



# Macro Manager Iteration 1

## Contents

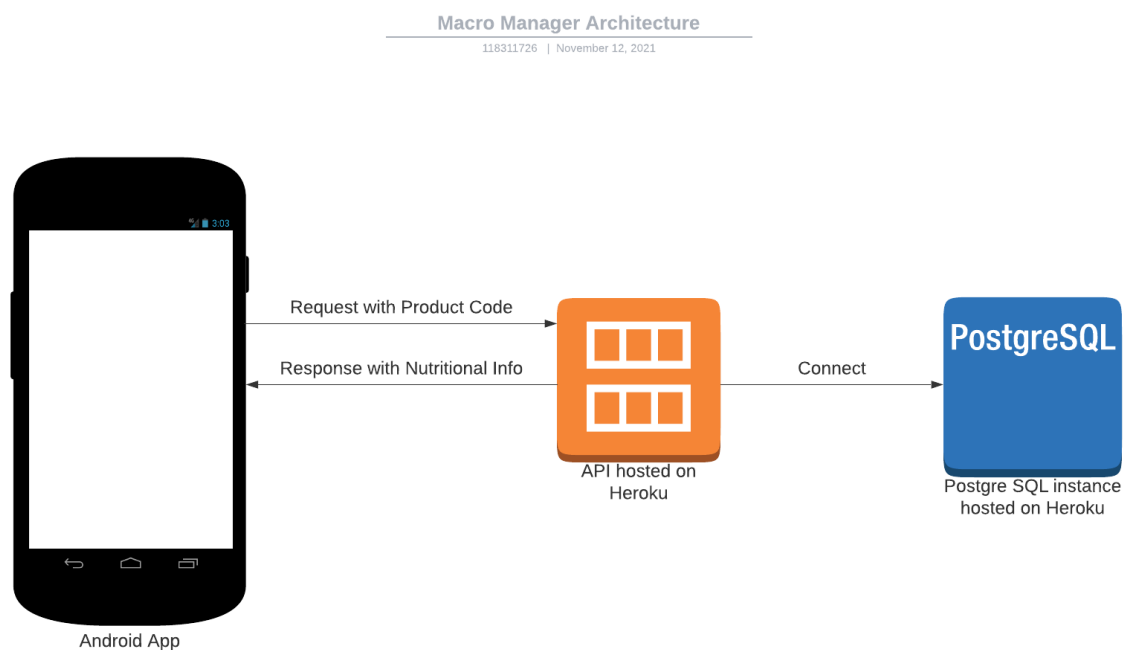
Executive Summary .....	2
Functional Model .....	3
Project Backlog .....	5
Sprint Backlog .....	7
Development .....	8
Frontend .....	8
API .....	9
Database .....	12
Kanban .....	17
Project Schedule.....	17
Issues, Risks & Learnings.....	18
Conclusion & Next Steps.....	20
Licenses.....	20

## Executive Summary

In the following document the progress for iteration one of the development of Macro Manager will be outlined. Functional Models of the current functionality of the project will be provided, a project backlog and sprint backlog will be broken down, steps taken in the development process will be outlined, a kanban board with current and upcoming tasks will be provided alongside an updated project schedule and, the issues, risks and learnings encountered during development will be discussed. Finally the next steps for iteration two will be laid out. The progress for this iteration of development went planned, however there were several issues which arose that slowed development down significantly. This iteration could have potentially been much more fruitful. The current status of the project is that a user may scan barcodes and receive nutritional information about the product. The Android app, API and Database are all connected (Android app <-----> API <-----> Database) with the Android app already making use of the API. The API connects to the Database but no transactions have currently been implemented.

## Functional Model

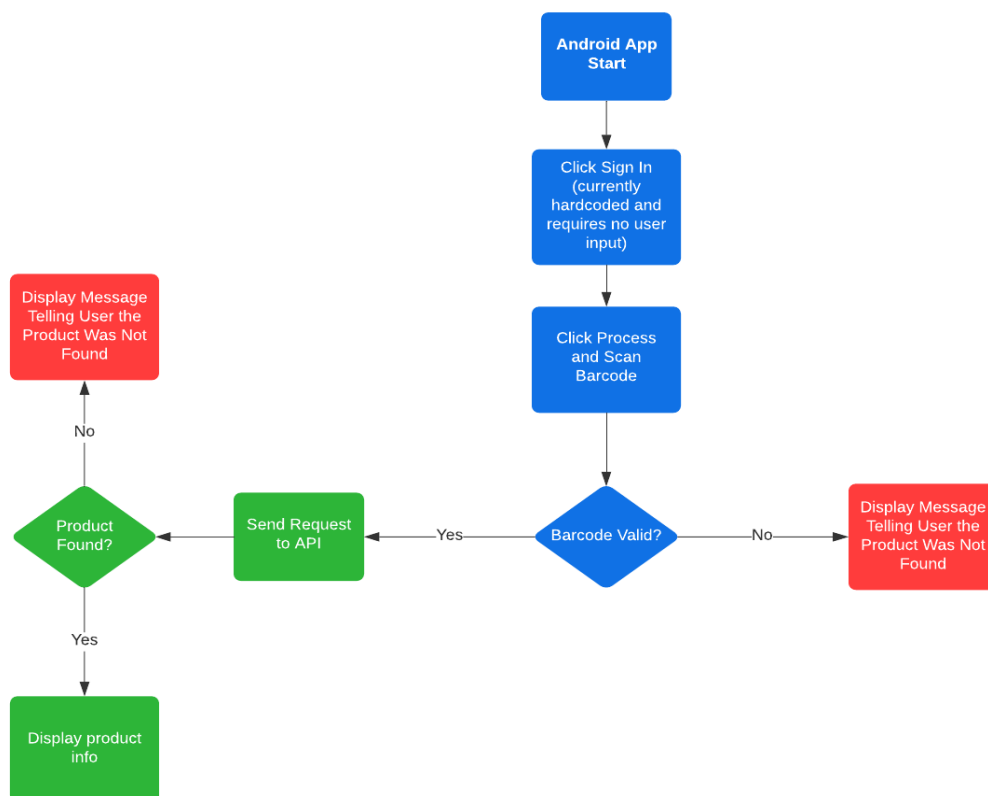
The following diagrams detail the current architecture of the MacroManager ecosystem. At a very high level, a user uses their phone to scan a barcode. This barcode is then sent as part of a request to the API which sends back the nutritional information of the scanned product. The API also connects to the PostgreSQL database but currently undertakes no read or write operations.



