
Durham Foodbank

Release 1.0.0

Group 12

Apr 15, 2020

CONTENTS

1	Introduction	1
1.1	Basic usage	1
1.2	Structure of the user manual	1
1.2.1	Installation	1
1.2.2	User Interface Walkaround	2
1.2.3	System Overview	2
1.2.4	Developer	2
1.2.5	Expansion	2
2	System Overview	3
2.1	Backend Overview	3
2.1.1	System Structure	3
2.1.2	Back-end Functions	4
2.2	Frontend Overview	6
2.2.1	Intro	6
2.2.2	Designer	7
2.2.3	Reports	7
2.2.4	StockTake	7
3	Installation	9
3.1	Authentication	9
3.2	Backend	10
3.2.1	Deployment	11
3.3	Database	11
3.3.1	Deployment	11
3.4	Frontend	11
3.4.1	Deployment	11
3.4.2	Git Repository	12
3.4.3	Command line	12
4	User Interface Walkaround	13
4.1	Overview	13
4.1.1	Login	13

4.1.2	Sign in page	14
4.1.3	Side Bar	14
4.1.4	Signing Out	15
4.2	Reports	15
4.2.1	What you can do	15
4.3	Stock Taking	19
4.3.1	Overview	19
4.3.2	Navigating the warehouse	20
4.3.3	Selecting trays	21
4.3.4	Swapping two trays within a bay	21
4.3.5	Modifying contents of selected trays	21
4.3.6	Modifying expiry date of selected trays	21
4.3.7	Modifying Weight of selected trays	22
4.3.8	Undo/Redo	22
4.4	Warehouse Manager	22
4.4.1	Overview	22
4.4.2	Actions to Perform	22
5	Expansion	27
5.1	API Expansion	27
5.1.1	Writing Back-end Functions	27
5.1.2	Routing Requests	28
5.1.3	Making Requests	29
5.1.4	Recompiling	29
5.1.5	Additional Information	30
5.2	Authentication Expansion	30
5.2.1	Adding and removing users	30
5.2.2	Allowing social integration	30
5.2.3	Customising the login page	31
5.3	Backups	31
5.3.1	Setup	31
5.4	Database Expansion	31
6	Developer	33
6.1	Tray functions	33
6.1.1	addTray	33
6.1.2	removeTray	34
6.1.3	switchTray	35
6.1.4	editTray	36
6.1.5	moveTray	37
6.1.6	nextExpiring	37
6.1.7	getAllCategory	38
6.1.8	addTrayMany	39
6.1.9	editTrayMany	39
6.1.10	removeTrayMany	40
6.2	Bay functions	40
6.2.1	addBay	40
6.2.2	removeBay	41
6.2.3	editBay	42
6.2.4	getTraysInBay	43
6.3	Zone functions	43
6.3.1	addZone	43
6.3.2	editZone	44
6.3.3	getBaysinZone	45

6.3.4	getZones	45
6.3.5	removeZone	46

CHAPTER

1

INTRODUCTION

Welcome to the user manual for the Durham Foodbank stock management system. This manual covers the basics through to more advanced content.

1.1 Basic usage

The website for this can be found [here](#).

All the credentials are the same and can be found in the credentials.txt file in our handover folder

1.2 Structure of the user manual

1.2.1 Installation

This section walks you through how to install and host the necessary components of the system so that it can be used in production. The four components are the front end, the back end, the database, and authentication. Each component has its relevant subsection with instructions on installation and deployment. Where relevant, there are links directing to outside documentation that will aid you in this process.

1.2.2 User Interface Walkaround

With the system up and running, this section will guide you through on how to use the system and its key functions. Each step is accompanied by pictures to aid you in using the system.

1.2.3 System Overview

This section provides a quick overview of the two main parts of the system - the front end and back end. This section adds context to the rest of the document, which delves into the more technical aspects of the system.

1.2.4 Developer

This is where the API documentation for the backend is described. Every function in the backend is split into three categories: tray functions, bay functions, and zone functions. Each function documentation describes the API URL, the input it receives, the parameters it requires, and the output it provides.

1.2.5 Expansion

Should you wish to develop the system further, the Expansion section details the structure of the back end functions, as well as information on adding to the database, and authentication.

It is recommended that someone with development experience works on expansion of the system.

CHAPTER

2

SYSTEM OVERVIEW

2.1 Backend Overview

2.1.1 System Structure

The whole system composes of three primary components: the MongoDB Database, the Express Server (which I will also refer to as the back-end server) and any number of clients running the front-end program.

The MongoDB database is where the data describing the contents and structure of the warehouse will be stored so that it can be accessed and edited by warehouse employees. We chose to use MongoDB as it is both easily extensible and flexible. Details on the exact structure of the database can be found in the Installation section.

The express server acts as a middle-man between the clients and the MongoDB and provides a number of useful API functions to interface with the database. It provides error handling between the client and server, and between the server and the database, ensuring that the whole system remains robust and to ensure consistency. Details on how to install and deploy the express server are also in the Installation section.

2.1.2 Back-end Functions

In this section I will give a brief overview of all back-end functions in `routes/stockTake.js`

addTray

When provided with a tray object and a MongoDB database object, this function will add the contents of the tray object to the database.

editTray

When provided with a tray object and a MongoDB database object, this function will update the contents of an existing tray object in the database, provided it exists.

removeTray

When provided with the location of a tray object and a MongoDB database object, this function will delete the tray from the database.

switchTray

When provided with the location of two different tray objects and a MongoDB database object, this function will swap the positions of two trays in the database.

addTrayMany

When provided with an array of tray objects and a MongoDB database object, this function will add all the tray objects to the database in a single command. This should be used if many trays are added at once, as adding them one-by-one will take an unacceptable amount of time.

editTrayMany

When provided with an array of tray objects and a MongoDB database object, this function will edit all the tray objects that are in the database in a single command. This should be used if many trays are edited at once, as editing them one-by-one will take an unacceptable amount of time.

removeTrayMany

When provided with an array of tray locations and a MongoDB database object, this function will remove all the trays at the positions specified. This should be used if many trays are deleted at once, as deleting them one-by-one will take an unacceptable amount of time.

getAllCategory

When provided with a category and a MongoDB database object, this function will return all trays with a matching category. This can be used to quickly obtain the locations of items of a specific type.

getNextNExpiring

When provided with a number N, an optional category argument and a MongoDB database object, this function will return the next N expiring trays. If a category is specified, it will return the next N expiring in that category only. This can be useful if we wish to find the items closest to expiring so that we can focus on taking them from the warehouse first.

getZones

This function, when provided with a MongoDB database object, simply returns all zones in the warehouse.

addZone

When provided with a zone object and a MongoDB database object, the zone will be added to the database. This can be used in the initial creation of the warehouse and to add zones temporarily when there is a need to meet a higher than normal capacity.

editZone

When provided with a zone object and a MongoDB database object, the zone in the database will be edited, provided that it does exist.

removeZone

When provided with a zone location and a MongoDB database object, the zone will be removed from the database.

addBay

When provided with a bay object and a MongoDB database object, the bay will be added to the database.

editBay

When provided with a bay object and a MongoDB database object, the bay will be edited, provided it does exist.

removeBay

When provided with the location of the bay and a MongoDB database object, the bay will be removed from the database.

getTraysInBay

When provided with the location of a bay and a MongoDB database object, the function will return a list of all tray objects inside that bay, provided the bay exists. This will be used to display bay contents in the front-end application.

getBaysInZone

When provided with the location of a zone and a MongoDB database object, the function will return a list of all the bay objects inside that zone, provided the zone exists.

moveTray

When provided with two tray locations and a MongoDB database object, the function will move the tray from one location to another provided a tray exists in the start location and does not exist in the end position.

mongoUpdate

This function takes a request body and a method code. It will first connect to the MongoDB database to get the database object. Then, using the method code, it will pass the request body to one of the functions described above. It is then responsible for handling errors that occur and returning the result to the front-end.

2.2 Frontend Overview

2.2.1 Intro

The front end is built with [ReactJS](#) an object orientated system aimed at building great user interfaces.

- It has a unique structure where in each item seen on the ui is effectively a react class component.
- These components contain subcomponents. some of these have been built buy us, others have been imported from a boot-strap component library.
- We use two of these, [Grommet](#) and [React bootstrap](#), these are primarily used for buttons and styling that have the benefit of improving user experience and decreasing development time.

