

Groceries - Crazy Butterflies

Luke Doukakis

Donald Turner

Osman Halwani

Ameer Abdallah

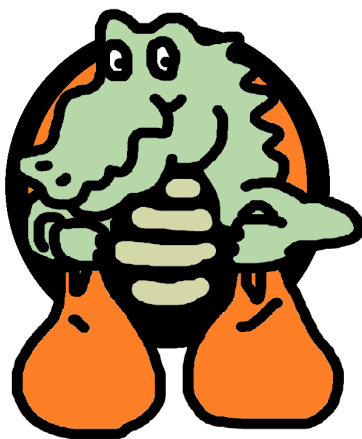


Table of Contents

- I. Introduction
- II. Approach
- III. Design of Solution
- IV. Implementation Issues
- V. Testing
- VI. Future Improvements
- VII. Conclusion
- VIII. Appendices
 - A. Software Requirements Specification
 - B. Database Relational Model

Introduction

Groceries is an easy to use online grocery ordering service which allows the user to create an order, place it, and have the items delivered from the comfort of their own couch not needing to move at all. Groceries allows nearby stores to add their wares to the website so that they can be better found by users on the site itself. Groceries main goal is simplicity with its very linear process in creating an order. A user simply needs to create an account, select a store(s) that they want to order from, select the items and then checkout. We want to be a very straightforward company with no tricks up our sleeves so what you see is what you get with our \$0 delivery fee on any purchase you ever make. The reason for this is that the main goal is to make the user happy and we can happily pay the drivers ourselves so everyone wins and we have slightly lower profits instead.

Other online grocery services are outdated and much too convoluted for their own good like Instacart with their poor user interface. The priority when selling a product is to make the customer happy and that is what we have done here with this simple user interface. Additionally, we will allow the stores themselves to easily log into our system and add their products so they have complete control over what users see on their store page with small amounts of customization available for them so they can distinguish their products according to their product line.

We have two main stakeholders in this project being: Professor Edwin Rodriguez and Crazy Butterflies LLC. Prof. Rodriguez, a professor at Cal Poly Pomona, contracted

us to create this project for him on February 26, 2021, and expected us to deliver the site to him by May 21, 2021. We have finished the site and met all of Prof. Rodriguez's criteria by May 10 with over a week to spare. As the majority owner, Prof. Rodriguez has full control over the website. Secondly, Crazy Butterflies LLC has invested its own resources into the project as well along with Prof. Rodriguez. Prof. Rodriguez owns 80% of the business with Crazy Butterflies LLC owning the other 20%. All profits made by the company will be split along the 80/20 margin accordingly. We are still open to more investors purchasing equity in the company in order to further develop the website with better features before we turn into a live site.

Approach

To create our product we had to use a multitude of different technologies to tie the entire website together to work in a functional manner. Starting off with the server host, we used Amazon Web Services(AWS) which offered the perfect platform to easily host our website on. AWS offers an easily scalable service so that we could start as a very small website and as we need more resources we could request more from them so our website was always supported without having to go through a huge process. Additionally, their site's stability and uptime are unparalleled in the industry giving us a huge amount of confidence in their ability to remain up whenever we need them. Also as students, we were able to receive a special plan in which we did not need to pay for this site. This is very important as a starting up company needs to be able to save as much money as they can in any situation possible and this saves us close to 80 dollars a month. We went with their EC2 instance which boasts some very large numbers when it comes to their hardware they give us access to. Firstly, we are given 32GiB of memory to work with and process data inside of 12 logical cores allowing for us to process data in a parallel manner. Secondly, they offer 10 Gbps Network bandwidth which allows users to talk to the website with low latency very quickly which is very important in today's online industry. Lastly, their systems are optimized to use as little memory as possible to leave as much free for other processes when they are required. Additionally, AWS instances are very easy to change with their easy to use UI and the

ability to ssh into your server at any point in time and change the code running on it. You have full control of everything as if you owned it yourself and were not renting the hardware through Amazon. All of these features together made the AWS instance an easy solution when we were determining which web server host we were going to use.

Secondly, for the frontend we used Javascript, Bootstrap, CSS, and HTML. Javascript and Bootstrap gave us complete control over the functionality of the website whenever a user interacted with a feature on the site itself. We were able to do almost everything using Javascript on its own as none of the features we were implementing was necessarily difficult to do, but Bootstrap came in handy when we needed to do forms or just need easy-to-use JavaScript. There is no need to reinvent the wheel so we chose to use a good wheel called Bootstrap. Bootstrap has a built in grid system which is very helpful for a site like selling products where you want to lay things out in a grid format or have a toolbar in a grid format. Additionally, Bootstrap is very easy to use when rapidly developing as there are no extra steps besides simply adding the necessary bootstrap code and you are good to go.

