University of Plymouth

School of Engineering, Computing and Mathematics

COMP3000
Final Stage Computing Project
2020/2021

The development of an integrated home stock database system

Struan Sharpe

10560031

BSc (Hons) Computer Science

Table of Contents

Word count –9886

Github: https://github.com/struansharpe/COMP3000

Office Planner: https://tasks.office.com/live.plymouth.ac.uk/en-US/Home/Planner/#/plantaskboard?groupId=51183dd1-11c6-4690-bd02-516a07d72881&planId=0xvt9-VuyUe9GYloTswMRJYABu-K

Chapter 1- Introduction

1.1 Summary

Summary

Project name: Stokk

Stokk Is a home database system. This application would be created in consideration of everyone who owns a home and has access to the internet: this application can assist with shopping, choosing meals and keeping stock at home, in one application.

This project comes in four main components, the database, the Application Programming Interface (API), the API Exerciser and a prototype User interface.

Figure 1, Stokk Prototype UI Homepage

1.2 Introduction

With Smart home systems on the rise the desire for a fully integrated smart home environment would be a nirvana for smart home technologies. There are already many smart systems out there in the world which are used to great effect to increase convenience at home. Smart systems allow the user to control aspects of the home by using technology i.e. heating, lighting and resource usage, they achieve this by creating a simple portable interface which allows the user to interact with the smart system, creating an smart home environment.

The Initial idea for Stokk came from looking at smart fridge technology where they used various means to determine what item was placed inside; this information was then sent to the cloud where the system uses this information to tell the user what meals they could make from the ingredients inside (Qiao, et al., 2017). This was an interesting concept, creating an application which echoes this for other items in the house would be a step closer to having a smart integrated home or smart home environment.

Initially the idea was to attach the information to QR stickers and attach that to the items for the phone to scan. But with smart systems, when expectations weren't met it was usually due to "the effort required to set up and control systems, rather than the lack of functionality." (Oliveira, et al., 2020). So constantly placing new QR codes on items would be the opposite of what the application is trying to achieve in efficiency and convenience as it adds extra steps for functionality.

So, the next logical step would be to create an application that attaches to multiple external API such as Tesco, Barcode readers and shopping carts, which utilise these API's to scan the items a user has, or if shopping online get the items from the Kart. This cuts out the middleman of having to manually put data into a system but still allow the user to alter and add items if they wish, whilst still being able to stock control/ shop and recipe handle, this would then be a smart home system.

1.2 Objectives

Problems That Were Set Out to Solve

The problems that the application strives to solve, are that of stock control. The main functionality of Stokk is to modernise the archaic and less efficient ways of keeping a record of what's in the home, seeing what recipes a person can make and what is needed to be bought at the shop with a shopping list which is automated but can be changed manually as well.

The problem of not knowing what is needed when shopping, allowing the user to write a shopping list using the app, and have the system automatically place items in the shopping list when they are getting low or are out.

The problem of not knowing what meals a user can cook with the ingredients they have in the house, showing them what they can make with what ingredients they have, also show other meal options by showing what they need to shop for.

There are many applications out in the world which give a user a great collection of recipes, but the functionality stops there - the application doesn't tell you if you have all the ingredients. The purpose of Stokk is to integrate home and errand life into an application that simplifies all the tasks by having one application that has a wide range of functionality, with the capability to share peoples recipes from around the world and add your own, as well as automatically change stock or shopping lists.

Goals

To make an application that makes the home life i.e. cooking and shopping more convenient and integrated.

	Goal
1	Increase home convenience by use of application for shopping, cooking, and stock control.
2	Steppingstone to smart home technology based in kitchen.
3	Make chores/errands easier.
4	System which utilises automation to help create a smart home environment, but also allow users to still add and edit if they wish.

1.3 Critical Success Factors

	Critical success factor
1.	Having a database created which can be the backbone of a home database system.
2.	Having an API that is published which allows access to data through the use of API endpoints.
3.	Have an application that allows reading, creation, editing and deleting (CRUD)of all the tables' data.
4.	Have automation on the shopping list functionality so that once items run low, they are added to the shopping list. If item quantity runs out to remove from the household items and add an urgency attribute to the item in the shopping list.
5.	Display recipes if the user can make them with the current ingredients and retain separate ones where extra ingredients are needed.
6.	Once meal has been chosen the ingredients are removed from the household stock list.
7.	If meal needs ingredients, those ingredients can be added to the shopping list.

1.4 Scope and Limitations

Designing for a Larger Scope

External API

The Project itself would be made far more effective and efficient if it could be attached to multiple external API, such as Tesco API, barcode readers/ QR scanners. Having many External API involved was

taken into consideration during development, and it was decided that although the final product would ideally have these feature to help the convenience, the early versions of the application, the one used for this project will not have them connected. This is due to the fact that connecting numerous API could take up a lot of time and money to get access to, so instead following core functionality to avoid scope creep. Although the literal API were not implemented a work around was used. A solution of simulating the connection using mock adapters and interfaces to simulate data coming in from an API. With this, it allows the application to demonstrate how the connection would be, using mock data to simulate the interaction, this can be seen in the barcode and shopping mock adapters and interfaces within the API application.

User Functionality

The Core functionality used to be very user oriented, so much so that one of the first coding sprints desired was to be the creation of the login and user creation page; this was more difficult than anticipated spending more time on the sprint than desired. The project started from other functionality and left the user detail till later, prioritising the storage and shopping functionality. Due to this, most user functionality had to be simulated and hard coded into the application.

User Interface

Although the project aims to be a user friendly experience, as most smart home systems strive to be, trying to make the interface between home functionality and the user as intuitive and easy as possible, it may be hard to make the application as user friendly as initially anticipated. Instead of having the web flow similar to my UI mock designs, a far simpler UI made for just core functionality was made, to accommodate testing main functionality over gold plating an application that doesn't work.

Variation in User Usage/Fuzzy Logic

During development, it was realised that not every user is the same, and therefore when using a recipe every user will use slightly different amounts. Originally, the system would allow the users to just add their own recipe variations, so that if they didn't like another recipe, they could just add their own. But realising not everyone would be interested in constantly typing out all their recipes, if there was a way to instead, track how much they used of each item in comparison to others. The idea to eventually implement some form of fuzzy logic where there is an attribute that is a float, which has a value of 0.0-1.0 (where 0.5 is the average and normal amount), if the user states that they used more ingredients than the actual recipe then the value of this float increases, and assumes that the user slightly overuses ingredients based on this float, which then increases the ingredient amounts for recipes for that household. Many issues arose with this as it hard to change the values of every ingredient in a recipe, so although it may be easy to implement with recipes which use 100g of flour, it's hard to change a recipe when only using one or 2 eggs, as a part of an egg cannot be used or taken from the storage. Even in this instance the algorithm would have to decide whether to keep it at one egg or change to two, although it seems logically sound, the difference in cooking between having one egg and 2 is vastly different, and as such meals would be inconsistent and not necessarily nice compared to its original form.

Modals

Given the change in architecture, the design decision to have modal popups over certain pages was dropped. As the MVC architecture already has numerous page views for each controller, so flow of

functionality was already functional; as well as the uncertainty of being able to use modals within an ASP.net MVC application.

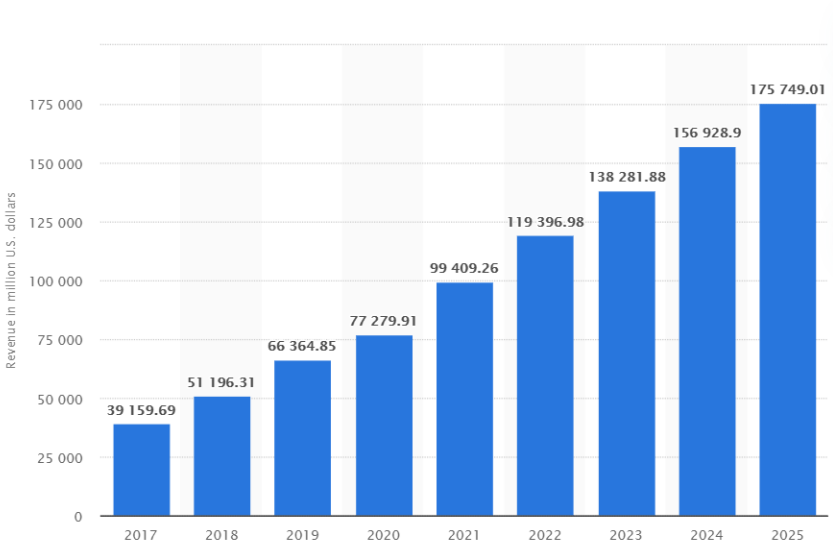
Rating System

A rating system would be an important user aspect of the application, eventually a user would be able to view different recipes and be able to tell if other users enjoyed the meal, and when searching for new recipes show higher rated . This would also be used to display to the user which recipes they preferred .

Chapter 2 Literature Review

2.1 Statistics on Smart Home Systems

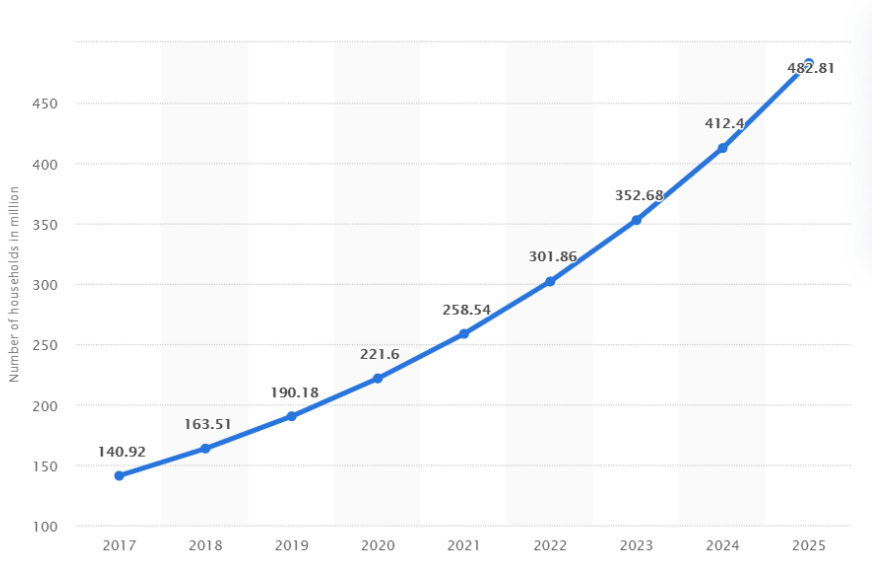
Smart Home Revenue Forecast Worldwide Until 2025



(Lasquety-Reyes, 2020)

Figure 2, This statistic presents a forecast of the revenue in the Smart Home market worldwide until the year 2025(in millions), According to the Digital Market Outlook.

Smart home Usage by Household



(Lasquety-Reyes, 2020)

Figure 3,This statistic presents a forecast of the number of Smart Homes in the Smart Home market worldwide until the year 2025(in millions). According to the Digital Market Outlook.

As seen by figure 3 smart home systems are projected to rise in usage, at a steady rate, that eventually leads to a possible 482 million households using some form of smart home technology by 2025. This

shows how many possible households could eventually lean towards using the application that this project wishes to develop. Along with this, Figure 2 presents the forecasted revenue, which if the calculations are accurate, by 2025 would be close to $175,750,000,000. If the application being created was able to hook the market by being a one of a kind application that is a first on the scene, with minimal or no competition as there aren't alternatives, then the possible revenue stream would be large as well as steadily increasing over the years.

2.2 Study On Digital Hybrid Shopping Lists

Whilst researching the area of digital shopping lists to help with the assistance of shopping, there was a study which touched on the area of hybridised digital shopping lists. Where the authors wanted to create a hybrid of writing a shopping list physically with a digital format to store the information. The authors postulated that "Shopping is one of the most frequently occurring tasks in our daily lives…small tasks that are carried out repeatedly" (Heinrichs, et al., 2011). Although I disagree that shopping is one of the most frequently occurring tasks, as normally a person usually shops 1-2 times a week, compared to other tasks in the household. But due to the sheer amount that a person cooks and shops throughout their life, an application which assists with these aspects of the home by adding elements of automation would be more convenient for a user than a pen and paper.

2.3 Smart Home Environment - Concepts and Solutions

This paper covers the introduction to the concept of a smart home environment, as Florian Kazmierzak puts it; smart home environments are "environments that try to facilitate the life of the user in many different ways and make it more comfortable by using technology" (Kazmierzak, 2012). Since Stokk desires to implement automation to help users shop and cook, the application is helping to make the kitchen a more integrated smart home environment.

2.4 Intelligent refrigerator using ARTIFICIAL INTELLIGENCE

In this study, Schweta is creating a smart System which operates in a refrigerator, which can tell the user when items are going out of date. In this Schweta states that "Developing Smart Appliances is directly proportional to Developing Smart Home environment. It is a critical factor in the realization of the smart home environment." (Shweta, 2017). This desire to create an integrated smart home environment where almost everything is eased by the use of technology, is what the project vision desired to be, and is indeed the end goal of most smart systems. Schweta also believes that the "Kitchen is one of the most important places for a Smart home as it consists of many Appliances which provide better services to the household." (Shweta, 2017), which is true as eating is one of the most important functions for the survival of a human. The environment a user works in to achieve sustenance(the kitchen), if made more streamline through smart home technologies, could save the user vast amounts of time in the long run,

if Stokk were to implement these ideologies in the design, then it would be one step closer to being a smart system.

2.5 Smart home technology—comparing householder expectations at the point of installation with experiences 1 year later

This study sets out to analyse the expected benefits and anticipated challenges to the introduction of smart home technology. They did this over studying 19 households, where they took questionnaires after a year of having the system. There were two findings that were interesting in this study, firstly that the “Results showed that approximately a year after installation, many of the initial expectations of the benefits were met” (Oliveira, et al., 2020) This shows that if a user goes in knowing what they want from a smart system then most of the time it works out as the user expected. The only times expectations were not met was usually “due to the effort required to set up and control systems, rather than the lack of functionality.” (Oliveira, et al., 2020), which is what Stokk desires to avoid. So, if Stokk has a good design which allows users to utilise functionality without having to add extra steps then Stokk would be one step closer to being a successful Smart home system.

2.6 Summary of Studies

For the success of Stokk, it must be realized that all the factors considered in these studies contribute towards Stokk’s design. Additionally, for Stokk to be a smart home system, it needs to bring convenience to its users whilst being desirable and innovative enough for the user to deviate from other archaic methods.

Chapter 3 Existing Products

This chapter shows research into products that are similar to Stokk, applications that were chosen to be reviewed share similar functionality and approaches, the applications share some similar elements but differ on how they interact with the user and what their reach of functionality is.

3.1 BBC Good Food /BBC Food

For the Web based websites both bbcfood.com and Goodfood.com were studied, and since they were so similar in functionality and design they have been grouped together.

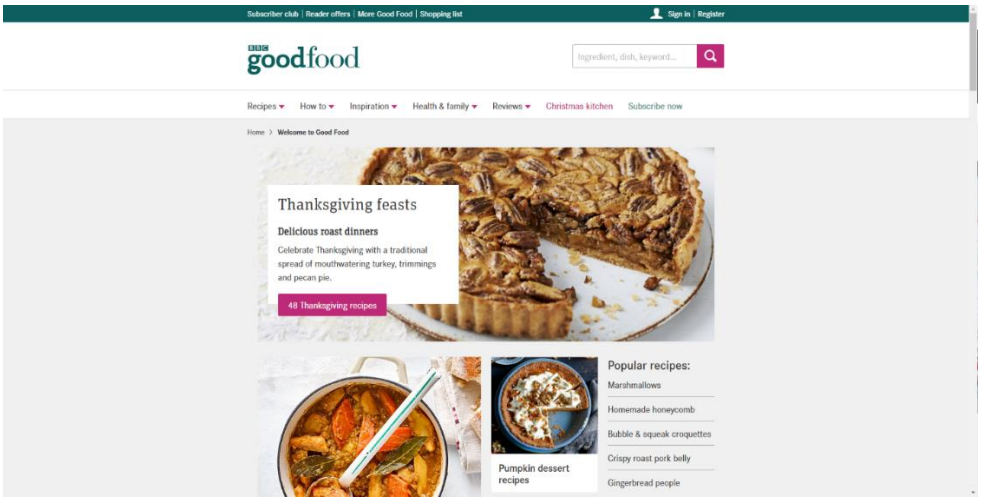


Figure 4, BBC goodfood website home page

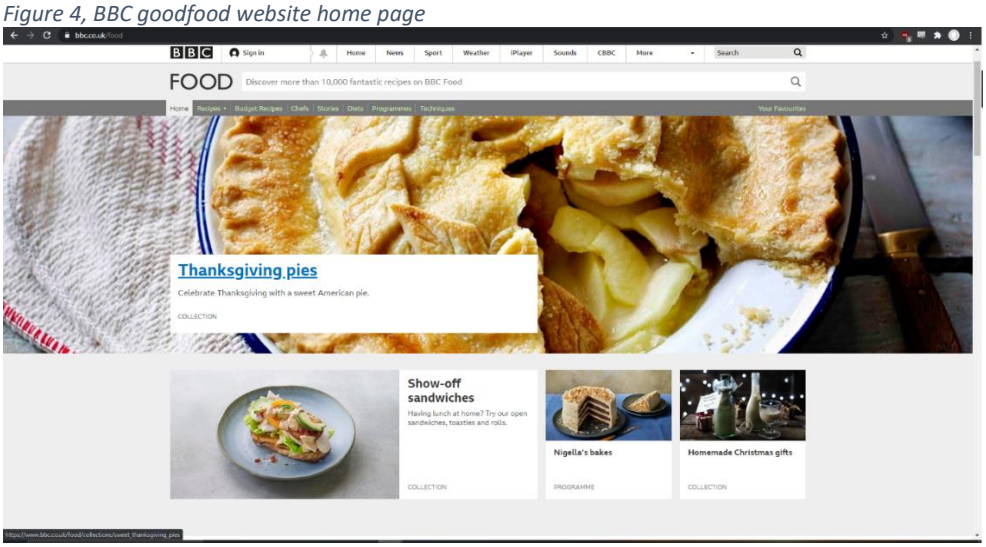


Figure 5, Home page for BBC food website

Key Features

on the main page of BBC food, it shows some suggested possibly seasonal meals, with some choices to navigate to other selections of recipes. They have a navigation bar at the top for quick access to pages

Speculative Functionality

Could use a feature of suggested meals either by popularity or by season, this would help users get meals which are deemed as better (user can still use their own recipes which they input).