The integral components that have been built by us are:

2.2.2 Designer

The component is a tool to build the warehouse, add remove, resize zones and their respective bays, this is a very simple component visibly, however functionally it is integral. This component uses Api Endpoints:

- addTray
- removeTray
- addTrayMany
- removeTrayMany
- getZones
- addZone
- editZone
- removeZone
- addBay
- editBay
- removeBay
- getTraysInBay
- getBaysInZone

2.2.3 Reports

This component takes and manipulates the data received from the api, the component interfaces with the server. This component uses Api Endpoints:

- getNextNExpiring
- getAllCategory
- getZones
- getTraysInBay
- getReports
- publishReport

2.2.4 StockTake

This component acts as a data structure, fetching and syncing with the server, the sub components below display the data stored in StockTake. The component itself not at all complex, however its interactions with the sub components and them with each other is quite a lot more complex than the sum of the api calls. This component uses Api Endpoints:

- editTrayMany
- editBay
- getZones
- getTraysInBay
- getBaysInZone

Bayview

This component displays a collection of tray items; which represent individual trays, it stores a list of the selected trays, and allows the manipulation of tray items, by means of category buttons and several forms.

CategoryButtons

this component displays and handles the change of category.

trayItem

a graphical representation of an individual tray and the event handler for when an individual tray is selected.

CHAPTER

3

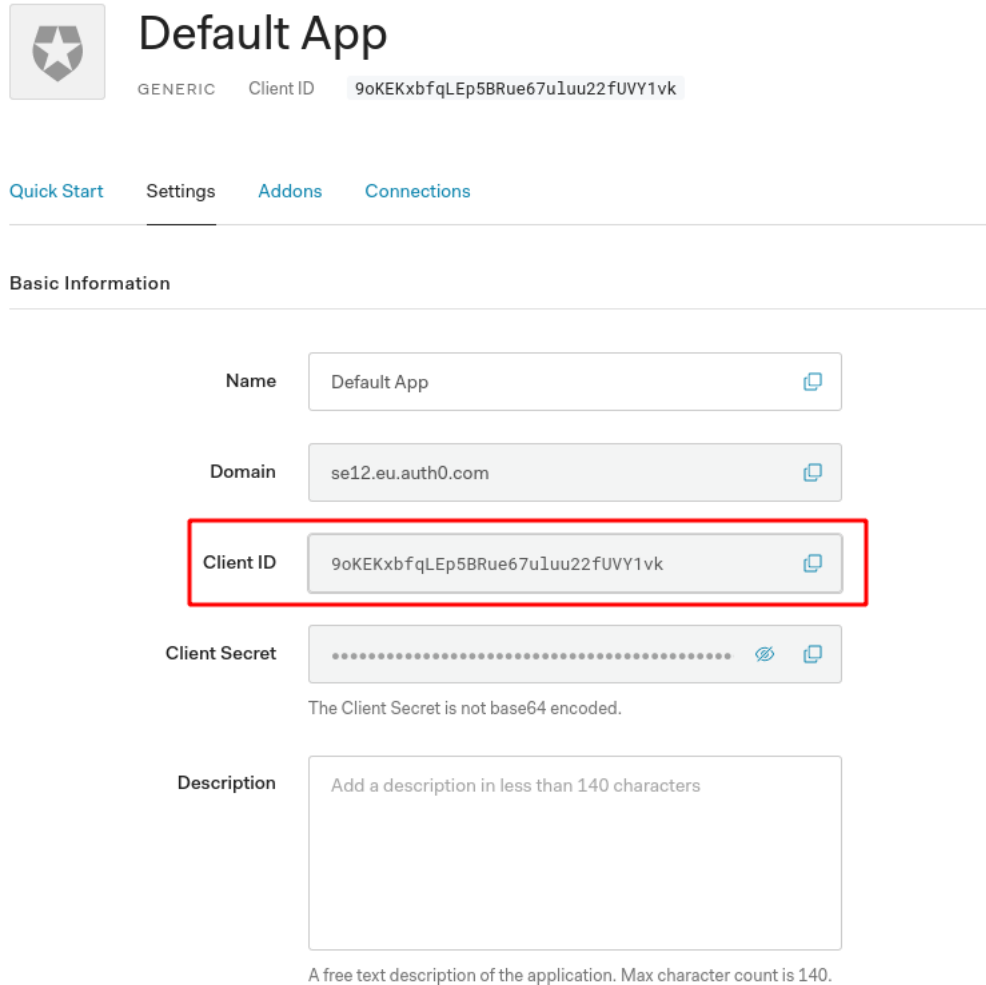
INSTALLATION

3.1 Authentication

Authentication for the website is handled through [Auth0](#) the free tier allows for up to 7000 users, which should be more than ample.

The process of transferring the account to one that you own is very simple:

1. Sign up for Auth0 [here](#)
2. Under the applications tab on the dashboard create a new Generic application
3. Copy the client ID from the application, as pictured below



Default App
GENERIC Client ID 9oKEKxbfqLEp5BRue67uluu22fUVY1vk

[Quick Start](#) [Settings](#) [Addons](#) [Connections](#)

Basic Information

Name Default App

Domain se12.eu.auth0.com

Client ID 9oKEKxbfqLEp5BRue67uluu22fUVY1vk

Client Secret
The Client Secret is not base64 encoded.

Description Add a description in less than 140 characters

A free text description of the application. Max character count is 140.

4. Paste this value under `clientId` in `auth_config.json`, which is located in the `src` folder of the frontend server
5. Also change the domain in the `auth_config.json` file, this domain can also be found in the application, as shown below
6. Disable signups by going to the connections tab of the dashboard, clicking on Username-Password-Authentication, then turning on the disable signups option as shown below

This is done as otherwise any user could come and sign up for the system without permission

3.2 Backend

In our handover there is a folder named `backend`, in this is stored the server that manages the requests made in the website. It is built using Node.js and Express and can be started with `npm install` followed by `npm start`, the server will then be hosted on port **3001**

Note that the frontend and backend should be run in separate terminal windows.

When both the front end have been started, requests can be made between them

3.2.1 Deployment

Our recommendation for deploying this is on [heroku](#) as it has a good free tier and has suitable pricing plans for expanding if usage increases.

It is also simple to set up:

1. Create an account [here](#)
2. Click the new button on your dashboard to create a new application
3. Give it a name and set the region to Europe
4. Follow the on screen instructions to get your code deployed

3.3 Database

For this application to run it also needs a database set up, for this we have chosen [MongoDB](#).

3.3.1 Deployment

Our recommendation for deployment is to use [MongoDB Atlas](#) this has a free tier but allows for scaling up to a paid plan if your requirements exceed the limits of the free tier.

To get started sign up [here](#) then choose any of the cloud providers, and select a local cluster with a free tier

Once this is done, a particular structure of the database will need to be established if you don't want to change the code

1. Create a database named `foodbank`
2. Create 4 collections, `bays`, `dummy`, `food` and `zones` in that database

If you do want to change the names of the database or collections, please consult the expansion section of this user guide.

3.4 Frontend

In our handover there is a folder named `frontend`, in this is stored the user interface with the program. It is a react app, and can be started with the command `npm install` followed by `npm start`, you will then be taken to a browser window with the application loaded.

Note that the frontend and backend should be run in separate terminal windows.

3.4.1 Deployment

The code can be deployed using any static hosting, our recommendation is [ZEIT Now](#) as they have a very generous free tier and it is very easy to get set up. There are two main ways of getting set up

3.4.2 Git Repository

If you don't know what this is, then proceed to the other option.

For continuous development, this is a better option as any changes you make will be automatically deployed to the website.