CSS and HTML gave us the ability to stylize and add components to our website such as images and HTML. We did not use any CSS wrappers as once again we were not doing anything complicated where that was necessary. CSS is a very easy, infinitely flexible stylizing system which allowed us to create a good looking responsive user interface for people to interact with. CSS also has a huge community backing it as it is necessary for any website so we had no issues finding help via the internet. Additionally, there are lots of free CSS elements out there that people allow you to use in your website whenever you want too.

Thirdly, for our middleware we used Django which is an extremely powerful platform which operates off of python which is a very easy language to pick up for the whole team and utilize quickly. Additionally, Osman has had experience with it making it even easier for the entire team to get into as we had some sort of direction when starting with Osman's past experience. Django offers us the capability to talk to the database and front end from one easy to use central system without having to connect the parts up ourselves. Since Bootstrap consists of only static files, Django is very easily able to talk to it and use the capabilities to the max. Django promotes high level design with rapid development making it the perfect tool for us since we need to complete our project in such a short period. Django is also open source meaning that at any point if we needed to change functionality of the platform that option was there for us to incorporate. Since it is open source, the core team of engineers quickly respond to any changes needed for it and push up the fixes so that everyone's site using it is fully protected. There are tons of examples and documentation on the internet for us to rely on as so many people have used it. Django was the perfect middleware tool for us with its flushed out functionality.

Lastly, we used PostgreSQL for the database. Django connects very easily with PostgreSQL to interface with it allowing us to not deal with talking directly to the database itself allowing us to speed our development time up. Just like Django, PostgreSQL is an open source platform allowing us to change it anyway we need to get maximum effectiveness out of it. PostgreSQL is also a free platform which means no barrier to entry besides learning how to use it. There is plenty of documentation and an easy to use GUI so that we can visually see any changes we made to our database

without purely relying on the terminal to tell us yes or no whenever we made changes. PostgreSQL is secure and scalable as well making it perfect for a growing website such as our own. Again, just as Django and AWS PostgreSQL has been used by numerous groups and organizations which means it is user tested and approved with all of its usage.

Using all of these platforms together, we were able to successfully use our website in a much easier fashion compared to doing everything from scratch ourselves. The ability for Django to talk to all of the other platforms and turn everything into one cohesive package made developing our project a joy rather than a pain which is something you always look forwards to when creating anything. If we had to do this again in the future, we think we should rely more on bootstrap and other wrappers of JS and CSS rather than rewriting anything ourselves as their solutions are more elegantly made and function better than what we made.

Design of the Solution

We knew that the components of our development stack worked well together, and we planned to take a measured and organized approach to the design of our web application. The Django middleware by default provided a convenient solution to tie all our front-end and back-end together to create a functional and scalable site. It operates off of the MVC (model-view-controller) model, which is the architecture we kept for the entirety of the development process.

Django allowed our models representing everything having to do with stores, items, and customers, to be programmed as python classes, which are then translated to models in our PostgreSQL database at our command (the credentials for this database are specified in a special file provided by the framework). We programmed the site navigation and functionality through our 'views' (python) files, which could perform all necessary database logic and return request objects that loaded the requested page. All this functionality is either called by other views functions or sent via requests from buttons in the user interface. These requests would come from both HTML (forms) as well as JavaScript code (Ajax requests). To put it all together, user input is handled through the front-end interface, and sent to the middleware through requests to view functions, and then these view functions perform the database logic and, if necessary, redirect the browser to the appropriate webpage. This approach is straight-forward and

streamlined, especially due the convenience that Django offers, but of course in practice nothing is a complete walk in the park.

Implementation Issues

Even though implementing the website was a relatively painless process, we still ran into a multitude of roadblocks when developing the website and the backend to support it. The very first issue came on day 1 when we were setting up the AWS instance as we dove into creating the AWS instance before understanding how it worked. This caused the creation of numerous faulty instances which were not made with the correct settings to support our django server and run properly. Solving this was easy as we just took a step back and read the documentation until we properly understood exactly what we needed. The next issue came when creating the crazy butterflies website's calendar as we tried implementing the calendar from scratch by talking to the google calendar and pulling in the information into the site from there. This created a long and slow process of not being able to figure out how to do this properly as it would not pull the data or format the resulting page as we were wanting. Luckily, we ended up finding an ulterior solution to this problem which was a service called Tockify which is a dynamic calendar made for websites that links directly up to google calendars. We found a perfect middleman to solve the issue for us.