A more in depth explanation of the current state of the Android app is shown in the below diagram. On start the user is shown a sign in screen. Currently this is hardcoded so that if the “Sign In” button is pressed the app progresses to the next strage regardless of what, if anything, the user has type into the email and password text boxes. This is because user functionality is not a key feature of the app and will be developed at the end. User functionality such as login also slows down testing if done straight away so it is better left until closer to the end of development. When the user progresses they are shown a screen which contains a “process” button. If they click this it will access their devices camera and look for a barcode. When it finds and scans the barcode it will send a request to the MacroManager API and display the response on screen in unformatted json.

## MacroManagerAndroidApp

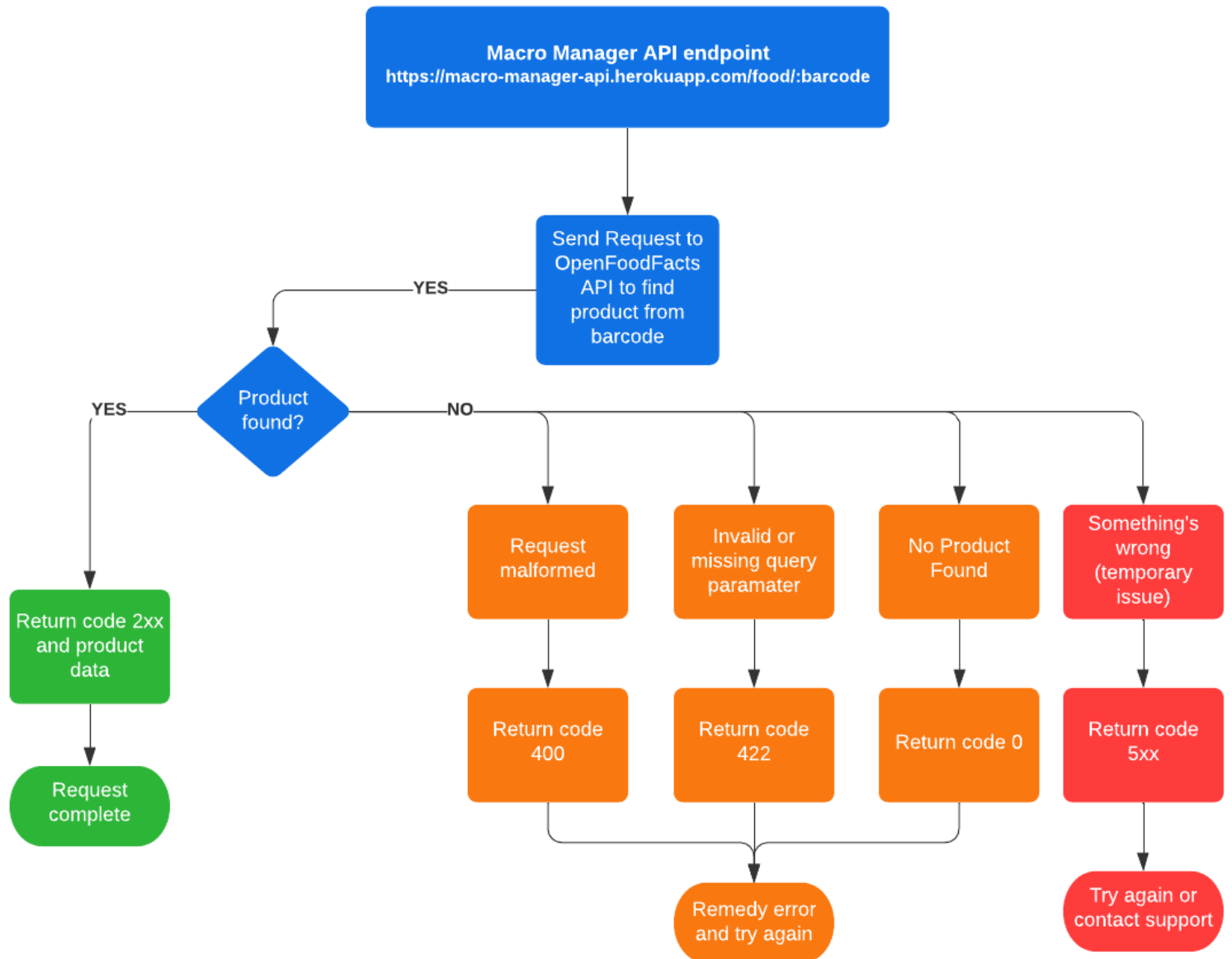
118311726 | November 12, 2021



Finally, the MacroManager API is laid out as seen below. The API listens for requests at the endpoint <https://macro-manager-api-herokuapp.com/food/:barcode>. If it receives a request it takes the barcode parameter from the request and places it into a new request to the OpenFoodFacts API. If the product is found the information is sent to the system which requested it and continues to listen for requests. If the product is not found it will respond with an error code and explain that the product was not found.

