Contents

Introduction

We have been developing a smart shelf inventory management system targeted primarily at small to medium sized labs and businesses.

By providing real-time stock level data, a fast item setup process and a convenient place to store re-order information, our system has been designed to streamline inventory related tasks, reducing the amount of time managers need to spend on them.

Small to medium sized labs and businesses were selected as our target for two main reasons. First, they are underserved by current inventory solutions. The stakeholders we surveyed generally performed inventory management either by eye, by hand or manually by updating spreadsheets. Second, their inventory needs are large enough that our concept of a shelf that reduces the time and effort required for inventory management is appealing.

The methods currently employed by stakeholders surveyed led to risks of inaccuracies and stock not being available when needed, but most importantly they took valuable time away from managers, who often saw inventory management as an essential, but boring chore.

We gathered insight and requirements through a series of structured interviews and prototype demonstrations with stakeholders. Responses gathered during consultations were, positive, a few showed strong enthusiasm for the value a system like ours could bring to their workflow.

A ‘minimum viable product’ was created from the gathered requirements, it has served as our target for functionality to implement over the past few months.

The process of development and evaluation, difficult at times, but overall, quite successful. All aspects of the minimum viable product outlined in our proposal were explored and attempted with eight of the ten identified features eventually completed in time.

We now have a functioning system that both serves as a good demonstration of the value a smart shelf system can bring and a foundation upon which a marketable product could be built if development continued.

The smart shelf built using an Arduino pressure sensor and other components works well and successfully sends real time stock information to the inventory management application.

The application has functionality to add, edit and delete items, as well as swap shelf positions, display stock information and present item details in a way useful for when re-ordering.

Some more work would be needed to reach a point where our product could be sold, but we have a firm belief in the value it could bring to our stakeholders.

Design and Implementation

The design process started with features identified through interactions with stakeholders, in our 'Minimum Viable Product' (MVP), outlined in our proposal and reproduced below (figure 1)

Minimum Viable Product

1. Add Items to shelf (set up)
    - set up/calibrate item weight
2. Remove items from shelf (delete)
3. Swap items/shelf spaces
4. Update shelf stock levels
5. Display stock levels:
    - Number of items
    - Absolute weight
    - Percentage weight
        - Manually set 100%
        - Auto Determine 100%
6. Stock level history
7. Low stock warnings within app
8. Form field to store re-supply information
9. Search
    - Search by name or search by 'tags'
10. Amend/ modify item details.

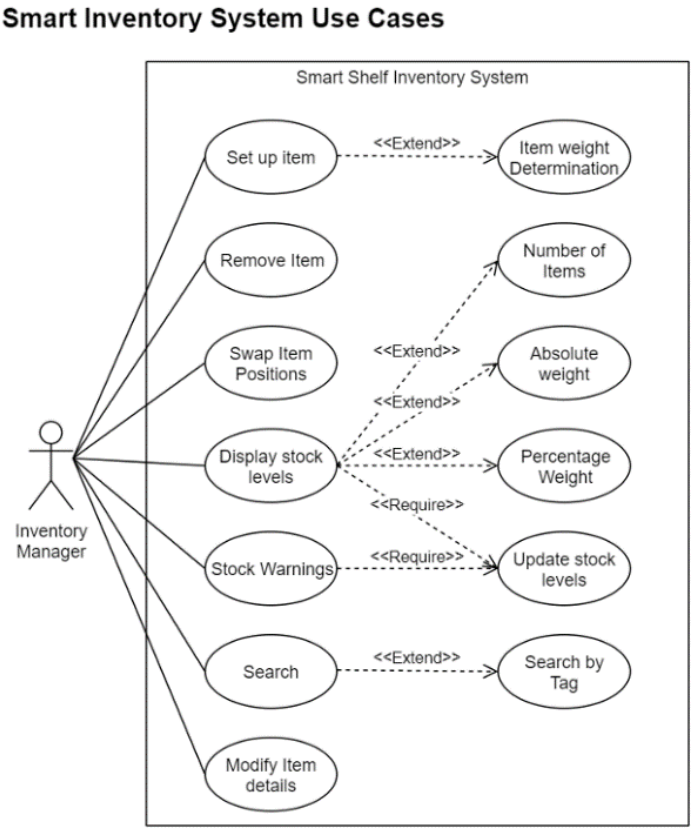


Figure 1: Minimum Viable Product specification and UML Use case diagram for proposed smart shelf inventory system showing interactions required by the system. <<Extend>> items are non-requisite extended functions. <<Require>> items are functionalities required for the linked use case to function.

Outside of these functional requirements, the main design requirement expressed by stakeholders was a desire for a desktop application. Managers primarily saw themselves using the shelf's application as a tool and reference while ordering and re-supplying inventory items at the computer.

W adapted our plans to build a mobile application and outlined a plan to build a java desktop application in the project proposal.

During the winter break having more time to reflect we decided that a web-application accessed through the browser would meet the user's needs better. A web application would allow users to check inventory levels on any computer, without the need to install a program and would ensure the user always had access to the most up to date version of the software.

A locally run desktop application, less open to attack and with higher guarantee of uptime, might be desired by large industry however for our target demographic of small to medium sized labs and businesses the convenience of a web based application would likely outweigh that risk.

Initial Design

After considering the MVP requirements, as well as the need to support our planned Arduino based shelf, we decided to build a Node.js, express web application fed by a MySQL Database.

These technologies were chosen for the following reasons:

Node.js and Express

- Node.js's asynchronous operation makes it good at handling 'Input Output' (IO) intensive operations. At the initial phase of development, we did not know if the Arduino would be able to interact directly with the database or whether requests would need to be processed by the server before being passed to the database. If the requests needed processing, stress would be added to the server, potentially with many requests from the shelf coming in every second. For this reason, Node, able to handle IO intensive workloads, seemed a good option.
- The 'Node Package manager' contains a large library of useful modules that can be used in projects, reducing the need to build things from the ground up. These packages facilitate rapid deployment, allowing developers to focus on building the functionality stakeholders requested without needing to build supporting frameworks. For example, the 'Express' and the 'MySQL' node modules greatly simplify the processes of handling requests and interacting with the database, saving a lot development time.
- Node is a JavaScript runtime. When used, the same language is used both on the front end and back-end. This allows for easier code re-use and makes it easier for developers, as they do not have to familiarize themselves with multiple languages, while working on the code.
- Express is a minimal and flexible server framework for Node.js. It is provenly robust and vastly simplifies the process of handling requests, creating routes, and rendering views in a web application. Express was chosen over other server frameworks because it provided all the features needed for a basic web app and its popularity means that there is extensive documentation and troubleshooting advice available.

MySQL

- We knew all the data we would be working with would be structured and that this data would need to be linked via relations. Relations can be modelled in NoSQL databases; however, the set-up process is difficult, and they run slower.
- We had examples proving the ability of an Arduino being able to communicate with a MySQL. The creation of a physical shelf that would work with the database was one of the biggest unknowns, so it was decided to stick with a MySQL solution to not potentially add unneeded stress.

Details of Implementation

During planning we decided it was realistically only be feasible to create and test one functioning shelf using an Arduino and pressure sensor during the development period.

The other shelves would be modelled as 'virtual shelves'. To remove complication, we decided to model a fixed number of shelves. Six was arbitrarily decided on as a good number of shelves to model.

Database Implementation

The functionality of a web application is dependent on the structure of the database or data structure supporting it.

Before creating the database, the functionality of the shelf needed to be modelled as a set of entities, relationships, and attributes. The properties of entities needed thought and the relationships required between these entities needed consideration.

The UML sequence and flow diagrams from our proposal proved valuable references, aiding the identification of attributes that would need to be modelled to ensure functionality (example diagram can be found figure 4 page 7).

Due to the databases importance, each team member was tasked to independently propose an entity relationship diagram that would allow all the functionality in the MVP to be achieved. The reasonings and explanations behind each diagram were discussed and a database schema was created incorporating aspects of several proposals (see figure 2 below and figures 16 – 19 Appendix 2).

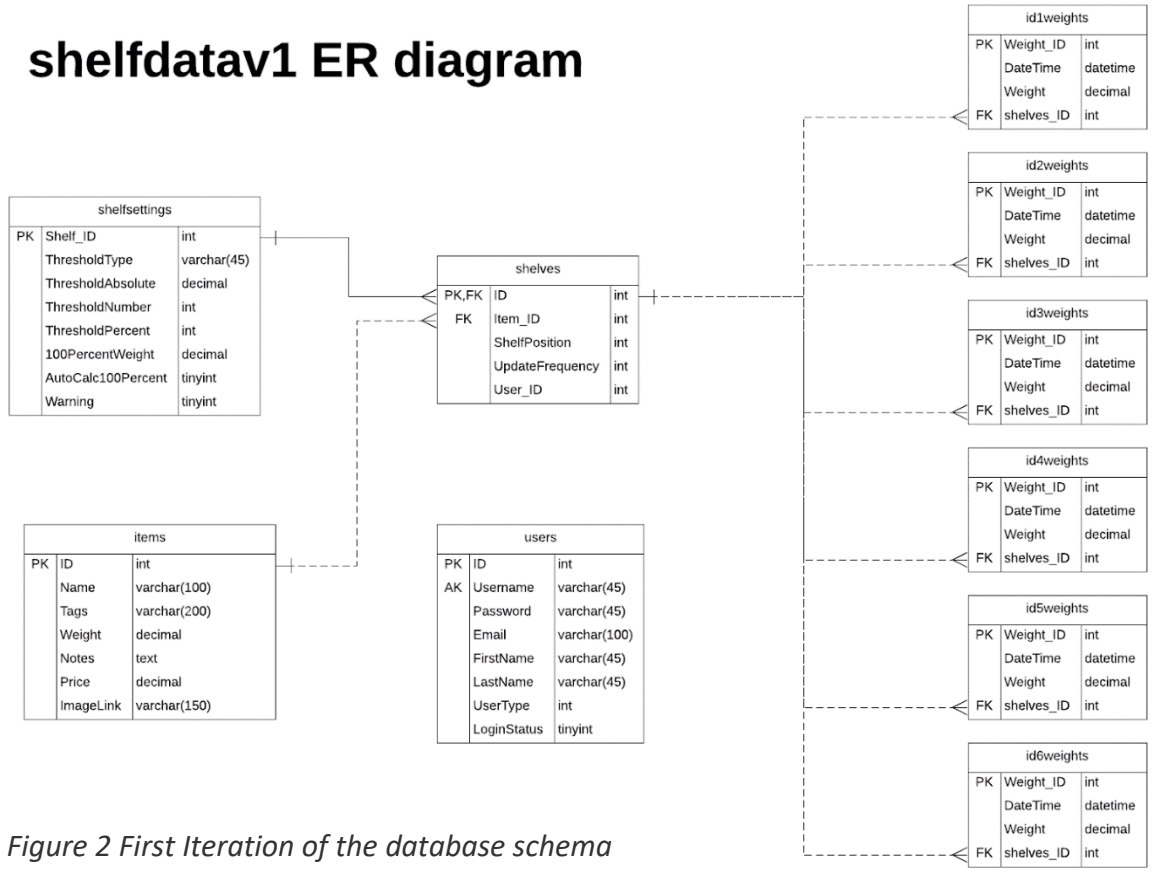


Figure 2 First Iteration of the database schema

Tables were created for each object as we saw them, primarily: shelves, users, and weight records. Shelf Settings were also initially given their own table. It was thought we may want to keep the settings constant after a shelf is cleared, however this was found not to be needed and later changed. Separate weight tables were given to each shelf because at this point, we did not know if the connected Arduino shelf would require different settings.

The database went through several minor revisions, however remained mostly consistent same with few major changes made. Version 2 merged the shelves and ‘shelfsettings’ tables into one. Version 3 replaced the three separate fields for holding warning threshold values and combined them into a single field 'thresholdValue'. This was done because only one warning threshold type could be selected at a time. We found instead that it was possible to prompt the user to enter a new value if the threshold type was changed. (see figure 3 below)

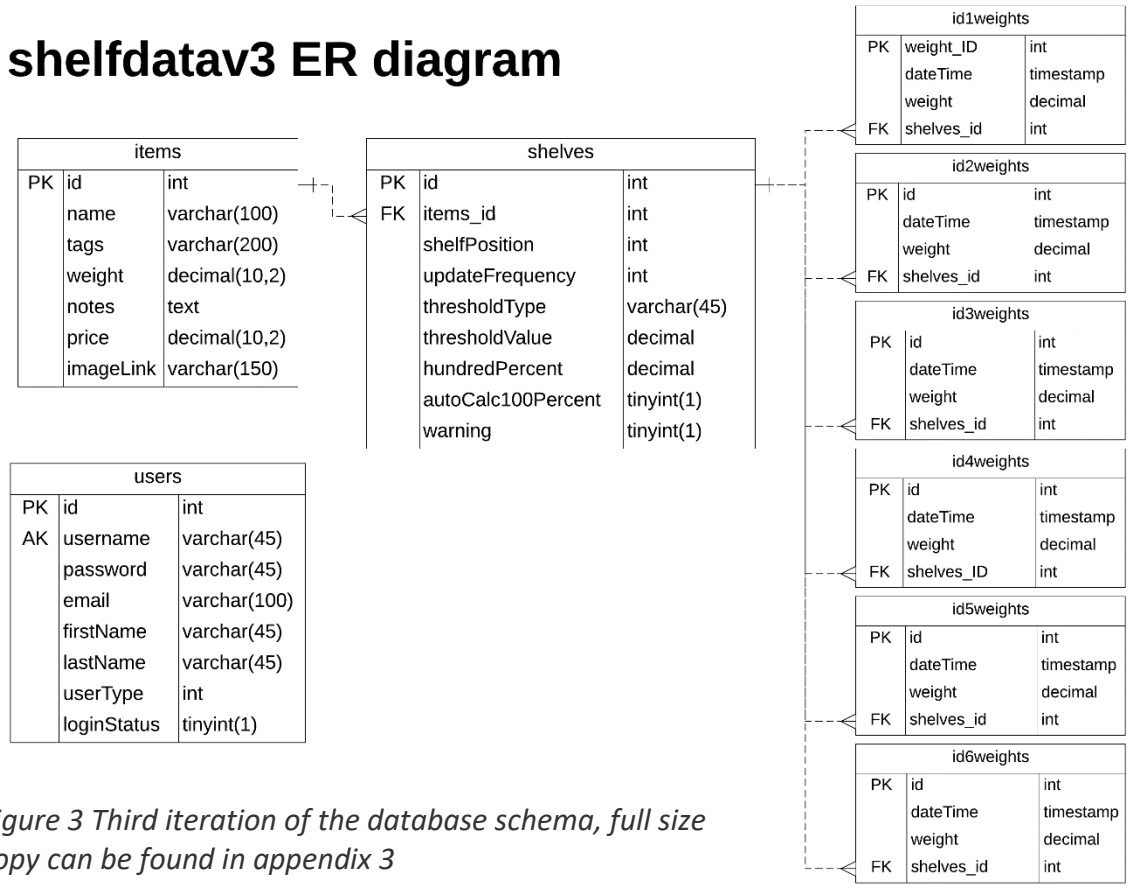


Figure 3 Third iteration of the database schema, full size copy can be found in appendix 3

Middleware Implementation

Generally, a Model View Controller set up was used for the web application, separating concerns, and allowing for greater re-use of code. Models were created to interact with the database. Calls related to different tables (or functions) were separated into different files and object constructors for different tables were created.

Controllers in the form of routing files were used. A separate route file was used for each of the sites major functions, with all paths in that file sharing a common domain path.

The EJS (Extended Java Script) templating engine was used to render views, allowing for dynamic content to be passed to pages and simple calculations to be performed. EJS was chosen primarily because it is closest to standard HTML. The similarity meant that if styling work were done independently on an HTML template page, the process of implementing those elements and styling would be relatively pain free. The 'includes' EJS feature for storing common page elements in separate files was also useful. The navigation, header, footer and a frequently used shelf selector form were all stored in this manner.

Building routes started off as a difficult and time-consuming job but got easier over time, both with experience and as the models got built up. Towards the end of development new routes could be generated quickly, often requiring only database calls already defined and tested in the Models.

A site map proposing the routes required to build the site with the most important functionality was created (see figure 5 on the next page).

As development went on and problems broken down and considered more deeply the size of the site and number of routes required increased (see figure 6 overleaf).

Routes and functions were constructed with reference to the UML sequence and flow diagrams from our proposal. The UML diagrams were an especially useful reference for the longer paths such as the item setup process, giving a clear idea of the decisions and actions to present to the user at each step (see figure x below).

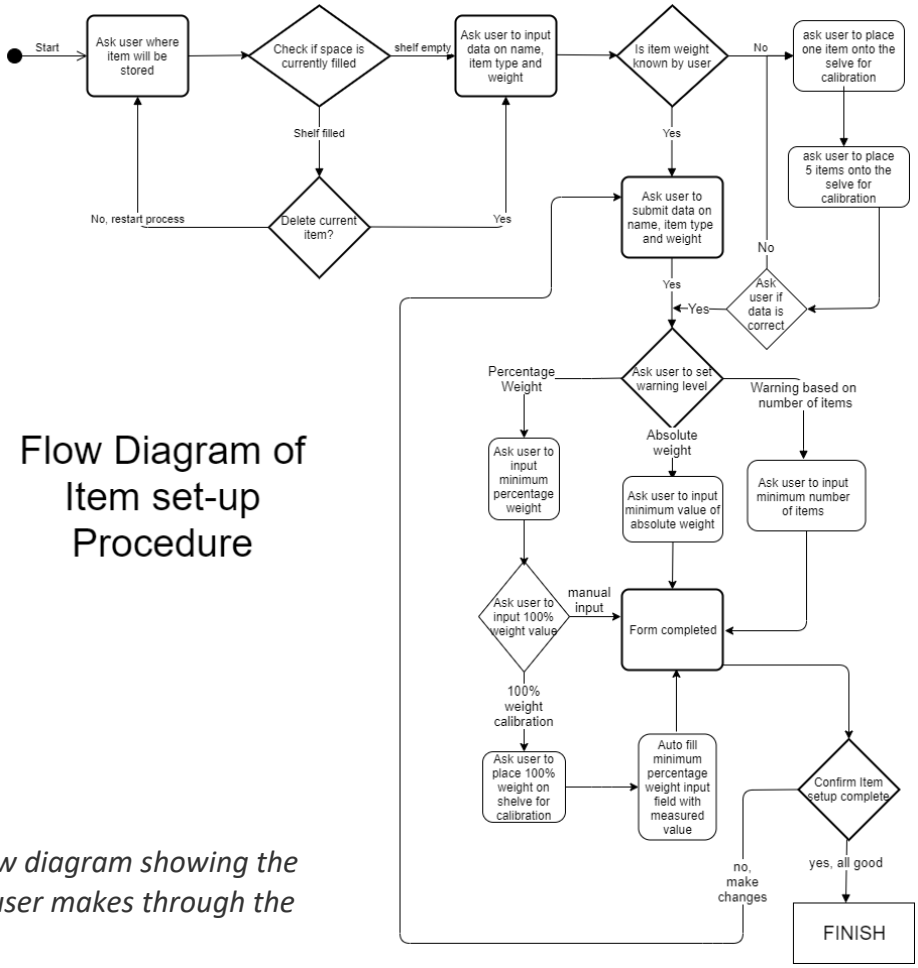


Figure 4 The UML flow diagram showing the series of decisions a user makes through the item setup process