A rating system for meals could be possible, after eating a meal, ask the user what they thought of the meal and what they would rate it on a scale of 1-10. Alternatively, the app could follow how many people used it and how many people wanted to save it to favourites.

3.2 Paprika

One of the applications most like Stokk's vision was paprika, which is a mobile application which does recipe handling as well as storage control and groceries. This is one of the only other applications that encapsulates nearly all the functionality needed for Stokk, excluding automated shopping list features.

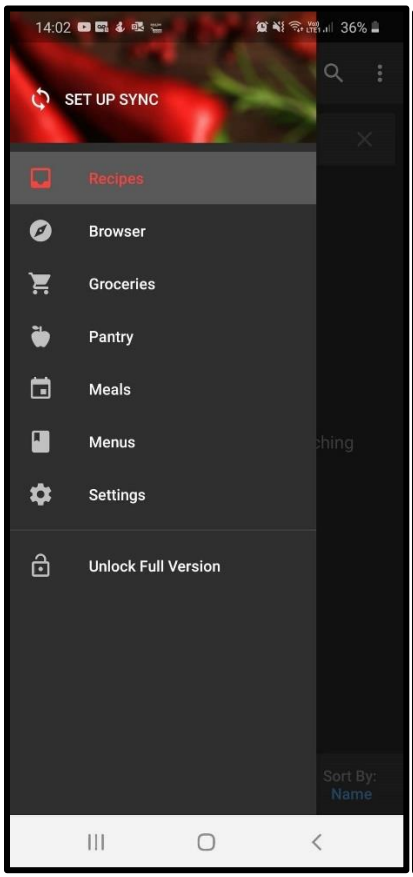


Figure 6, Paprikas, side bar navigation

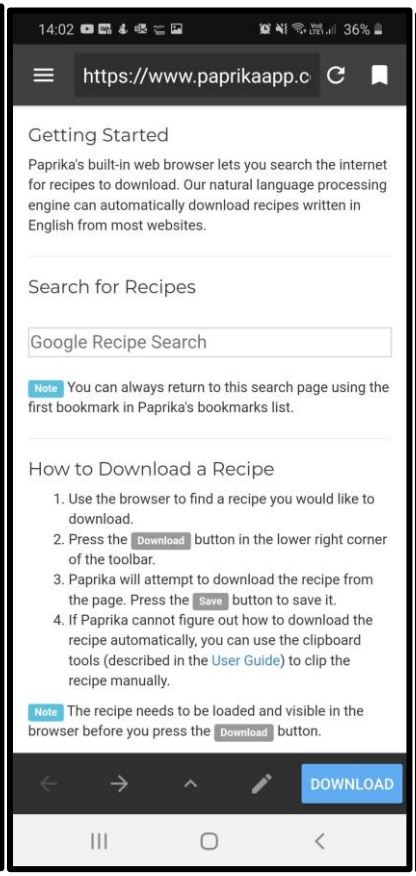


Figure 7, Paprikas recipe search feature

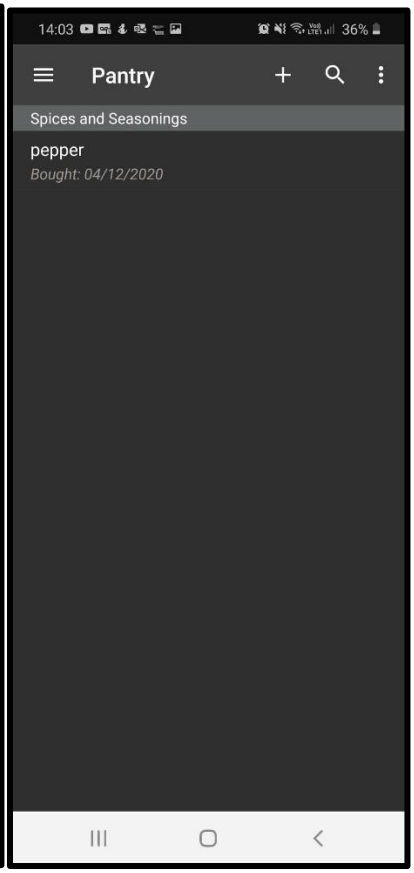


Figure 8, Paprikas pantry page

Key Features

Paprika has a google API attached to the Recipe page, seen in figure 7, where a user is able to search any website and add the recipe to their collection, Paprikas pantry page which contains a user's ingredients which can be seen in figure 8.

Instead of going for a main page design, paprika elects for a side navigation bar to go from page to page, which gives quick easy access to the user without having visual clutter of different page buttons, only showing when the user requests, as seen by figure 6.

Speculative Functionality

The storage feature will be included with Stokk, where the user can say which room or cupboard the ingredient is in.

The recipe and shopping list functionality will be similar but adapted so that a user can see what meals they can make with the ingredients they have available in their household.

The use of a search feature using Googles API would be a great addition to the application, although not within scope of this project but within the vision of what Stokk would be if in development on market. The recipes for this project will all be stored in the system so anyone can view any recipe and add their own.

3.3 Yummly

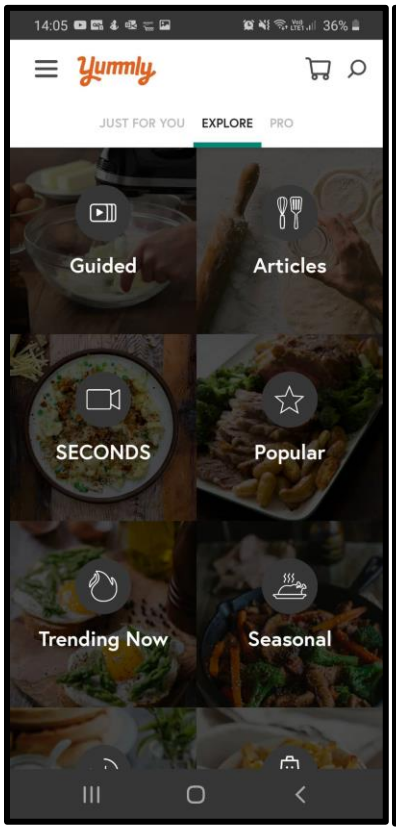


Figure 9, Yummly Home page

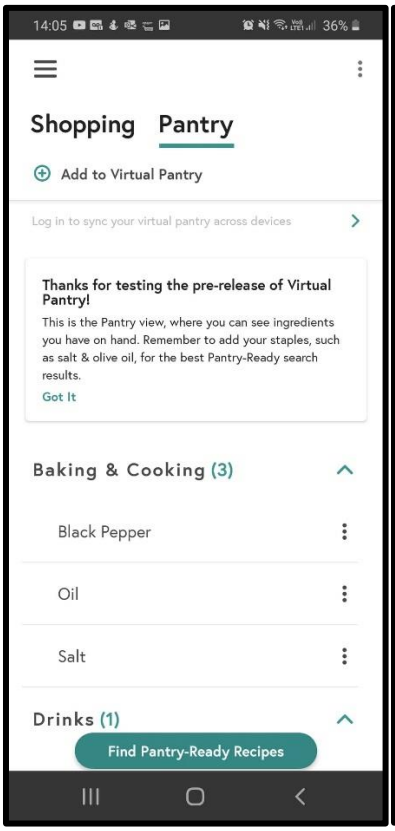


Figure 10, Yummly Pantry page

Figure 11, Yummly shopping list

Similar to Paprika Yummly has shopping list and pantry (stock) functionality. But Unlike Paprika, Yummly elects to have a home page for navigation through pages.

Speculative functionality

The option to view seasonal meals related to the time and also the trending now features seen in figure 9, Yummly's home page. This was be another great feature to help the user decide what to cook but not within scope of this project.

Chapter 4 Method of Approach

The desired product at the end of this project is a system which can facilitate the functionality of day to day kitchen use for a user that helps accommodate recipe handling, shopping list and stock taking functionality. The end product will simulate connections to mock adapters which simulate connections to external API.

4.2 Legal Social and Professional Issues

Two questionnaires which can be found in appendix B, were carried out during the development for Stokk, the initial questionnaire was used to gauge if people would be interested in an app like this, and another later on to test the mock User interface. Ethics approval for the use of Plymouth University student led responses to the questionnaire was obtained from the university ethical board located in appendix C. This means that the ethical principles of nonmaleficence, transparency and informed consent were adhered to.

None of the questionnaires aimed to mislead or misdirect anyone, for the first questionnaire simple questions were asked around the subject area, to gauge if users around the 18-25 age range would use an application like Stokk.

As the application only holds dummy data (data in which the developer implemented) no one's personal data is at risk, as the system isn't fully complete including security features. It would be irresponsible to hold genuine information about a user in this form, as even the passwords are only a varchar currently, later implementations would include security.

4.3 Functional Requirements

4.3.1 Unauthenticated Users

	Unauthenticated users' Functional requirement
1	Create an account
2	Sign into an account
3	Request a password reset

4.3.2 Authenticated Users

	Authenticated users' Functional requirements
1	Add Household
2	Edit Household
3	System creates Shopping list for Household
4	Add Storage space
5	Add Room
6	Edit Room
7	Edit storage space
8	Add item
9	Add item type
10	View households' items
11	System Household Items ordered by room and storage space
12	Add household item
13	System sets Desired Stock quantity
14	Edit household item
15	Remove household item
16	System adds to shopping list
17	System adds urgency and removes item from household item when depleted
18	Add to shopping list
19	Remove from shopping list
20	Delete shopping list item
21	Add Recipe
22	View Recipes
23	System displays recipes which can be cooked with current ingredients, separated by those that can't be made.
24	System can add ingredients needed for meal into shopping list

4.3.3 Non-Functional Requirements

	Non-functional Requirements
1	Usability
2	Consistency
3	Convenience

4.3.4 Functionality Iterations

Functional requirements change, research found the idea of telling the user where the items are before a meal is unnecessary as most people know where their items are.

There was an issue with determining how much of the ingredient the user actually used and keeping track of it. The system is flawed if the user has to manually input data every time there is a minor change in a recipe, for this project it is assumed that the user can just add their own recipe with minor changes.

4.4 Architecture

This section goes through numerous diagrams which display design choices in architecture for the system.

System Level Design

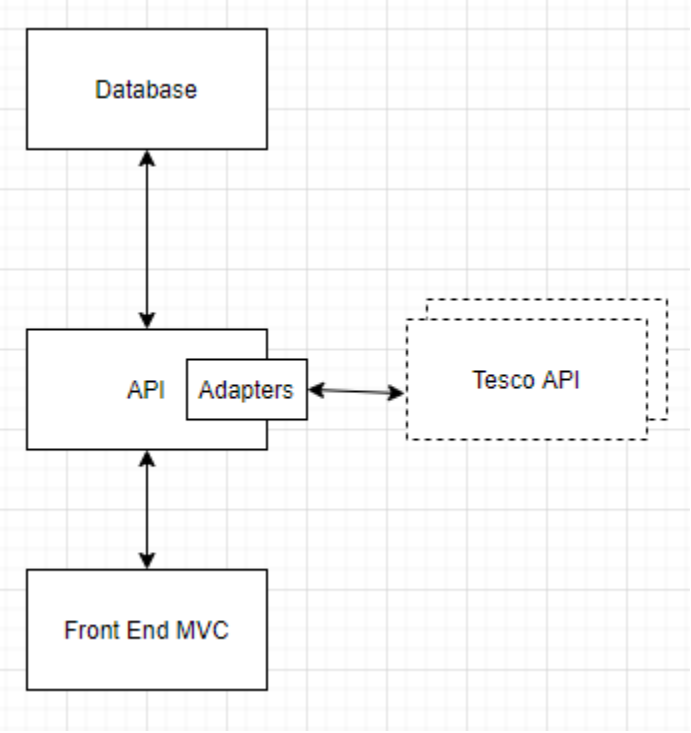


Figure 12, low level architecture design

Figure 12 shows a simple architecture of how the system would work, including the external API's, the applications API is communicating with both the database, and through adapters within the API, it is able to communicate with external API's such as Tesco API and Barcode Reader API's. The front end MVC application receives information and displays it using the API endpoints, here if the front-end application CRUD's any data then it should be reflected in the Database.

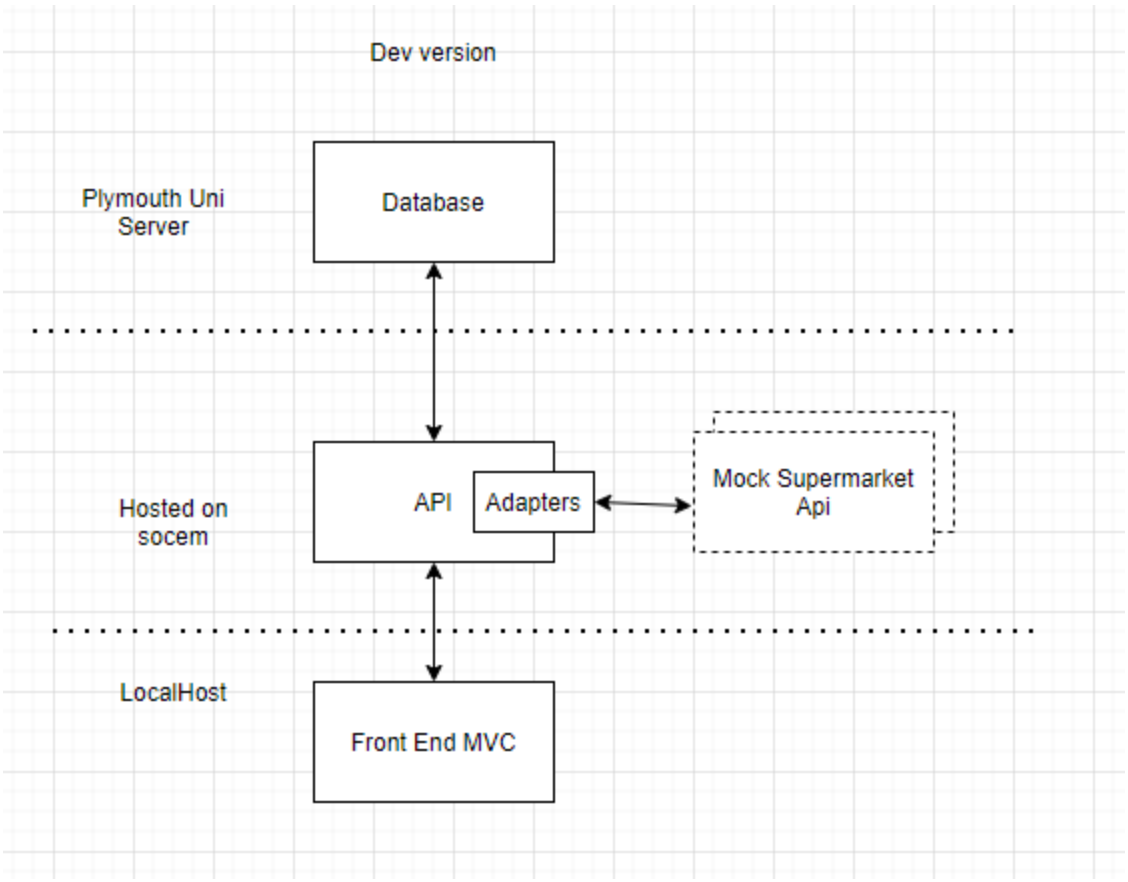


Figure 13, Developer version architecture

Figure 13 demonstrates the architecture of the system during development. The database was hosted on the University server, the API after being published is hosted on the web Socem server. Instead of being attached to real API's this system merely simulates a connection to one by having mock adapters/ interfaces that create some mock data, which on a query can display the relevant item for both the mock barcode and shopping adapters. Finally, the front-end application was tested on LocalHost.

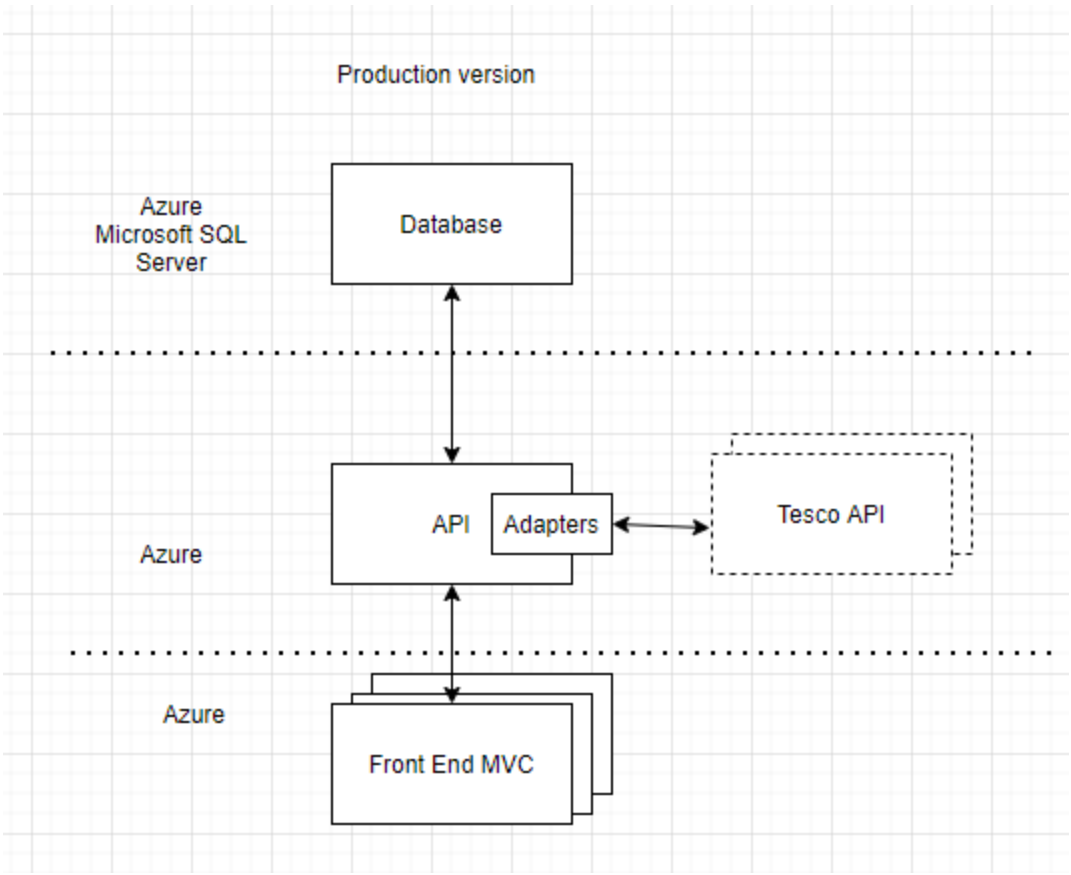


Figure 14, production version architecture

Figure 14 is the Desired Architecture if ever brought into further development/ production, as the University wouldn't have space for hosting after the project being finished, it would have to be hosted on another service. Where everything could essentially be hosted on Azure, this was avoided as it would cost money to get hosting space.