## MacroManagerAPI

118311726 | November 12, 2021

























## Project Backlog

In the high-level design document for this project the following project backlog was devised. Currently this is the same backlog as is being followed. Anything with a priority of 10 is not a requirement but rather a feature which would be nice to have, should time allow for their development.

User Story	Estimated Size	Priority
As someone who tracks calories I want to be able to scan barcodes on food and receive calorie and macro nutrient contents so that I can track my calories quickly and easily.	Small	1
As someone who tracks calories I want to be able to scan menus in restaurants and receive the calorie content so that I can make better food choices while eating out.	Large	1
As a diabetic I want to be able to scan menus in restaurants and receive macro nutrient content so that I can make better food choices while eating out.	Large	1
As someone who tracks calories I want to be able to create custom recipes with these scanned foods so that I can track my calories quickly and easily.	Medium	3
As someone who tracks calories I want to be able to filter a list of local restaurants based on calorie content so that I can make better decisions about where to eat.	Large	5
As a coeliac I want to be able to filter a list of local restaurants based on ingredients used so that I can go out to eat.	Large	5
As someone who tracks calories I want to be able to register and login so that I can save the foods that I scan and calories from these foods.	Small	7
As someone who owns a Fitbit/uses the Fitbit app I want to integrate my Macro Manager diary with the Fitbit app so I can track everything in one place.	Medium	10
As someone who likes to create recipes I want to be able to share my recipes with other users so they can enjoy them as well.	Medium	10

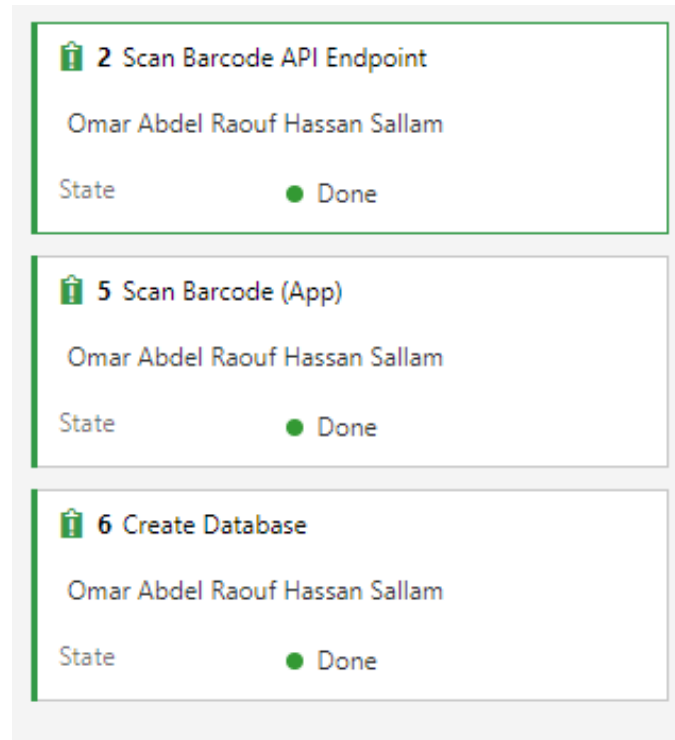
The existing backlog at the end of this sprint can be found below, each feature has been broken down into several tasks. This may need to be broken down even further in the next iteration as tasks are larger than initially anticipated.

 	Order	ID	Title	Assigned To	State
	1	20	<div>   Calorie Diary         </div>		● To Do
		12	<div>  Write to Database from API         </div>		● Doing
	2	14	<div>   Calorie Estimator         </div>		● To Do
		9	<div>  Estimate allergens API Endpoint         </div>		● To Do
		8	<div>  Estimate calories API Endpoint         </div>		● To Do
		7	<div>  Scan menu (App)         </div>		● To Do
	3	16	<div>   View, Search and Filter Local Restaurants         </div>		● To Do
		11	<div>  List Local Restaurants API Endpoint         </div>		● To Do
		10	<div>  Display local restaurants (App)         </div>		● To Do
	4	15	<div>   Create Custom Recipes         </div>		● To Do
		13	<div>  Recipe Creator API Endpoint         </div>		● To Do
	5	17	<div>   User Functionality         </div>	...	● To Do
		3	<div>  Register and Login         </div>		● To Do
	6	18	<div>  Share Recipes         </div>		● To Do

## Sprint Backlog

The backlog for this sprint has been cleared with three tasks being completed:

- Creation of the Database
- Development of an API endpoint to get the nutritional information of a product based on its barcode
- Development of a frontend which allows users to scan barcodes and receive the the nutritional information from above