This can be done in 3 steps

1. Add the files to a GitHub, GitLab or Bitbucket repository
2. Create an account [here](#)
3. Link together the repository and ZEIT by clicking `Import Project`

3.4.3 Command line

1. Install npm using the instructions [here](#)
2. Create a ZEIT account [here](#)
3. Install the Now CLI using the instructions [here](#)
4. The code can then be deployed with the command `now`, as documented [here](#)

CHAPTER

4

USER INTERFACE WALKAROUND

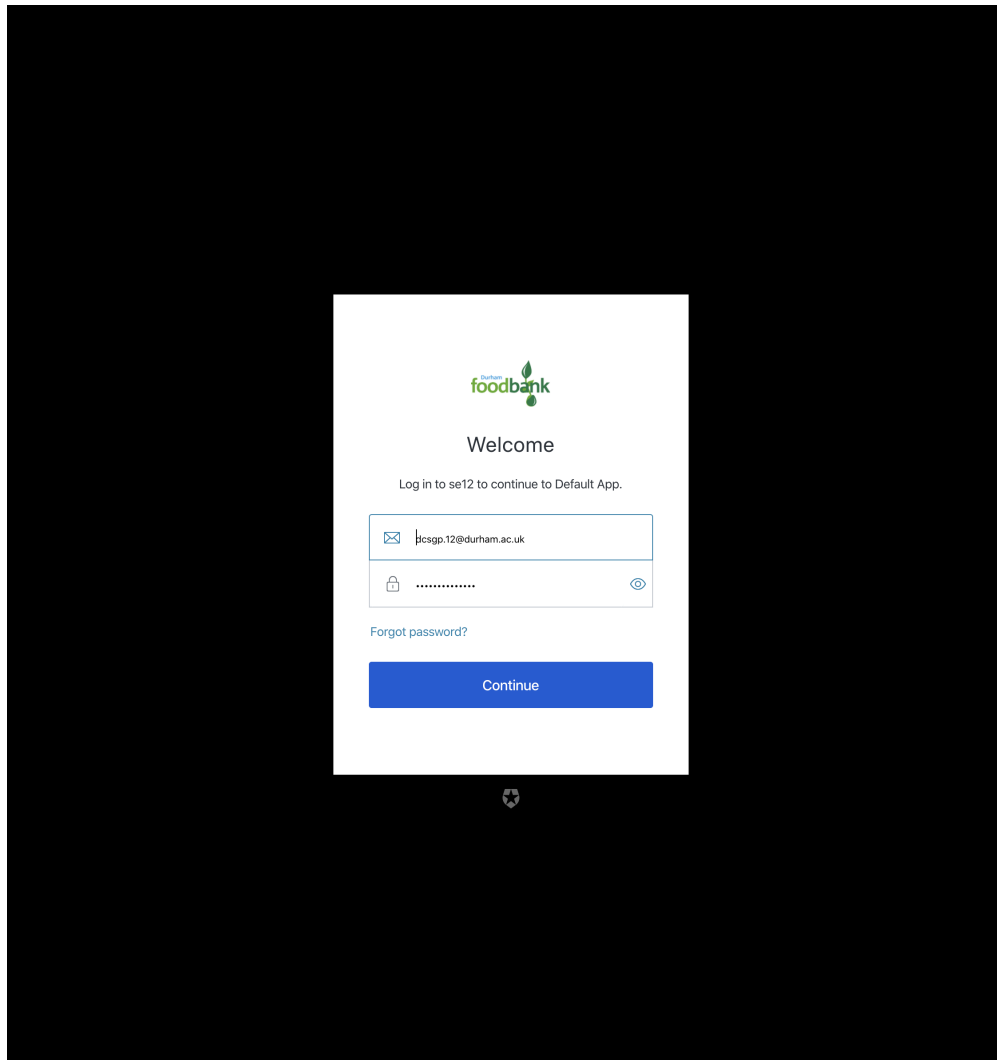
4.1 Overview

4.1.1 Login

To sign in, Click on the hamburger menu button in the top left hand corner and click login.

4.1.2 Sign in page

Once you've clicked login, you will be redirected to Auth0's signin page, you are prompted to enter the credentials either you created, or the development credentials provided.



4.1.3 Side Bar

You can click any icon or text to navigate to the respective target. To view the full menu you can click the hamburger menu in the top left hand corner to expand the side bar.

- By default the Documentation tab is always accessible with or without logging in.
- Once signed in, all warehouse features are accessible.

4.1.4 Signing Out

To Sign out, click on the circular user icon image in the sidebar, and then click log out.

4.2 Reports

Access by clicking reports in the side bar.

4.2.1 What you can do

- Search for all items that match a certain category, and where to find them.
- View a snapshot of the warehouse categorised by location.
- View a list of all trays ordered by their expiry date.
- export a csv file containing all the trays in the warehouse.
- Publish a report as a csv file, and keep it stored for future trend reporting.

- See a percentage of all free trays in the warehouse to get an accurate representation of how full the warehouse is.

The screenshot shows the Shelfie V2.0 interface with three main sections highlighted by red boxes and labeled with red arrows:

- Section 1 (Reports):** Contains a 'Publish Report' button (Control Panel 3) and an 'Export an old report' section (Control Panel 4) with a date/time selector and a 'Download' button.
- Section 2 (Trays occupied):** A circular gauge showing 35.2% occupancy.
- Section 3 (Sorted by expiry Date):** A table of inventory items with columns for Category, Expiry, Weight, Zone, and Bay. A 'Search by' dropdown (Control Panel 5) is located above the table.

Additional controls include a 'Search by' dropdown (Control Panel 1) and an 'Export to CSV' button (Control Panel 2) in the 'Sorted by Location' section.

Category	Expiry	Weight	Zone	Bay
Preserve	09/2019	3095	RED	A
Canned Fruit	09/2019	5792	RED	A
EMPTY	09/2019	3647	RED	A
UHT Milk	09/2019	6536	RED	A
Canned Beans	09/2019	243	RED	A
Biscuits	09/2019	6417	RED	A

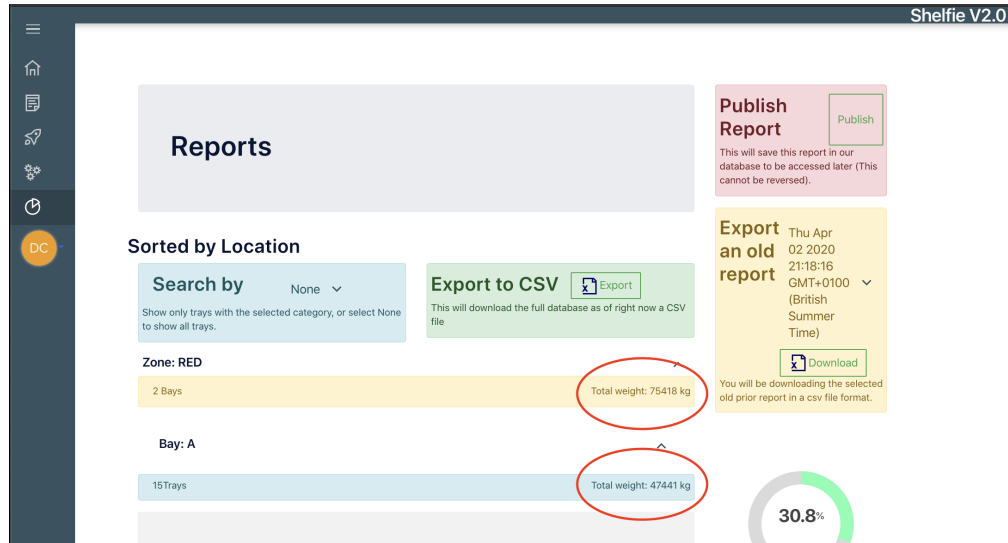
Overview

1. Section 1 on the diagram shows you the snapshot of the warehouse.
 - you can filter this snapshot to only show trays that contain a desired category by choosing it from the drop down menu in Control Panel 1
2. You can export the full database of trays at this point in time into a CSV file and download it.
 - You can do this by clicking Export in Control Panel 2.
3. You can save the snapshot of the warehouse on the server for guaranteed future access, by clicking Publish in Control Panel 3.
 - This cannot be reversed, so only publish if and when it is absolutely necessary to do so, e.g. on a weekly basis after stock is taken.

4. You can download published reports in a csv format by selecting the report that was published from the drop-down list in Control panel 4 and clicking the Download button in Control Panel 4.

Navigating the warehouse

- Clicking on the target zone to expand.
 - You will see the total weight of all products in the zone



- Followed by a list of bays that the zone contains, or the bays that contain a category to be searched for.
 - * Expanding on a bay will show you the full weight of trays in that bay, this can be useful to see if a bay is too heavily laden.
 - * Beneath is a list of all the trays that match the search criteria, (their weight, category and expiry).

Reports

Sorted by Location

Search by None ▾
Show only trays with the selected category, or select None to show all trays.

Export to CSV **Export**
This will download the full database as of right now as a CSV file

Publish Report **Publish**
This will save this report in our database to be accessed later (This cannot be reversed).

Export an old report Thu Apr 02 2020 21:18:16 GMT+0100 (British Summer Time) **Download**
You will be downloading the selected old report in a csv file format.