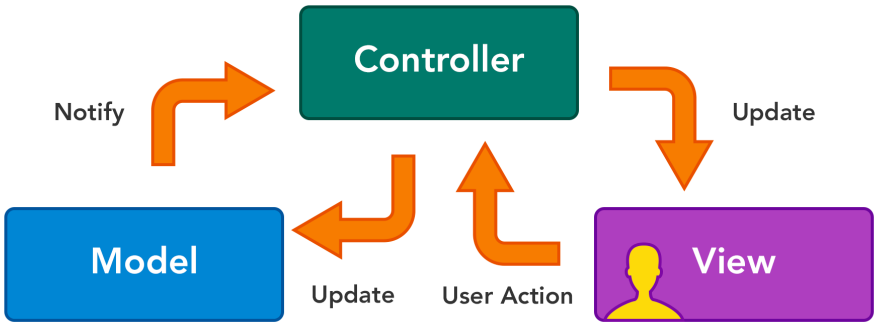


Figure 15, MVC Design pattern/architecture

The front-end application has the architecture of MVC, this is where the different components of the structure of an application are kept into their separate compartments. The model is the shape of the data, like a schema, these models are essentially the same models seen within the API. The View deals

with the User Interface and what the user interacts with, within ASP.Net, the view is represented by numerous files for different controller pages and within those files are cshtml files or razor pages. The controllers deal with the functionality, they have the methods as well as the connections to the API, once an action takes place a function is usually called which updates the model data. The controller should then be notified of the changes and as such updates the view to either display the new data or go to another page.

Implementation started off with the database following the database first methodology, using Microsoft SQL server manager to create database tables using queries, including some triggers and sequences for ID's, this database is on the Universities Socem System.

Then an ASP.Net API was created, this is where the ADO Entity models were utilised to connect to the desired database tables, here the API Endpoints are created. Once created it is then published and hosted on Web Socem, so the endpoints are reachable by typing into the URL of any internet browser.

This is where The API Exerciser was created to help test connection and interactions between applications, while practicing the technologies involved for the application, here each table can be viewed and fully capable to CRUD while using the published API Endpoints.

Finally, a prototype user interface using Adobe XD, to create a mock design that captured the image of what Stokk would look like if it went into production.

4.4 Technologies

Database

Was created using Microsoft SQL Server Management Studio, where Queries were used to create tables, sequences, and triggers for the systems core functionality (SQL statements for the creation of the tables and triggers are in the appendix A).

API

The API is developed in ASP.NET framework web application, the API Application is in the architecture of MVC, where the models hold the shape of the data from the desired tables, so for each table in the database there is a model to accompany it, these were created by adding an ADO Entity Model . The API application has a set of API controllers which create the API endpoints for all the Get, Post, Put and Delete request, these are autogenerated from the wizard, choosing the data entity model that each controller works off of, these controller classes derive from the API Controller class rather than the usual controller class used by MVC applications. The views in this application didn't have much use apart from a home page where it can direct to the raw data (xml), of each API GET endpoint.

This API Application was published so a separate front-end Application could be made in parallel to receive the API endpoints and push the data to and from the database using the URL "web.socem.plymouth.ac.uk/FYP/SSharpe/". This proved beneficial as testing could be done using the endpoint instead of local host, so FortiClient didn't have to be used, as well as not having to run 2 applications at once for testing.

Front End (API Exerciser)

The desire to keep confusion to a minimum and abide by the separation of concerns was prioritised, and as such a separate application was created for the front end, the front-end application is an ASP.NET MVC web application, it has sort of the same structure as the API application apart from the fact that the Model doesn't derive from an ADO entity model, and the controllers don't derive the API Controller class.

Model

This means that the models within the MVC application are just a mirror of the Model.cs files within the API application, where it has the same attributes as the selected table, with gets and sets within each.

Controller

The controllers in the MVC application are used to control the pages and the functions within them, by calling the API endpoints for the relevant page. Although the views were pretty much ignored in the API application, the front-end application will be dealing with the User, so the views were utilised.

View

After creating a controller, a folder is created which has the same name as the controller, this folder holds the cshtml files for each page/controller. Within the User Folder there are many cshtml files which were created using the wizard where each table has an Index, Edit, Create, Details and Delete cshtml page. Within the User Folder There are methods for each page and functionality, which all interact with the API endpoints.

Adobe XD

Although the first mock design for the user interface was created using PowerPoint, the second after testing input was Adobe XD as it was able to create a more complex real prototype for testing.

4.5 Code Walkthrough

This code walk through covers the MVC functions within the API exerciser, with examples of the displayed data and how it was achieved.

Index (GET) Function

```
public async Task<ActionResult> Index()
{
    List<User> UsersInfo = new List<User>();

    using (var client = new HttpClient())
    {
        //Passing service base url
        client.BaseAddress = new Uri(Baseurl);

        client.DefaultRequestHeaders.Clear();
        //Define request data format
        client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

        //Sending request to find web api REST service resource
        HttpResponseMessage Res = await client.GetAsync("api/Users");

        //Checking the response is successful or not which is sent using HttpClient
        if (Res.IsSuccessStatusCode)
        {
            //Storing the response details recieved from web api
            var UserResponse = Res.Content.ReadAsStringAsync().Result;


            //string UserResponse = await HttpRequest("", null, "Get",null);


            //Deserializing the response recieved from web api and storing into the storage list
            UsersInfo = JsonConvert.DeserializeObject<List<User>>(UserResponse);


            //returning the storage list to view
            return View(UsersInfo);

        }
        return null;
    }
}
```

Figure 16, Index function in MVC application

This code goes through the index user function within the MVC application (API Exerciser), this function would be used to display all the data within the user table.

The base address or Baseurl “web.socem.plymouth.ac.uk/FYP/SSharpe/” sits at the top of all the MVC controllers. In figure 16 the index function Initialises a list with the User datatype, this will hold all of the data from the API call.

```
List<User> UsersInfo = new List<User>();
```

Then it initialises a new http client by using the “using” action word.

```
using (var client = new HttpClient())
```

Within this using Http client, it gives the client a base address by passing the base URL.

```
client.BaseAddress = new Uri(Baseurl);
```

Then it defines request data format, the data being dealt with is Json.

```
client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
```

Then it sends a request to find the web API REST service resource, as this is a get function client.Getasync is used where the string parameter notates the desired endpoint. So for index users, the base URL is used and within the Getasync it's parameter is the string "api/Users" to reach the desired endpoint.

```
HttpResponseMessage Res = await client.GetAsync("api/Users");
```

The function then checks res to see if it reached success status code, if not the function returns null, if success then.

It then Stores the response details received from the API into UserResponse using readAsStringAsync function.

```
var UserResponse = Res.Content.ReadAsStringAsync().Result;
```

The function then deserializes the response received from web API and stores it into the list using Json.Convert and deserializeObject.

```
UsersInfo = JsonConvert.DeserializeObject<List<User>>(UserResponse);
```

Then the User List is returned to the view.

```
return View(UsersInfo);
```

Home Users House Holds Rooms Storage Space House Hold Items Items Item Types Shopping Lists Shopping List Items Recipes Recipe Items

Create New

UID	FriendlyName	Password	Email	
1	Joshua	Stokkpassword1.	Joshua@hotmail.co.uk	Edit | Details | Delete
2	Steven	Password1.	Steven@outlook.com	Edit | Details | Delete
3	Jonathon	password1.	Johnathon@yahoo.co.uk	Edit | Details | Delete
4	Tiffany	password1.	Tiffany@AOL.com	Edit | Details | Delete
5	Alice	password1.	Alice@outlook.com	Edit | Details | Delete
6	Tiffany	password1.	Tiffany@AOL.com	Edit | Details | Delete
7	barry	Stokkpassword1.	Joshua@hotmail.co.uk	Edit | Details | Delete

Figure 17, shows index page for users

```
    ViewBag.Title = "Index";
}

<h2>View</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.UID)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.FriendlyName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Password)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Email)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.UID)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.FriendlyName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Password)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Email)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id = item.UID }) |
            @Html.ActionLink("Details", "Details", new { id = item.UID }) |
            @Html.ActionLink("Delete", "Delete", new { id = item.UID })
        </td>
    </tr>
}
```

Figure 18, MVC app, User, View Index page

Figure 17 shows the displayed index page for the users, the index function is used when looking at any of the collections of the tables on the index pages (this one for the user), it displays all values in a table which is based off the model within the MVC application. At the bottom of the cshtml page (which is used as the View page) seen in figure 18 you can see the action links for other pages, these redirect to another page taking in User ID as a parameter.

Get Deserialize Function

Most Get functions within the MVC application are similar in structure to the Index function, although instead of a list, the datatype of the model is used, so instead these functions return the information of a single user.

Also instead the client.GetAsync function uses (“api/Users/” + id), as a parameter, passing an individual ID after the desired endpoint.

```
HttpResponseMessage Res = await client.GetAsync("api/Users/" + id);
```

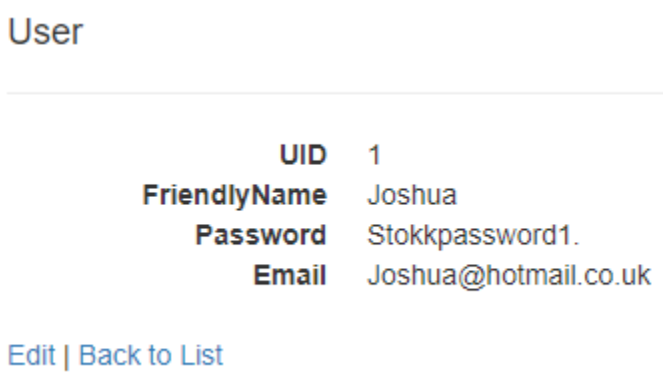


Figure 17, MVC page showing details of specific user

The Get Deserialize Function is used on many pages’ controllers. Get Users Details, Get Users Edit, Get Users Delete, all relocate to the desired page passing the ID of an individual user across to the delete, details and edit pages as seen in figure 19. The only other GET function that doesn’t use GetDeserialize is the Create User page, as data doesn’t need to be passed to the page, it only needs a blank template to send the information across.

Post/Put Functions

All post functions don’t require the 2 lines using client.DefaultRequestHeaders, as they are not receiving any data only posting.

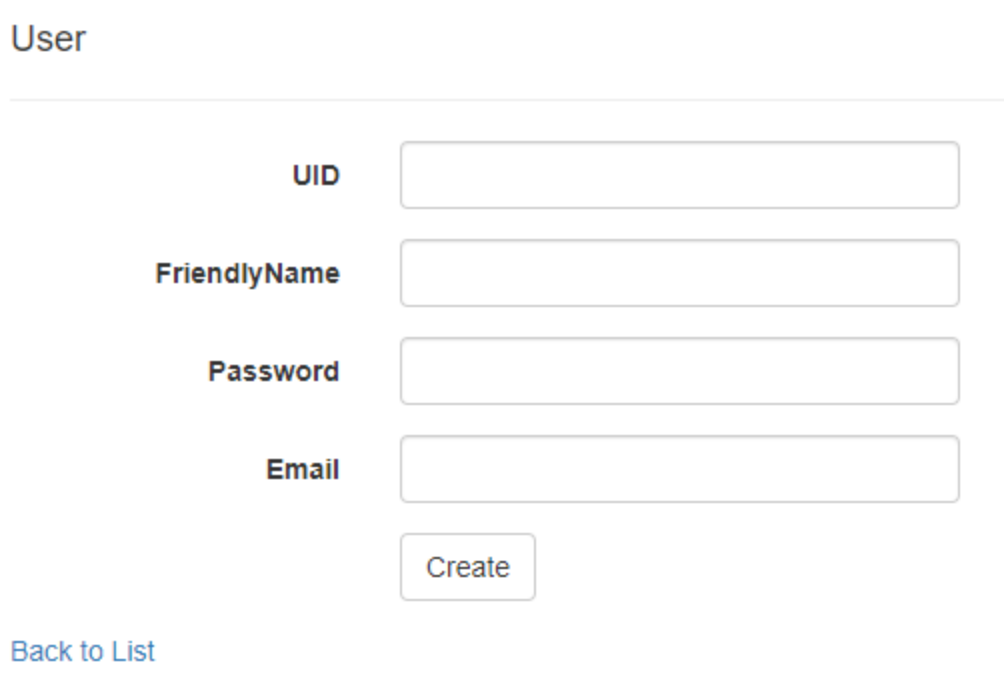


Figure 20, View for creation of user

Create Functions

Create user is a Post function using PostAsJsonAsync method, this converts the body to Json format sends it to the API. Once completed the user is redirected back to the index page after, the function uses client.PostsAsJsonAsync, as it takes in the input boxes form the cshtml page as seen in figure 20.

```
var postTask = client.PostAsJsonAsync<User>("Users/", user);
```

Delete Functions

Delete User, although a delete Async is used, the cshtml page passes the information in post format so [httpPost] needs to be declared before the function, but still uses DeleteAsync, this function also redirects back to the index page at the end of execution.

```
var deleteTask = client.DeleteAsync("api/Users/" + id.ToString());
```

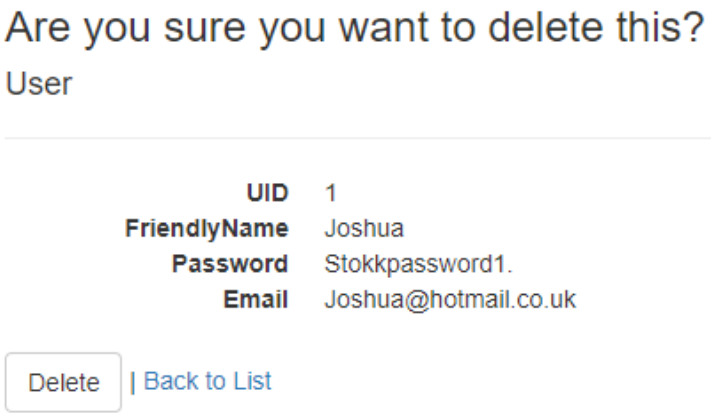


Figure 21, delete user view page

Edit Function

Edit User takes in the parameters of user for the body of changes and ID to refer to the user's ID and endpoint. Yet again, due to the way cshtml pages send data this function had to be denoted as [httppost]

Even though it uses putAsJsonAsync, which also serializes the input for the API endpoint.

```
var putTask = client.PutAsJsonAsync<User>("api/Users/" + id, user);
```

Filter Function

Certain controllers have a filter function and page. For instance, the items table can be filtered by ItemType using the function FilterByIT. where the action link in the index cshtml page calls the function FilterByIT and passes the Item type ID as a parameter, then redirects to the filter page (Which is in the list format like the index page).

The function is similar to the Index function up until the last, which instead of just returning the ItemInfo list will filter it where item.ITID (itemID) is the same as the id passed through as a parameter.

```
return View(ItemInfo.Where(Item => Item.ITID == id));
```

Create New

IID	ItemName	ITID	
1	Heinz Beans	1	Edit | Details | Delete Filter By IT
2	Tescos own Bananas	2	Edit | Details | Delete Filter By IT
3	Sainsbury's Sausages	3	Edit | Details | Delete Filter By IT
4	Branston Beans	1	Edit | Details | Delete Filter By IT
6	Hovis Bread	6	Edit | Details | Delete Filter By IT
7	Davidstowe Cheese	5	Edit | Details | Delete Filter By IT
5	Tesco's Sausages	3	Edit | Details | Delete Filter By IT

Figure 22, Index item page displaying the filter by Item type

So if the user clicked the filter by IT Button on the first item as seen in figure 22, the ITID of the generic beans which is 1 is passed to the filterByIT function where it returns the Filter list page where Items ITID == 1.

Create New

IID	ItemName	ITID	
1	Heinz Beans	1	Edit | Details | Delete
4	Branston Beans	1	Edit | Details | Delete

Back to List

Figure 23, filter items by item type

Once pressed, the page displays all items under the item type of beans which is Heinz beans and Branston beans as seen by figure 23.

Mock Adapter for External API/ Barcode and Shopping Cart

To simulate different API being connected to the system, a folder was created in the API application called Adapters, this folder holds for each desired connection a C# interface class that defines the adaptor's interface and a class that conforms to the interface, along with some adaptations to how the model is formatted.

For example, there is a dummy barcode adapter and interface named ImpBCDUmmy and IBarcodeAdapter respectively. The Ibarcode interface defines item findproductByBarcode passing barcode as an int parameter.

```
Item FindProductByBarcode(int barCode);
```

Within the Model, Item.cs, this code was added so the data being passed conforms to the model structure, where it can be passed no parameters, parameters of string or id, or those and TypeID.

```
        public Item() { }
        public Item(int ID, string Name)
        {
            IID = ID;
            ItemName = Name;
            ITID = null;
        }

        public Item(int ID, string Name, int TypeID)
        {
            IID = ID;
            ItemName = Name;
            ITID = TypeID;
        }
    }
```

ImpBCDUmmy class creates a dictionary collection of fake data called BCValue, this dictionary has integer and item data being passed to it. Fake data was input where the first number of the dictionary is the id of the barcode that it receives from the external Barcode API. Additionally, in the second section of the dictionary a new item is created with item name and id being given for that item.

```
IDictionary<int, Item> BCValue = new Dictionary<int, Item>() {
            { 88887777, new Item(1234567, "Beans")},
            { 77776666, new Item(2345678, "Sausages")};
```

There is also a class that returns the data in the format of the datatype Item

```
private Item ReturnTranslate(object data)
        {

            return (Item)data;
        }
```

The system has another class called findProductByBarcode that takes int item as a parameter, it then returns the ReturnTranslate function where BCValue[item] is the parameter.

```
public Item FindProductByBarcode(int item)
      {
          return ReturnTranslate(BCValue[item]);

      }
```

Another similar dummy adapter was also created for receiving shopping from a basket online, where instead of looking at the Item Table it looks at the HouseHoldItem Table.