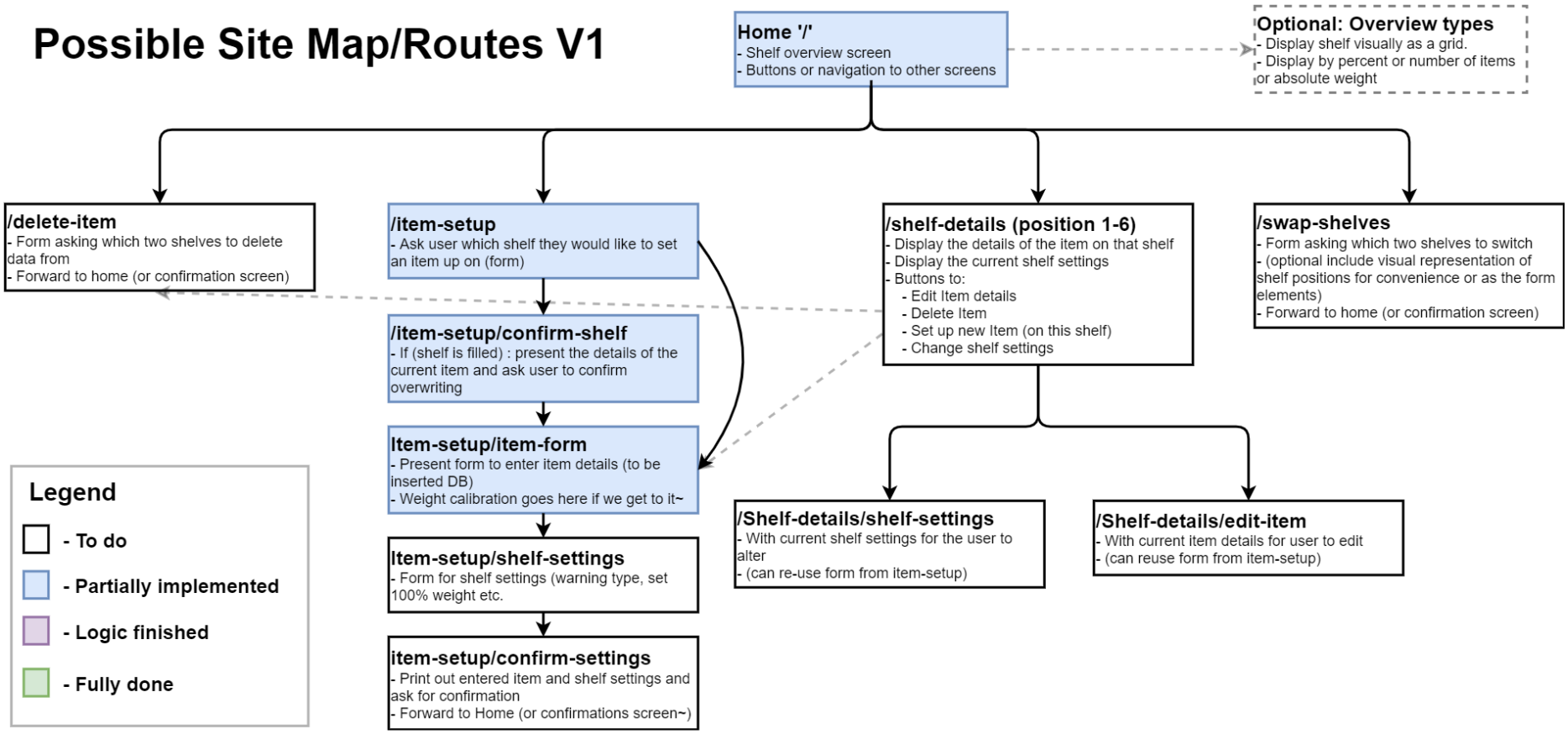


Figure 5: The proposed site map and set of routes thought to be required to achieve functionality early in development

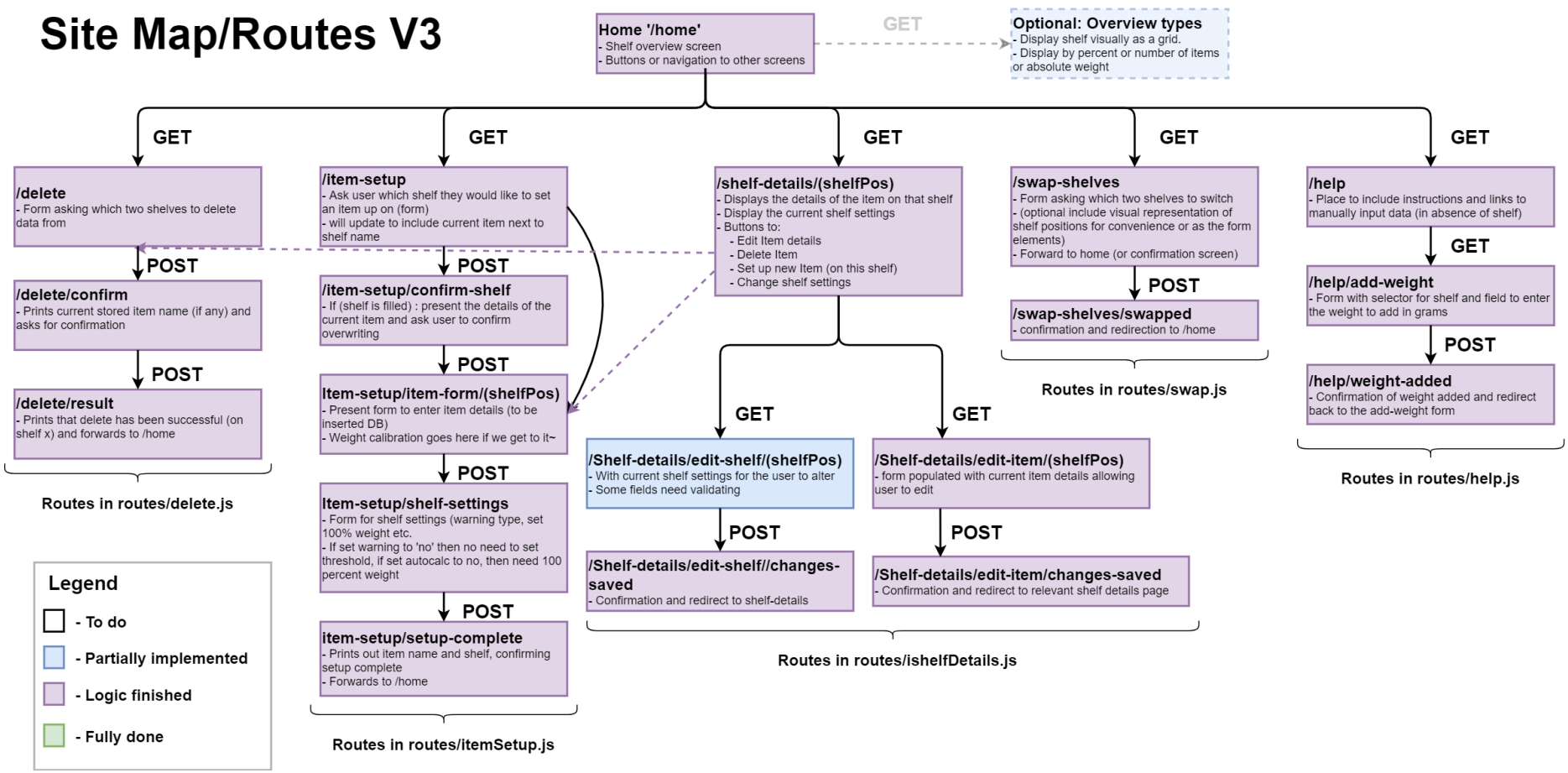


Figure 6: The site map and set of routes developed towards the end of development. Routes are still marked as 'logic finished' as there may have been small edits to UI and helper text still to do and some testing had not finished

User Interface Implementation

The site styling and UI went through rocky development. A temporary UI was used through the development process and was eventually adapted to the current styling (see figure x). A UI overhaul was planned, but due to unforeseen circumstances the changes got delayed and were not quite ready when development halted.

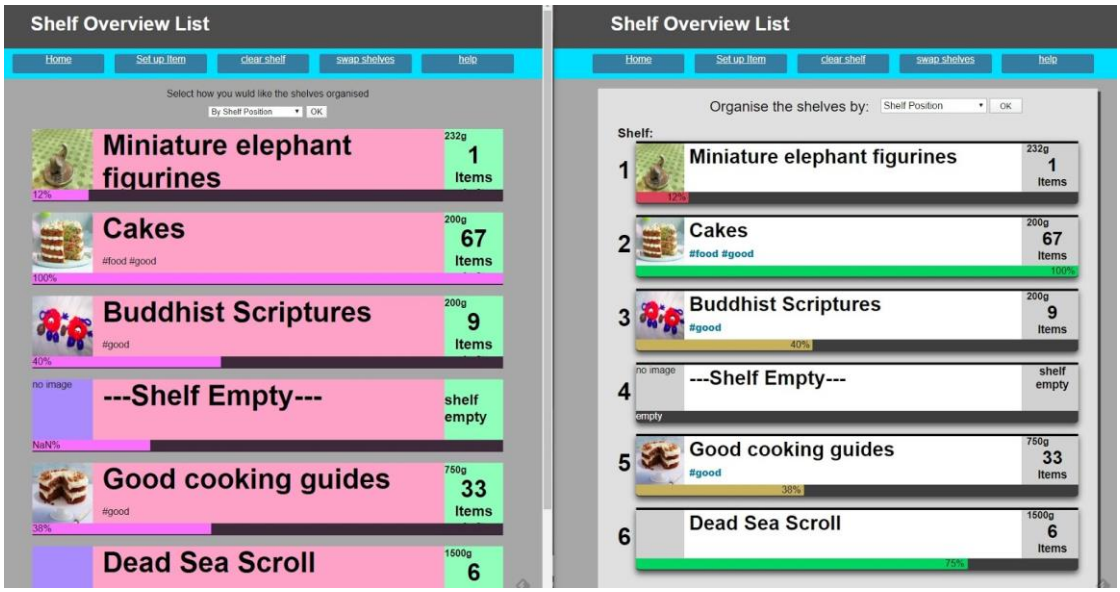


Figure 7: The main overview page immediately before and after updating the UI styling

In the early stages of development mock-ups were created for a user interface, to plan out the site and present to stakeholders.

Later in development when the functionality of the site was being built, a website containing a cleaner, more modern UI was created using the bootstrap framework (See figure 8 below, more pictures in appendix 3).

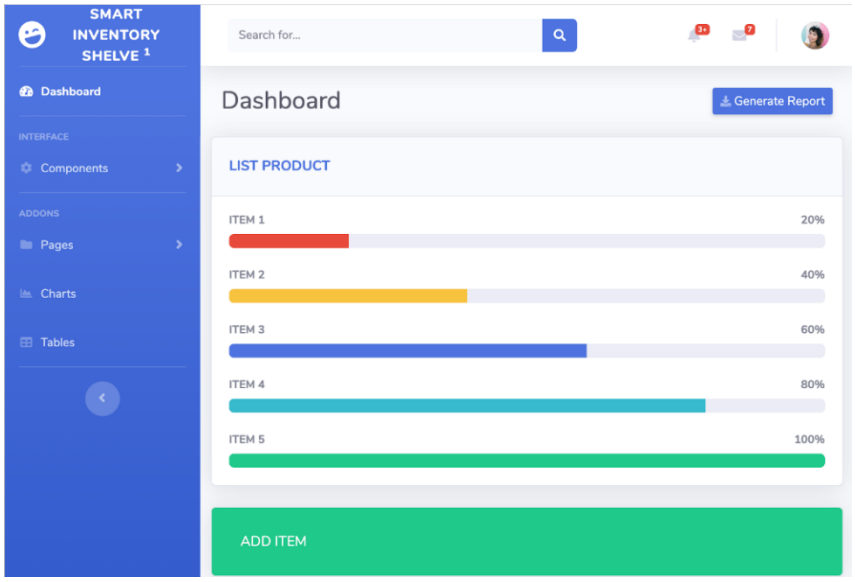


Figure 8: Proposed homepage developed as part of the planned UI overhaul containing more modern UI

The updated UI gives insight into how the site might be made to look if development continued. It also contains space for some features that were not included in the MVP, but could be useful to inventory managers, such as direct access to the database as well as charts and analytics that could show item usage over time.

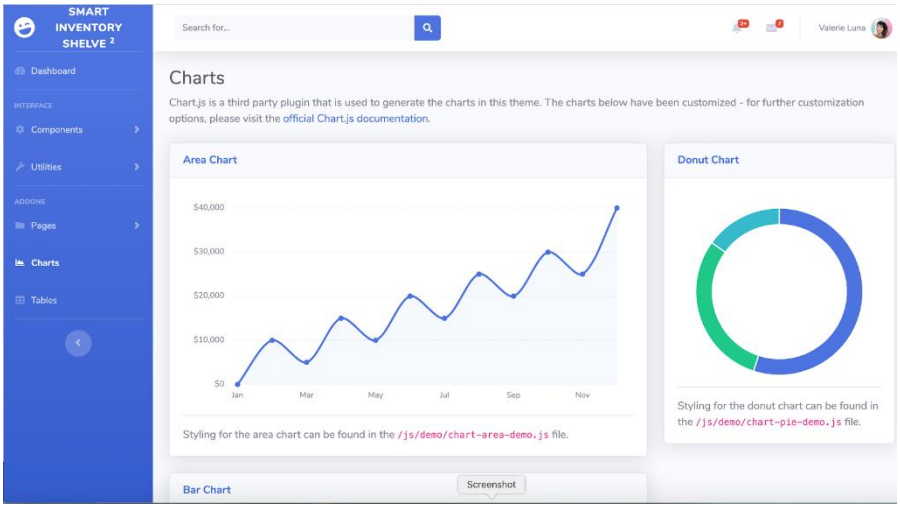


Figure 9: Chart page in planned UI overhaul that could plot a shelves stock level history

Implementing this updated UI to work with the site's dynamic data should not be too difficult and would be one of the next steps taken if we continued development.

The current site's UI was built to be simple and functional with a basic navigation bar used to navigate the site and display the site's functionalities.

The overview and shelf details pages were given more attention than the rest of the site as they are important elements and it was thought they might carry or be adapted to work with the new UI.

SASS and CSS were used to build the layout for the site. Flexboxes were used for the general page layout and CSS Grid was used for the information boxes on the overview and shelf details pages.

The site tried to use a consistent colour scheme and layout with large font sizes and good contrast to make the site easy to understand.

EJS 'includes' files were used to template the parts of the site common to multiple pages. This made including the consistent navigation bar easy to implement on all pages, including the 404 not found page (see figure 10 below).

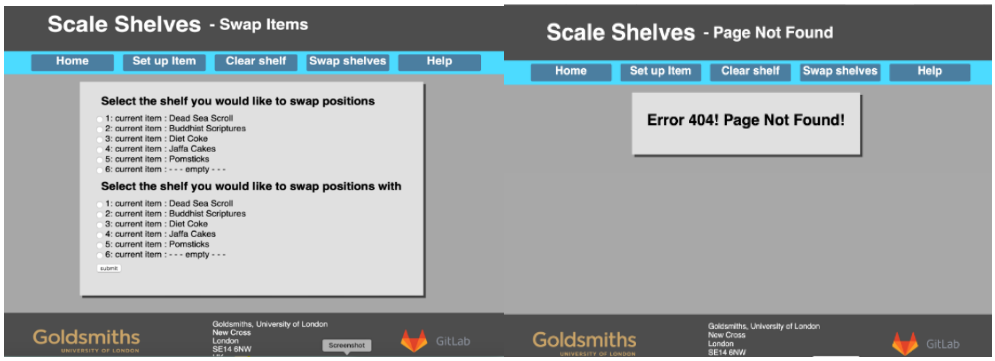


Figure 10: Two representative site pages showing the consistent UI even on the 404 page

Reflections on Implementation

Overall development went well. The main systems of a smart shelf inventory system have been implemented and tested, working robustly with the Arduino shelf connected feeding live data.

Live, stock information is displayed well and is sortable. Items can be set up, edited, and deleted easily. Shelf positions can be swapped and an ‘autocalculate 100%’ function was implemented which monitors the stock history and selects the most recent highest weight value to serve as the 100% weight (if this option is selected).

We believe these features in combination produce a compelling demonstration of our shelf’s feasibility.

The user interface still needs work and two items from the MVP were not implemented, namely a warning system and search. Search was deemed unnecessary for our demonstration of 6 shelves. The warning system was important but ran into difficulties, the groundwork has been laid for another attempt. Both features should not be difficult to implement now if development were to continue.

Things to do differently

Over the course of development lessons were learnt and problems with our original design cropped up. Most were not major, but some do limit the ability of the current system to be expanded and would need reworking if development continued.

Some of the technical and design choices we would do differently if starting again are:

- Use a single weights table for all shelves. The current database schema has a hard limit of 6 shelves. This could work if shelves were sold in units of 6 but does not account for if a user wants to keep track of two sets of shelves. In theory the current system could be extended with scripts issuing commands to create new weight tables for shelves as they are set up. Storing all weight records in a single table might be a better solution as long as it does not impact performance when pulling records too greatly.
- SQL - like databases work well for this application, the problem of multiple users requires consideration though. Our system is currently set up for a single shelf. The database would need to be modified with a greater emphasis on user ownership of shelves and records in order to facilitate multiple users with multiple sets of shelves or each user would need to be given a separate shelf database or set of tables.
- We generated SQL queries to create tables and interact with the database manually. This worked fine for a project of this size, however it may have saved time and made the process easier if we used use an API and tool set such as ‘sequelize’ to manage models and generate statements.

- At certain points in code the number of shelves is assumed to be six and checked. These statements would need to be replaced with checks for the number of shelves the system currently has if the system is to be expanded.
- The current UI is not responsive only currently looking as intended on devices >960px. The current UI extended from a temporary one used in development. We considered making the UI responsive towards the end but found it would require a complete re-work of the site's flexbox structure. It might have been better to adopt a 'mobile first' UI implementation strategy where the UI was built for mobile first and desktop stylings were added on top.

Development Record

Overview of Development Process

We began development with two problems that prevented the full adoption of agile methods. First each member's schedule varied massively, leaving only Monday and Friday available for meetings. Second despite having done research over the winter period none of us individually knew how to start building a web application.

Though not ideal we resolved to have Monday and Friday meetings and increase communication on the group chat.

Waiting before we understood all aspects of the shelf and application before starting would have taken too long.

We decided, for the first two sprints, members would choose different areas of the application and shelf to research and study the implementation. Those members would be primarily responsible for that area going forward and share what they had been working on and their findings in group meetings and help to break down the tasks and steps necessary in their area.

Deciding on a database structure was an important task that affected how the rest of the application could be built. Due to this importance it was decided everyone should engage in the process of designing the schema and each propose a database structure ERD (figures 16 – 19 Appendix 2).

By the end of the fourth week the applications structure and first database version had been discussed, generated, and uploaded on to GitLab. A summary of how each functional component might be implemented using the database had also been uploaded to the wiki and the different tasks associated put on the Trello board (pictures of the Trello figures 27 - 28 appendix 4).

Following week 4 a more conventional sprint structure was followed, but on a weekly rather than bi-weekly basis. Members would take on tasks, talk about progress in meetings, using the Trello board used to keep track of the development stage of each task.

Due to members schedules most development work occurred on weekends. It made more sense to set targets on a weekly basis Friday to Friday, with progress updates on Monday, after the weekend, and a re-evaluation of targets and progress during Friday meetings.

Tasks were prioritized based on a combination of how essential the function was and the difficulty of the task. The shelf overview page and item setup process were both difficult tasks, but also essential, so development started on them early.

Tasks were not assigned; members chose what they wanted to work on. Prioritized tasks were suggested, but if a member displayed interest in an area, they were able to pursue it.

Development of the physical Arduino based shelf was taken on by Marisa, the team was updated with the progress and the setbacks encountered. Help was offered if needed, but that proved unnecessary.

Towards the last few weeks of development priorities switched from implementing more features to testing the currently built functions and validating them with users. Eventually it was decided to fully halt development of new features and alterations to ensure there was adequate time to finish the validation and testing of the features already completed.

What worked well

Information on which tasks needed doing and who was working on tasks was maintained through use of Trello, meetings, and communication over WhatsApp. This communication was generally successful. At no time did two members end up working on the same task, doubling effort, and no difficult to solve merge errors occurred through people working on the same lines of code.

Git was successfully used to simplify the development process. Branches were used allowing members to work on their own versions of the repository and merge in their changes only once satisfied. Git and branches allowed members to work without worry of overwriting others changes or pushing unstable code that could break the app for others (see figure 11 below).

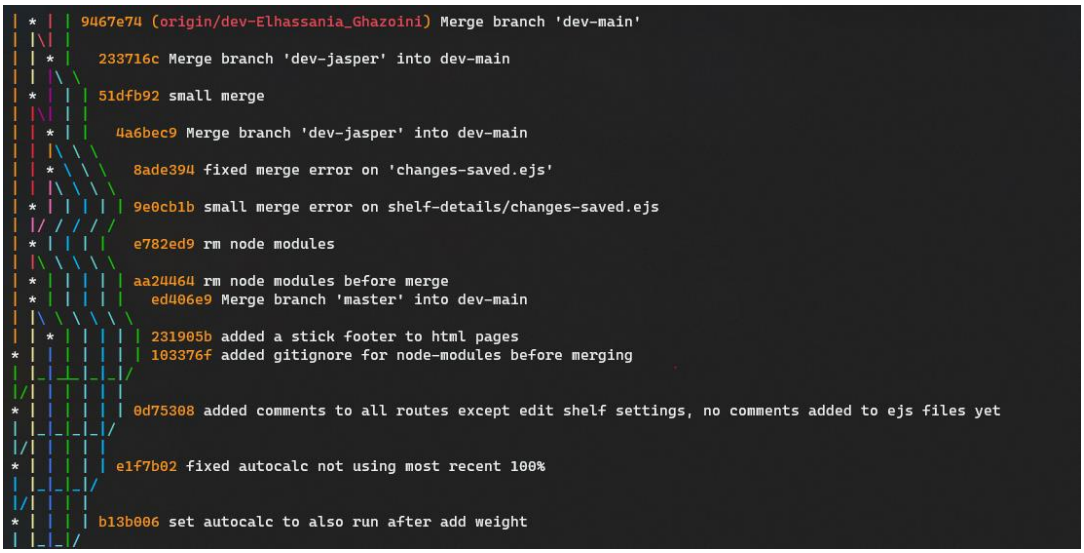


Figure 11, section of the git history, showing commits on multiple branches and merges.