Zone: RED Total weight: 75418 kg

2 Bays

Bay: A Total weight: 47441 kg

15 Trays

Misc 1000 kg

Preserve 3095 kg

Canned Fruit 5792 kg

EMPTY 3647 kg

UHT Milk 6536 kg

Canned Beans 243 kg

Bay: B ▾

Zone: YELLOW ▾

Zone: GREEN ▾

Zone: INDIGO ▾

Sorted by expiry Date **Search by**

35.2%
Trays occupied

Click here to expand upon "RED" Zone

Click here to expand upon Bay "A", in the "RED" Zone

- Note: you will have to expand on each accordion manually to see the contents. This is meant to create a clean environment to view the data, if this is not suitable you can:
 - export to CSV,
 - Open the file in Microsoft Excel, and apply filters to each column and perform a search based on any criteria, sort by any field. A full feature list can be found here: [Help with Microsoft Excel Filters](#)

Determine how full the warehouse is

- The progress wheel in `Section 2` shows the percentage of the tray spaces in the warehouse that have been declared as empty on the most recent stock take.

View an ordered list of the next Expiring trays

In `Section 3` of figure 1, you can see an ordered list of all the trays in the warehouse.

- You can filter the list by a chosen category in drop down menu within `Control Panel 5`.

Category	Expiry	Weight	Zone	Bay
Cherios	03/2020	5792	YELLOW	B
Canned Fruit	03/2020	7788	YELLOW	E
EMPTY	03/2020	1662	INDIGO	C
EMPTY	03/2020	7343	INDIGO	C
Canned Fruit	04/2020	7064	YELLOW	A

4.3 Stock Taking

Access by clicking Stock Take in the side bar.

4.3.1 Overview

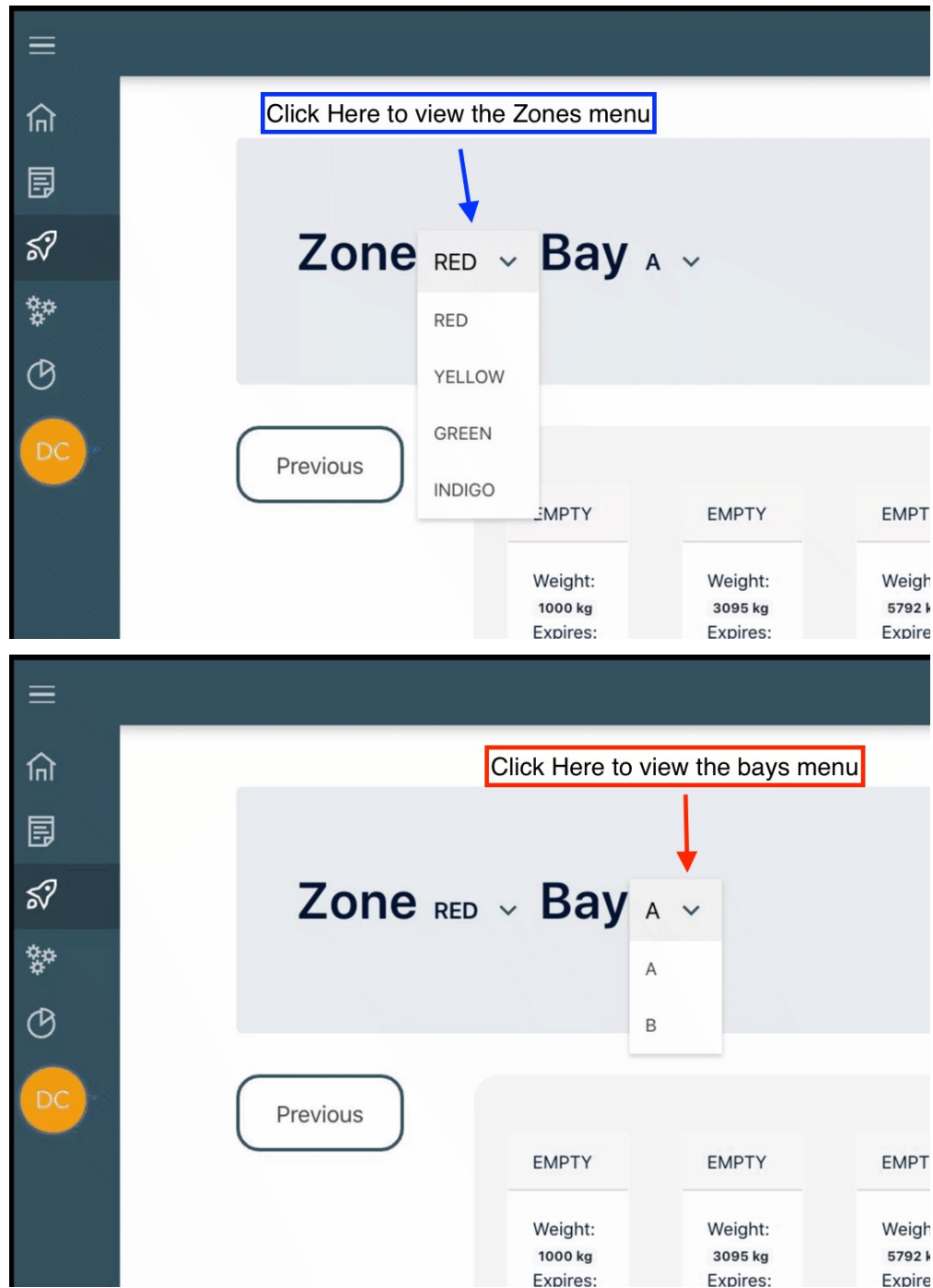
Whenever a change is made to any tray, these changes are immediately synced with the server, and so you can have multiple users taking stock and these changes will be reflected accross all devices that are active.

You can choose to take stock whenever you want, if you wish to only count a single bay you can do so.

There will be a loading screen, it will only take about half a minute depending on your internet connection. Please be patient as the whole database is being pulled, this will help you save time while counting.

4.3.2 Navigating the warehouse

1. Start by selecting the zone you are taking stock in.



2. The default bay will be the one added first when designing the warehouse.
 - You can define your ideal tour through a zone when building your warehouse.
 - This will help you rely on the next and previous buttons, to save time.
3. You can navigate between bays by clicking the next and previous buttons .

4.3.3 Selecting trays

A single left click on the desired tray, will high light it in blue, indicating it's selection. Deselection is just as easy, just click again to deselect it.

If you wish to select all the trays in the bay, you may click the `Select All` button.

note: once a modification occurs on any selected tray it is automatically de-selected; this again is meant to save you time.

4.3.4 Swapping two trays within a bay

You may only swap if you only have two trays selected. Clicking this will swap the contents of the two trays, their expiry dates and their weights.

Note: you can only do this for trays within a given bay. The backend supports cross bay/zone swaps and moves however due to time limitations in the project it self, these features were ommited as they were too unstable to implement in the front end.

4.3.5 Modifying contents of selected trays

Once the trays to be modified are selected, navigate to the category tab in the bottom half of the interface. Then you can:

- Choose a `EMPTY` to consider this tray as an empty, however you may still have weight and expiry attributes as requested by the client.
- Choose a category that you have chosen before by any of the quick toggle category buttons.
- Manually type in a category and click set.
 - node: you will not immediately see this custom category show up in your quick toggle categories, you will need to refresh the page to do so; however `EMPTY` will always be in the quick toggles.

4.3.6 Modifying expiry date of selected trays

Choose your selected trays, navigate to the expiry tab in the interface and then:

- Choose `None` to give no expiry date,
- to give only a month as an expiry date, select `None` followed by a desired month.
- To choose an expiry year simply give a desired year,
 - immediately followed by a month to give an expiry date in the `MM/YYYY` format.

4.3.7 Modifying Weight of selected trays

Choose your selected trays, navigate to the Weight tab in the interface and then:

- Simply type the desired weight as a number, with or without a decimal, and click update.
- or Click `None` to give a tray no weight.

4.3.8 Undo/Redo

You can undo the changes you just made in any zone/bay by clicking undo, and continue to redo. changes are saved locally and made on the server if you choose to undo changes that were made.

4.4 Warehouse Manager

Access by clicking Warehouse Manager in the side bar.

4.4.1 Overview

When loaded, this page collects all of the zones in the database, as well as the bays contained within them, and displays them to the user in an accordion style menu.