For each of these dummy adapters an API endpoint was created, to use the adapter the code below had to be instantiated at the start of the main class.

```
private IBarcodeAdapter barcodeAdapter = new ImpBCDummy();
```

Making sure that the Response type is the Item data type, calls the GetBarcode function.

```
[ResponseType(typeof(Item))]
        public async Task<IHttpActionResult> GetBarcode(int barCode)
        {
            return  Ok(barcodeAdapter.FindProductByBarcode(barCode));
        }
```

If using postman or browser, the endpoint can be reached by typing:

“http://web.socem.plymouth.ac.uk/FYP/SSharpe/api/Items?barCode={88887777}”

Which displays the dummy data related to that barcode lookup which is beans.

Requirements for Application

Internet connection and access to an internet browser, and when user functionality would be implemented a user will require an email which must be verified.

4.5 User Interface Design

For this project, two prototype user interfaces were created. The process involved creating an initial mock UI which was simple using PowerPoint, test it with users, then create another more complex User Interface using Adobe XD improving upon design with the input of the user testing.

Usability Testing

Between each of the iterations, questionnaires were given to test users to gauge if the design was desirable, the questionnaires can be found in appendix B.

After the first round of testing it was clear what changes need to take place on the user interface of Stokk. One prominent complaint was that the users did not like the idea of not having a form of navigation bar or a form of quick navigation through the pages. This can be seen implemented on the second prototype where at the bottom of the page there are symbols representing each function page, as well as a home button that takes them to the main page.

Other improvements were made such as that of the colour pallet used, in the initial design the idea to use a theme of blue, but users found it too unattractive, so in the second design uses more vibrant colours as well as having a mix, while still staying in the relative colour area.

The tested users also noted that the use of having images would be a good feature that would help the see both the ingredients and the finished meal looked like.

These are some examples of the UI design and the changes that were implemented.

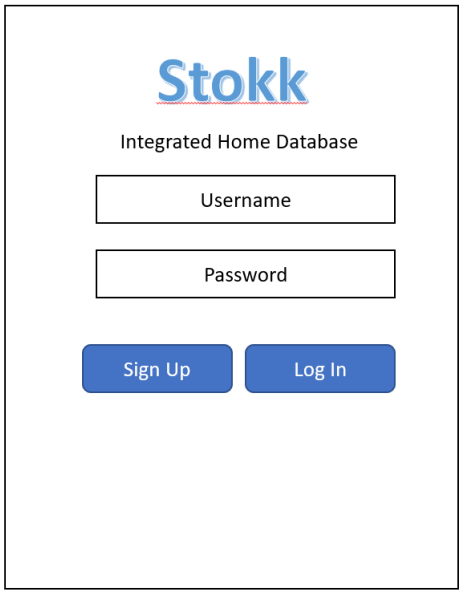


Figure 24, initial mock Login page

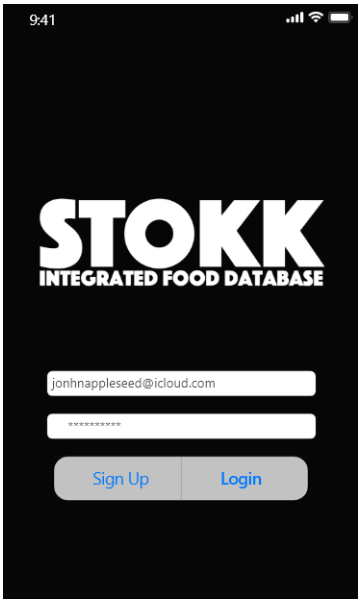


Figure 25, prototype Login page using adobe XD

In figures 24 and 25 the prototype login pages are displayed. Simple Login Page where the user has to input username and password to enter into the main page. If a user doesn't have an account yet, the sign-up button allows the user to create an account, similar design in both the mock and prototype.

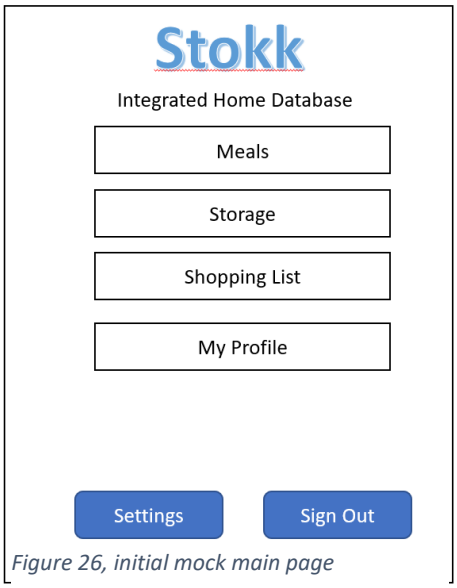


Figure 26, initial mock main page

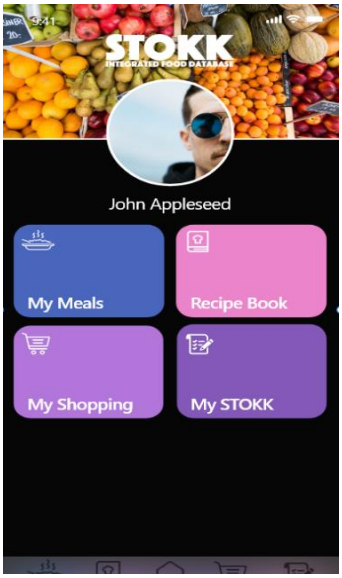


Figure 27, prototype main page using adobe XD

Figures 26 and 27 display the main page for the mock/ prototype UI design this will act as the first page a user will see once they log in, this page works as a hub to direct to other pages. Each button on the main page navigates to the other pages of functionality, Shopping list, recipe handling and stock control.
Sign out button signs the user out and redirects the user to the login page.

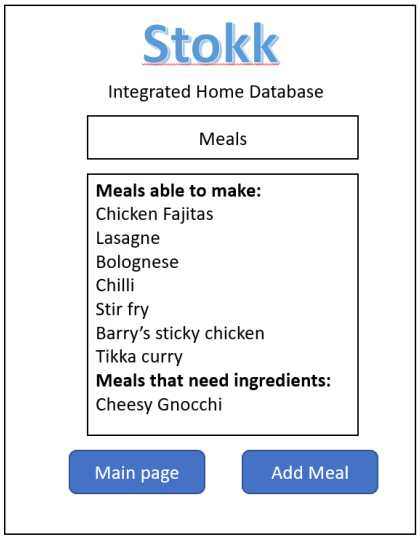


Figure 28, Meals mock UI page

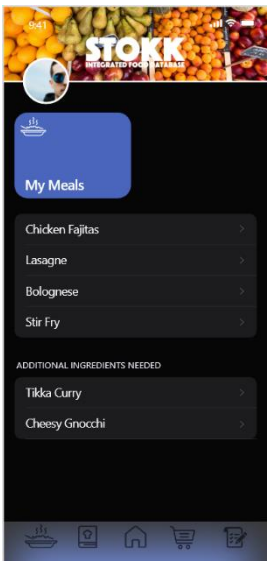


Figure 29, Meals page prototype, adobe XD

The meal page is displayed in figures 28 and 29, the initial design includes a main page button to take user back to main page directory, the adobe XD prototype prefers a navigation bar which has a house denoting the home/ main page.

On the meal page, a user can view meals that they can make, much like a recipe book, allowing the user to view other recipes made by others, or add their own.

This is separated out into meals they can currently make with the ingredients they have and the other section being all the other meals, where the user will be told what ingredients they need to buy to successfully make that meal; possibly here the user would be asked if they wanted to add those ingredients to the shopping list.

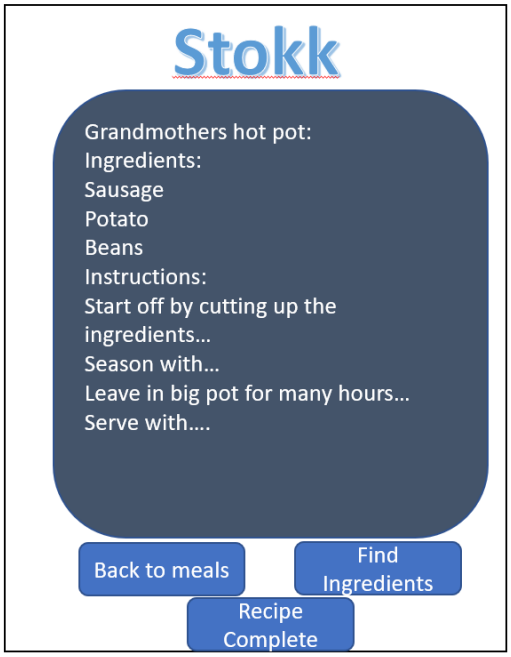


Figure 30, mock UI selected meal

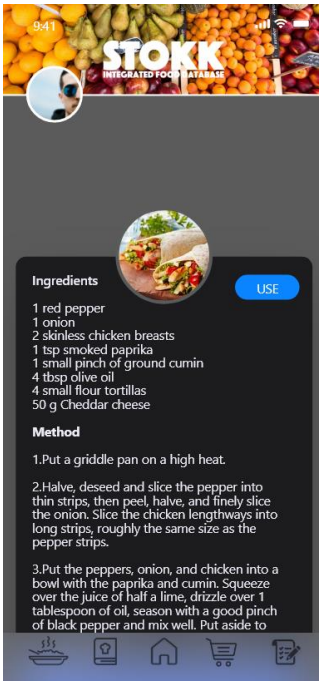


Figure 31, prototype UI selected meal, adobe XD

Once a meal is selected, a modal popup will come over the current page for the user to interact with where the user will see the ingredients involved and how it's made.

If the user presses find ingredients the user is given another pop-up modal which shows each of the ingredients and where they are positioned, this was elected to be removed from the later versions, and so is not seen on the adobe XD prototype.

Once the meal is complete the user then presses recipe complete in figure 30 or use in figure 31 where the database will take away the ingredients used in the meal away from the database. Possibly here is where we could ask the user how they would rate the meal.

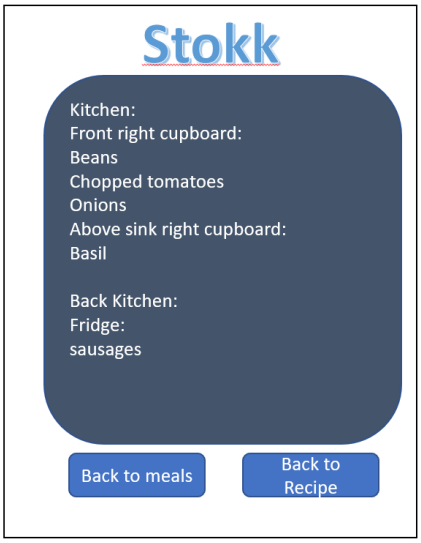


Figure 32, mock UI ingredients needed for meal

Figure 32 shows the page used to direct the user to the ingredients location, but this feature was avoided as it can be assumed that a user will know where the ingredients are located in their house, making the flow of the pages more simplified.

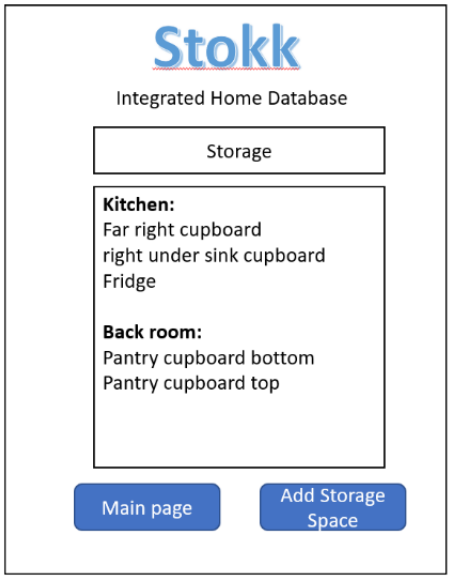


Figure 33, Mock UI for storage

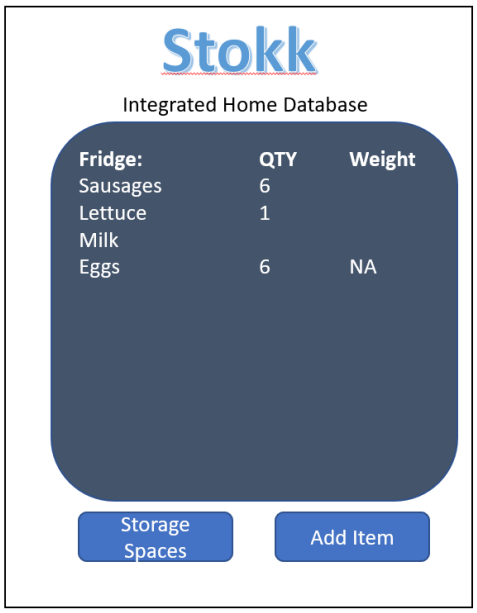


Figure 34, Mock UI for a storage space

These pages displayed by Figure 33 and 34 were deemed to be ineffective, and instead the rooms and storage spaces would be used to help order ingredients in a specific household. On this page, all ingredients would be shown but still grouped by the room and storage space, but it is not necessary for the user to navigate through all the rooms and storage spaces to see if they have an ingredient.

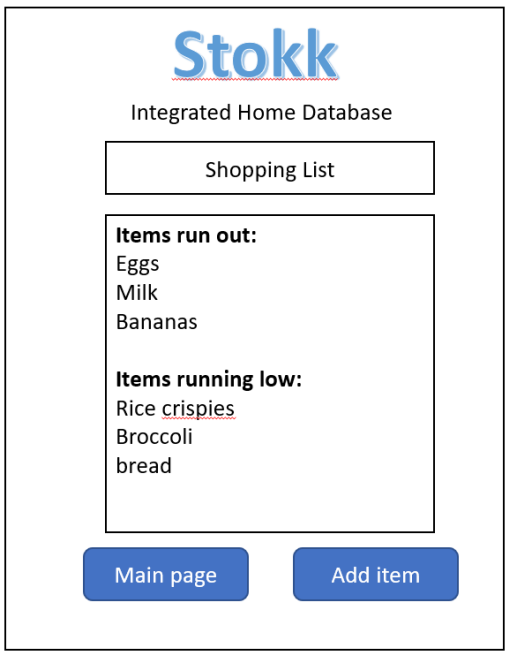


Figure 35, Mock UI for shopping list

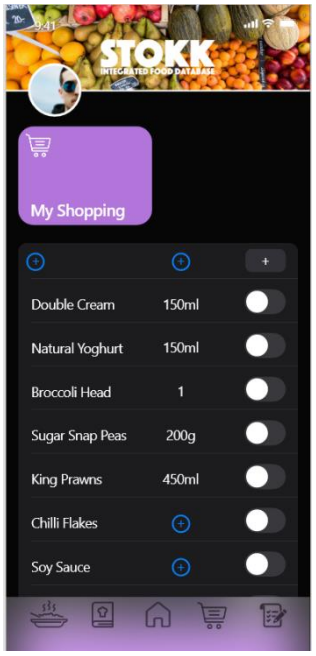


Figure 36, Prototype UI shopping list, Adobe XD

Most of the shopping list functionality will be attained by utilizing triggers within the database, to remove data from one table and move to the shopping list, although both designs in figures 35 and 36, allow the user to edit the shopping list and add to them if they so wish.

Although the application could not be made as a mobile application the use of web-based applications can help bring a wider audience of users as well as keeping the UI to a shared and conformed design.

This means that no matter if the user uses a mobile phone or desktop the application will share the same UI features and as such “Accuracy of recall was higher at longer retention intervals, in contrast to the usual fall of accuracy with time, which is seen when retention is measured over a period of minutes or longer.” (Crawford, et al., 1966). This means that no matter which system a user chooses, they will be able to remember the flow of the web page as they are the same.

4.6 Database Design

SQL code for the tables and sequences can be found in Appendix A.
Within the Database, there are triggers in place for certain functionality, such as the creation of a households shopping list once a household is created, removing items from household item and adding them to the shopping list, if the household item reaches 0 it is removed from household item and the urgency Boolean is added to the shopping list item. And the final trigger is one which creates a desired stock value, which is used for the functionality of the shopping list, as the quantity changes with every meal or usage. Once an item is added to the shopping list from the house hold items it needs to refer to the amount that the user desires to buy, not the current value, so when a user places an item into household item the original QTY is stored.

Similarly, to achieve the functionality of removing an item from the shopping list, when the QTY = 0 then the item is removed from the shopping list and added to the household item with the original QTY before being turned to 0.

These functions all work similarly, to start with the trigger declares variables which will hold the separate values for QTY both before the update and after the update, using the updated value to check if the function shall execute and the other value used as the QTY to be transferred.