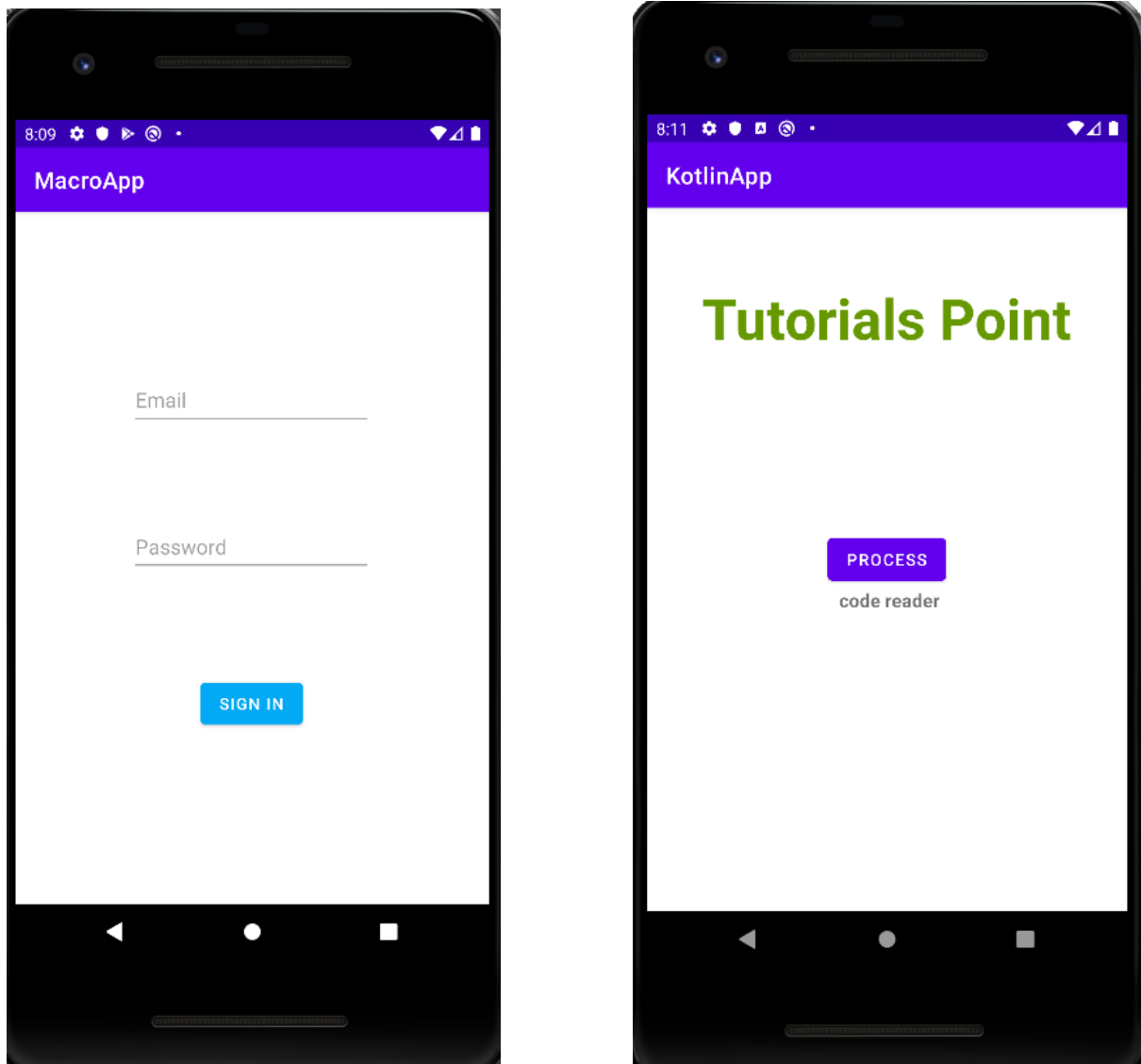
## Development

### Frontend

The Android app frontend was the biggest challenge in this sprint. Its current state contains a login page which has been hardcoded to take the user directly to the screen where they can scan barcodes and receive nutritional information when they press “Sign In” regardless of what is typed into the form (if anything). It also contains a second page consisting of a single button and some text. When the button which says “process” is pressed the devices camera activates and starts looking for a barcode. When the barcode is identified and scanned it sends a request to the API with products code and if the request is successful it displays the nutritional information of the product on the screen. The app is currently very ugly and unappealing, however it is not yet intended for use and was merely intended to be connected to the API for this sprint. The second page has also been taken from this tutorial:

<https://www.tutorialspoint.com/barcode-scanning-in-android-using-kotlin>.





## API

The API has been set up to run on any server capable of running a Go program. This has been done by setting environment variables and storing the access port for the project in one of these. By doing so, the port can remain dynamic and change from system to system. An endpoint has been set up to receive food product codes in requests from external sources. These requests are then handled and a request is sent to the OpenFoodFacts API to retrieve the nutritional information. Models have been implemented to transform these products from the format they are received from the OpenFoodFacts API in to the format which was decided upon in the Macro Manager ERD's.

Some of the below code comes from this tutorial on building web applications in Go using Gin:

<https://semaphoreci.com/community/tutorials/building-go-web-applications-and-microservices-using-gin>.

```
//taken and adapted from this site https://semaphoreci.com/community/tutorials/building-go-web-applications-and-microservices-using-gin
var router *gin.Engine

func main() {
    port := os.Getenv("PORT")
    db, err := sql.Open("postgres", os.Getenv("DATABASE_URL"))
    if err != nil {
        log.Fatal(err)
    }

    err = db.Ping()

    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Println("connection is good")
    }

    godotenv.Load()

    router = gin.Default()

    initialiseRoutes()

    router.Run(":" + port)
}
```