A site map showing the progress of routes and functions that needed building was made. The map helped display the current status of the project in a more visual way to the Trello board, helping show the current status of the site and what could be worked on (see figures 5 and 6 on pages 8-9).

The GitLab wiki again proved useful as a place to store information and instructions. It was used to store implementation plans for functionality, explanations of the project structure as well as information on how to get the project running locally and interact with the web server. (see figure 12 on the following page).

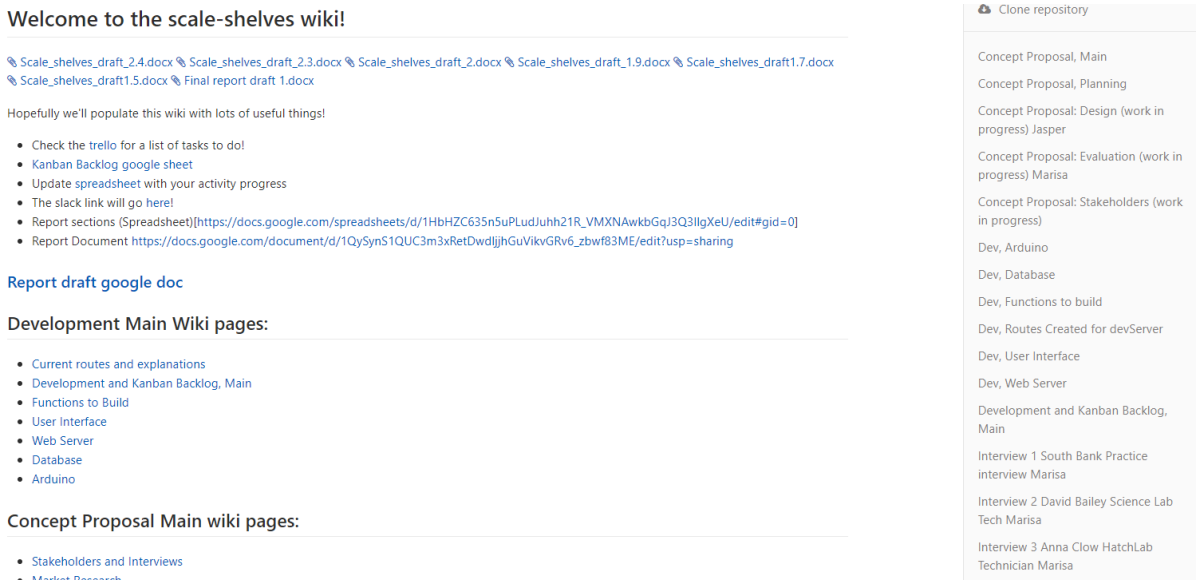


Figure 12 screenshot of the GitLab wiki used throughout the project duration, it currently has over 50 pages (link in appendix 4)

All members contributed something towards the final application and the majority of planned functionality was implemented. The Arduino based shelf, which was a big unknown, works and successfully interacts with the application and database.

What could have been done better

With differing schedules and few times to meet a week, updates on progress and some solvable issues were sometimes not heard for a while.

We generally felt that face to face meetings were better and strived to find times when all members would be available. On reflection it may have been better to have more frequent meetings with any members who could attend that day, or to embrace online meetings via Microsoft teams or some other conferencing software.

There was little getting around the difficulties of learning how to build a web application at the same time as doing so. Each member learnt from different sources and so ended up utilising different techniques writing their code. An example being the two methods of calling the database used in the routing logic. One method uses calls defined in models that generate promises (using the sql2 package) and the other uses database calls embedded directly in routes using callbacks (using the sql package). Both methods were facilitated and worked fine separately and in combination, however the style and structure of the promise code was unfamiliar and hard to follow for some. This made it difficult for members to adapt and extend existing code. This was eventually clarified but explaining code should have been given greater precedence at team meetings and perhaps been brought up as a repeated point of discussion at the beginning of each meeting.

Quality Assurance

Unit and Integration Testing

A quality assurance and control approach was employed throughout the development of the web application, database and Arduino scale shelf components of the Smart Shelf in order to prevent and detect errors early on. This is desirable because it provides greater confidence in the quality of the product components created at each stage, which are likely be reused or called upon in components developed later on. Also it hopefully reduces the number of errors found later on in the development process, which can be expensive in terms of time and effort taken to understand and resolve them.

Unit and Integration Testing: Web application and Database

Quality assurance and control during development was initiated by the following of a Model View Controller (MVC) approach. This entailed creating a model for each table in the database and then outlining and storing in the model the database calls and functions required in the application layer to create, update, read and delete objects in a particular table. This approach allowed for the calls and functions to created and tested before they were implemented in application layer routes and provided confidence in their quality if they needed to be reused elsewhere (see figure 13 below).

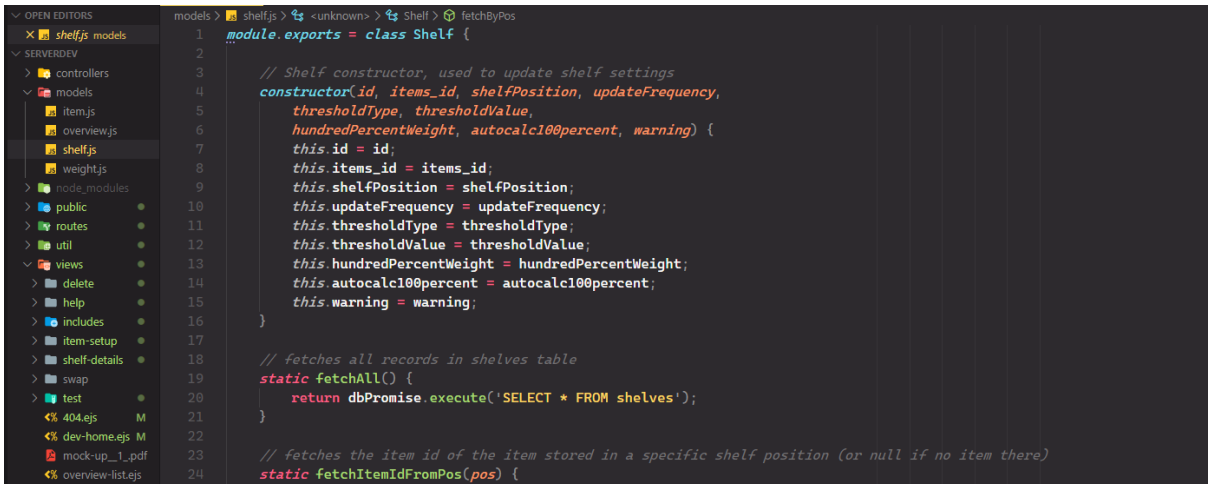


Figure 13: The top section of the Shelf Model, showing the constructor and first defined method.

Before creating a route the database model was consulted and the desired outcome considered. After considering what should happen, database calls and functions would be written in the models to achieve those changes. The newly generated functions were then each run in a test route with valid data, where the results where checked using console.log output as well as directly in the database in MySQL Workbench.

Once it was established that the model functions worked as expected for valid data inputs, the rest of the route was built up, ending with all the desired data available to be rendered in a response. After implementing each database call in the route, the route would be tested with console.log and res.send to check that behaviour and output was as expected. When the requirement for the route logic to function

correctly with valid input was satisfied, an EJS template was created to check that the route would accept valid inputs and then display the desired information correctly without crashing.

Next invalid inputs were considered such as shelf values outside of expected ranges, retrieving information from empty shelves and inputting the wrong data type into form fields. For example, wherever shelf selectors were used or the shelf position was inserted into the URL as a parameter in a GET request, code was added to check the shelf value was in the correct range of 1-6 before a database call was performed. Blank spaces in numerical fields were dealt with by converting the input to a null value.

After a route (or set of routes) was completed, a final 'real world' test was conducted on it by running through it a few times with valid input data and then running through it with invalid input data or closing forms at various stages to see if it would break. Due to time constraints, the testing at development time was not exhaustive but it ensured the route worked when using valid data and also addressed some of the more obvious problems that could arise.

Example list of tests and investigations conducted on components of the system:

Component: Querying a shelf or item in the database.
Test: Investigated best way to query a shelf or item in the database
Observations: A shelf position can change and more than one item with the same name could exist.
Actions: Concluded that where possible it is best to query by shelf_id and item_id rather than shelf position or item name.

Component: Swap Shelves
Test: Swapping empty shelves with each other
Observations: No problems
Actions: n/a
Test: Swapping shelves with themselves:
Observations: This caused issues.
Actions: Check added to make sure the two shelves selected are different and within range 1-6.

Component: Delete
Test: Deleting empty shelves
Observations: It was possible to do this
Actions: Check added to prevent this from occurring.
Test: That no data is retained when setting up an item on a shelf that has been emptied.
Actions: Ensured that all weight data is also cleared for an emptied shelf.
Test: An emptied shelf will retain it's shelf position
Actions: Prevented loss of shelf position or id by not actually deleting the record in the 'shelves' table, only resetting everything apart from those two fields to default.

Component: Item setup:
Test: Incorrect input into form
Actions: Validation put in place to prevent this
Test: Exiting the form before finishing
Observations: If quit before entering shelf settings, an item record is created but is not placed on a shelf.

Test: setting up two items with the same name, or same name as a previously set up item
Observations: Initially got data only for the first item set up
Actions: Database call changed to select the most recent record of that name during the item setup process.
Test: Uploading images that are too large or files that were not images
Actions: Created a route returning user to the previous page without losing data in item setup

Component: Item setup with shelf
Test: That data was successfully passed through several routes back to a form (that looks the same) without being lost
Action: Acheived using JSON.stringify and JSON.parse. An alternate and possibly easier way would have been to insert the current data, then modify it after the process (as in edit item).
Test: Setting up shelf where no new weight data is being received
Actions: Check added to detect lack of new data and prevent proceeding

Component: Shelf Overview
Test: That correct data was pulled with information (or lack of) attained both for empty and filled shelves.
Observations: Original 'items LEFT JOIN shelves' statement used was wrong.
Actions: Corrected to become 'items RIGHT JOIN shelves'
Test: Make sure that shelf and weights records matched from the database.
Observations: Both are originally called in order of shelf id not position. They are then pushed to a 2D array and sorted. Each of the sorts for the different overview preferences were tested with dummy data and expected outcomes. Found that empty shelves appeared in random positions after sorting.
Actions: Checks placed for NaN in the sort logic.

Component: overview-list.ejs
Test: rendering information on empty shelves, shelves and shelves with no weight records
Observations: 'infinity%' displayed when attempting to divide by 0 or NaN. Same behaviour occurred when creating the status bar, which shows shelf filled percentage.
Actions: Displayed 'shelf empty' or 'no weight records' instead, or placed a check to cancel the calculation if it broke the page. Also added a check for the status bar so it would display 'empty' instead.

Component: Shelf Details
Test: Representation of shelves outside valid range and representation of empty shelves
Actions: Redirect to 404 for former and to item set up for latter.
Test: Check shelf details for a shelf with no weight records
Actions: Placed a check and assigned the weight to null if no weight record found. Rendered to 'no weight records available' if null.
Test: Check that all the information on the item and shelf settings are rendered on the page
Observation: All information is rendered but text overflows may occur in layout if text is too long
Action: Further work required on CSS to resolve this.

Component: Auto calculate 100% Weight
Observations: Originally it found the highest weight and set that to 100%
Actions: Changed the behaviour to set the most recent highest weight to 100%
Test: Check with sets of data with a dip, then rise then dip (e.g. for '1000, 500, 300, 600, 400') the new 100% weight should then be 600
Notes: Did not test with the shelf that may have 'shaky' data slightly up or down which could mess with the function, it probably needs a tolerance of 5g or so introduced.

Unit and Integration Testing: Scale Shelf

Quality assurance and control practices were adhered to throughout the development of the Arduino Scale hardware and associated code.

Integration of Load cell, HX711 and jumper wires:

To ensure best practices were followed, the Arduino scale developer completed a Department of Computing Hatch Labs online soldering and electronics workbench induction before beginning the hardware construction phase. Online guides for constructing Arduino weighing scales and soldering them to a HX711 load cell amplifier were also consulted beforehand. Continuity testing, using a multimeter was carried out after soldering was completed on each pin of the HX711, each wire of the load cell bar and each jumper wire to ensure that each connection was reliable before moving on to the next soldering task.

Integration of Arduino, IDE, Laptop:

Both Arduino models used during development were tested in a number of ways prior to connecting them to the other parts of the scale. The first test was to ensure that the Arduino was able to communicate with the Arduino IDE via a USB connection to the laptop and that it was able to upload and execute the standard blink test sketch available in the IDE. The next test checked the connection between the Arduino and the IDE via the laptop serial port. This was achieved by uploading and executing a sketch to the Arduino that would connect to the serial port and then use it to output data in the IDE serial monitor tool on the laptop.

Integration of Load Cell, HX711 Arduino, IDE, Laptop:

Once it was confirmed that the Arduino was operating as expected, the previously soldered together load cell bar, HX711 and jumper wires were connected to the Arduino as directed in the online guides consulted and then tested with the example HX711.ino sketch provided in the online Instructables tutorial on how to build Arduino weighing scales. This test confirmed that the connected components were sending reliable weight data to the Arduino which then processed and outputted the data in the serial monitor tool of the IDE. The load cell was now attached to the scale platform and base. Once this was completed, the weight readings test described above was repeated to check that data sent was still reliable.

Scale Calibration:

So far, the weight readings sent to the Arduino had been using the pre-set calibration value in the example HX711.ino sketch. This value was designed for a load cell bar capable of weighing up to 1 kg. Therefore, a correct calibration value for a 5kg load cell bar was calculated using the calibration process described in the Instructables guide (see Appendix 1: Shelf Construction). Various items with known weights were then placed on the scale platform and the subsequent weight readings received in the serial monitor checked for accuracy. Once the scale bin had been attached to the scale platform, the calibration process was repeated to ensure that scale performance was not affected.

Network connectivity:

The ability of the Arduino to connect to a Wi-Fi network was initially tested by writing an Arduino sketch which would attempt to connect to a home Wi-Fi network and then output information such as the IP address obtained or an error message into the IDE

serial monitor in order to signify success or failure. When this initial test was passed, it was repeated with the eduroam network, the Goldsmiths guest network and the Department of Computing Hatch Labs G11 network. These further tests revealed that only the G11 network would be accessible on campus.

Integration of Arduino scale, MySQL Connector/Arduino library and MySQL Server database:

A test user account and test database were first created on the MySQL server that had been set up on the developer's laptop. A basic test of the user account's direct access to the test database on the MySQL server was then conducted both on the developer's laptop and from a different laptop on the same home network. This was followed by a test that used a SQL INSERT statement to directly insert weight type data (float type) into the test table created in the test database to ensure the data was inserted and represented correctly. Both of these tests were successful. Next the ability of the Arduino to connect to the MySQL server was tested. This was achieved by using a modified version of the Sample Connection Test - Wi-Fi sketch (Listing 2) from the MySQL Connector/Arduino User Guide by Dr Charles Bell. This test initially failed at the at the MySQL Server connection stage but after some online research and subsequent modifications to the sketch and MySQL Connector/Arduino library files, it ran successfully. Finally tests were run to check that the Arduino could use the MySQL Connector Arduino library to connect to and send an INSERT statement to the MySQL Server in order to insert data into a table in the test database. Again, two example sketches from the MySQL Connector/Arduino User Guide were adapted for these tests. The first, Listing 3: Simple Data Insert Sketch allowed for testing the use of a static query string to insert either integer or float data into a table. The second, Listing 4: Complex Insert Sketch allowed for testing the use of a query string that is continuously rebuilt to contain the latest data processed from a sensor. Once both of these tests were successfully passed, a first draft of an Arduino sketch that combined setting up the scale, processing weight readings connecting to Wi-Fi and the database server and inserting readings into a database table was created. This was then used to successfully test inserting actual processed weight data from the scale at regular intervals into a database table.

Integration of Scale with Application Database and Web Application (minus scale UI and code):

Now all the components of the Arduino scale had been successfully tested, it was time to test inserting the processed scale data into a replica of the application database at regular intervals. This was achieved by first recreating the application database on the developer's laptop from a database dump file. Then the draft sketch was modified to insert data into one of the replica database shelf tables (id1weights) and run while adding and removing items of known weight to and from the scale. This was a successful test of the integration of the application database and the Arduino scale. Although there was no user interface or associated code relating to the scale in the version of the web application available at this point, it was possible to test how the web application would process and display the weight data sent to the database table by running the draft sketch on the Arduino with the application database and web server also running. Checks of the user interface while adding and removing items to/from the scale bin demonstrated that the web application correctly detected, processed and displayed the changing weight data sent from the Arduino to the database.

Review, fine tune and clean-up of draft sketch:
The version of the web application and database described above were sufficient for use in reviewing, fine tuning and cleaning up the draft sketch code. Changes were made to existing code that was deemed to be redundant, code was reworked to be more efficient, sections of the sketch were moved around to create a more logical order and more comments were added. After each change the sketch was regularly rerun to test that everything was still running correctly and no new errors had been introduced. More significant changes such as introducing code at the beginning of the sketch loop to check and resolve the status of the MySQL server connection and adding a time delay to control the rate at which weight data was sent to the database were also tested in this way to check for any positive and negative consequences of the changes.

System Testing

Methodology:
Functional, black-box system and integration testing was carried out using suites of test cases, developed after reviewing the functional requirements of the Minimum Viable Product (MVP) and the user interface of the web application. The complete spreadsheet of test cases and required input data is available in the project repository (‘Report/Testing Materials/SmartShelfTesting.xlsx’) along with a folder of product images used in testing (‘Report/Testing Materials/TestImages’).

A suite of tests was developed for each page of the web application. Each suite contains a series of test cases designed to exercise specific functions on the page in question e.g. navigation bar links or a form requiring completion and submission. The test cases include a description of what the test examines, required perquisite input and actions, steps required to carry out the test, expected results and space to record actual results and the test status. This structure was intended to make the tests easy to execute by anyone, whether they had detailed knowledge of the system or not. An example test case is contained in the testing appendix (appendix 5).

Conducting the tests in test suite order allows the tester to examine both the look and feel of the application, whether data input, output and processing is functioning correctly, and navigation of the application. The modular nature of the test cases also allows for test cases from different test suites to be combined into sequences that can test an entire functional requirement such as being able to set up an item on a shelf.

Input data required for initial setup prior to testing and during testing was carefully selected both to fully exercise the system and to reflect the nature of the products that users in a laboratory setting would be likely use the system for managing. Due to the Covid-19 outbreak it was not possible to obtain the laboratory products themselves so each product is a supermarket item that can mimic the weight and usage patterns of a laboratory item. For example, a bin of small and light electronics components is represented by 500g of dry macaroni, a container of solder paste is represented by a jar of Harissa paste, small containers of liquids are represented by Yakult yoghurt drinks.

Testing Sessions:

General Test Environment:
Testing took place with a replica of the web application and database running on an Acer laptop which was running on an Ubuntu Linux 18.04 64 bit operating system. The web application and database were started with the node index.js command in a Terminal window. Once the web application was up and running, the application home page was accessed in a Firefox Browser window using the localhost loopback network interface. All shelves were first emptied by using the clear shelf facility in the user interface. Next the shelves were set up according to the data and instructions in the initial setup section of Test Input Data tab of the SmartShelfTesting spreadsheet. Once the general test environment was ready, testing could begin.

Test1:
This session was performed on the version of the web application and database that was stored in the git repository on 14/04/20. This version was very similar to the final version of the system except that it did not yet contain a means of interacting with the scale shelf from the user interface and lacked some of the application layer code required to process and facilitate those interactions. This test session therefore was used for two purposes. First to test the performance and integration of the web application and database before connecting the scale and second to ensure that the test suites and input data created were correctly designed for the final full system testing.

The general test environment was set up and then the test cases in test suites A-F of the SmartShelfTesting spreadsheet were run through in order (with the exception of test cases B5.1 -B5.5) . Actual results were recorded below the expected results for each test and at the end of each test a test status of Pass or Fail was recorded.