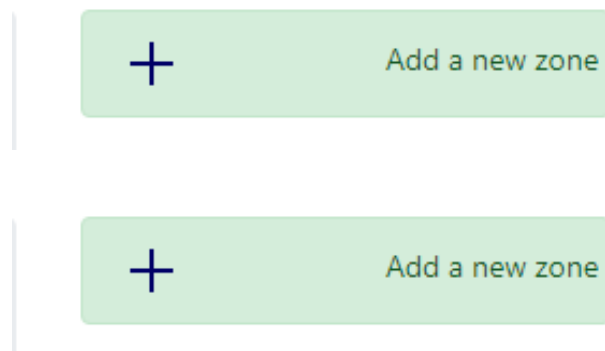
On the left is a wheel that displays the percentage of trays filled out of the total potential number of trays in the warehouse, and also lists the exact number of trays that are empty.

Whenever a (valid) change is made to any of these zones and bays, the page will be refreshed accordingly to display the changes.

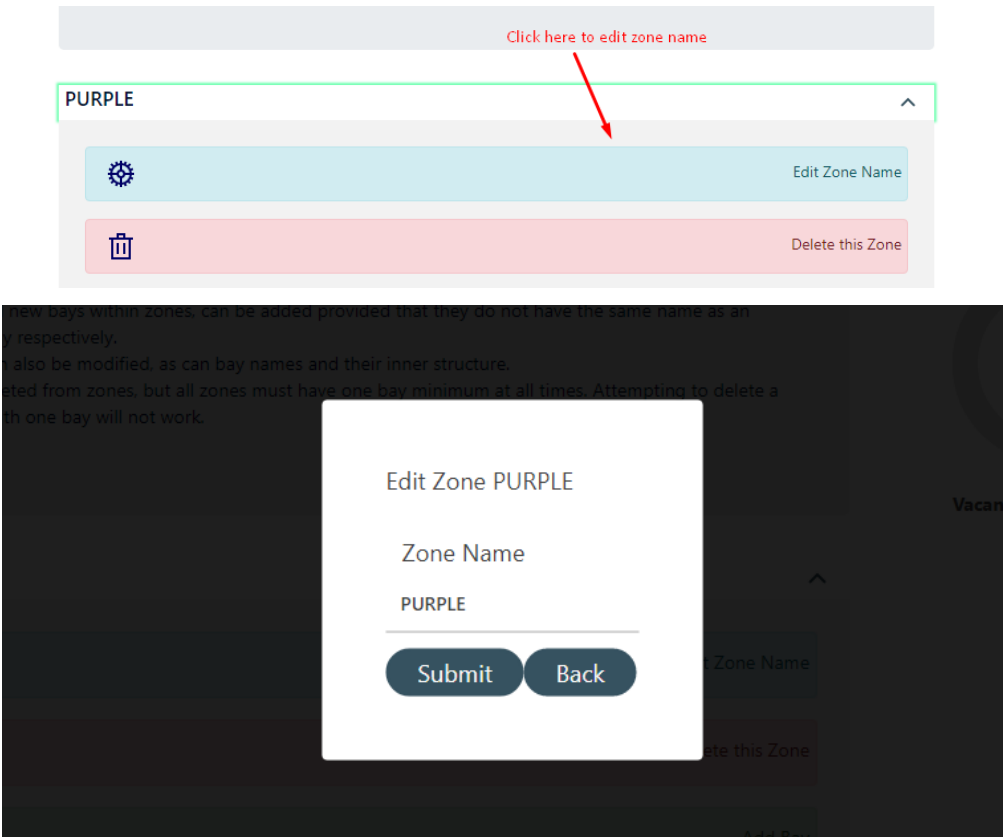
There are various types of changes and actions that you can perform to the zones and bays, as follows.

4.4.2 Actions to Perform

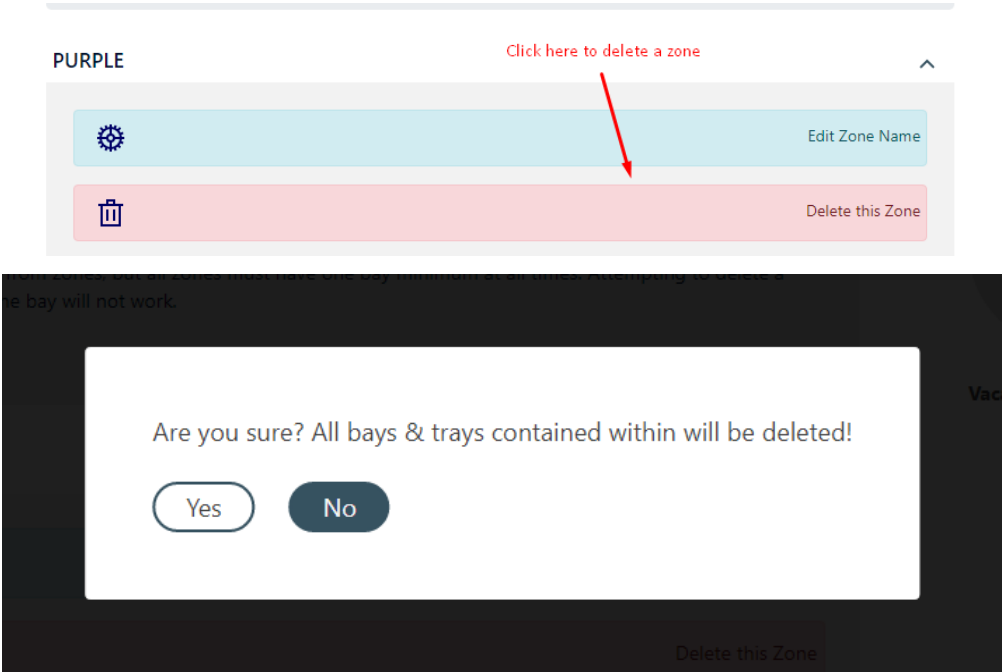
1. A zone can be added by selecting the button on the far right and entering valid data into the dropdowns. Zone names must be unique.



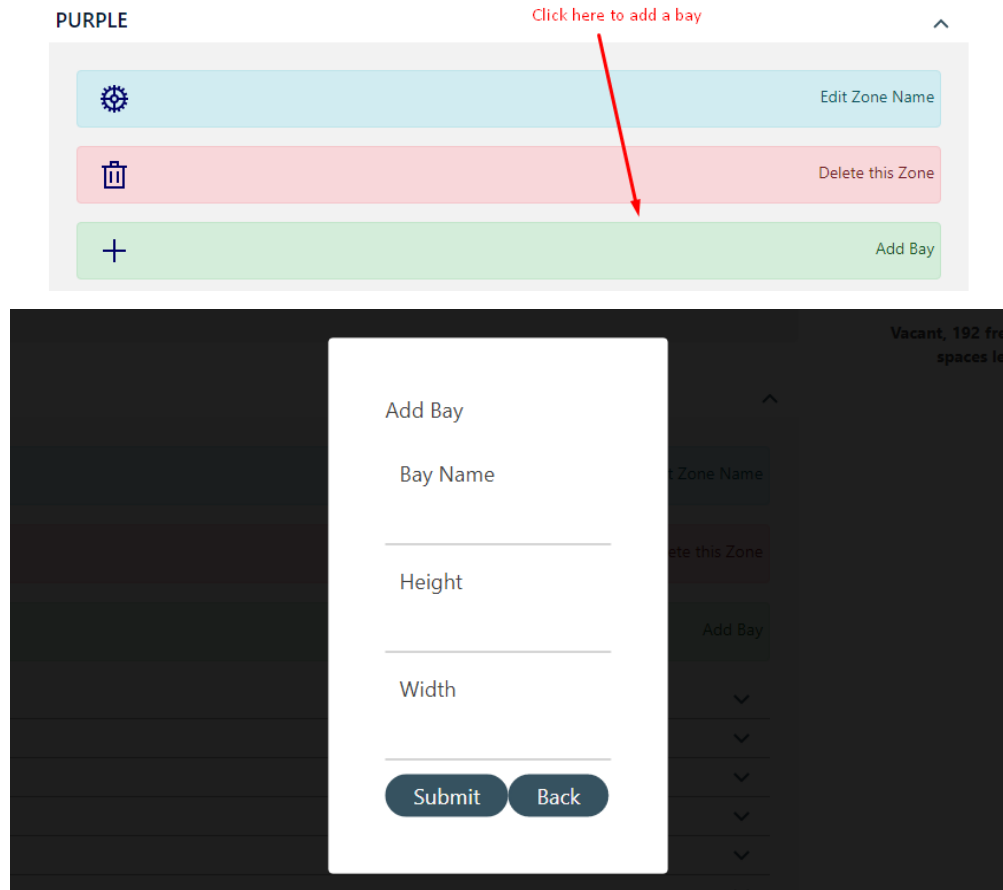
2. Zone names can be changed by selecting a zone in the dropdown, then clicking the edit zone name. Enter a zone name that is currently not in use.



