security	N/A
Constraints	N/A

Type	Performance Requirement
Metrics	The system must never hang / crash in the event of an error, say no more than ~ 20 seconds
Security	N/A
Constraints	N/A

Type	Performance Requirement
Metrics	From a fresh start, a user should be able to log all of the inventory in a warehouse of ~ 2,000 trays in under 3 hours. That is ~ 5.4 seconds per tray For regular inventory updates (where 50% of stock has remained in its previously logged place) should take no longer than 1 hour to complete
Security	N/A
Constraints	N/A

Type	Performance / Latency Requirement
Metrics	The system must respond with visual feedback to 95% of user actions within 0.5 seconds of that action.
Security	N/A
Constraints	Database query may take a while depending on the action, for example generating a report of all stock may take longer than 0.5 seconds

Type	Capacity Requirement
Metrics	The system must be capable of recording 2,000 trays (at this moment in time), but for the future it must be able to handle up to 10,000 trays for expansion.
Security	N/A
Constraints	N/A

Type	Availability Requirement
Metrics	The system should function the same in the absence of a network connection, where the main database is updated on the restoration of the network connection.
Security	N/A
Constraints	N/A

Type	Manageability / Maintainability Requirement
Metrics	Errors should be logged
Security	N/A
Constraints	N/A

Type	Portability Requirement
Metrics	Would be preferable to have this system be set up on other warehouse sites just as easily to allow for wide distribution of the system
Security	N/A
Constraints	Other warehouses will have to adapt to the same system and may need to modify their way of doing things

2.3 – Risks and Issues

As a group, we have agreed that the biggest issue is the client's insistence on updating the records once a week instead of continually making updates to it as they receive and shelve donations. While this may not have too much sway over how we build our system and we don't stand to lose much from it, we feel it is imperative that the client change their current way of doing things if they really wish to make a positive change.

Another risk is that the system is currently being developed with the expectation that not many people are going to be using at any one time. The client currently has one tablet for shared use and maybe one or two admins who would use our system on their phones. However, the client is also planning on scaling. If the client eventually gives many more people access to the system, the integrity of the data could be compromised by multiple entries being made simultaneously.

One more issue is that the client receives an overwhelming amount of donations near Christmas time. So much so that for a few weeks they completely change their methods of categorising goods. The big problem here is that with such an extreme increase in supply over such a short span of time, a lot of goods go unsorted, causing problems down the line in terms of updating large amounts of data at once.

3 – Project Development

3.1 – Development approach

From the onset, we have been keen on an Agile approach to this project. The main benefit to this would be the opportunity to iteratively work on improving our project while also getting feedback from the client at various points. At this stage, the client may have some reservations about some aspects of our project, and we believe that the best way to quell these worries is by allowing the client a greater part in the development of the project. In addition, the client is not entirely sure about what they want out of the project. An Agile approach allows us to deal with new demands and requirements as they come.

Of the various implementations of Agile, the one we are most interested in is Scrum, with each sprint lasting 1-3 weeks. A big part of why we want to use Scrum is because as a group, we have not really gotten to know each other too well yet so the constant development cycle seems to be a solid way of building team relationships. Not just relationships within the team, we also hope to be able to build a good relationship with the client through constant interaction. A good relationship with the client would give us more room to bargain and may make the client more open minded to change.

Another upside of using Scrum is that it gives some breathing room in between periods of intense work. Although we do plan to produce the best product we can, we understand that not all our team members are free at all times. The period of planning in between each sprint should enable the team to function better as a single unit by ensuring each one of us is on the same page as to what we should be doing and that as students, we'll each be able to devote some time to other endeavors.

The Scrum implementation also allows each member of the team to contribute to the project development while also ensuring that no member is left 'dead in the water' so to speak. While each member is expected to make their contributions in terms of coding and development, the daily scrum meeting gives the less experienced members of the group an opportunity to voice any problems they may be facing in their part of the project to the Scrum master. Also, though the less experienced members are admittedly more likely to face problems, it is also perfectly reasonable to think that even the most experienced team member may run into some trouble down the line. The beauty of the daily scrum is that when done right, most issues can be resolved quickly without creating a significant bottleneck.

The Scrum Product Backlog is also useful in that it can act as a sort of visualisation of how much the client values each of the planned features in the project. This can save time by allowing us to shift our collective focus onto higher priority features and leave low priority features as bonuses that we may or may not complete. We are also planning on using a Gantt chart to schedule our development as a way to more easily visualise exactly how much time we expect to put in to the project.

Overall, while in name we're doing Scrum, there's a fair chance we won't follow it to the letter. We're looking at the Scrum approach as more of skeleton that we can base our own implementation around to find what works best for us.

3.2 – Project Schedule

The main milestones for our schedule and their deadlines are:

- Completing the Requirement Spec Documentation: 15/11/19
- Prototype 1: Design Video 29/11/19
- Prototype 2: Client Demo 31/01/20
- Final Project Completion: 20/03/20
- Minimal Viable Product (MVP)

A minimal viable product (MVP) is needed to be completed long before the final project completion and hopefully also the client demo. This would be the minimal acceptable outcome of the project with only the essentials, and it would be demoed to the foodbank to adjust our future goals, deliverables, and schedules.
Based on the functional and non-functional requirements, our interpretation of the MVP must: (note this is very flexible and is likely to develop and change as the project progresses)

- Store essential data elements (tray location, tray contents, tray expiration date, tray weight (net), a user definable element, and date recorded)
- Be able to uniquely identify every tray location
- Accurately record and identify tray contents
- Accurately record tray expiry date
- Accurately record the weight of each tray
- Present a clear, legible, and logically ordered interface
- Produce well formatted and clear reports.

Formal software development planning would begin after the requirement specification documentation - in sync with creating the design video. This planning would be all conceptual and mostly be about discussing different tools, development environments, and vague project architecture as the video is to do with the user interface and navigation of the final product. The tools and environment used have a big impact on this.
Software development would begin after the design video completion and demonstration. This is when we would have our concept solidified and have a good understanding of how to go about creating that concept. As described in the previous section, it would be through the agile scrum methodology and working towards our MVP.

3.3 - Definitions

Front-end: the part of the system with which the client interacts

Responsive front-end: a front-end that resizes and reflows based on the size and layout of a client's device

UI (User Interface): an aspect of the front-end, often most focused on how the front-end looks

Back-end: the part of the system that handles the bulk of the data storage and processing, which the client does not interact with, except through the front-end.

MVP (Minimum Viable Product): the first iteration of the solution given to the client which contains enough functionality to solve their problem, if not perfectly

Schema-less database: a common phrase in database terminology which should technically be "dynamically typed schema database", a type of database where the structure is defined by the data it contains

B Tree Index: a system used in a DBMS to speed up data retrieval

Sharding: a horizontal (distributed across several process nodes) partition of data used to spread load

Transaction: an operation on data in a database; a search, addition, modification or deletion

Latency: the delay between a user performing an action and that action being reflected by the system

Agile: the division of work into short phases to deal with changing requirements and to improve development efficiency

Scrum: a framework based on Agile principles, where work is time-boxed, and teams complete a certain amount of work in that time