Test2 (Test Suites A-F, no scale):
This session was performed on the final version of the system. The scale shelf user interface had been now been added and also the underlying processing and facilitation code. The purpose of this part of the test session was to run this version of the web application and database through the same tests and in the same way as Test1 in order to compare the performance of the two versions and detect any error corrections or new errors introduced.

Test2 (Test cases B5.1 – B5.5 with scale):
This part of the session focussed on a full integration of the web application, database and Arduino scale. The general test environment was set up as above, then the Arduino scale was powered on and allowed to run through its setup process. Once the scale was ready and configured to send data to shelf 5 of the application/database, test cases B5.1 -B5.5 were conducted and actual results and test statuses recorded. These test cases are concerned with the Set-up item process using the scale to determine the item weight.

Test2(Weights and warnings, with scale):
This part of the session was also focussed on a full integration of the system but was concerned with testing the stability of the whole system, the reliability of the scale readings being sent to the web application, the ability of the web application and

database to regularly store, process and display those readings and the accuracy of the item status bar as a warning indicator.

The general test environment and scale were set up as in the previous part of Test2 with scale data able to be sent to shelf 5. The item Knorr stock cubes from the Test Input Data tab of the spreadsheet was set up following the instructions in the G_Weight Tests tab of the spreadsheet. When the shelf was set up and 100% of the item had been added to the scale shelf, the absolute weight, no of items, and percentage/colour values displayed in the item's home page overview section were recorded. A 100% weight value for the item using kitchen scales had also been recorded for comparison. Next portions of the 100% weight of the item were removed in a series of 7 steps. At each step the item(s) removed from the scale were weighed on a kitchen scale and the weight recorded for comparison. Also, the absolute weight, no of items, and percentage/colour values now displayed in the item's home page overview section were recorded. At the end of these steps the scale was empty and another 7 steps were carried out to refill the scale to 100% whilst recording the changing displayed data.

The stock cubes were tested in two ways, firstly initially stored in the shelf bin as 24 individual stock cubes which were removed or added in groups of 1-6, and secondly as 3 x packs of 8 stock cubes which were removed or added as individual packs. The Harissa paste, Yakult drink and Macaroni items from the Test Input Data tab were also tested informally in a similar way and any additional observations about the system behaviour with regard to these items was recorded.

Test Results:

(A brief summary of the results is available in appendix 5: Testing)

Test 1.1: Suitability of test suites, test cases and input data
Overall, the structure of the test suites and test cases proved to be correctly designed and suitable for use in final full system testing. Although running through the tests in test suite order only exercised each component of a page individually, the test cases and data were designed so that completion of all the test suites in this order also implicitly tested entire functional requirements such as setting up, swapping or deleting items on a shelf. A few minor tweaks were made to the test cases and input data before proceeding to Test 1.2.

Test 1.2 and Test 2.1: Test Suites A-F, no scale
As the results for these tests were very similar for both versions of the system tested, they will be discussed together here.

Test Suite A: Home Page Functionality

Test cases A1.1 – A1.7, which covered the ability to navigate the user interface using the navigation bar and other page links were all passed. Test case A3.1, which checked that shelf overview data for each filled or empty shelf was displayed correctly also passed. Test cases A2.1-A2.5 were concerned with the ability to use a drop down selector to order the display of the shelf data. The ability to order by shelf number and absolute weight were both fine and passed. The ability to order by percentage weight and absolute weight produced mixed results. When the shelves

all had identical percentage weights of 100% and some or all were empty, they organized into a seemingly random order. It appeared as though they were defaulting to being ordered by an underlying database table value rather than displayed shelf number. This was classed as a fail. When percentage weights or number of items were different then the filled shelves were correctly ordered but the empty shelves seemed randomly ordered. This was classed as a partial fail.

Test Suite B: Item Set Up Page Functionality

Test cases B1.1-1.3 covered the use of the shelf selector form and B2.1-2.3 the confirm shelf dialog. All these tests were passed. The replace/set up item form was tested in test cases B3.1-3.5 and the shelf settings form in B4.1-B4.4. These tests also all passed.

Test Suite C: Clear Shelf Page Functionality

Test cases C1.1-C1.4 covered the delete selector form. All these tests were passed. Test cases C2.1-C2.3 covered the confirm shelf dialog. These tests also passed but during the test on the final version of the system there was an instance in Test case C2.3 of the redirect to the Home page hanging after the user had confirmed deletion of a filled shelf. This test was repeated and the issue was not replicated, but it may be worth repeating the test several times more for different scenarios to see if an intermittent error is present.

Test Suite D: Swap Shelves Page Functionality

Test cases D1.1-D1.8 covered the swap items form. All tests were passed. It should be noted however that these tests only covered swapping shelves in a system where the web application and database were integrated. As there is only one physical scale shelf built at present, the Arduino scale and the web application do not yet contain testable code/functionality to communicate with each other in order to coordinate shelf swapping.

Test Suite E: Help Page Functionality

Test case E1.1 covered navigating to and displaying the Help page. This test passed. Tests E2.1-E2.11 covered use of the Add Weights page. All tests passed except some of those that tested the input of invalid data or invalid actions. It was found in Test Case E2.5 that a weight could be added to an empty shelf. An invalid action such as this needs to be blocked by the system with an error message displayed. Allowing the weight to be added is problematic because the shelf was still regarded by other parts of the system, such as the Clear Shelves functions, as being empty. Test case E2.6 revealed that it was possible to add a minus weight to an empty or filled shelf and Test case E2.7 showed that a weight above the maximum weight limit (5000g) of the scale could be added to an empty or filled shelf. These three tests were all classified as having failed. Non-numeric data input was successfully detected and blocked as shown by Test case E2.8 however. A final observation was that when the add weight function is used to add weight data to a shelf for the first time, there is a delay of a few minutes before the status bar for that shelf on the Home page updates from empty to showing a colour/% value. This may confuse the user into thinking the data has not registered correctly, so perhaps a

wait message could be added or the frequency of updating the underlying data could be increased.

Test Suite F: Shelf Details Page Functionality

Test case F1.1 covered navigating to the shelf details page for an empty shelf. This test passed. Test case F1.2 covered navigation to and display of the shelf details page for a filled shelf. All data displayed was correct apart from the price, which had been rounded up to £1 from a set up input of 1.09. This was classed as a partial fail as the price data is not currently used in any calculations. Test case F2.1 covered use of the set up a new item link. This test passed. Test cases F3.1-F3.4 covered use of the edit item details link. F3.1: It was possible to navigate to an auto filled edit item details form and the majority of the content displayed was correct, there was however the issue of the auto filled price being the rounded up version and for the older version of the system there was no display of any existing uploaded image. In F3.4 some issues also occurred when the form was edited and submitted. If the item title was edited to become longer, then it would partially obscure the tags, and completely obscure the price in the shelf details page. The rounded-up price issue was repeated even if the user had edited the auto filled price from £1 back to £1.09. For the older system version, the existing uploaded image was lost and had to be re-uploaded. Both F3.1 and F3.4 were classed as partial fail. Test cases F4.1-F4.4 covered use of the change shelf settings link. These tests all passed. Test cases F5.1-F5.3 covered use of the set up a new item on this shelf link and Test cases F6.1-F6.3 covered use of the delete items and reset settings link. All these tests were successfully passed.

Test 2.2: Test Suite B5.1-B5.5 with scale

This test suite examined the setup of an item on the system using the integrated Arduino scale shelf to obtain the weight data during set up. The process worked very smoothly and overall, this set of tests passed. The web application was able to correctly detect when the shelf was emptied and set a base value accordingly. The measured item weights were accurate and correctly displayed and the facility to reweigh the item worked well. The only caveats would be that the item image needed to be uploaded after the weight setting process had completed and the user returned to the item set up form, otherwise an existing image or newly uploaded image would be lost. Secondly it might be useful to inform the user to wait a short time for the scale readings to stabilise before clicking OK to obtain a base reading, as it was possible to obtain an incorrect base reading if clicked too soon after emptying the shelf.

Test2.3: Weights and Warnings, with scale

(See SmartShelfTesting.xlsx - G_Weights Tests tab in repo for results tables)
The formal tests with the individual and packaged stock cubes demonstrated that items were being measured accurately during initial set up, and that weight increases and decreases were detected accurately +/- a few grams by the scale (a table of data for the packaged stock cubes is contained in the results appendix 5). This data was also correctly processed by the application and reflected in the displayed absolute weight and no. of item values and in the colour and % full values of the item status bar. As the warning system by percentage weight, absolute weight

or number of items was not completed by the end of the development phase, this function could not be tested. It was pleasing therefore to find that the informal warning system provided by the item status bar was functioning correctly. A slight issue that was noticed was that when the scale was emptied, sometimes the fluctuations in the scale reading would mean a value of -2g was displayed rather than 0g. This should be resolved by for example zeroing any minus values received into the database from the scale (as long as they are within reasonably accurate range). The informal repetitions of the weight tests for macaroni and Yakult drinks produced similar behaviour to the stock cubes. The Harissa paste however threw up an issue with the measurement of a light item in heavy packaging, in this case a glass jar. The jar almost doubled the weight of the item and made all the displayed readings during the increase and decrease of the item unreliable. This suggests that a more sophisticated tare mechanism at either the scale or the application level needs to be researched and implemented if the scale is to be used with these types of items that cannot be easily decanted from their original containers. Finally, it was noted that the integrated Arduino scale, web application and database system was very stable and ran for almost three hours during this test without any loss of network connectivity, accuracy or performance.

Some additional observations during made testing

Home page:
It was noticed that the drop-down list text was partially obscured in its default position. Also, there was a misspelling of ‘absolute’ in the drop-down list.
Item Set Up Page:
There is a misspelling of the word ‘separate’ in the tag field text of the set up/replace item form.
Help Page:
It is not possible to input a decimal value using Add weights to manually add an item weight.
Shelf Details Page:
When a user chooses to set up an item on an empty shelf, they are taken to the replace items form rather than the set-up items form. There is no significant difference between the two except the title. Also, the title ‘Details and Settings’ on the shelf details page is partially obscured.
Scale Tests:
There is possibly an intermittent bug where the measured weight value is not always displayed straight away when an item is added. Also, if the Arduino scale Wi-Fi connection drops during a measurement being taken, the page will hang.
Set Up/Replace/Edit item form:
The selector for the weight field in this form increments in hundredths of a gram which makes selecting a value such as 57g very slow. It is possible however to type the weight in.
Weight values:
The tester felt that it was possible for a user to get confused between the weight set manually or with scale in the set up items form, the weight set manually in add weights function, the percent full value and the 100% weight value. It is important that any accompanying user documentation clearly explains the differences between and usage of these values. Secondly, it was noticed that if a weight was added to an item and the weight was above the 100% weight value, the status bar would go off

the screen and would display a value > 100%. This needs to be addressed by preventing values > 100% for the status bar and percent full values.

Auto Calculate:

When activated for an item, this function can be overzealous in changing the current 100% weight based on the recent weight history for an item. For example, 500g of macaroni is set up on a shelf and the 100% weight is set at 500g, when the item has been reduced to 276g for a short time, then 276g becomes that new 100% weight and the data from the status bar and no. of items value becomes unreliable as an indicator of amount of item left.

Conclusions:

Overall, the majority of test cases contained in the tests suites were passed and the results of the item set up with scale test and weights and warnings tests proved the full integrated system to be reliable, accurate and stable. Most of the failed or partially failed tests were due to fairly superficial issues that can be quickly investigated and resolved in a future development phase. For example, preventing invalid input and actions simply requires code to block these inputs and actions and/or to send warnings to the user. Correcting issues with obscured displays, shelf display order and rounded up values should also be straightforward. Issues such as the need to upload the image last on the set up/edit items page or waiting a short time before clicking to get a base reading can be resolved by giving clear direction to the user either via a redesign of the user interface or by improving product documentation. Slightly deeper issues to resolve would be the need to find a way to tare an item that cannot easily be removed from its container or packaging and reworking and clarifying the relationship between the various weight values and the auto calculate function. It would also be necessary to investigate further any possible intermittent bugs; in case they are only triggered in a particular scenario or are indicative of a deeper underlying issue in the application logic.

The system testing carried out has therefore demonstrated that our current product version has fulfilled the majority of the functional requirements of the minimum viable product and this fulfilment of requirements could be enhanced further by carrying out the recommended investigations and resolutions detailed above and by developing and carrying out tests for any functionality that was not completed during the development phase.

Formative Evaluation

Initial Approach To Stakeholders

During the early planning stages of our web app, we approached 3 different stakeholders from within various departments in the university to gather their opinions on what they thought of our initial development idea.

Demonstrating our Product

Our Stakeholders seemed to like the idea and thought it would be useful if it were put to use in the various different departments that they all worked in. 2 weeks later, we came to them with 2 different interactive prototypes that resembled our initial ideas for what our software would do. This was version 1 of our software. We initially had the mutually agreed idea that we would make an app but from this early interaction with our stakeholders, it was clear that they preferred to use it on a computer (mainly a laptop instead of a phone

During Development, One of the stakeholders, David Bailey pointed out that he would like to see an option in our smart shelves software that allows the system administrator to be able to add images to all of the items that are registered on the system making it easier for the user to be able to identify them. We thought of this logically and decided the best place to set this image for each of the items would be through our item setup form where the user adds information about each of the items on their shelves. Therefore, one of our team members did some research and figured out how to create a functionality that allows users to upload images to the server which then gets stored in an SQL database. We successfully managed to implement this feature into the final version of our web app.

Another One of our other stakeholders that we visited during development was Anna Clow from the hatch lab. She Seemed to Really like the idea and was really impressed by the prototype we presented to her. She said that she had a concern with the fact that we were using weight as a method to determine stock level. She knew a lot about the type of shelf we planned to develop a web app with and knew that it will not have accuracy when it comes to low-weight items. Also, this item is common in the hatch labs as there are a lot of small components such as solder that do not weigh much which students work with. She suggested using something like a light sensor to identify when products are removed from a shelf. With the knowledge and expertise we had in physical computing, we knew we wouldn’t be able to implement this with our given time frame and we wanted a full working product by the end of the 10-week development period we were given.

Summary of Our Interaction

As the weeks progressed, we went on to develop our product keeping in mind some of the improvements that had been suggested to us. Obviously, we couldn’t implement some of these but we implemented most of them as well as all of the key features of our product that were essential to it in performing it's designed task. We decided to stick with a shelf which uses weight to determine stock as initially planned instead of the suggested alternatives and managed to successfully make a web app which runs on the pc platform.

We have tested the product thoroughly for bugs and glitches and after fixes, we are pleased to say that all of the features we initially planned to include all work really

well and do exactly what they are supposed to. By the end of the 10 week development period we ended up with a fully working product which we are proud of and are sure that our stakeholders will be impressed with.

Showing Our Final Product

Most of the stakeholders we had consulted with worked in some facility at the university. By the time development was far enough that presenting and gathering feedback would be constructive the strikes were occurring, which filled up the schedules stakeholders we had previously contacted with a backlog of work. This followed shortly by the lockdown made validation with our target stakeholders difficult to obtain

Unfortunately, we didn't really show our product to many people whilst it was in development and because of the sudden emergence of the coronavirus pandemic and with lock-down procedures in place, we can't show many people our final product either. However, I have managed to demonstrate our final product to my older brother who lives with me as this was the only option available given the circumstances. Overall, his opinion on what we have made was good and there are only a few minor suggestions that he made. One of the things he liked was the simplicity of the GUI we had used. He said that this lack of sophistication allowed him to easily understand what all of the different functions on our app do without requiring much background explanation from me or the aid of a user manual, making our software more or less self-explanatory.

One of the improvements he suggested was with regards to the fact that we had not implemented functionality so that our software can be used by multiple people. We currently only can accommodate one user to make use of our system. This kind of software obviously would not be practical in the real world where lots of people may want to make use of our product to aid them in managing their stock. In relation to this he also had a concern with the fact that we did not add scalability in terms of the number of shelves a user can operate. In our software we hard coded it so that we can have a maximum of 6 shelves operating at any one time but what if the user wants to add more shelves to accommodate a larger amount of stock? There is no feature that allows us to do this and so this would definitely throw of anyone with a larger range of items to store. He described our software as "merely being a demonstration of what it would do for each user who makes use of it without having the ability to actually have more than one user operating it at any one time".

Apart from these 2 concerns with our app, he seemed to be pleased with how far we had come as a group and appreciated all the effort we had made to get to the final product we had presented to him. He could definitely see the need and potential for it in the software market being someone who has aided in developing software for massive corporate firms before.

Summative Evaluation

As outlined in the previous two sections Unit, Integration and system testing have been performed thoroughly verifying the features implemented. Overall, most features produce expected and desirable behaviour.

Regrettably however, we did not manage to arrange for sufficient user evaluation through the development period leaving a lot of our design choices unvalidated. This would need to be urgently addressed if development continued.

Some bugs and unexpected behaviours brought to light by the testing and quality assurance process that have not yet been fixed. Below these bugs are outlined with a brief comment on the cause of each.

- Position of empty shelves is not consistent between all sorting methods for the overview page. Empty shelf positions should appear consistently at the top or bottom of the page for all sort methods other than sort by shelf position.
    - *Caused by sort logic in controllers/mainController.js exports.getShelfOverviewList. Shelf details are retrieved from the database in order of shelf id then sorted by comparators which do not handle null values from empty shelves properly.*
- Weights can be added to empty shelf positions on the help page.
    - *Intentionally left in, was useful for developing the Arduino item setup route to simulate receiving data on an empty shelf, but should be removed.*
- Negative weights can be added to shelves - Ideally negative weights should be entered as zero.
    - *Caused by the choice to set the weights fields as signed integers, this was done to prevent problems if the shelf sent negative values.*
- Users can add weights above the max load of the shelf (5kg)
    - *Not considered, could be checked for and a warning displayed.*
- Price values entered for items are rounded up to the nearest whole number
    - *Caused by setting the data type incorrectly. The price field currently has a DECIMAL(10,0) type, it should have been DECIMAL(10,2).*
- Boxes to display the name of items on the shelf details page do not have overflow, part of the name is cut off.
    - *Caused by bad CSS, overflow was not considered*

Overall, beyond the pressing need to re-validate designs with stakeholders there are very few unexpected behaviours in the features we have implemented.

During development we referred frequently to suggestions made in stakeholder consultations and prototype demonstrations, trying to match our design as closely to the requirements they outlined

We can not know for certain until we re-validate those designs, but we are confident that what we have built contains both the core functionality that is required by any smart shelf solution as well as many of the features requested.

No matter the feedback, we are confident that what we have created could serve as a solid foundation for future development even if requirements shift.

Next Steps

We are of the firm belief our system would be of value to small labs and businesses, who often still rely on outdated or improvised inventory management solutions. Our system which accurately monitors stock and automates inventory tasks has the potential to offer considerable value to small and medium size stock environments.

What we have made serves as a god demonstration of the viability of such a smart shelf solution. Though a good demonstration and foundation for future development, the system is not shippable in its current state. Several actions would need to be taken and some features implemented before an initial alpha or beta release to consumers.

There are also several features that would add great value to the system that were not included in the ‘Minimum Viable Product’. A good example being the statistics and analysis of past stock history which could provide accurate measurements of usage over month or week time periods and offer predictions of stock depletion based on past rates.

Current Status

We have a functioning shelf and site that serve as a good demonstration of our concept. Due to circumstances towards the end of the project we have not had opportunity to verify our design fully with stakeholders, such as lab managers and business owners.

The first step that needs to be taken is a round of thorough user testing and evaluation. Feedback needs to be gathered from stakeholders to ensure that the design decisions made so far are good and whether there are any flaws or lacking features.

Without validation of design at this stage, any future development could be going down the wrong path and may be more difficult or costly to fix.

Once stakeholder feedback has been analysed and evaluated, a clear roadmap to an alpha release can be created.

We will not know for sure the steps needed until feedback is gathered, however below are some of the features and considerations we believe would need addressing before an initial release.

What needs to be done

The current shelf is based on an Arduino and pressure sensor and communicates over Wi-fi. Before an initial release, the design of a commercial shelf would need to be created and validated with stakeholders. The manufacturing of the shelf would also need to be arranged.

Our current Arduino shelf has had issues connecting to secure networks, such as the eduroam network at Goldsmiths. This networking issue must be addressed in the commercial hardware.

The inventory application architecture was created for only one set of shelves and one user. Before a release, the architecture will need to be altered to allow both

multiple users and for a single user to have multiple sets of shelves. Whether this will be achieved by giving each user a separate database or if the database will be expanded into a single large one that accommodates many users will need to be worked through and tested.

As a connected smart device aimed at businesses and labs security would be need major consideration before release. Over recent years several smart devices have raised security concerns. The potential security risks our device could introduce would need thorough investigation, mitigation, as well as testing.