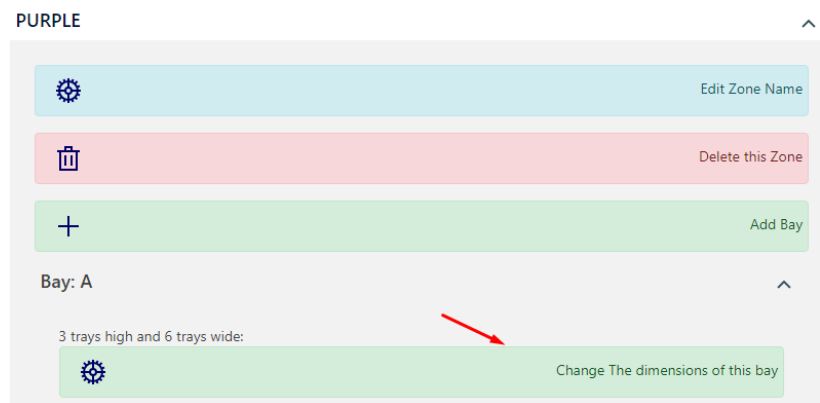
3. Zones can be deleted by clicking the delete zone button in its dropdown and then confirming.
- NOTE: All bays and trays contained inside the zone will also be deleted.

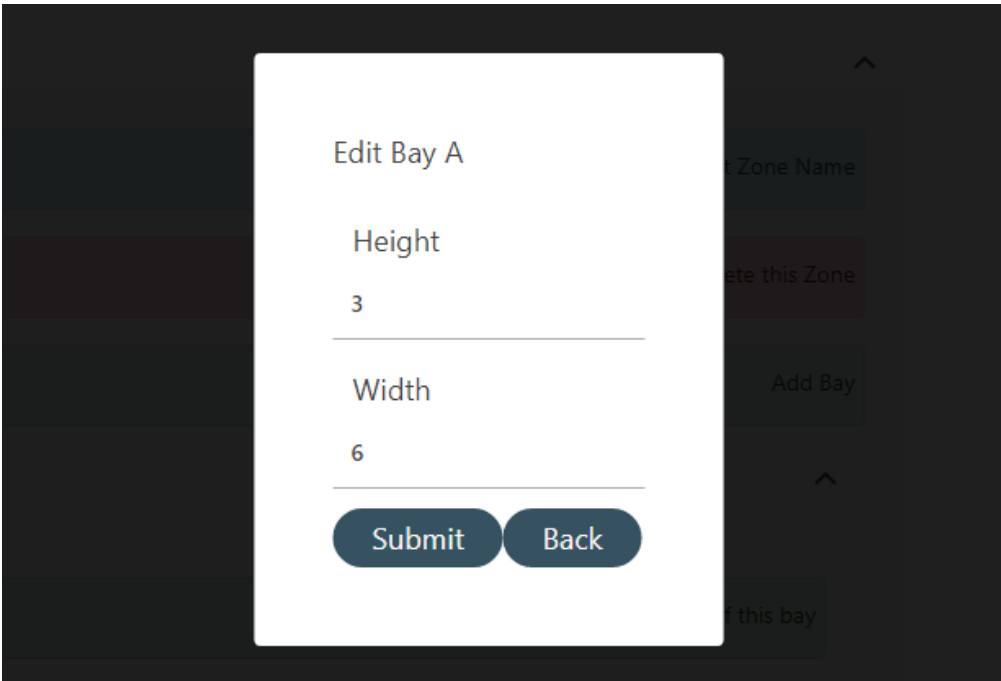


4. Bays can be added to any zone by clicking on the zone you wish to add the bay to and entering valid data. Bay names must be unique within zones.



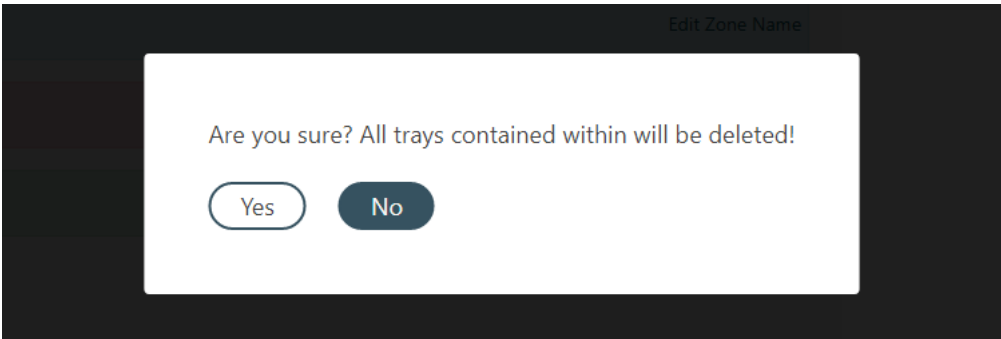
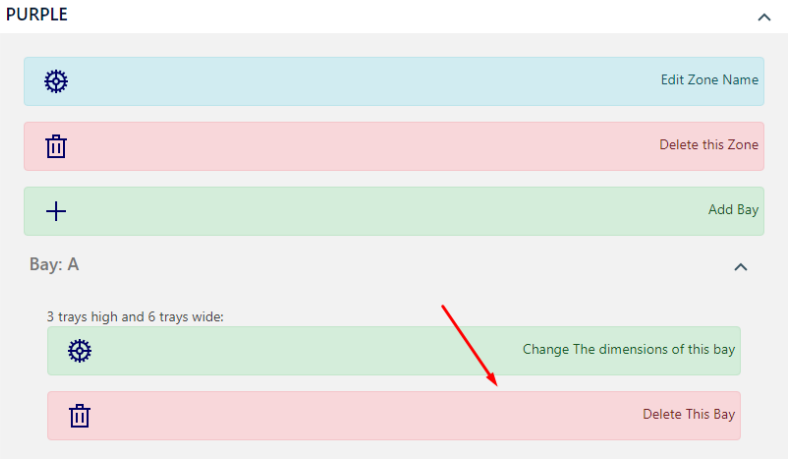
5. Bay dimensions can be altered by selecting a bay inside a zone then entering new values.





6. Bays can also be deleted - provided that it is not the final bay in a zone.

- NOTE: All trays contained inside the bay will also be deleted.



CHAPTER

5

EXPANSION

5.1 API Expansion

Once the handover has been completed, you may wish to add your own API endpoints to expand the functionality of the application further. In this document I will detail how to do this. This document is aimed towards technical users who have some experience with javascript before.

5.1.1 Writing Back-end Functions

Most of the back-end functions are written so that they can be tested with only a connection to a MongoDB server (No connection to the front end required). Connections with the front end and MongoDB server are handled by other functions in `routes/stockTake.js`, so when writing back-end functions we can assume that we have an existing connection to a MongoDB server and that some other function will handle communicating the result to the front end.

With this in mind, all the back-end functions take parameters `body` and `dbo` only, where `body` is a JSON object representing the request body and `dbo` is a MongoDB database object. The parameters of `body` will depend on the function being implemented. When writing new back-end functions, it is a good idea to emulate this, however it is not enforced.

Back-end functions are typically in the following format:

```
async function functionName(body, dbo) {  
  // Check if the body is well formed  
  if (!checkBody(body)) {  
    return "FAIL";  
  }  
  
  try {  
    // Build a request if the request is more complicated  
    let req = buildRequest(body);  
  
    // Call one, or many MongoDB functions
```

(continues on next page)

(continued from previous page)

```
    let res = await dbo.collection("food").someMongoDBFunction(req);

    // Handle the response, returning "FAIL" if necessary
    handleResponse(res);

  } catch (ex) {
    return "FAIL";
  }

  // either return "SUCCESS" or some data that was requested.
  return "SUCCESS";
}
```

For most cases this rough structure will be sufficient. The function must return either “FAIL”, “SUCCESS” or the requested data in order to be handled correctly by its calling functions.

5.1.2 Routing Requests

Within `routes/stockTake.js` there is a function called `mongoUpdate` that handles the calling of the created back-end functions, as well as the connection to the MongoDB server. It also handles the returning of data and error codes to the correct API endpoint. In order to use this for your new function, you will need to add an entry to the switch statement contained within this file.

This is simple to do, add the following code:

```
switch "functionID":
  code = await functionName(body, dbo);
  break;
```

to the switch statement.