The below code takes in the barcode from the parameters of the request sent to the MacroManager API and places it into the url of a request to the OpenFoodFacts API. The response is then unmarshalled into the food model which is described below. Once the food object has been populated it is sent as a response to the device which requested it initially. Some of the code has been adapted from <https://blog.logrocket.com/making-http-requests-in-go/>.

```

//takes in a barcode and sends a http request to the OpenFoodData API https://world.openfoodfacts.org/api/v0/product/
//adapted from https://blog.logrocket.com/making-http-requests-in-go/
func ScanFood(upc string) food.Food {

    resp, err := http.Get("https://world.openfoodfacts.org/api/v0/product/" + upc)
    if err != nil {
        log.Fatalln(err)
    }
    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        log.Fatalln(err)
    }

    sb := string(body)
    var foodProduct food.Product
    json.Unmarshal([]byte(sb), &foodProduct)
    return foodProduct.Food
}

func GetFoodProduct(c *gin.Context) {
    upc := c.Param("upc")
    foodProduct := ScanFood(upc)

    c.IndentedJSON(http.StatusOK, foodProduct)
}

```

The below code outlines structs (which are the equivalent of classes in Go) which unmarshal the json response received from the OpenFoodFacts API into the model which has been devised for MacroManager.

```

//all the structs needed to take in and manipulate data from the spoonacular api and the database
type Product struct {
    Food Food `json:"product"`
}

//custom structure for db table
//handles unmarshalling of json from OpenFoodData API
type Food struct {
    FoodID      int
    PantryID    int
    Title       string `json:"product_name"`
    Nutriments struct {
        Calories int `json:"energy-kcal_serving"`
        Fat       int `json:"fat_serving"`
        Carbohydrate int `json:"carbohydrates_serving"`
        Protein   int `json:"proteins_serving"`
    } `json:"nutriments"`
    ServingSize string `json:"serving_size"`
    Misc        []string `json:"allergens_tags"`
}

```

## Database

The database has been set up and hosted on Heroku. Initially it was setup on the developer's local machine to ensure correct function.

Install PostgreSQL, ensure the ODBC driver is installed with it in the stack builder section of the install. Add the Postgres bin and lib folders to the PATH environment variable.

Run `psql -U postgres` and input the password you set up during installation.

Commands to create database and tables.

```
CREATE DATABASE macromanager CREATE DATABASE macroManager WITH  
ENCODING 'UTF8' LC_COLLATE='English_Ireland' LC_CTYPE='English_Ireland';
```

Using the *CREATE DATABASE* command is only necessary on the local machine. If hosting on Heroku this is not necessary.

To create database in Heroku go to a Heroku app and navigate to the “Resources” tab. Here find the “Quickly add add-ons from Elements” search bar and search “Postgres” select “Heroku Postgres” and follow the instructions.

Connect to this database through the command `heroku psql -a [heroku-app-name]`.

Once connected run the following commands.

```
CREATE TABLE "Pantry" (  
  "UserID" int,  
  "PantryID" int,  
  PRIMARY KEY ("PantryID")  
);
```

```
CREATE TABLE "Food" (  
  "PantryID" int,  
  "FoodID" int,  
  "Title" varchar,  
  "Calories" int,  
  "Fat" int,  
  "Carbohydrate" int,  
  "Protein" int,
```

```
"Serving Size" int,  
"Misc" json,  
PRIMARY KEY ("FoodID"),  
CONSTRAINT "FK_Food.PantryID"  
FOREIGN KEY ("PantryID")  
REFERENCES "Pantry"("PantryID")  
);
```

```
CREATE TABLE "User" (  
"UserID" int,  
"FName" varchar,  
"LName" varchar,  
"Email" varchar,  
PRIMARY KEY ("UserID")  
);
```

```
CREATE TABLE "Diary" (  
"UserID" int,  
"DiaryID" int,  
PRIMARY KEY ("DiaryID"),  
CONSTRAINT "FK_Diary.UserID"  
FOREIGN KEY ("UserID")  
REFERENCES "User"("UserID")  
);
```

```
CREATE TABLE "DiaryEntry" (  
"DiaryEntryID" int,  
"DiaryID" int,  
"Title" varchar,  
"Calories" int,  
"Fat" int,  
"Carbohydrate" int,  
"Protein" int,  
"Servings" decimal,
```

```

    "Misc" json,
    PRIMARY KEY ("DiaryEntryID"),
    CONSTRAINT "FK_DiaryEntry.DiaryID"
    FOREIGN KEY ("DiaryID")
    REFERENCES "Diary"("DiaryID")
);

CREATE TABLE "DiaryEntryFood" (
    "DiaryEntryFoodID" int,
    "DiaryEntryID" int,
    "FoodID" int,
    PRIMARY KEY ("DiaryEntryFoodID"),
    CONSTRAINT "FK_DiaryEntryFood.FoodID"
    FOREIGN KEY ("FoodID")
    REFERENCES "Food"("FoodID"),
    CONSTRAINT "FK_DiaryEntryFood.DiaryEntryID"
    FOREIGN KEY ("DiaryEntryID")
    REFERENCES "DiaryEntry"("DiaryEntryID")
);

CREATE TABLE "Recipe" (
    "UserID" int,
    "PantryID" int,
    "RecipeID" int,
    "Title" varchar,
    "Calories" int,
    "Fat" int,
    "Carbohydrate" int,
    "Protein" int,
    "Serving Size" int,
    "Misc" json,
    PRIMARY KEY ("RecipeID"),
    CONSTRAINT "FK_Recipe.PantryID"
    FOREIGN KEY ("PantryID")