As the system may also be used for business-critical tasks the security of the application and user accounts would also need to be of a particularly high standard. The site would need thorough testing for vulnerabilities, user data would need to be stored securely and accounts would need several layers of protection / authentication.

Beyond these issues, general improvements would be made to the application throughout its life cycle. The User Interface would be updated to a more modern feel, assisting the functionality and ease within of program for its users. More extensive helper documentation or troubleshooting service could be provided. A forum where users can post problems or bugs they encounter with the application would help the team see what problems need addressing as well as provide solutions to users.

Lastly, the group must continue to evaluate feedback as more and more users are brought in to test. Feedback will ultimately determine our implementation strategy.

Analysing the direction our product should take from stakeholder feedback is an important step the group must overcome first before the product can be sellable to a wide range of users.

If the feedback on our current solution is worse than expected, the group must then reflect and address what went wrong before development of another prototype.

If the feedback is positive, then suggestions can be taken in and development should continue swiftly to an alpha release to a small group of representative stakeholders.

If positive feedback is obtained and development proceeds the group must also decide on the business plan for the application, such as: Will the shelf be a one-off purchase? Will there be a rolling subscription? How much are our stakeholders willing to pay for our solution? How much do we need to charge to cover costs? Etc.

Other business considerations may also crop up, such as patents, marketing, funding, and securing manufacturing for the shelf.

DevOps

DevOps is the continuing process of integrating development, operations, and deployment. It shares many common features with agile, constantly delivering and gaining feedback from stakeholders, but on a larger scale.

DevOps can help solve issues such as bugs and errors within a program, and disagreements between project teams on goals and mistakes. DevOps addresses these problems by working as a cross-functional team that shares responsibility for the way software is planned and developed. This enables the various teams to

prepare the software and address feedback and issues that may arise. DevOps improves the collaboration between stakeholders and developers throughout a products lifecycle and ensures that software developed meets expectations.

Having teams within the product's development cycle can help stable growth of the product and in turn, gather more stakeholders for investment. A team that has an organised system for the way the product is updated and released will be more effective at delivering software and integrating feedback.

The first phase of the DevOps plan, before any integration is done, is 'Waterfall Development'. This phase allows the team to create code for the first stages of the product, eventually merging them for it to be tested. This step often takes months or years, but the group had a limited time space and started coding at the start of the spring term. The next phase is 'Continuous Integration'. In this phase, new code is constantly integrated with existing code in a shared repository. Constantly checking and testing code for it to be usable in the later stages. After this, Continuous Delivery is done. Here the code is tested without any additional automation and the code is almost ready for deployment. Lastly, Continuous Deployment practices deploying the product into production without human intervention. Here testing is done on a small number of users such as stakeholders who have had an interest in the product and is monitored to see the quality and effectiveness of the code before releasing it to a larger target market. Continuous delivery continues as additions that have been through the processes of validation and testing are released in short cycles.

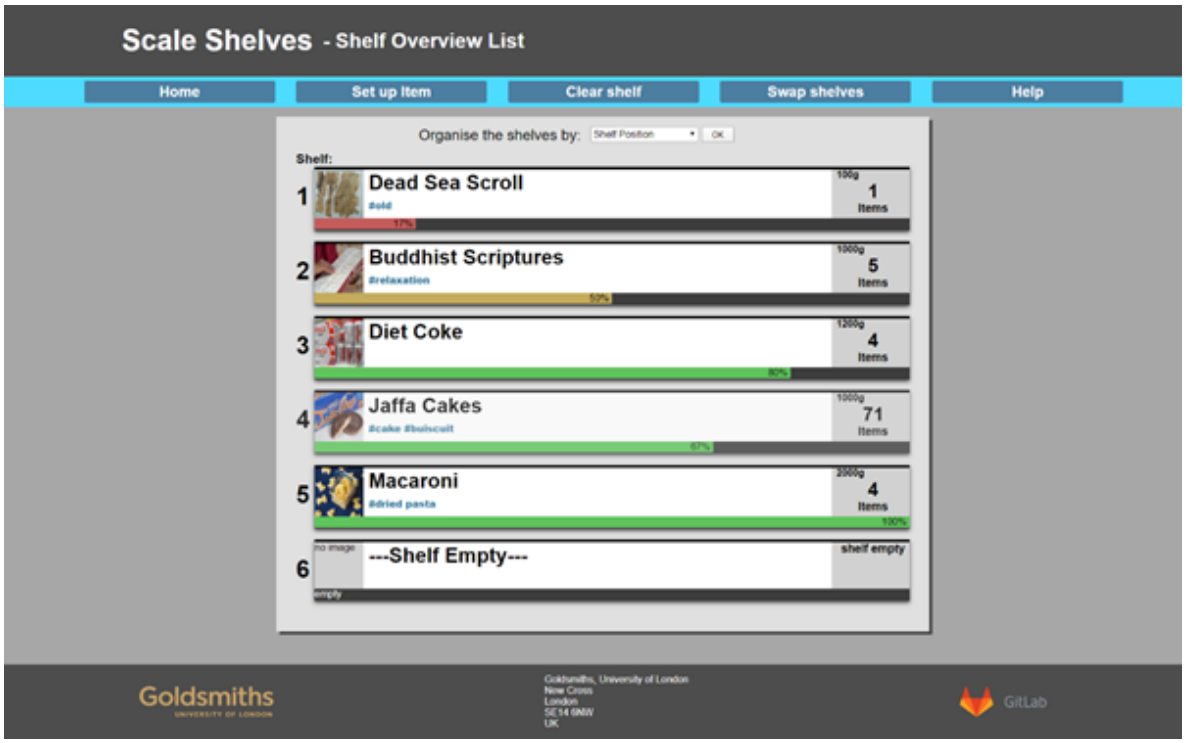
To implement DevOps practices the team may be split with different members taking responsibility for deployment, operations, monitoring and planning. Frequent meetings to discuss the status of each part of the cycle would be needed.

DevOps would be used to clarify which direction the product should go in the future and help plan what needs to be done to create a viable working product and then improve the product beyond.

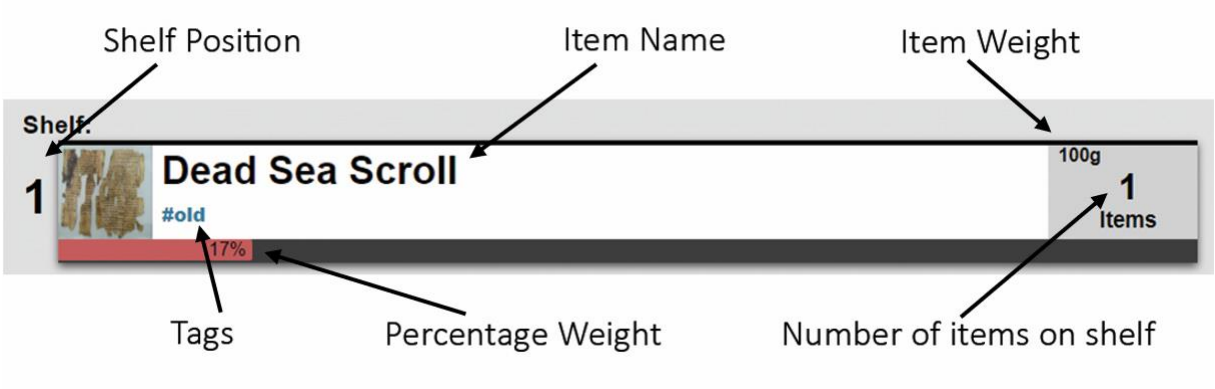
In the near future, we plan to implement a whole range of features to improve the application for user usability. A basic feature once the product has been released such as a login/register page for multiple users to access can be implemented to ensure users can make the most out of the application. Features such as a working warning system that can send a warning message to the user if a shelf or shelves are full, close to being empty, empty. Notifying the user when the shelf would reorder items or even tell the user that the items are on their way. Also, having item recognition for shelves may also be implemented in the later stages of the products cycle. More features are yet to be discussed, depending on the feedback we can gather on a stakeholder.

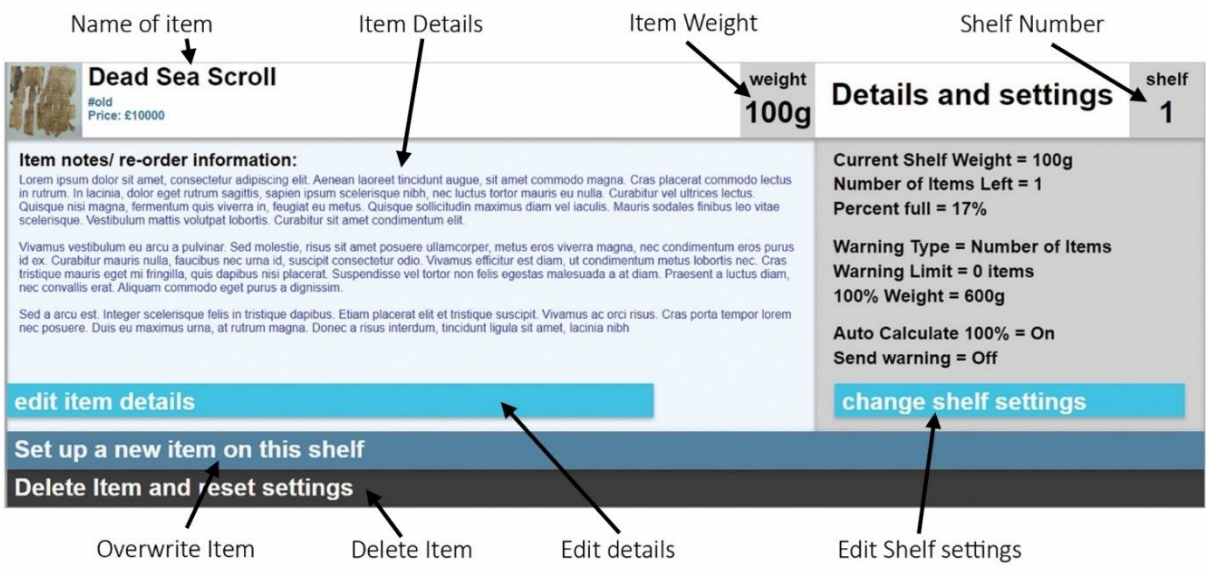
User Guide

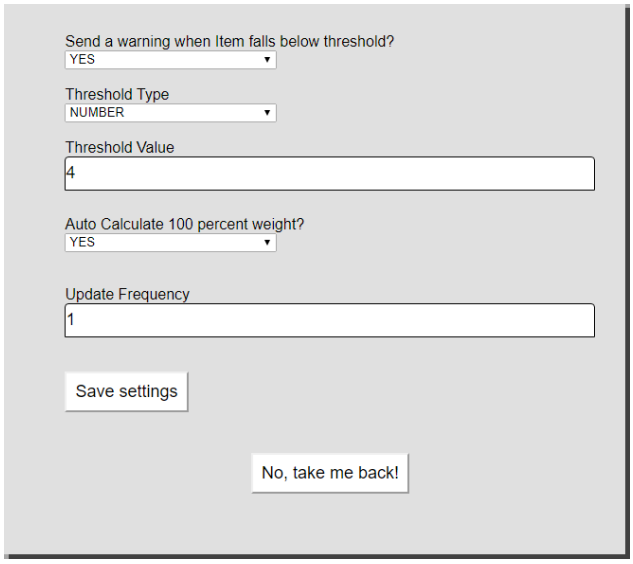
The home page is immediately displayed upon entering the web app. From here, a list of the shelves and their contents are presented. This includes the shelf number, name of item, weight, number of that item on shelf and tags related to the item. The shelves can also be sorted by shelf position, percentage weight, number of items left or absolute weight via the dropdown at the top of the page.

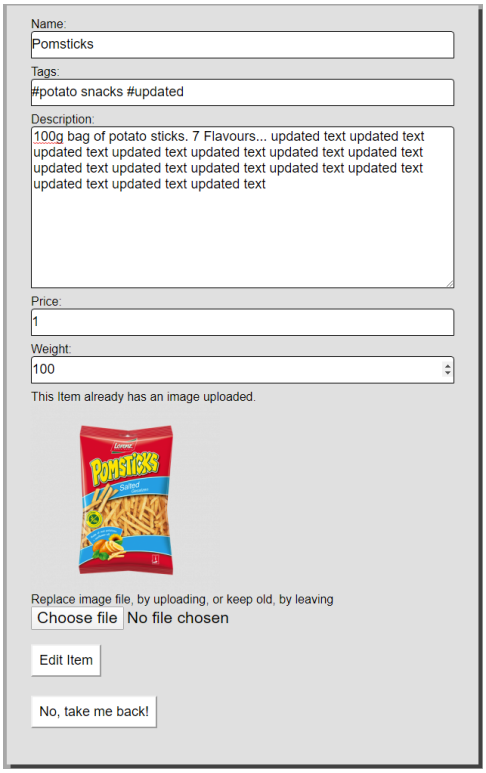


Upon clicking on a shelf, a table containing a more detailed overview of the elements and settings is displayed. The item notes and re-order information appear on the left and the details regarding weight and warning limits on the right. From this page, the user can edit the item details or change the shelf settings. They also have the option to set up a new item on the shelf or delete and reset the shelf position.





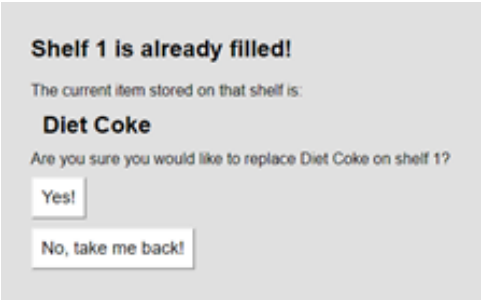




Editing the item details allows the user to change the name, tags, description, price, and weight of the item, or change/upload an image.

Editing the shelf details allows the user to turn the threshold warning on or off and change the type. The 100% weight can also be altered from this page.

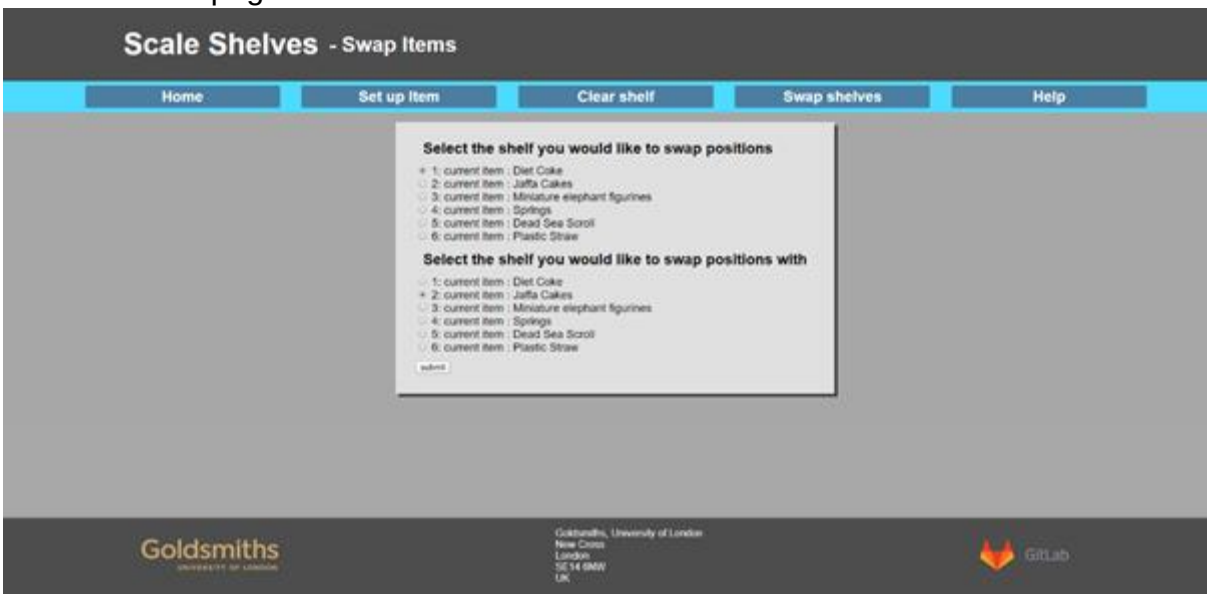
Selecting the ‘set up item’ from the navigation bar at the top of the screen presents a list of the shelves with their current item if it contains one, or ‘empty’ if not. The shelf to set up can be selected and a form will be displayed upon clicking submit. If a shelf with an item on is selected, the user will be notified and asked if they either want to replace the item or go back to the selection screen. The set-up form contains a field for the item name, tags, description, price, and weight. Any information regarding restocking or safety can be written in the description to be easily viewed after the item has been added. The items weight can be set automatically by inputting the individual item weight, or the scales built into the shelf can be used if the weight is unknown. After selecting this option, the item will need to be placed on the shelf and confirmed within the webapp. An image for the item can also be added at this stage which will be displayed in the list.



The ‘clear shelf’ page can be selected using the navigation bar at the top of the screen. From here, the user can delete the data stored on one of the shelf spaces. A form containing the shelf numbers as well as items stored is displayed, and after selecting the shelf and clicking submit, a confirmation screen will appear.

After selecting ‘yes’ it will be cleared from the database and the space can then be set up to contain a new item if required.

The ‘swap shelves’ page allows two shelves to have their positions swapped. This includes all data related to those items. Select the shelf you would like to change positions and the shelf that you would like to swap it with and click submit. Upon submitting, the shelves will be swapped, and the changes can be viewed in the list on the home page.



The help page is currently being used for testing and allows for weights to be manually added to the shelves while the scales were not connected. This would have otherwise been the page for guiding the user through various parts of the app, like this user guide.

Appendix 1 Shelf Construction

A: Marisa: Detailed account of development of Arduino scale shelf and associated code

The scale shelf was constructed from the following items:

Scale:
100mm x 45mm x 10mm pieces of softwood x 2
Nut and bolt x 2
55mm x 12mm x 12mm 5 kg Load Cell Bar x 1

Shelf storage bin:
76mm x 101mm x 167mm plastic stackable storage bin x 1

Load Cell Amplifier:
Sparkfun HX711 Load Cell Amp Breakout Board x 1

Microcontroller and IDE:
Arduino Uno R3 (Initially)
Arduino Uno Wifi R2 (For most of development and final product)
Arduino IDE version 1.8.12

Miscellaneous:
70mm x 50mm Perf board x 1
Jumper wires x 4
100mm x 20mm Velcro strips x 4
5 pin Header pins x 2

Scale Construction:

The pin holes on both sides of the HX711 were soldered to the perf board with the header pins. Each of the red, black, white and green wires coming from the load cell was then soldered to the perf board and connected to the RED, BLK, WHT, GRN pins of the HX711 by means of solder bridges. The same technique was used to connect jumper wires to the VDD, VCC, DAT, CLK, GND pins on the other side of the HX711. The information on how to connect the load cell and the Arduino to HX711 was gained from the online Sparkfun HX711 hookup guide[2].

Two pieces of wood were cut to create a wooden base and scale platform. A hole was drilled in the front of the platform piece and the back of the base piece in order to attach the load cell to both in such a way that any weight placed on the platform would cause the load cell to pivot downwards causing a strain that can be detected by the strain gauges in the load cell, amplified by the HX711 and then sent to the Arduino. The load cell was attached to the platform and base with nuts and bolts. The nuts also served as spacers between the platform/base and the load cell. Finally, the plastic storage bin was attached to the scale platform by means of Velcro strips. The strips keep the bin stable on the scale but allow for the bin to be removed if required (e.g. for cleaning). The scale design was based on the online Instructables tutorial on how to build Arduino weighing scales[1].

Connection to Arduino:

Once the basic construction of the scale was completed it was ready to connect to an Arduino microcontroller for calibration. Initially an Arduino Uno R3 was used, which does not have built in Wi-Fi or ethernet capability. This was adequate for the initial set up and calibration of the scale. The Arduino was connected to an Acer Laptop via a USB cable. The laptop operating system was Ubuntu Linux 18.04 and it also had the Arduino IDE version 1.8.12. installed on it. The demonstration Arduino sketch from the instructables guide[1] HX711.ino was downloaded and stored in the IDE and the required Arduino library HX711.h was added to the IDE[3]. The demonstration sketch was uploaded onto the Arduino itself which was then ready to be connected to the scale.

The scale was connected to the Arduino using the jumper wires previously connected to the HX711 in the following way:

HX711 Pin	Jumper Wire Colour	Arduino pin
DAT	White	A0
CLK	Orange	A1
GRD	Green	GRD
VCC	Black	5V
VDD	Green	3.3V

Once the scale and Arduino and laptop were all connected together with the HX711.ino sketch running, it was possible to view scale weight values processed and output in the Arduino IDE Serial Monitor window. An initial glance at the output confirmed that the scale was providing credible looking data, but to be sure the scale was accurate it next needed to be calibrated.