Entity Relationship Diagrams Through Development

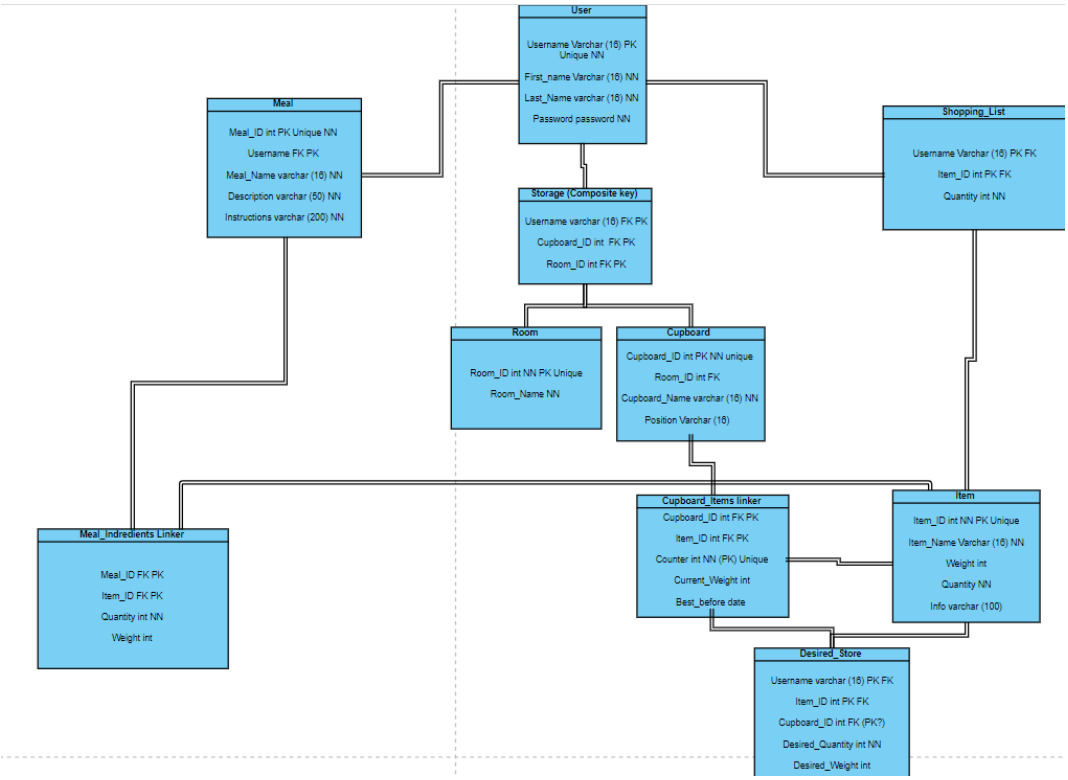


Figure 37, Initial ERD for system

figure 37 was the initial ERD which was conceived early in the project design phase, although this is ideally how the final database design would look like (without the correct relations), a simpler form of the application was built initially to try and build up the functionality, with a simple table that contains the Item with its relevant information such as quantity and the location of the item by room and cupboard attributes.

In this design, it was important to make sure that each item was connected to a user so that a user could see their own storage spaces as well as items within them. The next step was to connect the user table and storage table where Username is a foreign key that is attributed to each item.

A realisation formed about the system, as in a household there could be numerous people living there, so even though there are many users they would still be using the same ingredients and spaces, so instead, it was decided the system should be based around households. A household can have a primary user or admin user and then there is the secondary user or restricted user which has limited functionality compared to the admin.

To fully understand how this should be implemented correctly the database was redesigned from the bottom up, starting off with core functionality so just stock keeping, ignoring the recipe handling and shopping list functionality was left out to start with.

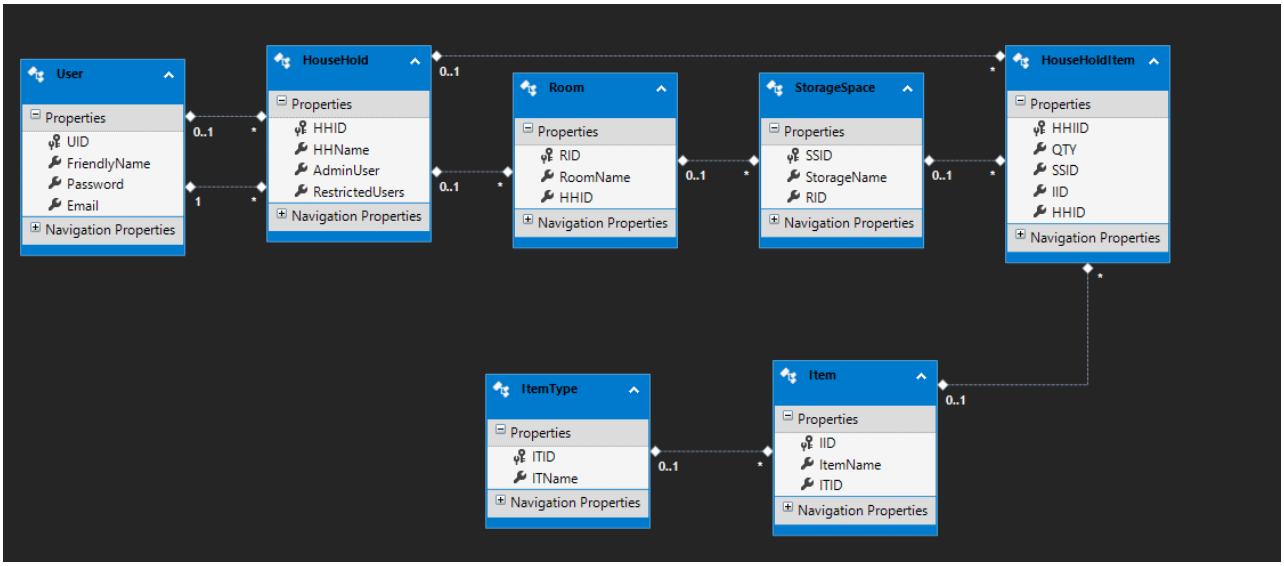


Figure 38, ERD for first phase of database, storage functionality

As shown in figure 38 the redesign took place, during which another interesting architectural idea came to form and that was that a household would have an instance of an item. This item references from the Item table which has the branded name of the Item for instance Heinz Beans, which also references Item type which is the abstract type of the item so for the Heinz beans item type would be beans. This way Multiple brands and shops can be chosen from, while still being connected to the core abstract that it is a tin of beans.

It was realised that rooms and storage spaces seemed like redundant data as it wasn't necessary for the user to know where items are as it can be assumed a user of a household (unless they are children) would know where most ingredient are. However, the tables were kept in to help display a household's items by room and storage space, this was kept in order to have some order to the household items, hopefully using the rooms and storage spaces to order the household items table on display. Another reason for keeping the tables in is that, essentially the application can be used for other storage/ stock control, by giving the user the ability to add items and item type a user could even add non meal related items.

From this database a simple API was created/tested and a front end application which was able to CRUD the data was created to get the practice and connections necessary for the final product, and so that

data could be manipulate and viewed so the API endpoints were tested and there were no flaws with that current system.

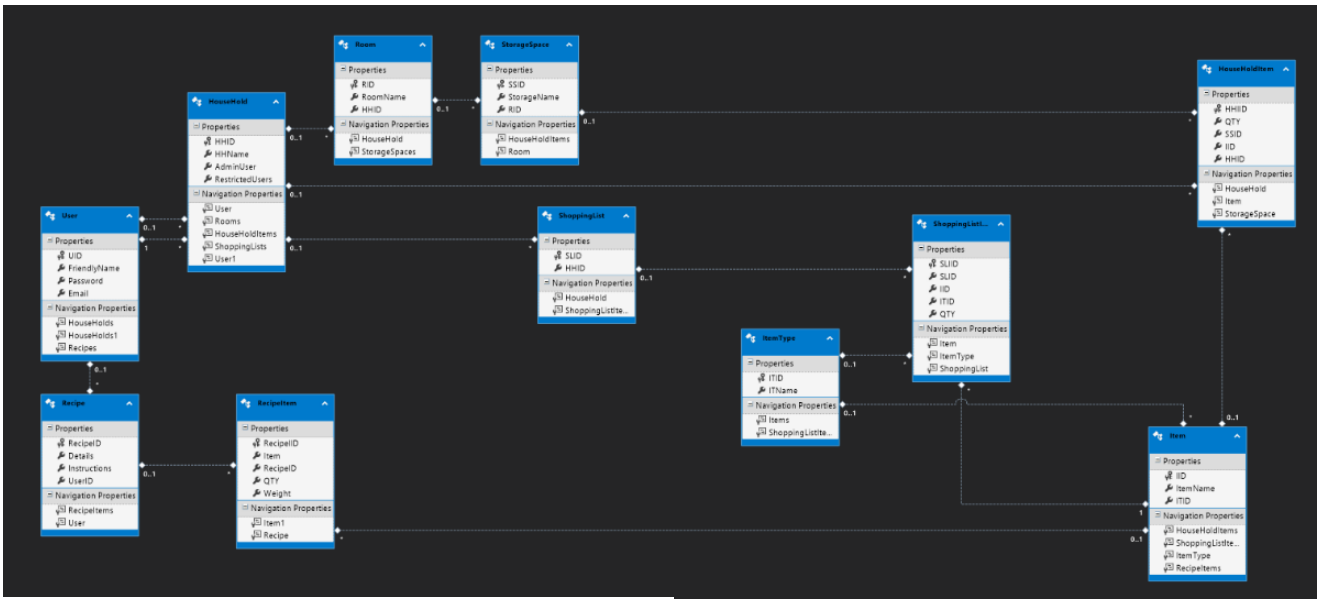


Figure 39, ERD Phase 2, shopping list and recipe functionality added

After the first build figure 38 was finished and tested, the shopping list and recipe tables were added to the database as seen by figure 39. Both tables have 2 tables for functionality – firstly, for the shopping list, there was a shopping list table and shopping list item table. Shopping list is a simple table which has the household as a foreign key and has the shopping list ID as the primary key, this table is used to cross reference which list is owned by which household.

And the shopping list items table has the item, QTY, and which shopping list it is a part of through SLID.

The Recipe Table has the recipeID, details of the meal and instructions for the meal as well as which user created it as a foreign key.

Recipe Item has the collection of ingredients needed for every meal where the recipeID is a foreign key as well as Item.

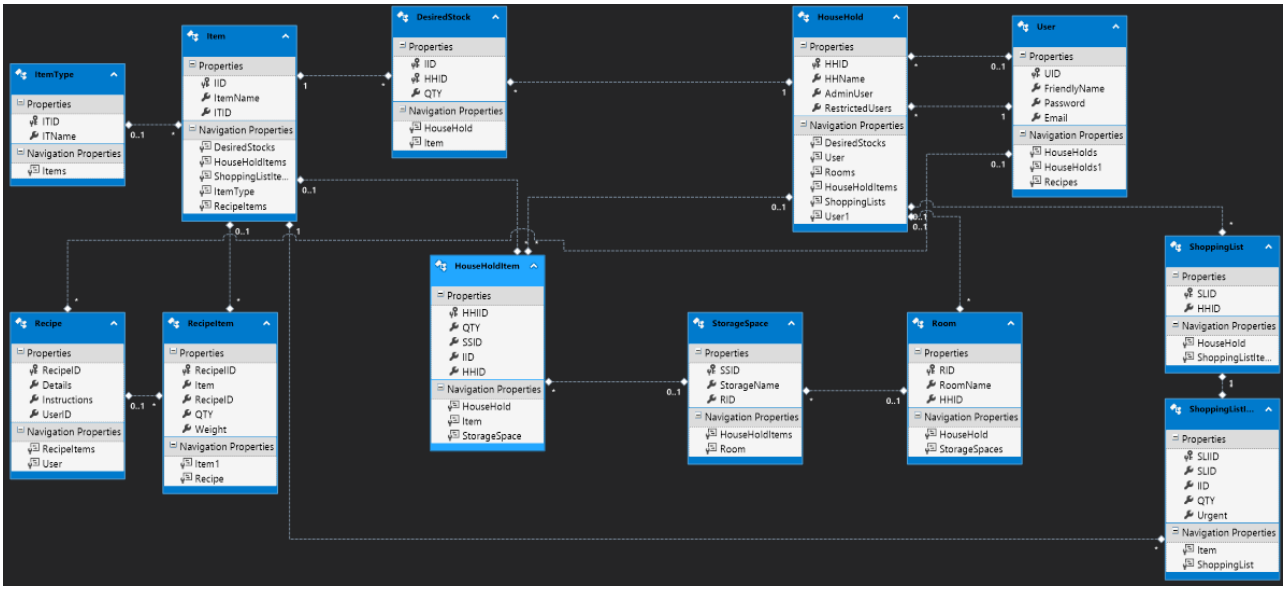


Figure 40, ERD with implemented Desired stock table

In figure 40's ERD some changes were made, such as the table of desired stock was added. This table's purpose is to hold the ideal QTY that a user wants of an item, when being added to the shopping list. This table is heavily utilised by the triggers in the database system, it is assumed that at some point either when during a user's first shop or after, a user will input the items that had bought into the household items table, as this value is the just bought QTY which means it is their preferred buying quantity and as such its itemID household and qty is saved. This table is then used when an items QTY in the household item table reaches 1 and 0, as the triggers work off of on update and checks the QTY values , if it is 1, then it is added to the shopping list items table, once it is 0, the item is removed from household items and the urgency Boolean is changed to true. In the mentioned function as the update relies on the changing of the QTY value, it is impossible to use it as a reference point as to what quantity of an item they want to have on their shopping list and buy.

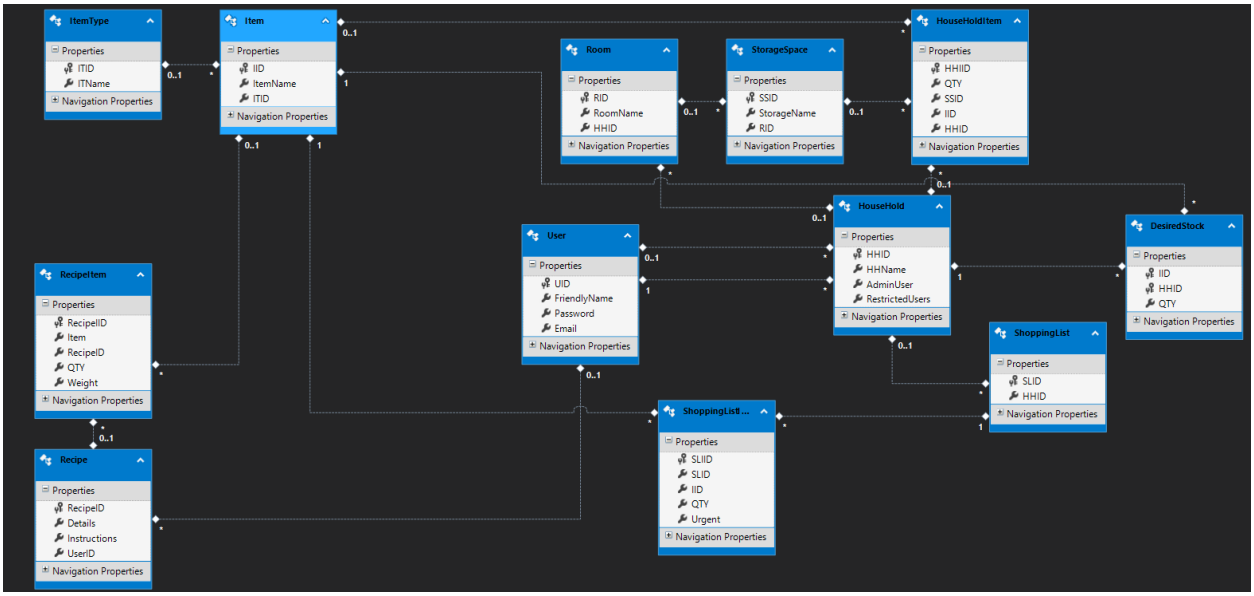


Figure 41, final ERD of system

Other changes can be seen in figure 41, such as adding urgency Boolean to shopping list items, this will help to display the shopping list in order of urgency, so most urgent items are top priority and displayed first, and item type attribute removed from shopping list item table as the item table already references item type.

One noticeable feature is that most of the tables have their own Primary key IDs, this was to help out with construction of the web API, as most API calls with asp.net refer to an individual ID for Updating, Deleting and Reading specific sets of data, so it would help for development to have IDs on most of the tables for easier reference.

4.7 Diagrams throughout development

Activity Flow Diagrams

These diagrams were created to help understand how the flow of the system would be, when a user interreacts with it, this stage greatly helped with gaining knowledge on specific functionality and buttons needed for each page.

In most of the diagrams a reference to a modal pop up is suggested, although this was not achieved as a web application was undertaken instead of a mobile application. The prototypes for the UI are still displaying a UI for mobile but hopefully these features can be shared to the MVC application.

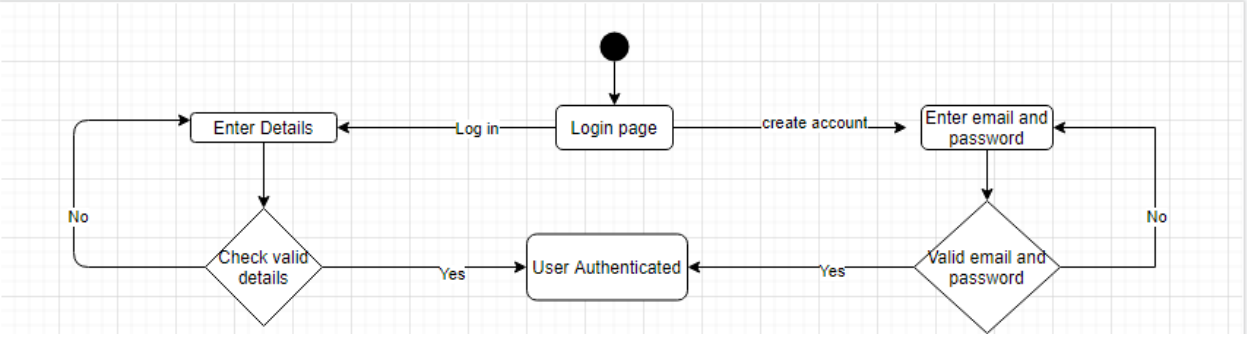


Figure 42, Activity diagram for log in page

Figure 42 Displays a simple interaction between the user and the system for the logging in functionality, features included are logging in and signing up for an account.