```

```

REFERENCES "Pantry"("PantryID")
);

CREATE TABLE "DiaryEntryRecipe" (
  "DiaryEntryRecipeID" int,
  "DiaryEntryID" int,
  "RecipeID" int,
  PRIMARY KEY ("DiaryEntryRecipeID"),
  CONSTRAINT "FK_DiaryEntryRecipe.RecipeID"
  FOREIGN KEY ("RecipeID")
  REFERENCES "Recipe"("RecipeID"),
  CONSTRAINT "FK_DiaryEntryRecipe.DiaryEntryID"
  FOREIGN KEY ("DiaryEntryID")
  REFERENCES "DiaryEntry"("DiaryEntryID")
);

```

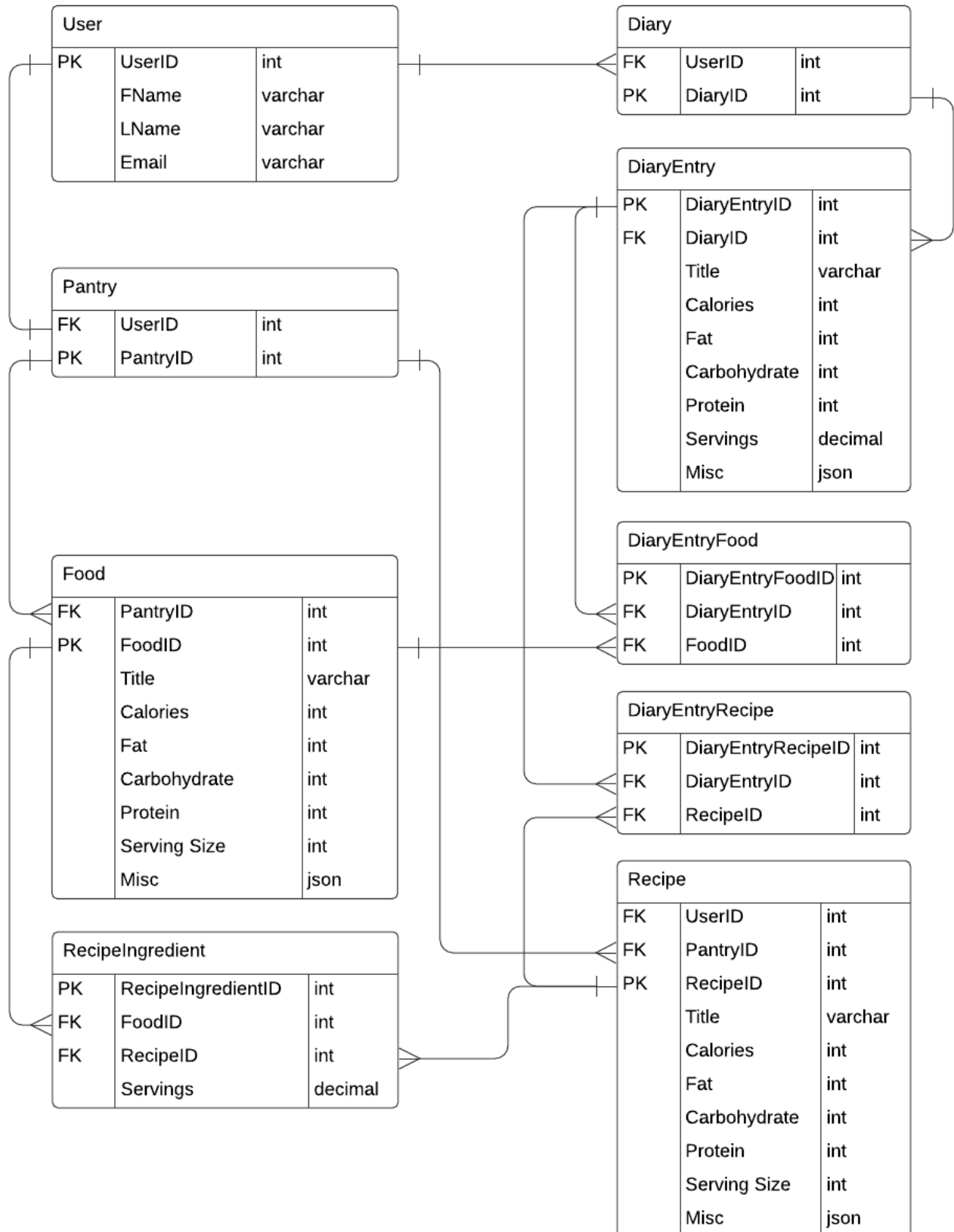
```

CREATE TABLE "RecipeIngredient" (
  "RecipeIngredientID" int,
  "FoodID" int,
  "RecipeID" int,
  "Servings" decimal,
  PRIMARY KEY ("RecipeIngredientID"),
  CONSTRAINT "FK_RecipeIngredient.RecipeID"
  FOREIGN KEY ("RecipeID")
  REFERENCES "Recipe"("RecipeID"),
  CONSTRAINT "FK_RecipeIngredient.FoodID"
  FOREIGN KEY ("FoodID")
  REFERENCES "Food"("FoodID")
);

```

These commands correspond to the physical ERD submitted with the Environment document. There has been a few alterations to datatypes as the data types used in the previous iteration were not compatible with PostgreSQL. Initially the database was going to be developed using MySQL.