The next step is to add a new router endpoint. This differs depending on what type of function you have implemented.

Functions that get something with no parameters

```
router.get('/functionEndpoint', async function (req, res, next) {
  let result = await mongoUpdate(req.body, "functionID");
  res.setHeader('Content-Type', 'application/json');
  res.status(200).send({'result': result});
})
```

Functions that get something with parameters

```
router.post('/functionEndpoint', async function (req, res, next) {
  let result = await mongoUpdate(req.body, "functionID");
  res.setHeader('Content-Type', "application/json");
  res.status(200).send({'result': result});
})
```

Functions that make changes and only expect success/failure as result

```
router.post('/functionEndpoint', async function(req, res, next) {
  let code = await mongoUpdate(req.body, "functionID");
  if (code !== "SUCCESS") {
    res.sendStatus(400);
  } else {
    res.sendStatus(200);
  }
})
```

Your back-end function should now be able to serve requests from the client.

5.1.3 Making Requests

We now need to add the end-point we have created to the public API so we can access it from the front-end server. The code for accessing the existing endpoints is in `public/api.js`. When this is imported by a front end program or part of the html of a website the functions can be called from there.

API functions are less strictly formatted compared to the back-end functions and can take any number of parameters. They should be written in a way that assists the developer when writing code by abstracting the awkward nature of hand writing JSON bodies.

Typically, the API functions should follow the following structure:

```
async function functionName(arg_1, arg_2, ..., arg_n) {
  let req = buildBody(arg_1, arg_2, ..., arg_n);
  let res = await fetch(URL + functionEndpoint, {
    method: 'GET or POST',
    mode: 'cors',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(req)
  });

  if (res.ok) {
    return "OK";
  }
  return "FAIL";
}
```

5.1.4 Recompiling

Javascript is a scripting language and does not require recompiling after changes have been made. However, changes to the back-end will require a restart of the back-end server before the changes will be reflected. Changes will also need to be redeployed to whatever hosting service you have chosen to use.

5.1.5 Additional Information

The above information should allow technical users to make simple changes to the back-end so they can add their own features. For more complex changes, you should consult the MongoDB documentation (for communication with the MongoDB database) and the express server and node documentation (for communication between the front and back end).

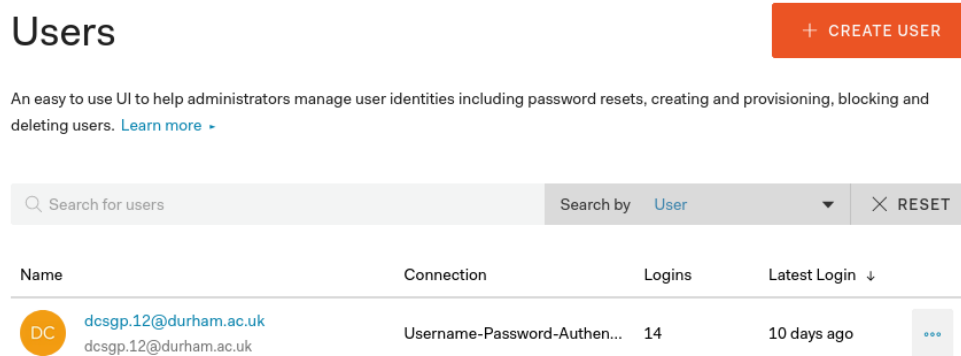
5.2 Authentication Expansion

Before expanding the authentication, it should first be installed, this can be done by following the guide in the installation section of this user manual.

5.2.1 Adding and removing users

One of the most common functions when expanding will be to add and remove users. This can be done from the Auth0 dashboard.

On the sidebar, click on the **Users & Roles** tab, this will take you to the following screen



Adding a user can be done by clicking the **CREATE USER** button, and removing a user can be done by clicking the three dots next to a given user.

5.2.2 Allowing social integration

Social integration is a very popular choice, such as signin with Google as it means you don't have to remember passwords.

This can easily be done in Auth0 by clicking on the **Connections** tab followed by the **Social** tab, then enable the switch of the social providers you want.

5.2.3 Customising the login page

This is something that would most likely be done when the company logo is updated to ensure brand consistency. From the dashboard, click on the **Universal Login** tab, and this will allow you to specify a company logo and colours.

5.3 Backups

This page will detail how to set up backups for your database if you have chosen MongoDB as your database of choice.

5.3.1 Setup

It is possible to create backups and restore data using the tools provided with MongoDB. However, this requires knowledge of command line instructions and will likely be difficult for users without this knowledge. You can find more information [here](#) on how to perform backups in this way.

The second way of creating backups is using MongoDB Atlas. However, this service is not available on the free tier (M0) and will require a paid plan, the pricings of which can be found [here](#).

If you choose an M2 or M5 cluster, Atlas will create daily snapshots of the cluster. They will be taken automatically, starting 24 hours after the creation of the cluster. These snapshots can then be viewed in a table. From there, you can restore or download your existing snapshots. Atlas will retain the last 8 daily snapshots. The full documentation on this can be found [here](#).

If you require custom policies (e.g. weekly backups), then you will need an M10 or larger cluster. It is then possible to create different backup policies and modify the retention time of these snapshots. Additionally, you can take on-demand snapshots, which occur immediately instead of at regular intervals. Full information can be found [here](#)

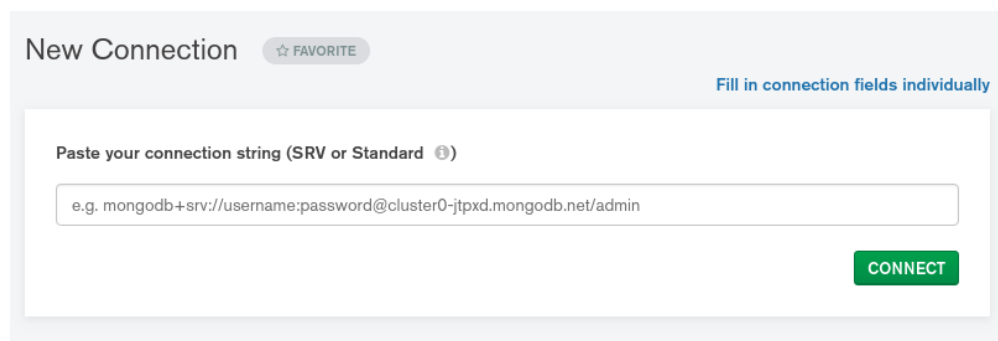
Please note that you will incur extra costs for Cloud Provider Snapshots if you are using an M10+ tier.

5.4 Database Expansion

The database we have chosen is [MongoDB](#) some details on the basic setup of this are included in the database installation section.

The easiest way to expand the database is using [MongoDB Compass](#) which is available for all operating systems. The full documentation for this software is available [here](#).

When you open the program you will be greeted with this input:







We will include this URL in the handover package, it can also be found in the `routes/stocktake.js` file of the backend assigned as URL.

Once you have connected, you will see a screen showing the databases, below is a screenshot of what this looks like for the default setup.

Databases

Performance

CREATE DATABASE

Database Name ^	Storage Size	Collections	Indexes	
admin	0.0B	0	0	
config	0.0B	1	0	
foodbank	81.9KB	4	4	
local	0.0B	6	0	

If you want to add more features to the database you can use `CREATE DATABASE` to add a new database for storing more information.

In each database there are collections, which store your data, you can access the collections in a database by clicking on the name of the database. If you want to add more collections, this can be done here in the same way as adding a database.

By clicking on a collection, you can see the stored data, this is useful for testing purposes as it allows you to immediately see the changes made to the data.

CHAPTER

6

DEVELOPER

6.1 Tray functions

6.1.1 addTray

Adds a new tray to the warehouse.

URL

```
POST /stockTake/addTray
```

Input

```
{
  "zone": "Zone specified",
  "bay": "Bay specified",
  "tray": "Tray specified",
  "contents": "What is in the tray",
  "weight": "How much does it weigh",
  "expiry": "When does it expire",
  "xPos": "The x position of each tray in the bay, i.e. leftmost tray has xPos = 0",
  "yPos": "The y position of each tray in the bay, i.e. topmost tray has yPos = 0"
}
```

Parameters

Parameter	Type	Description
zone	string	The zone in which the tray is
bay	string	The bay in which the tray is
tray	string	The tray to edit
contents	string	What is in the tray
weight	integer	The weight of the contents (give as a number in Kilos)
expiry	string	The expiry date of the food
xPos	integer	The horizontal positional element of the tray within its bay
yPos	integer	The vertical positional element of the tray within its bay

Output

OK if successful

FAIL if unsuccessful

6.1.2 removeTray

Removes a tray from the warehouse.