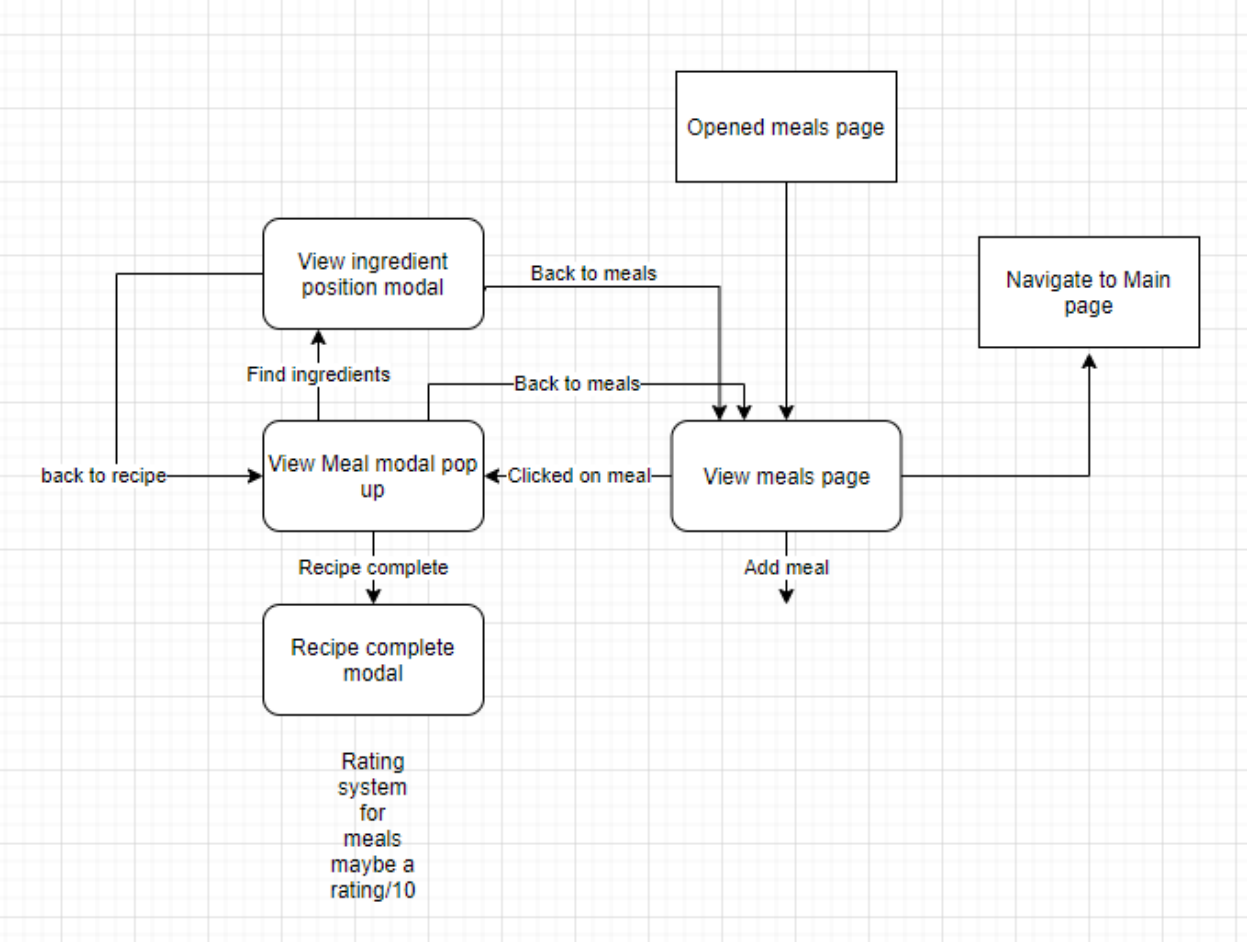


Figure 43, activity diagram for meals/ recipe functionality

Figure 43 contains the activity diagram for the recipe/ meals functionality for the Stokk system, adding meal is directed to another page for separate functionality which takes user input for the creation of a

recipe. A user views the possible meals and chooses a meal, this should take the ingredients out of their store in the database. The functionality of telling the user where ingredients are was abandoned.

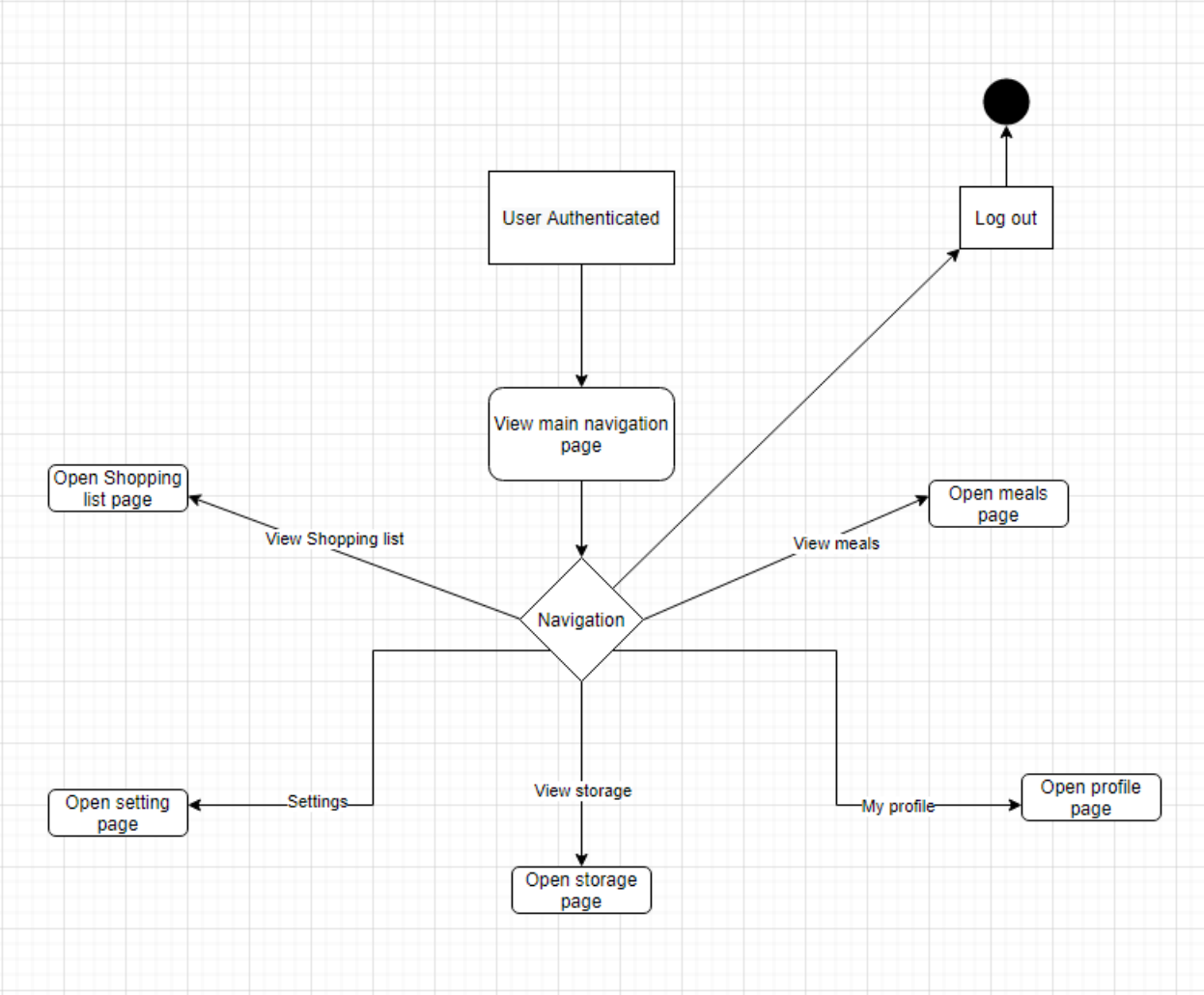


Figure 44, Activity diagram of main/ home page

Activity diagram for the home/ main page is displayed in figure 44. The functionality of this page is simple to redirect to other pages for their functionality, within the prototypes for the UI the main page format is kept as well as having a bottom navigation bar which acts the same as the home page.

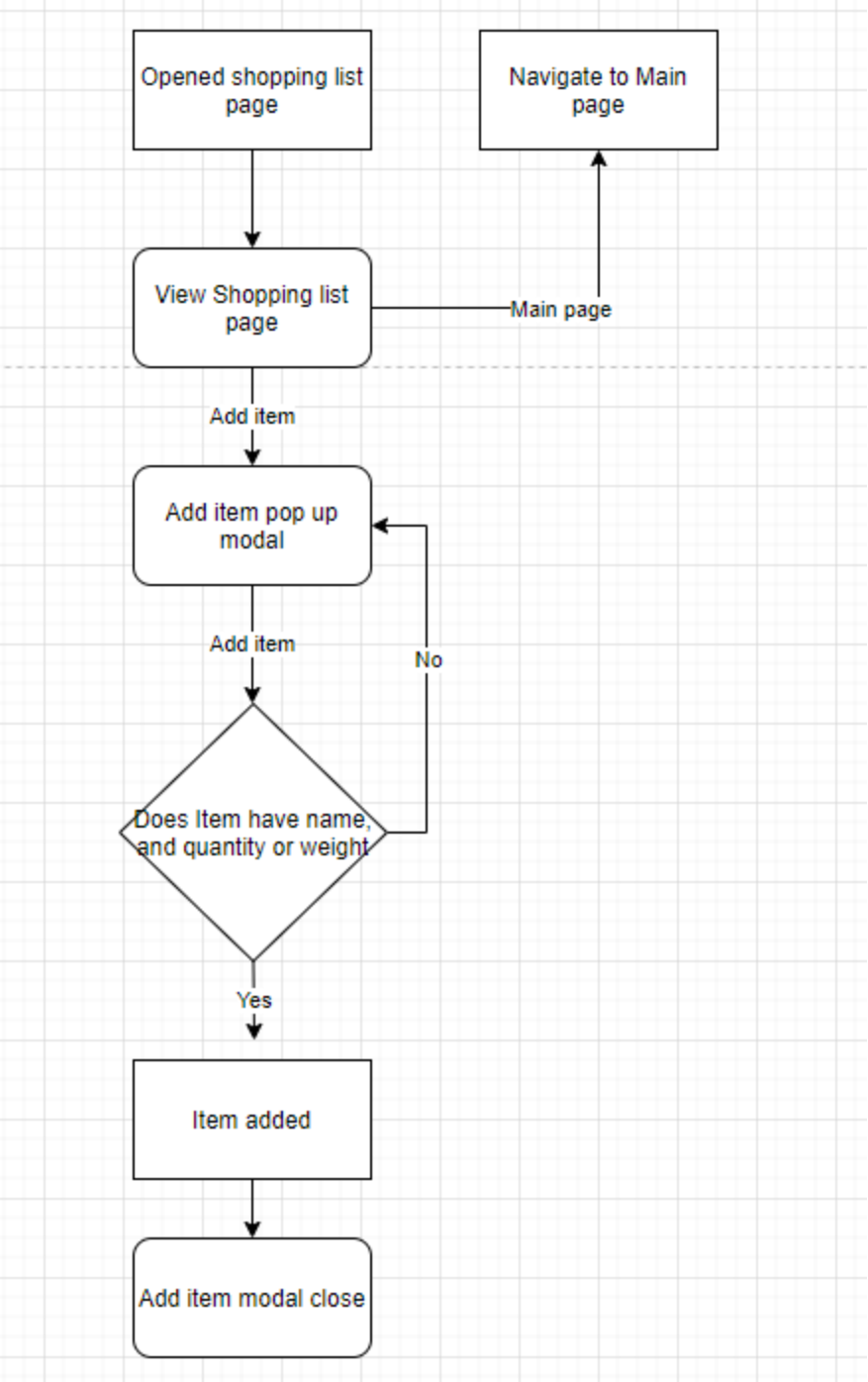


Figure 45, Activity diagram for shopping list functionality

Figure 45 displays the activity diagram for shopping list functionality which includes viewing the shopping list, adding items and editing items.

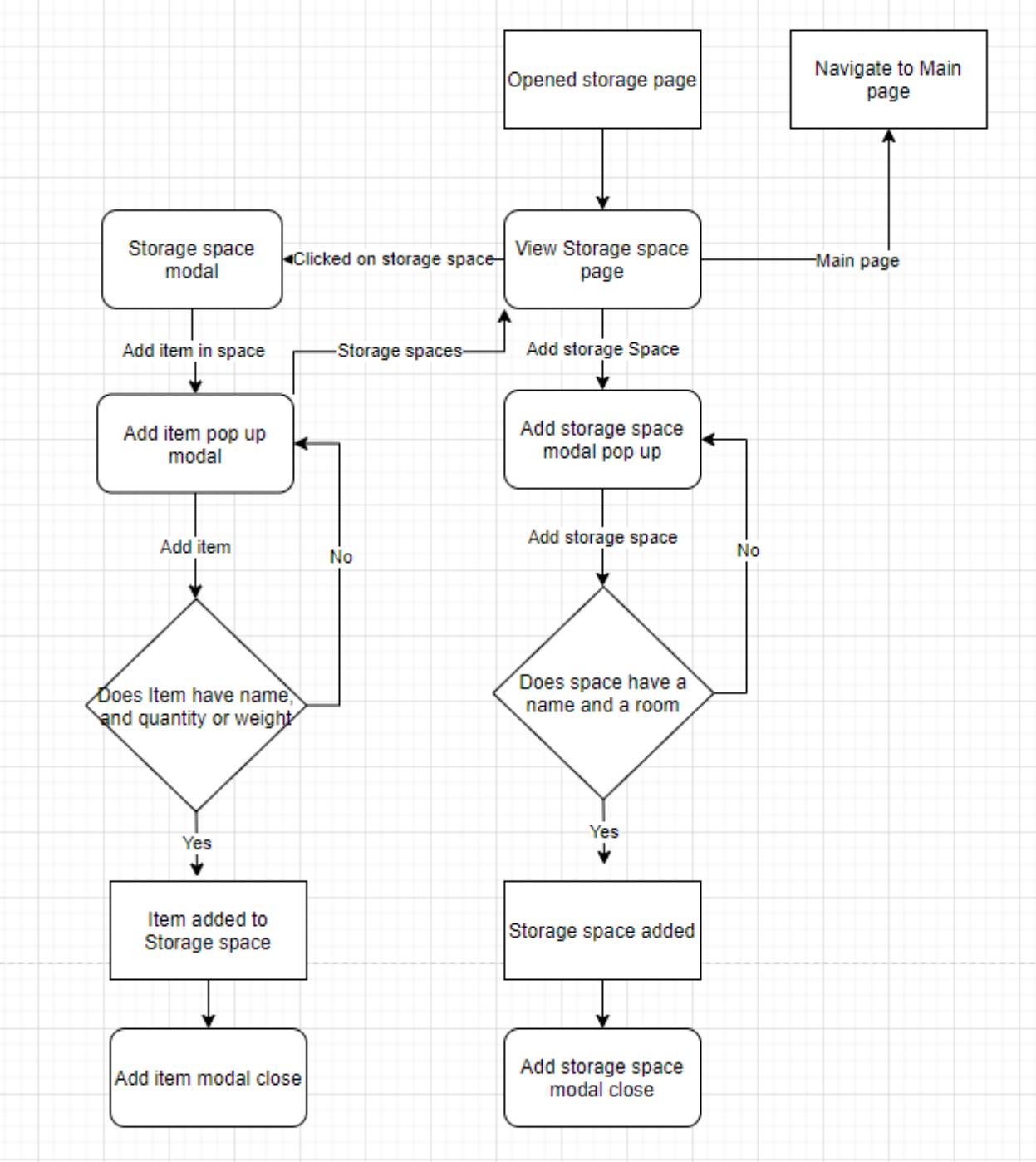


Figure 46, Activity diagram for storage space functionality

The activity diagram for adding, editing for storage space and items within those spaces can be seen within figure 46. This design to have a desired page which allows a user to create a room and a storage space would be separate from the household items functionality, to not confuse the user.

Class Diagrams

These class diagrams display the classes within the project, for the API application created in asp.net and the MVC application (API exerciser), they display the functions and data held within each page. Both of these applications share the MVC structure, and bother are connected as the Published API endpoints are consumed by the front end MVC application.

API class diagrams

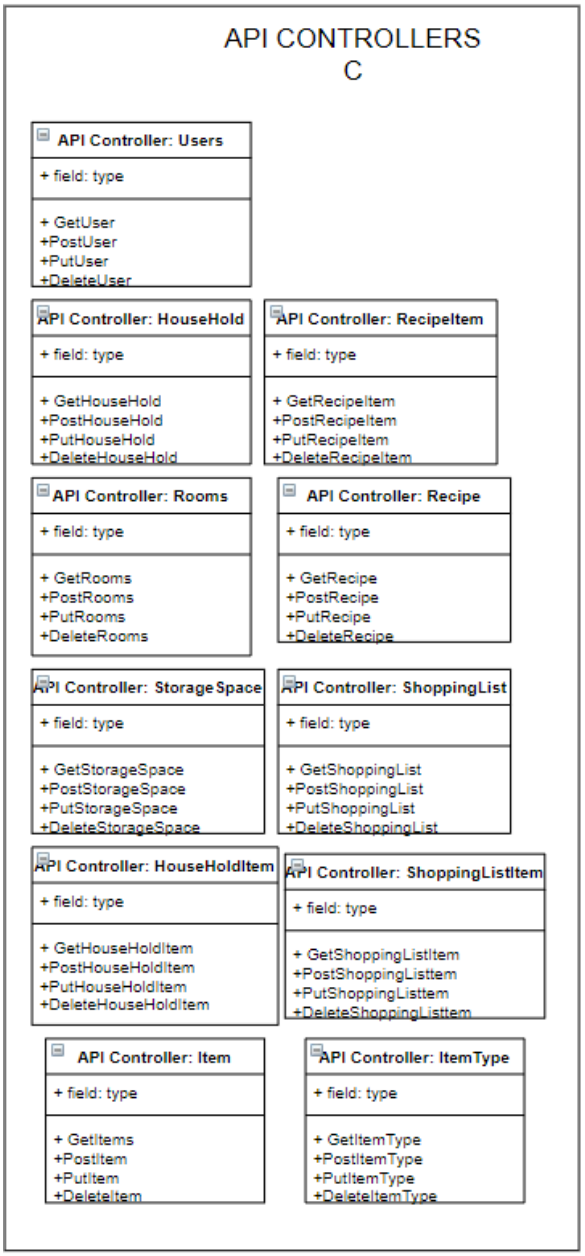


Figure 47, UML for API Controller

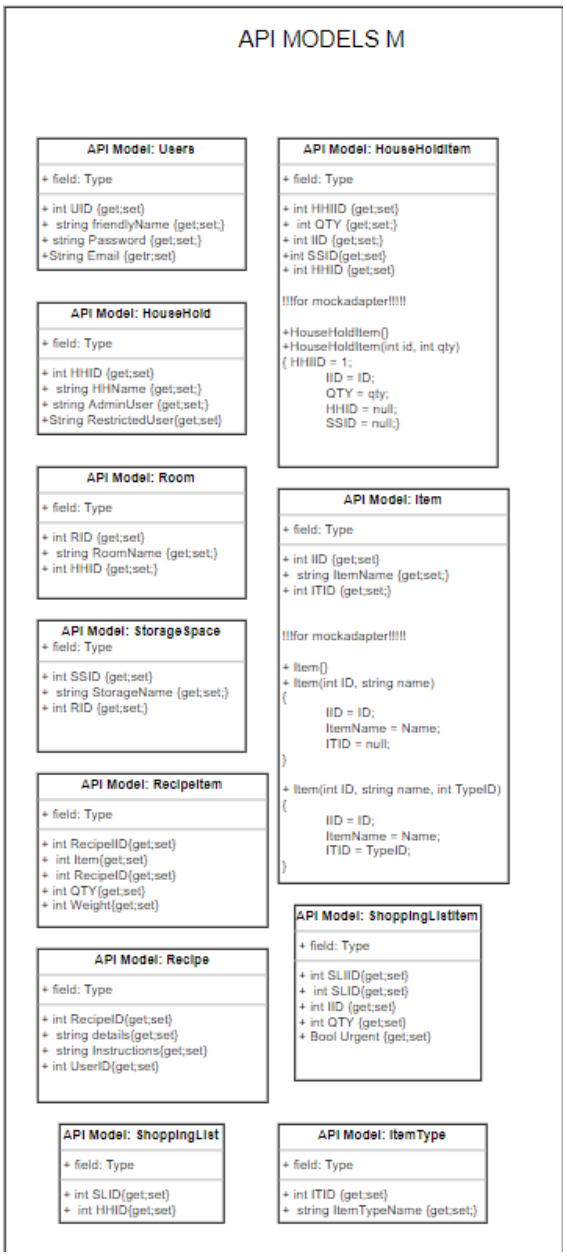


Figure 48, UML for API Models

Both the controller and models for shown in figure 47 and 48 are within the API application, these diagrams show the functions within each of the controller and model classes. The API doesn't need the

use of Views as the front end is handled by a separate MVC application that handles the then endpoints created by this application.