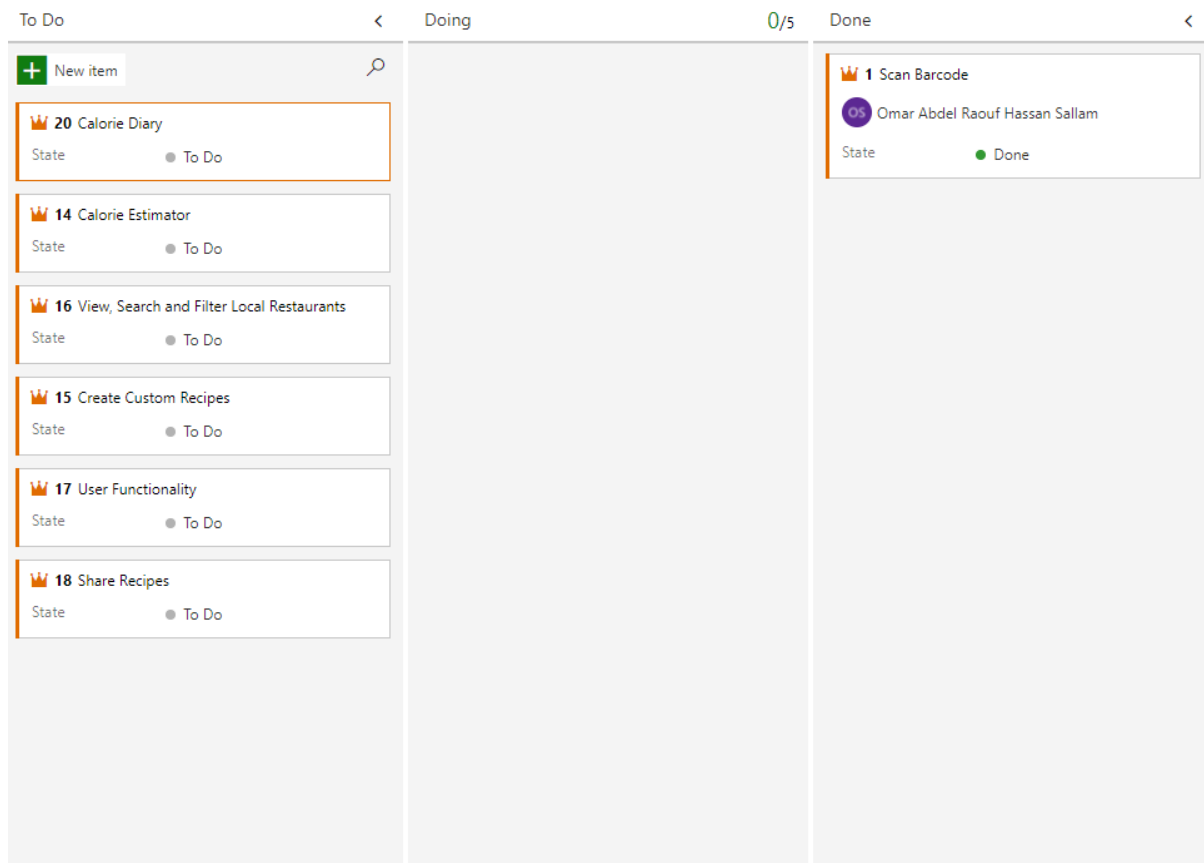
## Macro Manager Physical ERD





## Kanban

Using Azure DevOps the team is keeping track of workflow and the backlog of tasks. Below is the Kanban board at the end of Sprint 1.



## Project Schedule

The project schedule which was devised as part of the project's proposal is still being followed. This may be revised at a later date but the team hopes that development will be kept on track to meet the expectations laid out in this schedule. Below is this same project schedule, accompanied by a project burndown chart for sprint 1. Development for sprint 1 started quite late and so the burndown chart is not at the ideal trend. This will be rectified in the next sprint.