Scale Calibration:

(This method comes from Step 8 of the Instructables guide[1].)

The HX711.ino sketch includes the line:

scale.set_scale(2020.f);

which sets the calibration value. The pre-set value of 2020 in the sketch is designed for obtaining readings in grams from a 1kg loadcell. In order to calculate the calibration value required to obtain readings in grams from a 5kg load cell the following steps were followed:

1. Start with an arbitrary calibration value (I retained 2020)
2. Obtain an item of known weight that is at least a tenth of the scale's maximum weight (I used a 500g package of dried macaroni) which was weighed on a post office scale to obtain a reliable weight value of 507g
3. Place item on the scale and obtain a weight reading

4. Multiply current scale factor by the weight reading, then divide by the known weight obtained in Step 2. The value obtained is the new calibration value.
5. Repeat the steps using the new calibration value in Step 1 until the weight readings obtained at Step 3 are accurate.

After following this process, a calibration value of 458 was calculated. Other items such as 9V batteries and tins of baked beans were then weighed on the scale and the values obtained checked with the item weights obtained from kitchen scales to ensure the scale was measuring consistently and reliably.

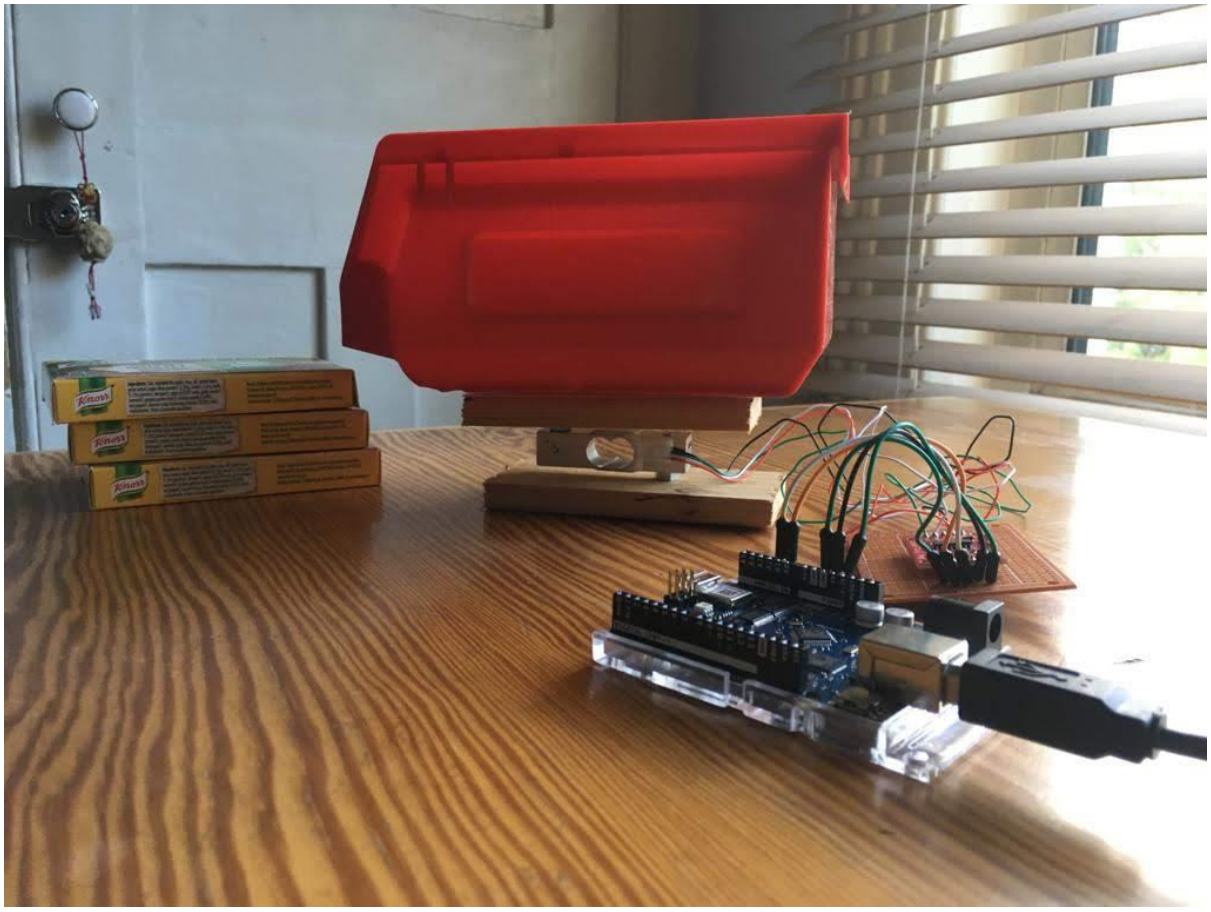


Figure 14: The assembled Arduino shelf with red bin attached for items to be placed in.

Sending weight data to the web application:

The next task was to get the weight readings calculated by the Arduino into the web application. This would entail sending the data either by Ethernet or Wi-Fi to the Computing Department network where our webserver and database would be running. Although an Ethernet connection would have been a more reliable option, it was problematic for two reasons. The first was that an Ethernet module would have had to be obtained and soldered to the Arduino R3. The second was that permission would be required to have the Arduino 'whitelisted' on the college network and configured for access to the Computing Department virtual servers. As development time was short, it was decided to pursue the Wi-Fi option instead.

Network connectivity:

The Arduino Uno R3 was replaced with an Arduino Uno Wi-Fi R2 microcontroller which has a built in Wi-Fi capability, and the required WiFiNINA.h library was added to the Arduino IDE. An initial test of the Arduino Wi-Fi connection was accomplished by writing an Arduino sketch which would attempt to connect to a home Wi-Fi network and then output information such as the IP address into the Serial Monitor of the Arduino IDE to signify success or an error message to signify failure. The information regarding how to write this sketch was gathered from an YouTube video tutorial[5]. The Arduino was able to successfully connect to the home Wi-Fi network.

Next an attempt was made to connect the Arduino to the Goldsmiths eduroam network. Unfortunately, this was not possible as the Arduino connects to Wi-Fi using just an SSID and a password whereas eduroam is a WPA2 Enterprise network which requires a username to be input as well as an SSID and password. Attempts were also made to connect to the Goldsmiths guest network (which had the same issue as eduroam). Being unable to connect to the network while on campus at this point was detrimental to the scale development process as a lot of time and effort was spent trying to find a solution to the issue and it meant that a lot of work on the scale carried out on campus could only be evaluated when the developer was at home, which slowed the process down considerably. It also led to integration of the scale with the web application taking place at a much later stage of development than was desirable. Towards the end of the development process a workaround was discovered whereby the Arduino and Laptop could both be connected to the Department of Computing, Hatch Labs G11 Wi-Fi network which only requires an SSID and a password. This was helpful as it enabled more development to take place on campus and would have meant that integration work could have taken place there. Unfortunately, the Covid-19 pandemic followed soon after and all further development had to take place at the developer's home.

Attempt to use Wi-Fi and POST method to send data to our webserver:

There were two choices regarding how and where to send the weight data from the Arduino. The first was to send the data using HTTP and POST to the Node.js/Express webserver to be processed in the middleware layer of the web application. The second was to send the data directly to the MySQL database using tools from an Arduino MySQL Connector library, where it could be retrieved and processed by the web application.

A test using adapted Arduino sketch code from a YouTube video tutorial on temperature information from an Arduino to a webpage[5] worked successfully for outputting random numbers and then scale weight values to the localhost/browser on the developer's laptop, using the Arduino as its own server. This method did not however use HTTP/POST, which would be required for sending the data to a Node.js/Express webserver. An attempt was made to adapt code from a second YouTube video tutorial which demonstrated how to send data from an Arduino Wi-Fi by POST to a webserver[4]. The developer could not successfully get the group Node.js/Express webserver to accept the data through HTTP/POST and there were also issues with permissions to access the virtual server ports. As the tutorial did not

specify the type of webserver used (it was probably Apache) and the server side scripting language used was PHP, the developer did further research on how to resolve these issues for Node.js/Express/JavaScript. As development time was running out however and progress was slow, it was decided to explore the option of sending the data directly into the MySQL database instead.

Use Wi-Fi and MySQL connector library to send weight data direct to a database

An excellent tool for getting data directly from an Arduino to a MySQL server is the MySQL Connector/Arduino library created by Dr Charles Bell[6]. In addition to the scale, Arduino and laptop configuration previously described, the following components were required:

- MySQL Connector/Arduino Library installed in Arduino IDE
- A MySQL 8.0 server (with legacy authentication) was set up on the developer's laptop.
- A database called test_arduino was created
- A user account was created with native authentication and granted all privileges to access the test_arduino database
- Access to port 3306 was enabled on the laptop
- The MySQL server and the Arduino must be connected to the same Wi-Fi network segment.

Once everything was set up, a basic test of the user account direct access to the test Arduino database on the MySQL server was conducted both on the developer's laptop and from a different laptop on the same home network. This was followed by a test SQL INSERT statement directly into a table created in test Arduino to ensure that the weight type data (float type) was inserted and represented correctly. Both of these tests were successful.

The next challenge was to connect the Arduino to the MySQL server on the developer's laptop. In order to do this an adapted version of the example sketch Listing 2: Sample Connection Test – Wi-Fi from the MySQL Connector/Arduino User Guide was used[6]. Initially there was an issue whereby, after the Arduino had connected to the Wi-Fi, the sketch would hang at the MySQL server connection stage. Some research online revealed a forum discussion between a user with the same issue and a response from Dr Bell with some very useful suggestions involving adding some extra libraries to the sketch and commenting out some code in one of the MySQL connector cpp files[7]. This resolved the issue and it was then possible to connect to the MySQL server with laptop and Arduino both on a home network.

Now that it was possible to connect to the database server from the Arduino, the ability to send an INSERT statement to a table in the test Arduino database was tested. The ability to send a static query string of either integer or float data was successfully tested using an adaptation of Listing 3: Simple Data Insert Sketch from the MySQL Connector/Arduino User Guide[6]. Sending readings such as weight data

from a sensor however requires that a new query string is built each time data is sent. This issue is addressed in Listing 4: Complex Insert Sketch from the MySQL Connector/Arduino User Guide[6]. This sketch was also successfully modified to insert regular weight readings from the scale into a table in the test Arduino database.

Putting it all together:

A first draft of an Arduino sketch that combined setting up the scale, processing weight readings connecting to Wi-Fi and the database server and inserting readings into a database table was created. The sketch combines and adapts parts of all the example sketches previously mentioned[1,3,4,5,6]. A database dump file of the project database was used to create a replica of the application database on the developer's laptop. The draft sketch was modified to insert data into one of the shelf tables (id1weights) in the replica application database and run while adding or removing items of known weight to and from the scale. This was a successful test of the integration of the database and the Arduino scale. A quick and dirty test of the integration of the components above with the latest version of the web application(running on the developer's laptop) was carried out by checking if the weight values being sent directly to the database were being correctly detected, processed and displayed by the web application. This was successful in that changes to weights caused by adding and removing items to/from the scale bin were correctly represented by the web application shelf details for shelf 1 when the web page was refreshed in the browser.

The draft sketch was reviewed at this stage and the following improvements were made:

- The sequence of activities was placed in a more logical order
- Redundant or commented out code was removed
- Each section was commented
- Code was written into the loop section of the sketch to check and resolve the status of the MySQL server connection before attempting to send data (as the sketch would hang if data was sent when the connection had dropped).
- A time delay was added to control the rate at which weight data was sent to the database

End of development phase:

As the development phase of the project was now drawing to a close and the scale was successfully sending data to the application database via the Arduino it was decided that development on the Arduino scale would halt here and attention would now be focused on formal testing of the integration of the Arduino scale, database and web application. For details of the testing phase, please view the Quality Assurance section of this report.

Next steps:

- **Connectivity:**
  - Use an Ethernet enabled Arduino and have it whitelisted for the Goldsmiths network and also with permission to access the virtual servers of the Department of Computing. This would hopefully be more stable, reliable and secure than a Wi-Fi connection.
  - Try to resolve issues encountered with accessing the database server on our virtual server space. Retry suggestions provided by Eamonn Martin regarding changing the port setting in the MySQL server config file to 8000, restarting MySQL server and changing the MySQL port to 58524 in the sketch I tried this with the Arduino on the G11 network and it didn't work. The MySQL Connector Arduino user guide states that the Arduino and the MySQL server must be on the same network segment? It could however be an issue with how to correctly express the server address and port of the group virtual server in the Arduino sketch.
  - Investigate the possibility of using a Raspberry Pi instead of an Arduino. The web server and database server could then possibly run on the Pi as well.
- **Sending data by HTTP/POST to web server:**
  - Do further research on how to send data by HTTP/POST to the web server. I found a solution online which creates a Node.js/Express HTTP server and uses web sockets to communicate with JavaScript embedded in an HTML file. I think this solution seems unnecessarily complicated though and perhaps just more research is required on how to send and handle a POST request outside of a browser/form using Node.js/Express/JavaScript
- **Scale Shelves:**
  - Try and develop a scale that can detect very light items individually. A much more expensive load cell would definitely be required for this.
  - Build enough scale shelves for all six shelf slots in the web application.
  - Investigate if all six shelves can share one HX711 amplifier and one Arduino
- **Sketch:**
  - Research and write code that will automatically restart the Arduino if the Wi-Fi connection drops.
  - Research and write code that will allow the web application to send instructions to the scale e.g. to reset a shelf or swap shelves.

Appendix 2: Architecture

Below is a brief outline of the project folder structure, found in the **'ServerDev'** folder of the git: https://gitlab.doc.gold.ac.uk/softproj24/smart-inventory-shelves/-/tree/master/ServerDev

- **controllers:** Some routes, specifically the item setup and shelf overview pages have their routing logic separated into controllers and then exported and called in the route file. The benefit of this approach is it allows the routing logic to be swapped out - a replacement set of logic for a route can be created without affecting the original. The route logic usually extracts or passes data from requests, then renders a page. They may call model methods to process that data or interact with the database.
    - ***itemSetupController.js, mainController.js***

- **models:** created to represent either tables in the database or used to contain database calls useful to specific functions, such as the overview. They have constructors allowing for data to be packaged conveniently for rendering on pages or inserting into the database, they also have methods which may interact with the database performing CRUD operations.
    - ***Item.js, overview.js, shelf.js, weight.js***

- **public:** for storing files public users will have direct access to - CSS files and images mainly
    - **CSS**: Stores CSS and SCSS files
    - **images:** Stores site images
        - **upload:** Stores item images uploaded by users

- **routes**: for storing routing files. Each file contains the routes for specific paths. Routes are split up this way because sites often end up with a lot of routes and the logic can be quite long - by splitting up the logic it is easier to see what routes a site has in the routes files.
    - ***delete.js, help.js, itemSetup.js, main.js, shelfDetails.js, swap.js***

- **util**: for storing any functions run separately from routes
    - ***autoCalcWeight.js***: Finds the most recent highest weight in a weight table and sets that to 100% (if this feature is turned on)

- **views**: For storing HTML EJS template files
    - **includes folder:** EJS allows for common HTML elements (such as the head and navigation bar) to be stored in separate files then imported into individual files. The benefit provided is that changes to common elements need only be done in one file to update every page.
    - **test folder** - for creating test views (in order to test routes)
    - **One folder for each domain path**: Each path has a separate folder for storing EJS files. Many files had similar names, so splitting them up this way seemed sensible

- ***index.js***: main express server file

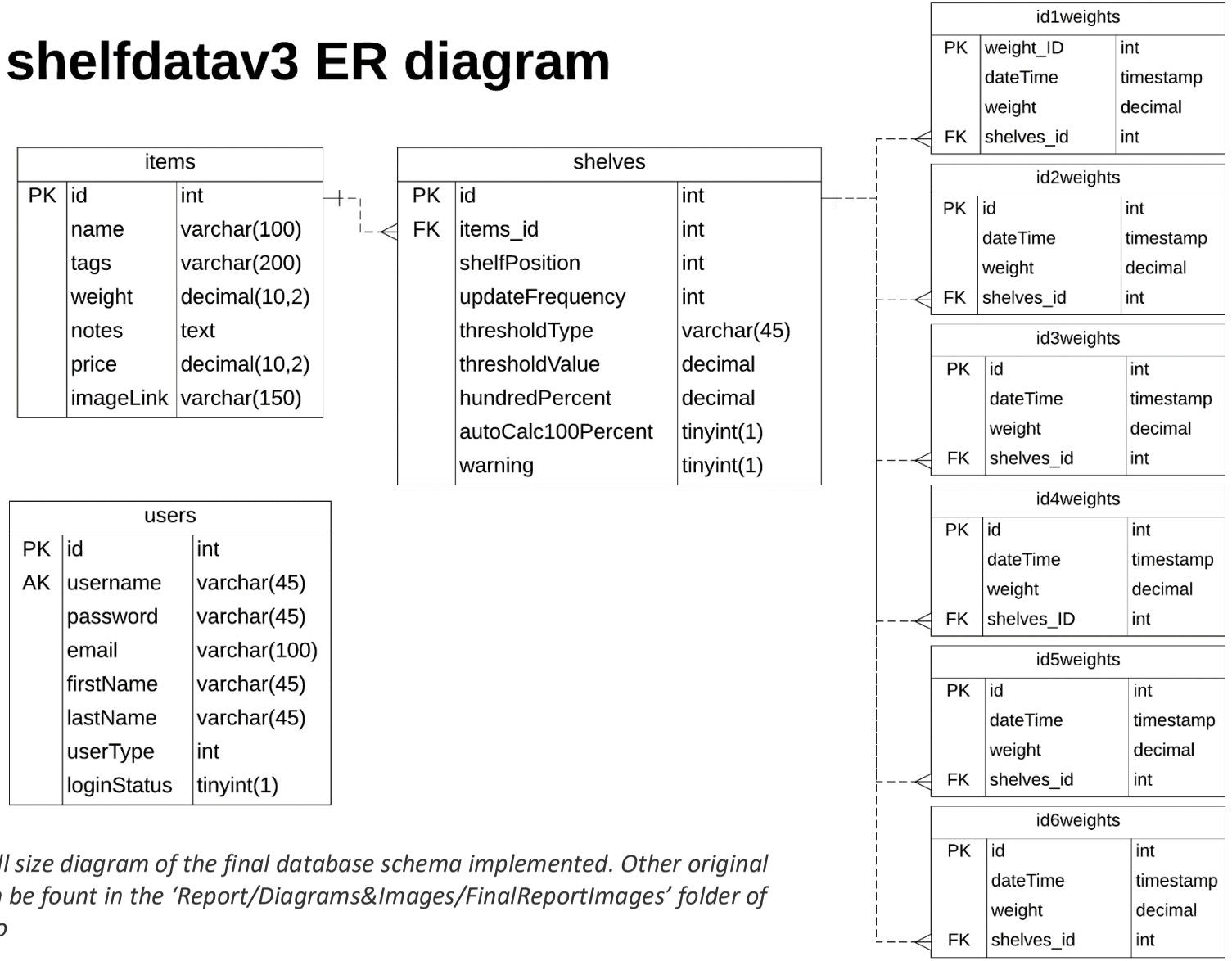


Figure 15: Full size diagram of the final database schema implemented. Other original diagrams can be fount in the 'Report/Diagrams&Images/FinalReportImages' folder of the main repo

Proposed Database ER diagrams

Figures 16 – 19 showing some of the database schemas proposed by different members during development.