MVC application (API exerciser) class diagrams

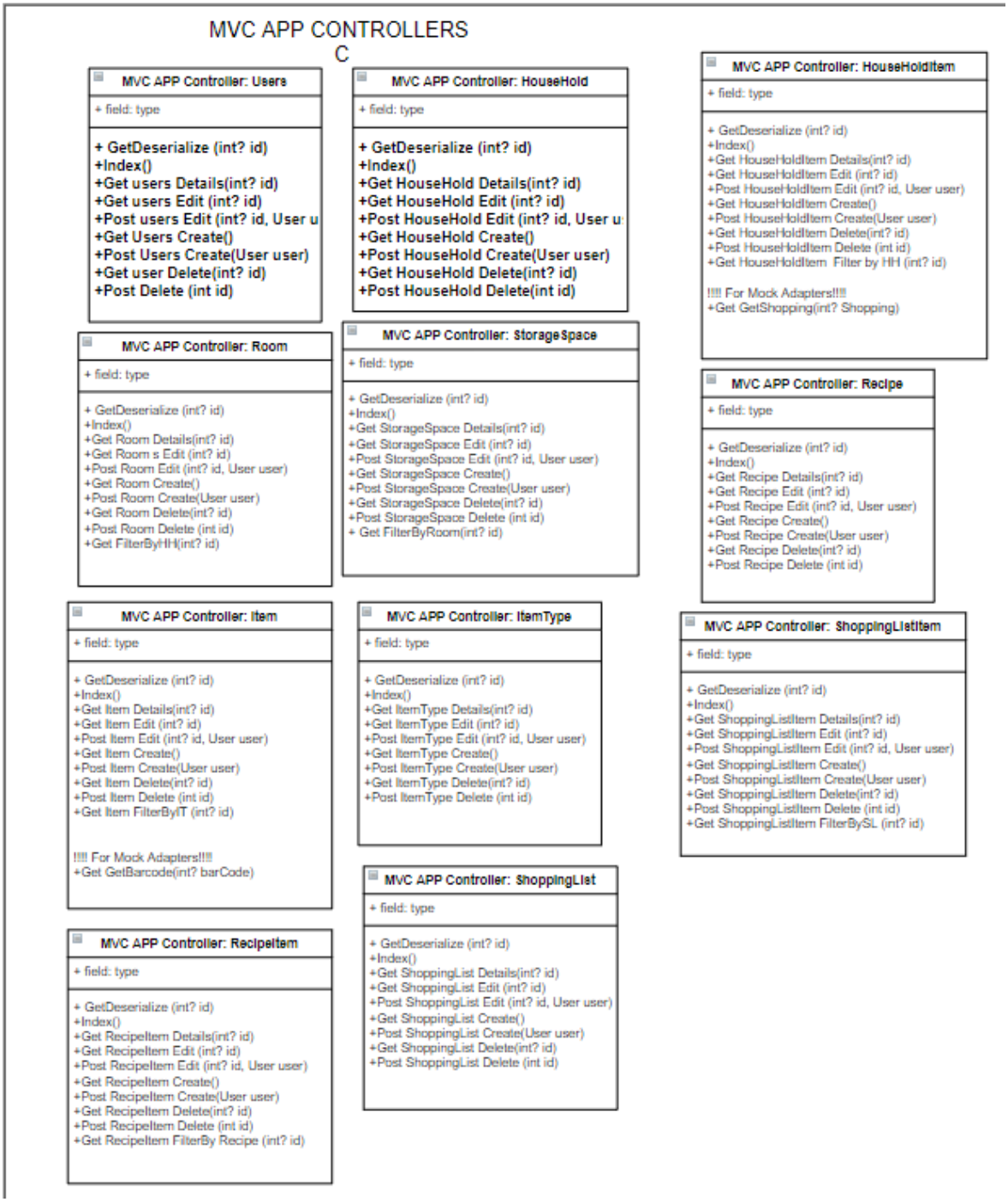
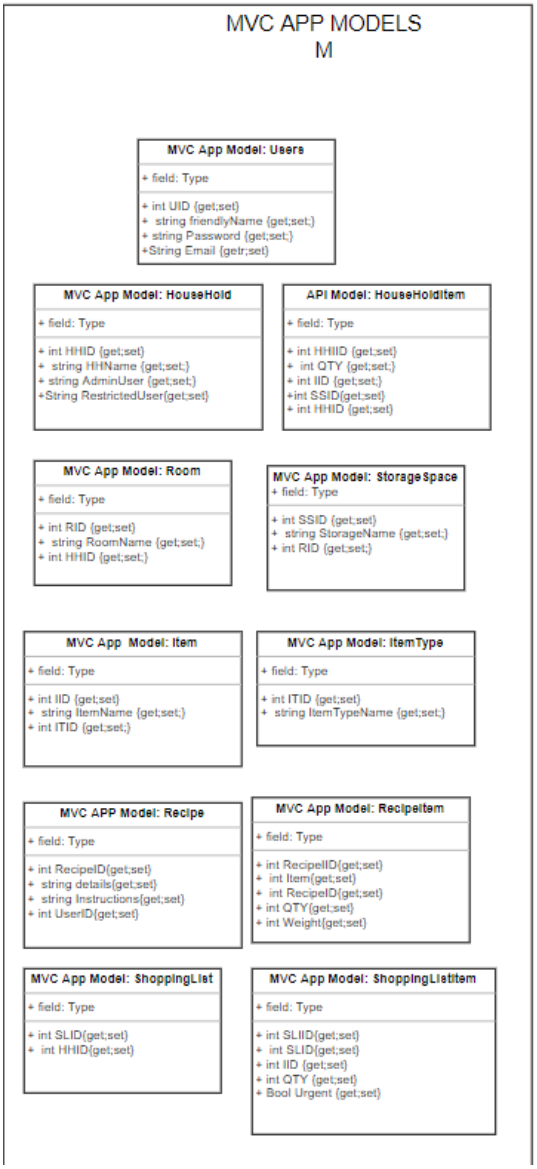


Figure 49, MVC application Controllers

Figure 49 displays the Classes for the controllers of the MVC application, which deals with the function which interact with the user and the models.



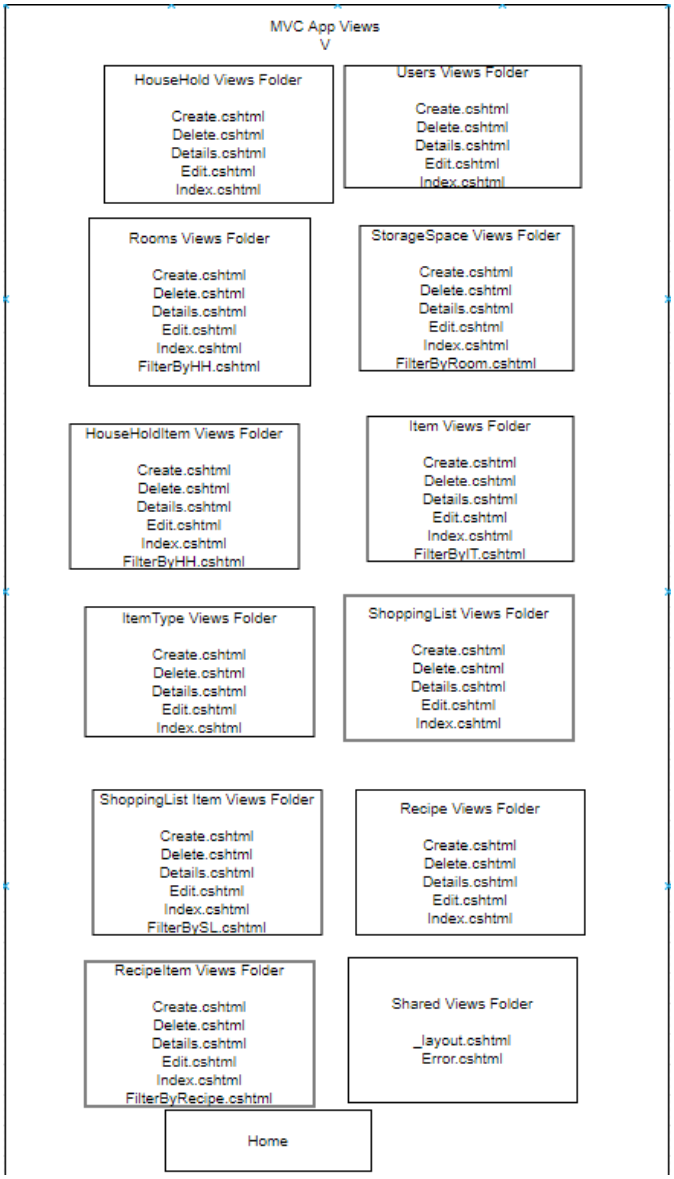


Figure 50, MVC application Models

Figure 51, MVC application Views

The models displayed in figure 50 are of the MVC application, these models are similar to that of the API application, although these are created manually rather than having an Ado Entity model created.

In figure 51 the Folders containing the view pages, each of which have a different functionality displaying different kinds of data at different points, like displaying all of the data or filtered objects by a certain parameter.

Conceptual Design diagram

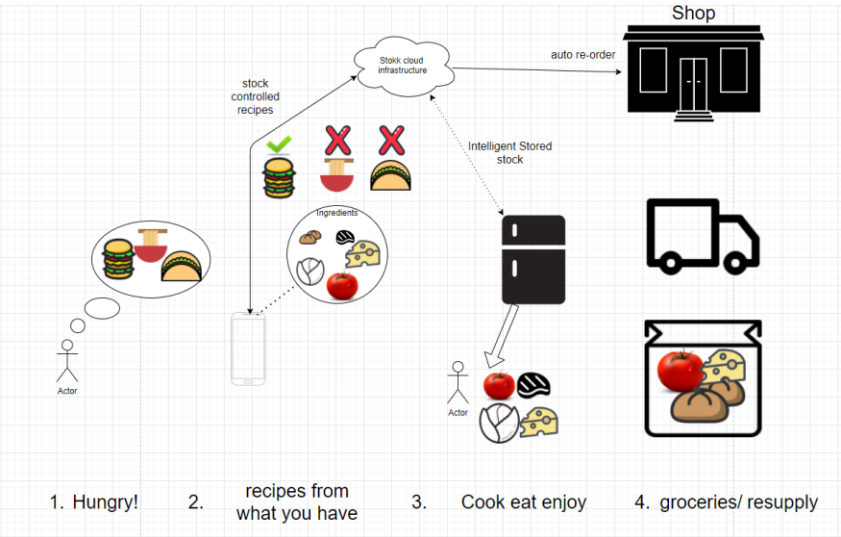


Figure 1852, conceptual diagram for system

Figure 52 displays the conceptual design for the application, giving an inciteful but simple explanation of the application, this was used with in the poster.

4.8 User Stories

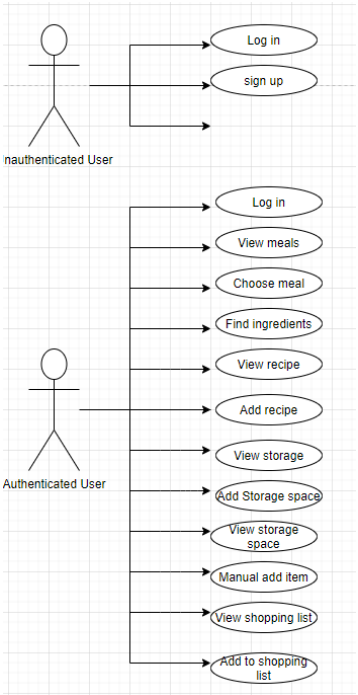


Figure 53, User case diagram

Figure 53 displays the first initial user case diagram; this would be one of the first steps for figuring out what the core functionality should and help for the creation of user stories.

	User story
1	As a user I want to View my houses shopping list
2	As a user I want to Edit/Add the shopping list
3	As a user I want to view my household's ingredients
4	As a user I want to view Recipes
5	As a user I want to be able to choose a recipe from the ingredients I have in the household.
6	As a user I wish to add my own recipe.
7	As a user I wish to choose a recipe and the system should remove the items from household items
8	As a user I want a to be able to look at my shopping list and see what items are getting low and what items are needed Immediately.
9	As a user I want to be able to log in and out of the system so they can check their stock.
10	As a user when I am low on an item, I want the system to add that item to my shopping list

Chapter 5 Implementation

5.1 Project Management

Every 2 weeks a meeting with a project manager/ tutor was held where every session was attended and a stand up took place, although a far cry from the daily stand up meetings which are used in the agile methodology in the work environment, this helped with vocalising issues and answering any queries.

On the alternating weeks a small blog post detailing what we did that week was also written for the tutor to follow.

Although these meetings were helpful, extra precautions were put into place to help this project move forward in a more structured manner. Biweekly meetings with another tutor were held, this helped keep the application stay on track.

For the final month or so of the project Daily stand-up messages were sent to a person who was keeping track. These were sent at the end of the day they outlined what progression was made during the day, what difficulties were had and what the plan for the next day was. This helped structure the working days on a day-to-day basis rather than a fortnightly, so progression was tracked more closely. This helped drastically with the progression of the project.

5.2 Stages of Sprints

Sprint	Topic	What was achieved
Sprint 1,2	Conception and Design	- Created simple ERD of a proposed system, State visualisation, Mock UI's and research on other applications similar in design. - Tried to normalise data - Report writing done a little bit every stage - Tutorials on the technologies was done - Write the Create table statements for possible tables.

Sprint 3	Attempted Xamarin/ UI	- Wanted to start off with User creation and Logging in functionality, but issues - created second prototype UI after testing.
Sprint 4-6	Redesign/Building test environment	- Redesign considered and taken place based on household. - Tutorials for Web forms, MVC and Asp.net API were done. - Many practice projects created. - Test environment Created with 1 test table, Fully CRUD front end that interacts with API endpoints.
Sprint 7	System Build Start	- Created some tables of the system not including shopping and recipe functionality. - Created and published API, which interacts with current tables - Create CRUD MVC web application (API exerciser) - Add filtering on certain pages
Sprint 8/9	Finishing off system and having fully finished API Exerciser	- Recipe and Shopping list tables added to the database - Endpoints created and published for those tables - Fully CRUD frontend added for the new tables, with some filtering - Added triggers for shopping list functionality - Added sequences for the ID's - Not allow user to Edit the ID - some mock adapters were created in the API to mock a

		connection to a barcode and shopping basket API.
Sprint 10/11	Report Writing and bug fixing	- writing report. -proof read report -improve the report - bug fix any issues

5.4 Security

As getting prototypes of each feature was prioritised, security was not deemed as a high priority within this project, although security is a very important feature within a consumer product, if this app were to go further into production many features would be implemented for the safety of the users.

Security/Error Handling

Although in the final system security would be a very important feature to have, as eventually the application may have personal user data that is needed to be kept safe which would be done through hashing the passwords, for the moment the system only uses varchar for the use of the password but this would not be in the developed version. Error handling has been held off in exchange for attempting to achieve the core functionality first.

Aren't able to edit the ID Primary keys of most of the tables, as they are sequences, this avoids duplicate data, which can cause major issues.

5.6 Functionality Testing

Between the changing of technologies many tutorials were taken and mock projects created to test if the architecture would work and practice those skills, It was decided that here a test environment should be created to demonstrate a full system connection of the chosen technologies.

For testing some of the API endpoints early on in development, Postman was used, this was helpful in that a front-end application wasn't needed to test if the API endpoints were published correctly and working as intended. Thus, it was possible to go into development for the front-end application knowing that the API worked.

For the purpose of testing and practicing the technologies, an API Exerciser was created, this application was a ASP.net MVC web application, which had fully CRUD capabilities (Create, Read, Update, Delete).

This application has the key components necessary for the main application, such as displaying data from the database using API endpoints, having the capability to edit certain rows of data, being able to

create a new set of data, delete a row of data as well as filter certain pages by an ID such as filtering the shopping list items table by each Shopping list. So, most of the Code within the application was reusable for the main application.

Chapter 6 Conclusion

6.1 Post-mortem

During the development for the application some issues arose which needed to be resolved so the project could continue.

One of main visions of the project was to make the application mobile friendly, hence why Xamarin was initially chosen. One of the key issues was that Xamarin was not working as expected. Previously when using Xamarin, all the work had been done within the university, in a Smeaton lab that has access to Xamarin and the necessary libraries. Due to COVID access to the lab was hindered. The assumption was that the computers that I had access to would be capable of coding in Xamarin like the lab, but there were continuous issues following through. On the main desktop where the project was originally created, due to having an AMD processor the computer was not able to run the emulator to run and test the front end. To try and resolve this issue a laptop with an intel CPU was used, this got passed the issue of being able to use the emulator, but another problem arose, where no matter what version of NuGet packages was used none would ever install correctly due to different versions, making development come to a standstill. As progression was halted, another solution had to be put in place, so instead of using Xamarin for the front end, a web based front end was considered. Although the application won't be able to be on the application store, the ability to access the application on any device that can access the web will increase the number of possible users able to access the application. To make sure the user is still getting the experience that the project desired the application will be scaled for mobile phones, so the UI is still user friendly to those using phones with smaller screens, although this may never be achieved within this project..