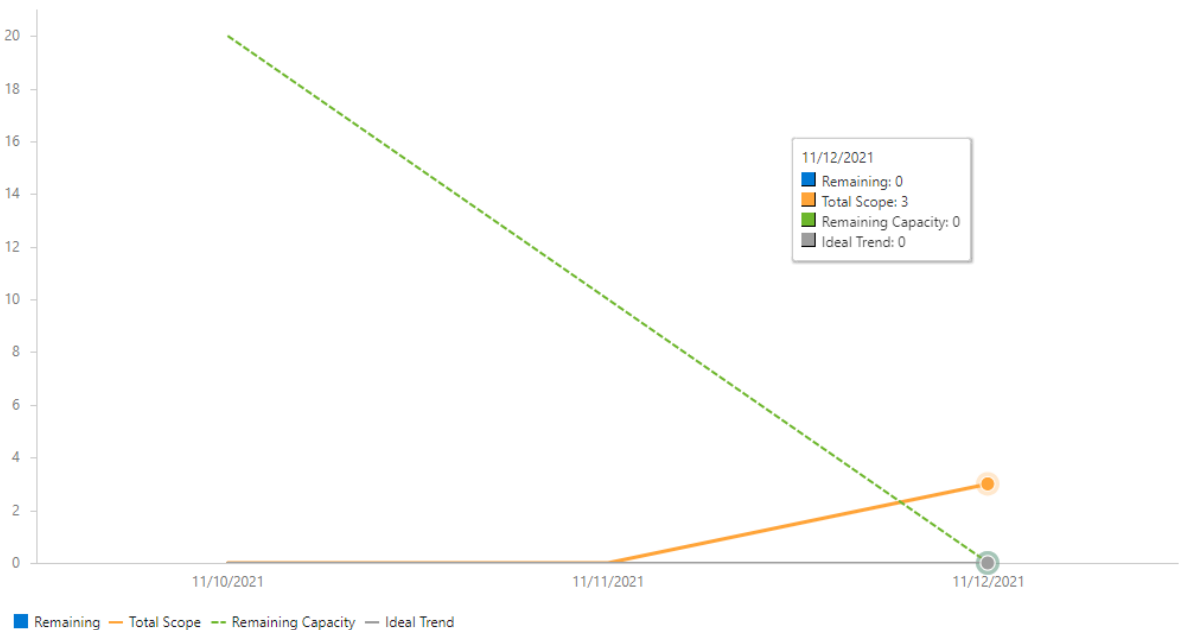
		01-Oct	15-Oct	29-Oct	12-Nov	26-Nov	28-Jan	11-Feb	25-Feb	11-Mar	11-Mar	15-Mar
Work Package	Tasks	Proposal	High-level Design	Environment	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Iteration 6	DEMO	Presentation
Work Package 1.0 Proposal												
	1.1 Executive Summary											
	1.2 Problem Statement											
	1.3 Proposed Solution											
	1.4 Business Model Canvas											
	1.5 Project Schedule											
Work Package 2.0 High-level Design												
	2.1 ERD											
	2.2 User Stories											
	2.3 Data Value Map											
Work Package 3.0 Environment												
	3.1 Database Platform											
	3.1 API Language + Libraries											
	3.2 Hybrid vs Native App											
Work Package 4.0 API												
	4.1 Calorie Diary											
	4.2 Calorie Estimator (Menu Items)											
	4.3 Restaurant Locator/Filter											
	4.4 Recipe Creator											
	4.5 User Functionality											
Work Package 5.0 Database												
	5.1 Models/Schema											
	5.2 Develop integrate and Test											
Work Package 6.0 Frontend												
	6.1 Wireframe/Design											
	6.2 Prototype											
	6.3 Integrate and Test											
Work Package 7.0 Demo												
	7.1 Film											
	7.2 Voiceover											
	7.3 Edit											
Work Package 8.0 Presentation												
	8.1 Slides											
	8.2 Present											

11/10/2021 - 11/12/2021

Completed 100%

Average 0  
burndown

Issues Remaining 0  
Total Scope Increase 3



## Issues, Risks & Learnings

There were several issues that have been encountered in this iteration. Conversely the team has learned a lot. Development started with the database, this was set up with little trouble bar a slight hump getting to grips with the CLI (command line interface) tools for PostgreSQL and Heroku. These are described in the “Development” section above.

One of the larges issues encountered was also the largest learning. Android Studio proved to have a much steeper learning curve than originally anticipated and as such development was delayed substantially. The learning of Android Studio was a good experience, however and the time spent at this stage will stand to the future development of the project in coming sprints. To address this issue, several tutorials were consumed, both on YouTube and various sites like TutorialsPoint. Code has also been appropriated from several sites, this code has been documented the “README” files of both code bases as well as in the comments of the code and the relevant sections of this document.

The next issue faced was an oversight with regards to the choice of food API to be used. Initially, the Go API developed for MacroManager was using the Spoonacular API. This seemed to be a good choice at first until the endpoints that were needed were finished and development of the frontend began. When the Android app had begun development and a barcode scanner was implemented the team realised there was a discrepancy between barcode formats which are available in the country of development (Ireland) and the ones which were capable of being searched by Spoonacular. Spoonacular only allows UPC formats to be searched, however in Europe the EAN formats are used. As such the food API being used needed to be changed. Thankfully, the OpenFoodFacts API contains products from all over the world and accepts all barcode formats, mnaking it perfect for MacroManager now and in the future if it is to expand and see global usage.

There are a few risks associated with the next iteration and beyond which need to be addressed:

- Lack of experience with Android Studio will make the next iteration difficult. Scanning text and deconstructing it into useful information will likely take substantial effort.

- Estimating calories in a menu item may take large amounts of compute power from a phone. Optimising this will be a difficult issue (achieving it will be a task unto itself)
- The food model in the MacroManager API will need to be changed to incorporate more tags in the Misc attribute, currently it only contains allergens but ideally it would contain such tags as “gluten-free”, “vegan-friendly” etc. Currently, the possibility of this is uncertain.

## Conclusion & Next Steps

In conclusion, the development of MacroManager this iteration has not been smooth but thankfully all deliverables have been completed. The team must take more care in the decisions they make in future iterations and do a better job of identifying issues.

For the next iteration the team will focus on enabling read and write functions between the API and database. They will also begin development of the Scan Menu features. Below is a backlog for the next sprint.

✓ Write to Database from API	Omar Abdel R... ● Doing
✓ Estimate allergens API Endpoint	Omar Abdel R... ● To Do
✓ Estimate calories API Endpoint	Omar Abdel R... ● To Do
✓ Scan menu (App)	Omar Abdel R... ● To Do

This backlog will likely stay very similar for the next several sprints as the tasks shown are rather large. The tasks may need to be broken down further to make progress more trackable.

## Licenses

Gin (Golang module imported for routing) – MIT License - <https://github.com/gin-gonic/gin/blob/master/LICENSE>

OpenFoodFacts (API and Database of Food Products worldwide) – Open Database License - <https://opendatacommons.org/licenses/odbl/1.0/>

Zxing (Kotlin/ Java library for barcode scanning) – MIT License – <https://github.com/zxing/zxing/blob/master/LICENSE>

Volley (Kotlin/ Java library for sending requests) – MIT License - <https://github.com/google/volley/blob/master/LICENSE>