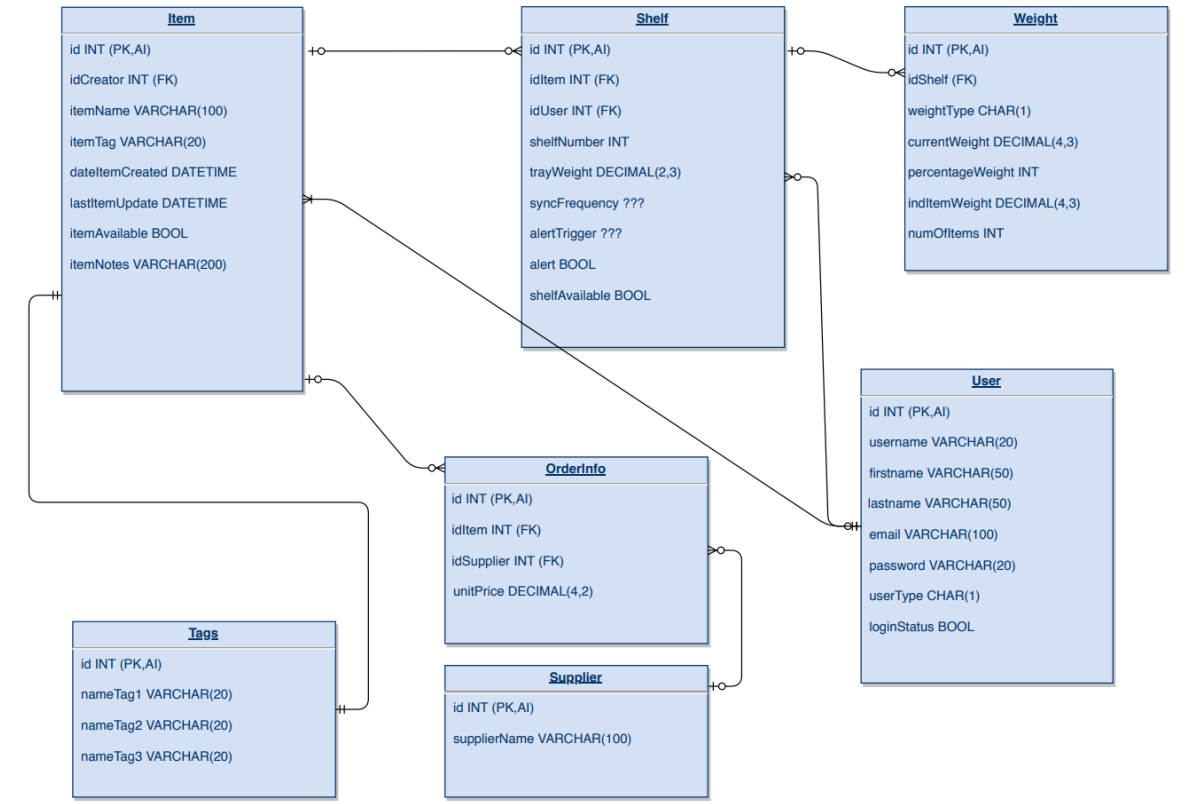


Figure 16

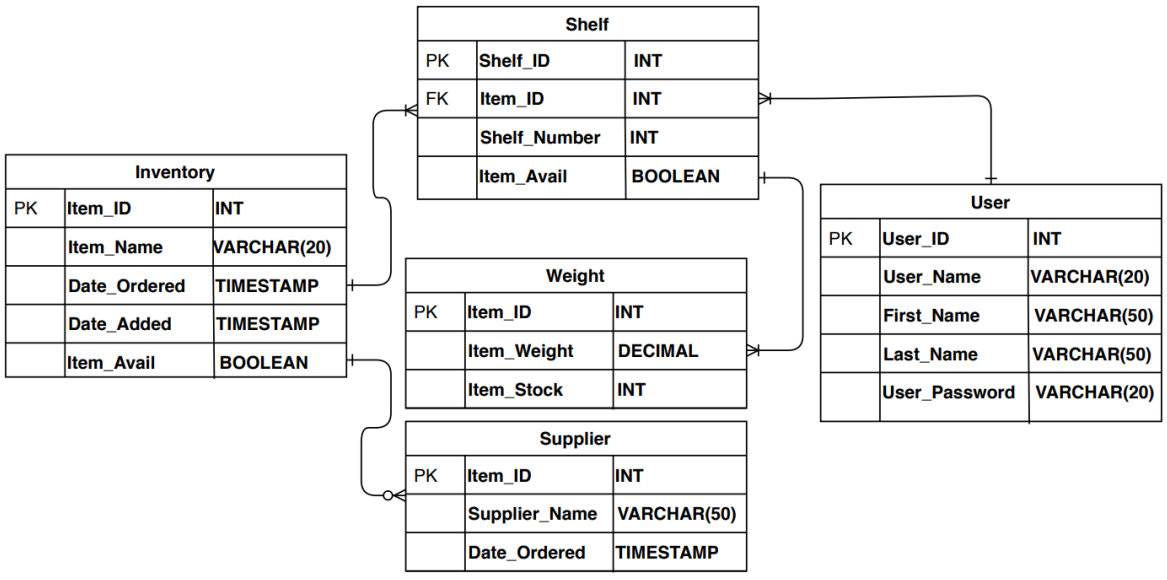


Figure 17

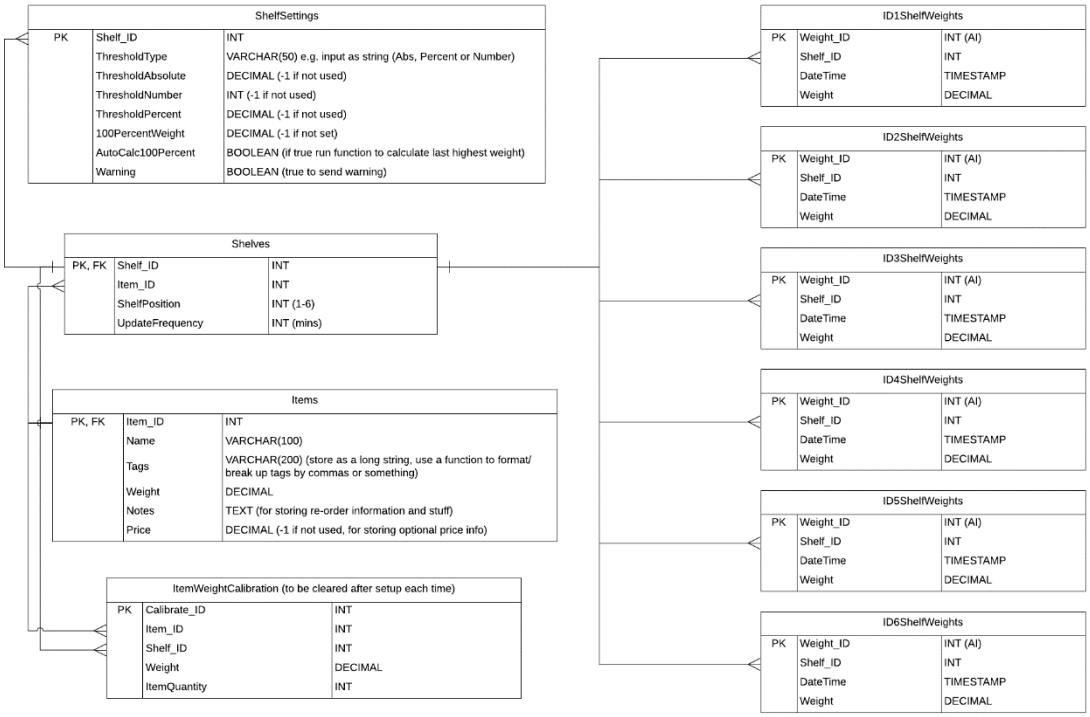


Figure 18

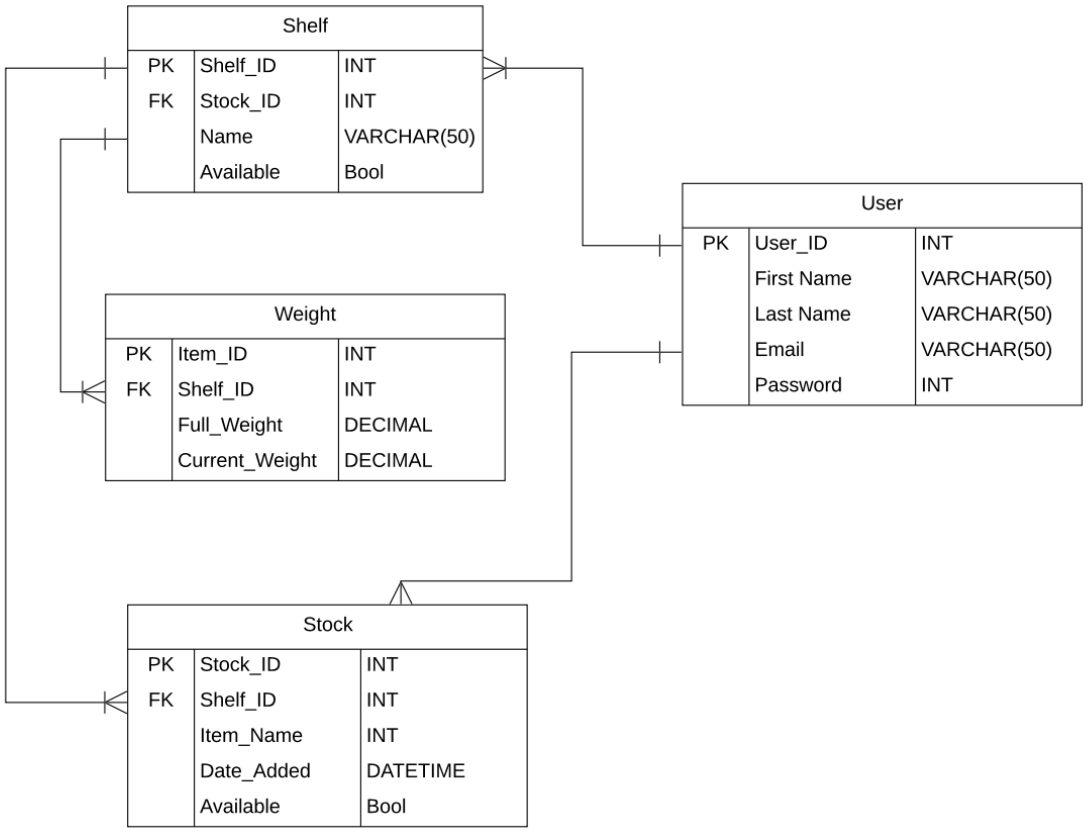


Figure 19

Appendix 3: Planned UI Overhaul

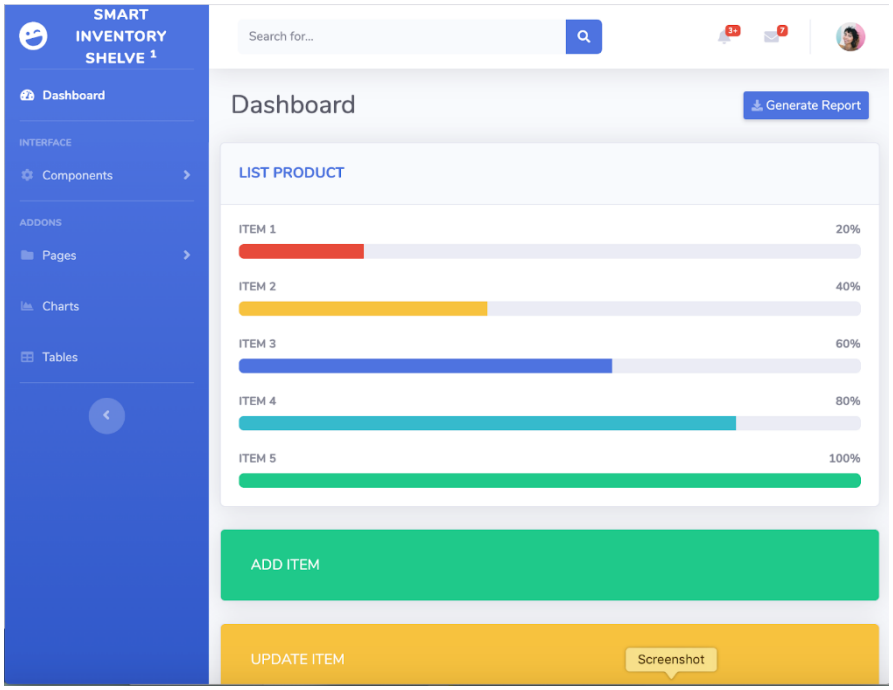


Figure 20: Home page from the Bootstrap based planned UI

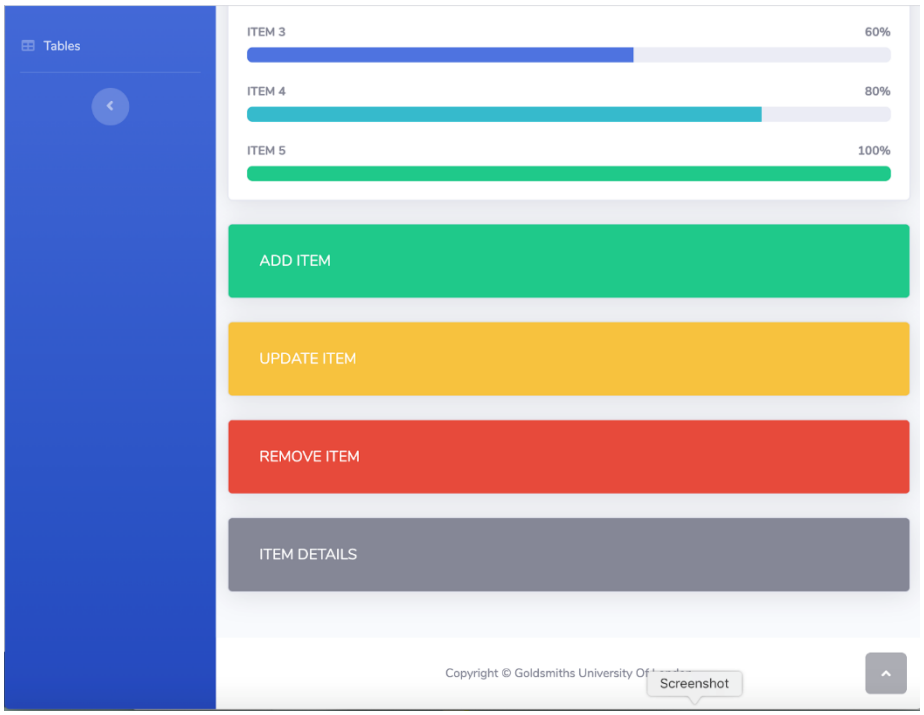


Figure 21: Bottom of the Bootstrap based planned UI homepage

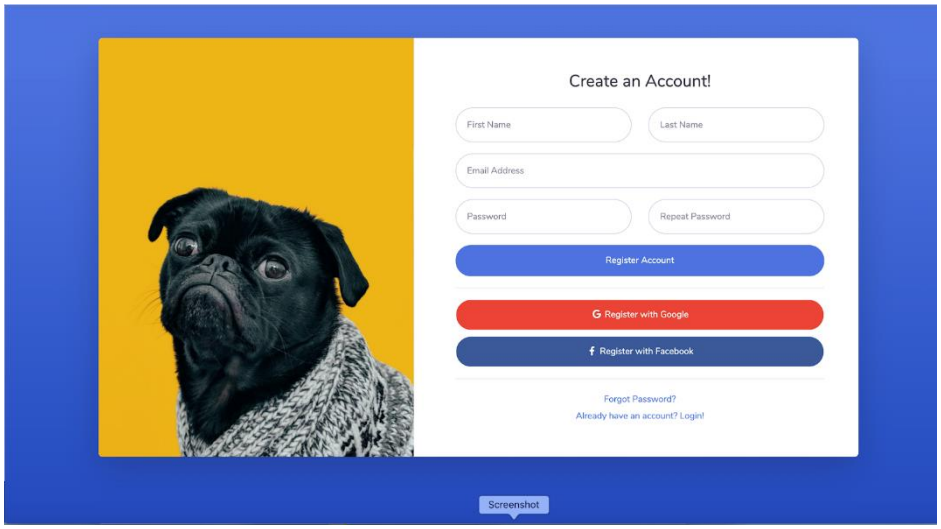


Figure 22: Sign up page from the Bootstrap based planned UI

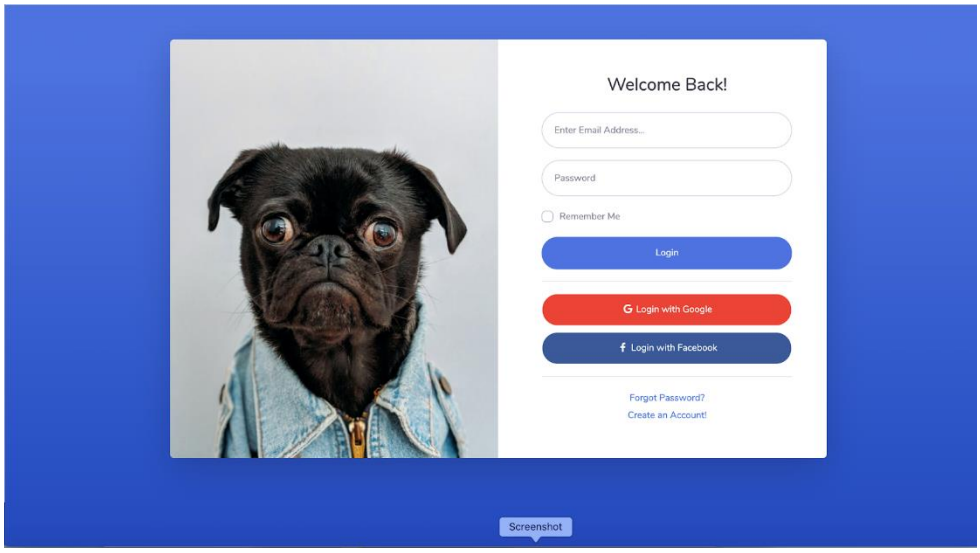


Figure 23: Login page from the Bootstrap based planned UI

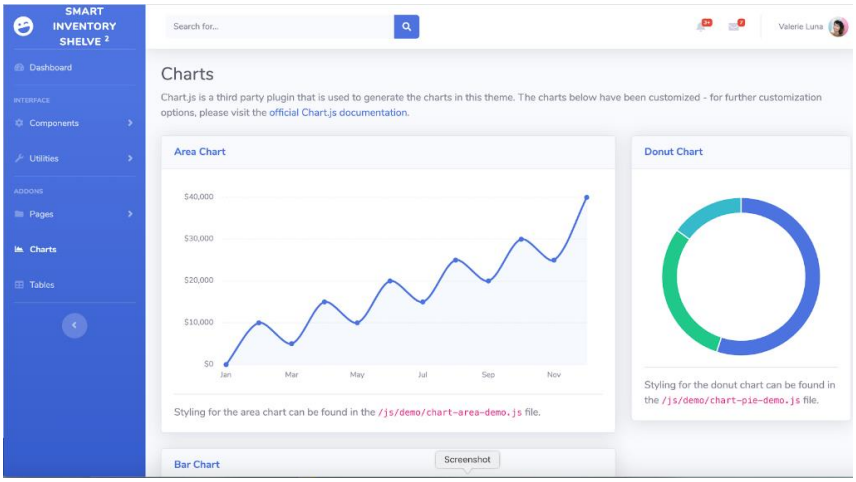


Figure 24: Chart page from the Bootstrap based planned UI, which could be used to show item usage over time or the cost of items used over time.

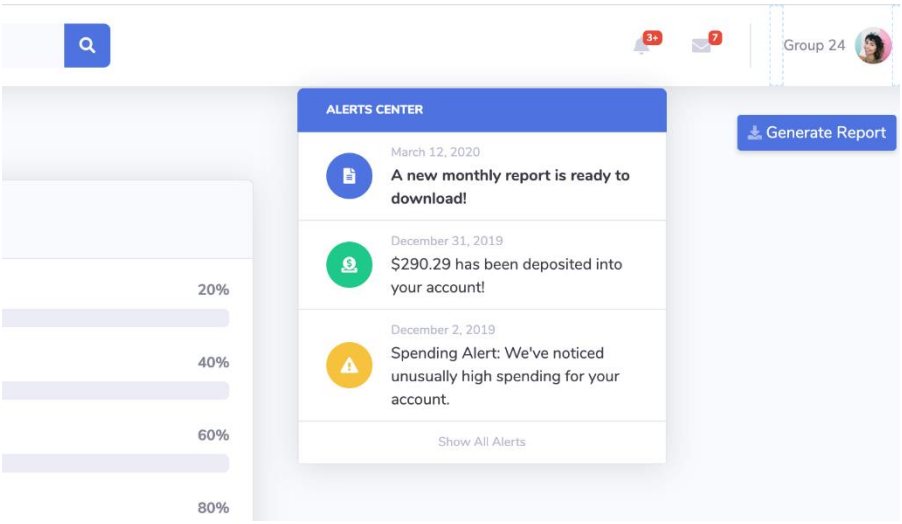


Figure 25: Notification page from the bootstrap-based UI. Could be used to send stock warning notifications, as well as monthly overviews of shelf usage (if a desired feauture)

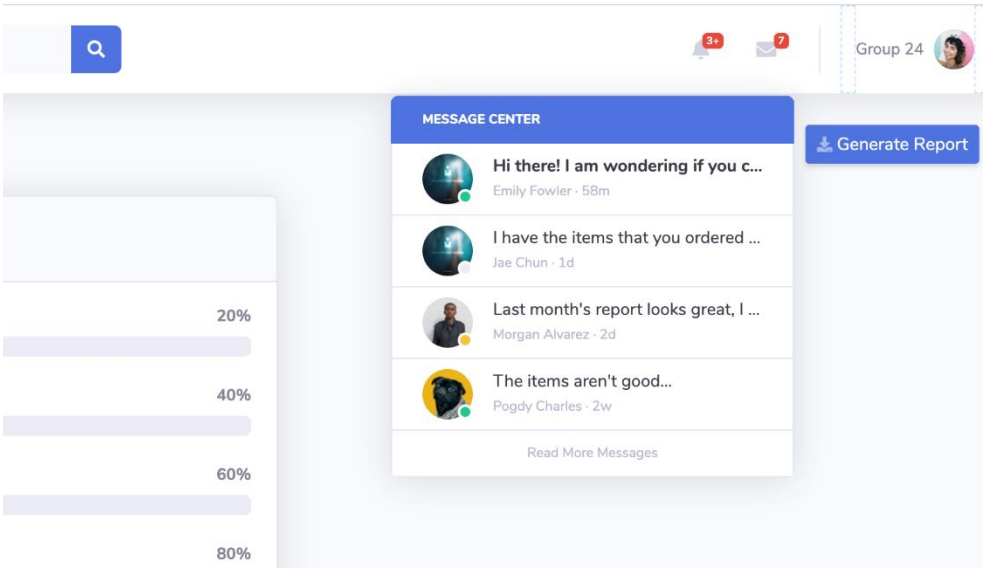


Figure 26: Messaging section of Bootstrap based UI. Could be used to offer troubleshooting advice to users.