After having decided a web based front end was best, many practice projects were used to both go through the tutorials of ASP.Net web forms, MVC and asp.net API. Web forms was attempted, but this was traded out for an MVC Front end.

To say that smart home systems are a necessity is a far cry, the desire to make homes more automated and convenient to control by means of smart technology, is mostly driven by the developers and sellers of the technology, which often fills a hole that never needed to be filled. Take the problems Stokk is trying to solve, there are already systems in place which can solve the issues, a simple pen and paper can create a shopping list, so what is the need for the application? Galen Gruman when speaking about smart home technologies mentioned that "They're mainly solutions in search of a problem" (Gruman, 2014), which at the start of the project was true. The Stokk system had to go through a few iterations before reaching the conclusion that to create a system that helped out in the home it first needed to be convenient for the user to use, compared to the physical alternatives that have been used for hundreds of years.

This links with another issue that occurs when using smart home systems, that when systems didn't work as expected it "was usually due to the effort required to set up and control systems, rather than the lack of functionality." (Oliveira, et al., 2020). So, an issue with going forward with the system was finding solutions which benefited the user, by not adding additional steps, to achieve the core functionality thus why the QR design was dropped.

6.2Main Functionality Not Achieved

Recipe Formatting

The desire to format and display the recipes was left out due to time limitations: The system would require the user, when writing the recipe, to have them denote each part of the instructions with a number, so that the system could go through a loop which displays the recipe in a more bullet point like fashion for the users, making it easier to read which is a key priority of a Smart home system.

Recipe Handling

Although recipe handling was one of the core features of the applications conception, being able to filter what meals a household could make with their current ingredients became a struggle to do, as time being spent on failed algorithms was taking its toll in other areas such as the report and other features. So, although this current application does not filter and display which meals a specific household can make, the page just displays a list of the meals. Eventually this would also include a Button which meant that the user had cooked the meal and that the system should remove the items that were used in the recipe.

Bought functionality

For the functionality of having a user say they bought an item, a button which says “bought” would reduce the value of QTY on that row to 0 subsequently setting off the Trigger in the database that adds that row from the shopping list items table, over to the household items table. Although the trigger was fully created and the functionality can be tested by editing the QTY value to 0, the button to do this functionality was never finished.

Friendly names

Desired to have friendly names displayed for each item instead of ID’s which is what is displayed at the moment, this is fine as the designer of the system, as testing doesn’t require the friendly names, but for the user application it would eventually reference to something by ID but display the friendly name.

6.3 Final Conclusion

Stokk is an aspiring smart home system that was created to help the user with shopping, stock keeping and recipe handling, with the use of automation to help create an integrated smart system for the kitchen and home, which can help be a stepping stone towards achieving a fully integrated “smart home environment” (Kazmierzak, 2012). Although not fully implemented the architecture for the database and API have been fully implemented with only minor tweaks to get the system to a higher standard for production. Stokk is not currently a smart home system but with a few adjustments it is well on the way to creating a smart home environment within the kitchen and when shopping, so the user’s tasks can be streamlined.

Bibliography

Crawford, J., Hunt, E. & Peak, G., 1966. *Inverse forgetting in short-term memory,* s.l.: s.n.

Gruman, G., 2014. Home automation is a solution in search of a problem. *Infoworld.*

Heinrichs, F., Schreiber, D. & Schöning, J., 2011. *The hybrid shopping list: Bridging the gap between physical and digital shopping lists,* s.l.: s.n.

Kazmierzak, F., 2012. *Smart Home Environment- Concepts and solutions,* s.l.: s.n.

Lasquety-Reyes, J., 2020. *Number of Smart Homes forecast worldwide until 2025,* s.l.: Statista.

Lasquety-Reyes, J., 2020. *Smart Home revenue forecast worldwide until 2025,* s.l.: Statista.

Oliveira, L., Mitchell, V. & May, A., 2020. *Smart home technology- comparing householder expectations at the point if installation with experiences 1 year later.* s.l.:s.n.

Qiao, S., Zhu, H., Zheng, L. & Ding, J., 2017. Intelligent Refrigerator Based on Internet of Things. In: s.l.:s.n.

Shweta, A., 2017. *Intelligent refrigerator using ARTIFICAL INTELLIGENCE,* s.l.: s.n.

Appendices

Appendix A SQL Statements with comments

```
USE COMP3000_SSharpe

GO

//all ID's are sequence

//Create user table where UID is primary key, friendly name is the users first name they want to be referred by, password and email


CREATE TABLE [User]

(UID int IDENTITY(1,1) NOT NULL,

CONSTRAINT AK_UID UNIQUE(UID),

CONSTRAINT PK_UID PRIMARY KEY NONCLUSTERED (UID),

FriendlyName varchar (50) NOT NULL,

[Password] varchar (50) NOT NULL,

Email varchar(50) NOT NULL

);

GO



//create household table PK HousHold Id , HHName is the house name, admin user is the primary user of a houshold and has full control of houshold, the users are foreign keys from the user table


CREATE TABLE HouseHold

(HHID int IDENTITY(1,1) NOT NULL,

CONSTRAINT AK_HHID UNIQUE(HHID),

CONSTRAINT PK_HHID PRIMARY KEY NONCLUSTERED (HHID),

HHName varchar (50)NOT NULL,

AdminUser int NOT NULL,

CONSTRAINT FK_AdminUser FOREIGN KEY (AdminUser)

    REFERENCES [User] (UID)
```

```
ON DELETE NO ACTION
    ON UPDATE NO ACTION,
RestrictedUsers int,
CONSTRAINT FK_RestrictedUsers FOREIGN KEY (RestrictedUsers)
    REFERENCES [User] (UID)
            ON DELETE NO ACTION
    ON UPDATE NO ACTION
 );
GO
```

//Create Room table Room ID is PK RoomName is the name of the room , and The House Hold ID as a FK from HouseHold

```
CREATE TABLE Room
(RID int IDENTITY(1,1) NOT NULL,
CONSTRAINT AK_RID UNIQUE(RID),
CONSTRAINT PK_RID PRIMARY KEY NONCLUSTERED (RID),
RoomName varchar (50)NOT NULL,
HHID int CONSTRAINT FK_HHID FOREIGN KEY (HHID)
    REFERENCES HouseHold(HHID)
ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
GO
```

//StorageSpace table where storage space id is PK, storage name is its name and Room ID as FK

```
CREATE TABLE StorageSpace
(SSID int IDENTITY(1,1) NOT NULL,
CONSTRAINT AK_SSID UNIQUE(SSID),
CONSTRAINT PK_SSID PRIMARY KEY NONCLUSTERED (SSID),
StorageName varchar (50)NOT NULL,
RID int CONSTRAINT FK_RID FOREIGN KEY (RID)
    REFERENCES Room (RID)
ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
GO
```

//Item Type table that has item type ID as pk and Item Type name this is the generic item like beans, sausages

```
CREATE TABLE ItemType
(ITID int IDENTITY(1,1) NOT NULL,
CONSTRAINT AK_IYID UNIQUE(ITID),
CONSTRAINT PK_ITID PRIMARY KEY NONCLUSTERED (ITID),
ITName varchar (50) NOT NULL
);
GO
```

//item table that has Item ID as the primary key, itemname which is the specific name or brand of an item like heinz beans, with Item Type ID as a foreign key

```
CREATE TABLE Item
(IID int IDENTITY(1,1) NOT NULL,
CONSTRAINT AK_IID UNIQUE(IID),
CONSTRAINT PK_IID PRIMARY KEY NONCLUSTERED (IID),
ItemName varchar (50) NOT NULL,
ITID int CONSTRAINT FK_ITID FOREIGN KEY (ITID)
    REFERENCES ItemType (ITID)
ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
GO
```

//HouseHoldItem table that has HouseHold Item ID as PK, and has HouseHold ID Item ID and storage space as Foreign Keys

```
CREATE TABLE HouseHoldItem
(HHIID int IDENTITY(1,1) NOT NULL,
CONSTRAINT AK_HHIID UNIQUE(HHIID),
CONSTRAINT PK_HHIID PRIMARY KEY NONCLUSTERED (HHIID),
QTY int NOT NULL,
SSID int CONSTRAINT FK_SSID FOREIGN KEY (SSID)
    REFERENCES StorageSpace (SSID)
ON DELETE NO ACTION
    ON UPDATE NO ACTION,
            IID int CONSTRAINT FK_IID FOREIGN KEY (IID)
    REFERENCES Item (IID)
ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION,
            HHID int CONSTRAINT FK_HHoldID FOREIGN KEY (HHID)
    REFERENCES HouseHold(HHID)
ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
GO
```

//create shopping list table, has shopping list id as primary key and household id as a foreign key

```
CREATE TABLE ShoppingList
(SLID int IDENTITY(1,1) NOT NULL,
CONSTRAINT PK_SLID PRIMARY KEY NONCLUSTERED (SLID),

            HHID int CONSTRAINT FK_HouseholdID FOREIGN KEY (HHID)
    REFERENCES HouseHold (HHID)
ON DELETE NO ACTION
    ON UPDATE NO ACTION,

);
GO
```

//shopping list item shopping list item ID as Primary key, shopping list ID as foreign key Item ID, QTY int and Urgent as a bool

```
CREATE TABLE ShoppingListItems
(SLIID int IDENTITY(1,1) NOT NULL,
CONSTRAINT PK_SLIID PRIMARY KEY NONCLUSTERED (SLIID),
```

```
SLID int CONSTRAINT FK_SLID FOREIGN KEY (SLID)
REFERENCES ShoppingList (SLID)
ON DELETE NO ACTION
    ON UPDATE NO ACTION,
IID int NOT NULL
CONSTRAINT FK_ItemID FOREIGN KEY (IID)
    REFERENCES Item (IID)
            ON DELETE NO ACTION
    ON UPDATE NO ACTION,
            QTY int NOT NULL,
            Urgent bit NOT NULL
);
GO
```

//create recipe table with Recipe ID as primary key, has details and instructions for meal and UserID as foreign key

```
CREATE TABLE Recipe
(RecipeID int IDENTITY(1,1) NOT NULL,
CONSTRAINT PK_RecipeID PRIMARY KEY NONCLUSTERED (RecipeID),
Details varchar (200) NOT NULL,
Instructions varchar (MAX) NOT NULL,
UserID int CONSTRAINT FK_User FOREIGN KEY (UserID)
    REFERENCES [User] (UID)
ON DELETE NO ACTION
    ON UPDATE NO ACTION



);
```

```
GO


//RecipeItem for ingredients of a recipe, Recipe Item ID as pk, Item  as Item Id and Recipe ID as FK with QTY and Wight attributes
CREATE TABLE RecipeItem
(RecipeIID int IDENTITY(1,1) NOT NULL,
CONSTRAINT PK_RecipeIID PRIMARY KEY NONCLUSTERED (RecipeIID),
Item int CONSTRAINT FK_RecipeItemID FOREIGN KEY (Item)
REFERENCES Item (IID)
ON DELETE NO ACTION
    ON UPDATE NO ACTION,
            RecipeID int CONSTRAINT FK_RecipeID FOREIGN KEY (RecipeID)
REFERENCES Recipe (RecipeID)
ON DELETE NO ACTION
    ON UPDATE NO ACTION,
QTY int NOT NULL,
Weight int
);
GO


//desire stock to dictate how much a user wants of each item and is the qty we add to the shopping list


CREATE TABLE DesiredStock
(IID int
CONSTRAINT FK_DSIID FOREIGN KEY (IID)
    REFERENCES Item (IID)
ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION,
            HHID int
CONSTRAINT FK_DSHHID FOREIGN KEY (HHID)
    REFERENCES HouseHold (HHID)
ON DELETE NO ACTION
    ON UPDATE NO ACTION,
            CONSTRAINT PK_HHIDIID PRIMARY KEY (IID,HHID),
            QTY int NOT NULL
)
```

//trigger tjat after a houseHold is inserted Insert a New Shopping List where HHID== the one just created

```
CREATE TRIGGER CreateHouseHoldShoppingList ON HouseHold
AFTER INSERT
AS
BEGIN
INSERT INTO ShoppingList(HHID)
SELECT i.HHID
FROM
inserted i;
END
GO
```

```
USE [COMP3000_SSharpe]
GO

//create / altered trigger for adding item to shopping list once low or 0


ALTER TRIGGER [dbo].[addingToShoppingList] ON [dbo].[HouseHoldItem]
For UPDATE
AS

//declare variables to hold values from differnt tables, to hold the household id
DECLARE @HouseHoldID int
SELECT  @HouseHoldID = HHID
FROM inserted

//item id from item

DECLARE @ItemID int
SELECT @ItemID = IID
FROM inserted
```

```
//QTY From theHouseholdItem

DECLARE @ActualQTY int
SELECT @ActualQTY = QTY
FROM HouseHoldItem
WHERE HHID = @HouseHoldID
AND IID = @ItemID


//Shopping list ID from Shopping List

DECLARE @ShoppingListID int
SELECT @ShoppingListID = SLID
FROM ShoppingList
WHERE HHID = @HouseHoldID

//Desired QTY from desired stock

DECLARE @DesiredQTY int
SELECT @DesiredQTY = QTY
FROM dbo.DesiredStock
WHERE HHID = @HouseHoldID
AND IID = @ItemID

//Declare Urgency if the actual QTY is more than 1 return the function trigger, if QTY =0 Delete from
household item where QTY =0 and add to Shopping list with urgency is true, else urgency is false

DECLARE @Urgency bit
if(@ActualQTY> 1 )
```

```
RETURN;
if(@ActualQTY=0)
BEGIN
DELETE FROM HouseHoldItem
WHERE QTY = 0;
SET @Urgency = 1;
END;
ELSE
SET @Urgency = 0;
```

//select attributes from shopping list items where the created variables = the accompanying attribute in Shopping List Items, update the the database, set urgency and insert into Shopping List Items

```
BEGIN
IF EXISTS (SELECT * FROM ShoppingListItems WHERE @ShoppingListID = SLID AND @ItemID = IID)
UPDATE dbo.ShoppingListItems
SET dbo.ShoppingListItems.Urgent = @Urgency
ELSE
INSERT INTO ShoppingListItems VALUES (@ShoppingListID,@ItemID, @DesiredQTY, @Urgency)
END;
```

//trigger on shopping list item, a button should make the value of qty = 0, when this happens item is removed form the shopping list and added to the householod item table

```
CREATE TRIGGER AddFromShoppingListToHHI ON ShoppingListItems
AFTER UPDATE
AS
DECLARE @TransferrableQTY int
SELECT @TransferrableQTY = QTY
FROM deleted

DECLARE @ShoppingListItemID int
SELECT @ShoppingListItemID = SLIID
FROM deleted

DECLARE @CheckingQTY int
SELECT @CheckingQTY = QTY
FROM inserted

DECLARE @ShoppingListID int
SELECT @ShoppingListID = SLID
FROM deleted

DECLARE @HouseHold int
SELECT @HouseHold = HHID
FROM ShoppingList
WHERE SLID = @ShoppingListID
```

```
DECLARE @BoughtItem int
SELECT @BoughtItem = IID
FROM deleted

DECLARE @CurrentStock int
SELECT @CurrentStock = QTY
FROM HouseHoldItem
WHERE IID = @BoughtItem AND HHID = @HouseHold


DECLARE @NewStockAmount int
SET @NewStockAmount = @CurrentStock + @TransferrableQTY


IF(@CheckingQTY = 0)
BEGIN
DELETE FROM dbo.ShoppingListItems WHERE SLIID = @ShoppingListItemID

IF EXISTS (SELECT * FROM HouseHoldItem WHERE IID = @BoughtItem AND HHID = @HouseHold)
UPDATE dbo.HouseHoldItem
SET QTY = @NewStockAmount
WHERE IID = @BoughtItem AND HHID = @HouseHold
ELSE
INSERT INTO HouseHoldItem VALUES (@TransferrableQTY, null,  @BoughtItem, @HouseHold)
END;
```

//trigger for adding desired stock values on insertiung into house hold item

```
USE COMP3000_SSharpe
GO

CREATE TRIGGER AddToDesiredStock ON HouseHoldItem
AFTER INSERT
AS
DECLARE @InsertedHH int
SELECT @InsertedHH = HHID
FROM inserted

DECLARE @InsertedItem int
SELECT @InsertedItem = IID
FROM inserted

Declare @InsertedQTY int
SELECT @InsertedQTY = QTY
FROM inserted

IF EXISTS (SELECT * FROM HouseHoldItem WHERE IID = @InsertedItem AND HHID = @InsertedHH)
RETURN;
```

```
ELSE

INSERT INTO DesiredStock VALUES (@InsertedItem, @InsertedHH, @InsertedQTY)
```

Appendix B Questionnaire Questions

Questionnaire 1: concept assistance.

Do you use an application to help with cooking?

Do you use an application to help with shopping lists?

Do you use an application for stock control?

Would you use an application that helped with cooking?

Would you use an application that helped with shopping?

Would you use an application that helped with stock control?

Would you use an application that integrated all of the previously mentioned functionality?

Questionnaire 2: Usability testing of Mock Design

Users shown design prototype demo.

What did you think of the design?

What would you change?

Would you use this application?

And why would you use this application?

Appendix C – Ethical Approval Confirmation

UNIVERSITY OF PLYMOUTH

24 April 2020

CONFIDENTIAL

Shirley Atkinson
School of Engineering, Computing and Mathematics

Dear Shirley

Ethical Approval Application

Thank you for submitting the revised ethical approval form and details concerning your project:

Requirements Gathering and Usability Testing of Software for Computing Undergraduate Project modules - Revision to cover remote usability testing in response to COVID-19.

I am pleased to inform you that this has been approved via Chair's Action

Kind regards

Rebecca Waghorne
Secretary to Faculty Research Ethics Committee

Mrs Jayne Brenen, Head of Faculty Operations, Faculty of Science and Engineering, University of Plymouth, Drake Circus, Plymouth, Devon PL4 8AA **T** +44 (0)1752 584584 **F** +44 (0)1752 584540 **W** www.plymouth.ac.uk