URL

http request POST /stockTake/removeTray

Input

```
{
  "zone": "Zone the tray is in",
  "bay": "Bay the tray is in",
  "tray": "The tray to remove"
}
```

Parameters

Parameter	Type	Description
zone	string	The zone the tray you want to remove is in
bay	string	The bay the tray you want to remove is in
tray	string	The tray you want to remove

Output

OK if successful

FAIL if unsuccessful

6.1.3 switchTray

Switches the location of two trays.

URL

http request POST /stockTake/switchTray

Input

```
{
  "first": {
    "zone": "Zone ID",
    "bay": "Bay ID",
    "tray": "Tray ID"
  },
  "second": {
    "zone": "Zone ID",
    "bay": "Bay ID",
    "tray": "Tray ID"
  }
}
```

Parameters

Parameter	Type	Description
first	JSON Object	The position of the first tray to swap
second	JSON Object	The position of the second tray to swap

Output

OK if successful

FAIL if unsuccessful

6.1.4 editTray

Edits the contents of a tray. The only required fields are the zone, bay and tray name; as well as any fields that are being edited.

URL

http request POST /stockTake/editTray

Input

```
{
  "zone": "Zone specified",
  "bay": "Bay specified",
  "tray": "Tray specified",
  "contents": "[Optional] What is in the tray",
  "weight": "[Optional] How much does it weigh",
  "expiry": "[Optional] When does it expire"
}
```

Parameters

Parameter	Type	Description
zone	string	The zone in which the tray is
bay	string	The bay in which the tray is
tray	string	The tray to edit
contents	string	[Optional] What is in the tray
weight	integer	[Optional] The weight of the contents (give as a number in Kilos)
expiry	string	[Optional] The expiry date of the food

Output

OK if successful

FAIL if unsuccessful

6.1.5 moveTray

Moves a tray to a specified empty tray position.

URL

http request POST /stockTake/moveTray

Input

```
{
  "posStart": {
    "zone": "Zone ID",
    "bay": "Bay ID",
    "tray": "Tray ID"
  },
  "posTarget": {
    "zone": "Zone ID",
    "bay": "Bay ID",
    "tray": "Tray ID"
  }
}
```

Parameters

Parameter	Type	Description
posStart	JSON Object	Starting (current) position of the tray to move.
posTarget	JSON Object	New position of the tray after it moves.

Output

OK if successful

FAIL if unsuccessful

6.1.6 nextExpiring

Get the next n expiring trays in the warehouse

URL

http request POST /stockTake/nextExpiring

Input

```
{
  "n": "Top n trays to fetch",
  "contents": "[Optional] Get trays that match these contents"
}
```

Parameters

Parameter	Type	Description
n	integer	Number of trays to fetch
contents	string	[Optional] String matching the contents of some tray in the warehouse

Output

Array of JSON Tray Objects if successful

FAIL if unsuccessful

6.1.7 getAllCategory

Get all trays that contain the specified item

URL

http request POST /stockTake/getAllCategory

Input

```
{
  "contents": "Optional, only get trays that match these contents"
}
```

Parameters

Parameter	Type	Description
contents	string	String matching the contents of some tray in the warehouse

Output

Array of JSON Tray Objects if successful

FAIL if unsuccessful

6.1.8 addTrayMany

Add many trays in a single request.

URL

http request POST /stockTake/addTrayMany

Input

Array of tray objects **as** described **in** addTray

Output

SUCCESS if all trays added successfully

FAIL if unsuccessful

6.1.9 editTrayMany

Edit many trays in a single request.

URL

http request POST /stockTake/editTrayMany

Input

Array of tray objects **as** described **in** editTray

Output

SUCCESS if all trays edited successfully

FAIL if unsuccessful

6.1.10 removeTrayMany

Removes many trays in a single request.

URL

http request POST /stockTake/removeTrayMany

Input

Array of position objects **as** described **in** removeTray

Output

SUCCESS if all trays removed successfully

FAIL if unsuccessful

6.2 Bay functions

6.2.1 addBay

Adds a new bay to the warehouse

URL

http request POST /stockTake/addBay

Input

```
{
  "bay": "Bay Name",
  "zone": "Zone Name",
  "xVal": "x Position of bay in zone",
  "yVal": "y Position of bay in zone",
  "xSize": "x Size of the bay",
  "ySize": "y Size of the bay"
}
```

Parameters

Parameter	Type	Description
bay	string	Bay identifier string
zone	string	Zone identifier of zone the bay is in
xVal	integer	x Position of the bay within the given zone
yVal	integer	y Position of the bay within the given zone
xSize	integer	x size of the bay (number of columns for trays)
ySize	integer	y size of the bay (number of rows for trays)

Output

OK if successful

FAIL if unsuccessful

6.2.2 removeBay

Removes a bay (and all trays within) from the warehouse.

URL

http request POST /stockTake/removeBay

Input

```
{
  "zone": "Zone that the bay is in",
  "bay": "The bay to remove"
}
```

Parameters

Parameter	Type	Description
zone	string	Zone identifier of the zone the bay is in
bay	string	Bay identifier of the bay to remove

Output

OK if successful

FAIL if unsuccessful

6.2.3 editBay

Edit the attributes of a bay.

URL

http request POST /stockTake/editBay

Input

```
{
  "bay": "Bay Name",
  "zone": "Zone Name",
  "xVal": "[Optional] New x Position of bay in zone",
  "yVal": "[Optional] New y Position of bay in zone",
  "xSize": "[Optional] New x Size of the bay",
  "ySize": "[Optional] New y Size of the bay"
}
```

Parameters

Parameter	Type	Description
bay	string	Name of the bay being amended
zone	string	Zone identifier of zone the bay is in
xVal	integer	[Optional] x Position of the bay within the given zone
yVal	integer	[Optional] y Position of the bay within the given zone
xSize	integer	[Optional] x size of the bay (number of columns for trays)
ySize	integer	[Optional] y size of the bay (number of rows for trays)

Output

OK if successful

FAIL if unsuccessful

6.2.4 getTraysInBay

Gets all of the trays in a specified bay.

URL

http request POST /stockTake/getTraysInBay

Input

```
{
  "zone": "Specified zone",
  "bay": "Specified bay"
}
```

Parameters

Parameter	Type	Description
zone	string	The zone you want info about
bay	string	The bay you want info about

Output

An array of the trays in the bay.

6.3 Zone functions

6.3.1 addZone

Adds a new zone to the warehouse.

URL

http request POST /stockTake/addZone

Input

```
{
  "zone": "Name of the zone to be added",
  "height": "Height of the zone",
  "width": "Width of the zone"
}
```

Parameters

Parameter	Type	Description
zone	string	The zone you want to add
height	integer	The height you want the zone to be
width	integer	The width you want the zone to be

Output

OK if successful

FAIL if unsuccessful

6.3.2 editZone

Edit the target zone's attributes.

URL

http request POST /stockTake/editZone

Input

```
{
  "zone": "Name of the zone to be altered",
  "newname": "[Optional] New Name of the zone",
  "height": "[Optional] New Height of the zone",
  "width": "[Optional] New Width of the zone"
}
```

Parameters

Parameter	Type	Description
zone	string	Target zone identifier
newname	string	[Optional] The new name of the zone
height	integer	[Optional] The new height of the zone (number of rows in the zone)
width	integer	[Optional] The new width of the zone (number of columns in the zone)

Output

OK if successful

FAIL if unsuccessful

6.3.3 getBaysinZone

Gets an array of all the bays in a specified zone

URL

```
POST /stockTake/getZone
```

Input

```
{
  "zone": "Target zone identifier",
}
```

Parameters

Parameter	Type	Description
zone	string	The name of the zone containing the bays we wish to retrieve

Output

An array of trays is returned if successful

FAIL if unsuccessful

6.3.4 getZones

Gets all zones in the database and their bay contents.

URL

```
http request GET /stockTake/getZones
```

Input

None

Output

Returns an array of all zone identifiers

6.3.5 removeZone

Removes a zone (and all bays and trays contained within) from the warehouse.

URL

http request POST /stockTake/removeZone

Input

```
{  
  "zone": "The zone to remove"  
}
```

Parameters

Parameter	Type	Description
zone	string	The identifier of the zone you want to remove

Output

OK if successful

FAIL if unsuccessful