Appendix 4: Project Management

Gitlab Wiki: https://gitlab.doc.gold.ac.uk/softproj24/smart-inventory-shelves/-/wikis/home

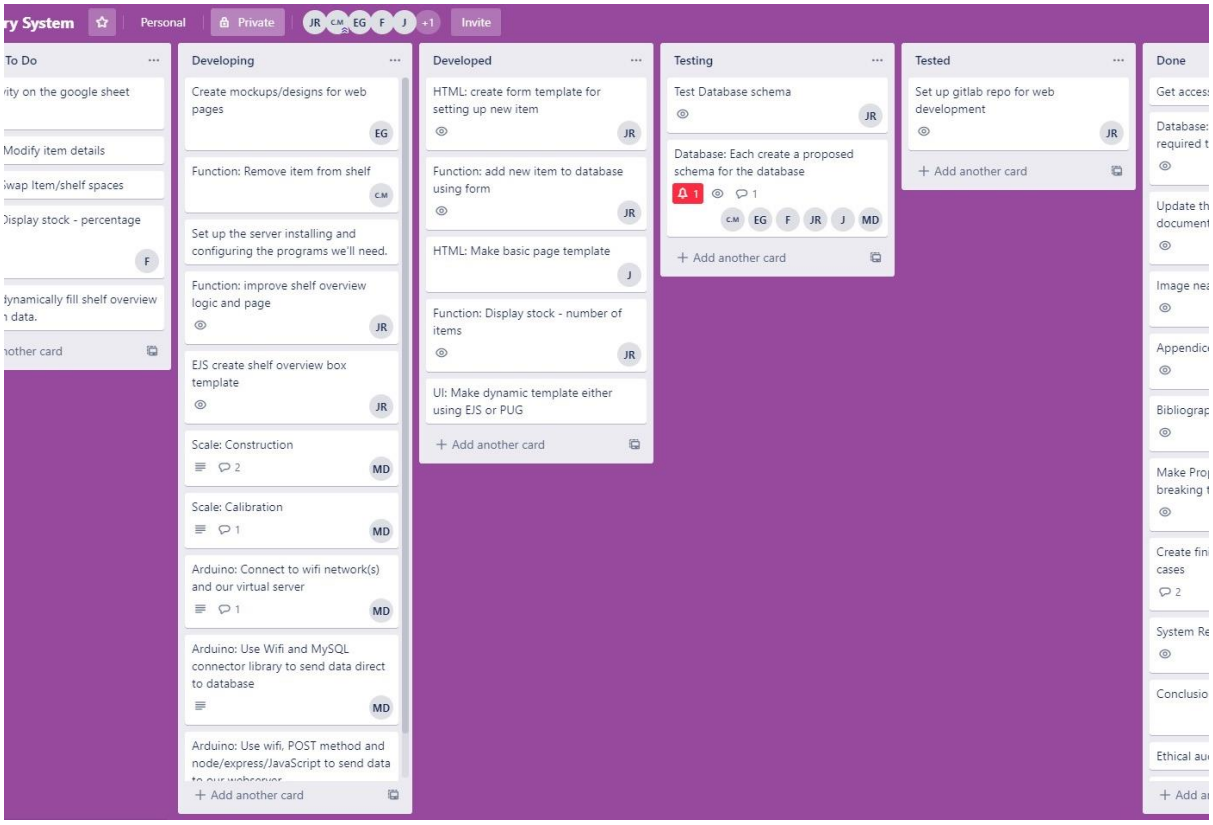


Figure 27: Trello board near the beginning of development

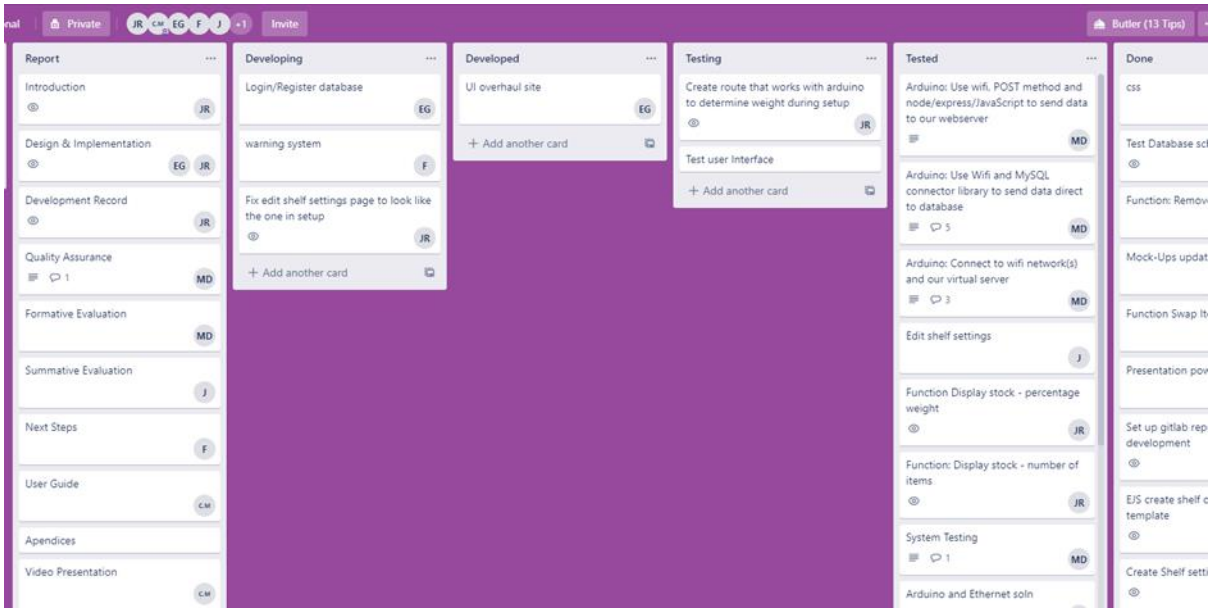


Figure 28: Trello board when development finished and work on the Report began

Ethical audit:

Our current system does not have the ability to hold multiple users, if developed further this would be an area where focus would need to be given to privacy and data protection. This is especially important for our target demographic, who's inventory data may be business critical. Encryption of user data and multiple levels of authentication would be needed, as well as extensive testing of the smart shelf and application for vulnerabilities.

During the development of the site steps were taken to not encroach on others Intellectual Property. Some images were needed for the site, all images used were labelled for non-commercial re-use. If development continued the rights to images used would need to be purchased or produced in house.

Several sources were referenced while developing the code for the application. All these references appear as comments within the code. Some of the more notable references used to troubleshoot problems encountered in the code are referenced below (links to references in bibliography).

- Setting up a 'mysql2' 'pool' that returns promises when the database is queried (lines 17-27 *'index.js'*):
    - **Evert Pot** [1]

- Making 'flash messages' available to all roots (lines 75-82 index.js)
    - '**Thad Blakenship'** StackOverflow [2]

- Setting up 'multer' for handling image uploads (lines 11 – 31 '*routes/shelfDetails.js'*):
    - **Brad Traversy** at Traversy Media [3]
    - **Maximilian Schwarzmüller** at Arcademind [4]

- Preventing NaN values affecting sorts when sorting the overview page (lines 51 – 70 *'controllers/mainController.js'*):
    - **Fabian N**. on StackOverflow [5]

- Help writing a series of synchronous calls as a promise stack for fetching all item names on shelf selectors (lines 30-61 *'models/item.js'*)
    - **'Josh**' on StackOverflow [6]

- Help passing EJS variables to JavaScript scripts using htmlDecode (lines 78-91 views/overview-list.ejs)
    - **'robertklep'** on StackOverflow [7]

- Help dynamically hiding un-needed form fields in shelf settings during setup (lines 75 – *93 'views/item-setup/shelf-settings.ejs'*)
    - **'radhika'** on stack overflow [8]

Appendix 5 Testing

For the full details of each test performed, please refer to the spreadsheet ‘SmartShelfTesting.xlsx’ which can be found in the ‘Report\Testing Materials’ folder of the git repository.

Example Test Case (See testing spreadsheet for the input data):
Test Case No: B3.4
Description: Test user can submit completed Replace/Set up Item form
Prerequisites: User has confirmed replacement for a shelf already in use (see B2.3) or user has selected an empty shelf via Shelf Selector form (see B1.2)
Step:1 Complete Replace/Set up Item form: (Please use Shelf 1)
Step:2 Click ‘Add item’ button
Expected Result: Shelf settings form is displayed with correct shelf number and new item name
Actual Result: Result as expected.
Status: Test1 PASS
Status: Test2 PASS
Notes: Go to B4.1

Test Suites:
A: Test Home Page Functionality
B: Test Item Setup Page Functionality
C: Test Clear Shelf Page Functionality
D: Test Swap Shelves Page Functionality
E: Test Help Page Functionality
F: Test Shelf Details Page Functionality
G: Weight Tests

Function tested/Test Case by Suite:
Test Suite A: Test Home Page Functionality
Function Tested: Page links are working
Test Cases: A1.1 – A1.7
Test Status: All tests passed
Function Tested: ‘Organise shelves by’ drop down is working (default and user selection)
Test Cases: A2.1-2.5
Test Status:
Test Cases A2.1-2.2,A2.5 passed
Test Cases A2.3-2.4 both fail/partial fail
Function Tested: Shelf overview data displayed
Test Case: A3.1
Test Status: Test passed

Test Suite B: Test Item Setup Page Functionality
Function Tested: Shelf selector form
Test Cases: B1.1-B1.3
Test Status: All tests passed
Function Tested: Confirm shelf dialog
Test Cases: B2.1-B2.3
Test Status: All tests passed
Function Tested: Replace/Set up Item form
Test Cases: B3.1-B3.5
Test Status: All tests passed
Function Tested: Shelf settings form
Test Cases: B4.1-B4.4
Test Status: All tests passed

Function Tested: Replace/Set up Item form (set weight with scale)

Test Cases: B5.1-B5.5
Test Status:
Test Case B5.1 passed
Tests Cases: B5.2-B5.5 partial pass

Test Suite C: Test Clear Shelf Page Functionality
Function Tested: Delete selector form
Test Cases: C1.1 -C1.4
Test Status: All tests passed
Function Tested: Confirm shelf dialog
Test Cases: C2.1 -C2.3
Test Status:
Test Cases: C2.1-C2.2 passed
Tests Case: C2.3: Test1 passed, Test2 partial pass

Test Suite D: Test Swap shelves Page Functionality
Function Tested: Swap Items form
Test Cases: D1.1 -D1.8
Test Status: All tests passed

Test Suite E: Test Help Page Functionality
Function Tested: Help page
Test Case: E1.1
Test Status: Test passed
Function Tested: Add Weight page
Test Cases: E2.1-E2.11
Test Status:
Test Cases: E2.1-2.4, E2.8-E2.11 passed
Test Cases: E2.5-E2.7 failed

Test Suite F: Test Shelf Details Page Functionality
Function Tested: Shelf details page
Test Cases: F1.1-F1.2
Test Status:
Test Case: F.1 passed
Test Case: F.2 partial fail
Function Tested: Shelf details: Set up a new item link
Test Cases: F2.1
Test Status: Test passed
Function Tested: Shelf details: Shelf details: edit item details
Test Cases: F3.1-F3.4
Test Status:
Test Case: F3.1 partial fail
Test Cases: F3.2-F3.3 passed
Test Case: F3.4 partial fail
Function Tested: Shelf details: change shelf settings
Test Cases: F4.1-F4.4
Test Status:
Test Case: F4.1-F4.3 passed
Test Case: F4.4 passed (but can inherit price rounding and price obscured issues)

Function Tested: Shelf details: set up a new item on this shelf
Test Cases: F5.1-F5.3
Test Status: All tests passed
Function Tested: Shelf details: Delete Items and reset settings
Test Cases: F6.1-F6.3
Test Status: All tests passed

Appendix 6: Arduino Sketch:

```
#include <SPI.h>
#include <WiFiNINA.h>
#include <MySQL_Connection.h>
#include <MySQL_Cursor.h>
#include <MySQL_Encrypt_Sha1.h>
#include <MySQL_Packet.h>
#include "HX711.h"

// Name: smart_shelf arduino sketch
// Author: Software Projects Group 24
// Version: 1.2
// Date updated:
// Notes: Designed to work with Arduino Uno Wifi Rev2, SparkFun HX711 ADC, 5kg Bar
Load Cell, MySQL Server 8 (legacy authentication).

// WIFI:

// Create WiFiClient object
WiFiClient client;

// Wifi Connection credentials and radio status
char ssid[] = "HOMEWIFISSID";
char pass[] = "XXXXXXXX";
int status = WL_IDLE_STATUS;

//MySQL:

// Create MySQL_Connection object
MySQL_Connection conn ((Client *)&client);

// IP Address of MySQL Server
IPAddress server_addr(192,168,X,XX);

// MySQL user details
char user[] = "testX";
char password[] = "cake123";

// MySQL Query
char INSERT_DATA[] = "INSERT INTO shelfdatav3.id1weights (weight) VALUES (%s)";
char query[128];
char weight[10];

// HX711 24-bit Analog-to-Digital Converter (ADC)
// Create HX711 scale object
HX711 scale;

// setup
void setup() {
 // start connection to serial port
 Serial.begin(9600);
 // Wait for serial port to be ready
 //while (!Serial);

 // HX711 scale set up
 Serial.println("Setting up scale...");
 scale.begin(A1, A0); // Connections from ADC to arduino pins: Data output: DAT =
```

```
pinA1, Clock input: CLK = pinA0

 // take and output the raw average of 20 readings to serial port
 Serial.print("Raw average of 20 readings: \t\t");
 Serial.println(scale.read_average(20));

 // Set the scale factor:
 // 458 is the scale factor value I obtained for a 5kg Load cell after calibrating
with known weights
 scale.set_scale(458.f);
 // Reset scale to zero
 scale.tare();

 Serial.println("\nScale set up complete!!!:");

 // output single raw scale reading to serial port
 Serial.print("Raw: \t\t\t");
 Serial.println(scale.read());
 // output raw average of 20 scale readings to serial port
 Serial.print("Raw ave(20): \t\t");
 Serial.println(scale.read_average(20));
 // output raw average of 5 scale readings minus tare weight
 Serial.print("Raw ave(5) - tare: \t");
 Serial.println(scale.get_value(5));
 // output calibrated average of 5 scale readings minus tare weight and divided by
scale factor(458)
 Serial.print("Calibrated ave(5): \t");
 Serial.println(scale.get_units(5), 1);

 // Commence wifi connection attempt
 while (status != WL_CONNECTED){
Serial.print("Attempting to connect to WPA SSID:");
Serial.println(ssid);
status = WiFi.begin(ssid,pass);
// delay of 10 seconds
delay(10000);
}

 Serial.println("Connected to network!!!");
 IPAddress ip = WiFi.localIP();
 long rssi = WiFi.RSSI();
 Serial.print("My IP address is: ");
 Serial.println(ip);
 Serial.print("Received signal strength(RSSI): ");
 Serial.println(rssi);


 // Connect to MySQL Server
 Serial.println("Attempting connection to MySQL Server...");
 if (conn.connect(server_addr, 3306, user, password)) {
Serial.println("Connection to MySQL Server established!!!");
delay(1000);
}
 else {
Serial.println("Connection to MySQL Server failed!!!");
conn.close();
}

}
```

```
void loop() {
 delay(5000);
 // re-check if MySQL Server is connected
 if(conn.connected()){
// calculate average of 20 scale readings:
// declare and initialize variables:
int t, i, n, T;
double val, sum, sumsq, mean, reading;
float stddev;
n = 20;
t = millis();
i = sum = sumsq = 0;
reading = 0;
// sum the readings
while (i<n) {
val = ((scale.read() - scale.get_offset()) / scale.get_scale());
sum += val;
sumsq += val * val;
i++;
}
// calculate time taken
t = millis() - t;
// calculate and output the mean and standard deviation of n readings
mean = sum / n;
stddev = sqrt(sumsq / n - mean * mean);
Serial.print("Mean, Std Dev of "); Serial.print(i); Serial.print(" readings:\t");
Serial.print(sum / n, 3); Serial.print("\t"); Serial.print(stddev, 3);
// Note: 2 sigma is 95% confidence, 3 sigma is 99.7%
Serial.print("\nTime taken:\t"); Serial.print(float(t)/1000, 3);
Serial.println("Secs\n");

// Insert data into database
Serial.println("Sending data to database...");
MySQL_Cursor *cur_mem = new MySQL_Cursor(&conn);
reading = sum/n;
dtostrf(reading,7,2,weight);
sprintf(query,INSERT_DATA,weight);
cur_mem->execute(query);
delete cur_mem;
Serial.println("Data recorded!");
}
else {
conn.close();
Serial.println("Reattempting connection to MySQL Server...");
if(conn.connect(server_addr, 3306, user, password)){
  delay(500);
  Serial.println("Reconnected to MySQL Server!!!");
  }
  else {
    Serial.println("Cannot reconnect to MySQL Server!!!");
    }
    }
}
```

Bibliography

Appendix 1: References:

1. Instructables Tutorial: Arduino Weighing Scales: https://www.instructables.com/id/How-to-Build-Arduino-Weighing-Scales/

2. Sparkfun Tutorial: HX711 Hookup Guide: https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide

3. HX711 Arduino Library: https://www.arduinolibraries.info/libraries/hx711-arduino-library

4. Eli the Computer Guy You Tube Video Tutorial: Write POST Data to Server with Arduino Uno with Wifi:https://www.youtube.com/watch?v=32VcKyI0dio&list=PLJcaPjxegjBUsCc8PDvalF9j9dvc1RpUh&index=26

5. Eli the Computer Guy You Tube Video Tutorial: Arduino Wifi Temperature Web Page Alert: https://www.youtube.com/watch?v=4IhSCSPQMA0&list=PLJcaPjxegjBUsCc8PDvalF9j9dvc1RpUh&index=28

6. MySQL Connector/Arduino Library: https://github.com/ChuckBell/MySQL_Connector_Arduino

7. Issue in conn.connect(server_addr, 3306, user, password) Arduino uno wifi rev.2: https://github.com/ChuckBell/MySQL_Connector_Arduino/issues/80

Appendix 4: References

1. Evert Pot article on own website: https://evertpot.com/executing-a-mysql-query-in-nodejs/

2. ‘Thad Blakenship’ on StackOverflow: https://stackoverflow.com/questions/23160743/how-to-send-flash-messages-in-express-4-0

3. Brad Traversy at Traversy Media: https://github.com/bradtraversy/nodeuploads

4. Maximilian Schwarzmüller at Academind: https://github.com/academind/node-restful-api-tutorial/tree/09-image-upload

5. Fabian N. on StackOverflow: https://stackoverflow.com/questions/17557807/javascript-how-do-you-sort-an-array-that-includes-nans

6. 'Josh' on StackOverflow: https://stackoverflow.com/questions/46067704/how-to-force-synchronous-loop-execution-in-javascript

7. 'robertklep' on StackOverflow: https://stackoverflow.com/questions/16098397/pass-variables-to-javascript-in-expressjs/16098699

8. 'radhika' on StackOverflow: https://stackoverflow.com/questions/17621515/how-to-show-and-hide-input-fields-based-on-radio-button-selection