One huge issue we ran into were github merge errors which happened when two or more of us were making changes to the same files without having proper lines of communication setup. This would cause an issue in which we would overwrite each other's code, breaking the site and deleting progress. To get around this, we started using branches in our git repository which would tell us when merge errors would occur

upon merging into another branch or main. By doing this we could proactively fix the files that would cause error before pushing them onto the site.



Issues with Django were rampant in the project as well due to our basic understanding of the framework when hopping into it. For starters we did not know we needed to “collect static” to tell Django what files were to be considered static and thus able to be reached via the framework. This caused us to quickly design something, implement it, then see no results when we pushed it to the live site which confused us for a long time until we realized the errors in our ways. Additionally, we ran into issues when using the migration service Django offered as we would again update a relational model and then try to use the updated version without migrating the files. Or we would delete the migrated files as we thought they were garbage slowly growing rather than something extremely pivotal to the project. When we did this, the entire project broke and we had to revert back to an earlier version and redo some code in order to fix the issues we had made. Additionally, we struggled to interface with Django from js code when first starting. This caused us to write quite a bit of code in a work around fashion that eventually needed to be rewritten after we understood we could use Ajax to talk from frontend to Django which could talk to the database for us.

We also experienced API issues when using the paypal API and the google maps API. The paypal API was very easy to add into the website but created a roadblock when we needed to determine whether or not the process was a success or failure. The Google Maps API was another beast all together with extremely confusing documentation and poor examples walking you through the customization and usage of their api. They boasted a powerful mapping service which was true, but their

documentation was very weak. Trial and error was the victor in this scenario as we had to keep trying slightly different variants of the same thing until the end goal was reached with exactly what we were hoping to come out of it.






Lastly, a big implementation issue was our huge initial misunderstanding of the way relational models should work. We spent many days of the process reworking the database and models to be properly done for maximum efficiency. When we first started, we did them in such a roundabout way it became very hard to interface with them and extra information for us to use. All in all, the biggest issue we faced was our lack of knowledge in which we had to learn a swathe of technologies in order to effectively create our site in the end.

Testing

Testing for this project was done manually, we did manual testing on each of the modules determining expected behavior beforehand. The below tables explain which module each test belongs too, the test itself, expected behavior, seen behavior and whether they passed or not. If the test passes, they will have a  while a failure will result in a .

I. Manual Testing

1. Landing Page

Test	Expected Result	Seen Result	Passed?
Navigating to the website	Landing page	Landing page	
Items resize dynamically based on screen size	Yes	Yes	
Selecting the user info page while not logged in.	Redirected to the login page	Redirected to the login page	
Selecting the cart icon while not logged in	Redirected back to landing page	Redirected back to the landing page	
Upon scrolling down selecting the smooth scrolling icon	Smoothly scrolls back to the top of the screen	Smoothly scrolls back to the top of the screen	

Selecting the register button	Takes user to the registration page	Takes user to registration page	✓
Selecting the login button	Takes user to login page	Takes user to login page	✓

2. Logging In

Test	Expected Result	Seen Result	Passed?
Entering an incorrect username and/or password	Receive an error telling us that the login attempt was not successful	Receive an error telling us that the login attempt was not successful	✓
Selecting register here	Redirected to register here page	Redirected to register here page	✓
Entering a correct username password combo	Redirected to user info page	Redirected to user info page	✓
Forgot password button	Sends the user an email to change their password	Does nothing	✗

3. Registering

Test	Expected Result	Seen Result	Passed?
Entering a username already taken	Warns us a user with that name exists and asks for a new username	Warns us a user with that name exists and asks for a new username	✓
Password entered with insufficient length	Warns us the password is not long enough	Warns us the password is not long enough	✓
Password contains same words as	Warns us that the username and password are too	Warns us that the username and password are too	✓

username	similar	similar	
Does not select agree to terms and conditions	Popup tells us to agree to terms and conditions	Popup tells us to agree to terms and conditions	✓
Being logged in and selecting back to log in	Go back to landing page	Go back to landing page	✓

4. Finding a Store to shop at

Test	Expected Result	Seen Result	Passed?
Initial map placement	Map is centered over Riverside	Map is centered over Riverside	✓
Selecting a map marker to go to that store	Can be redirected to that stores page	There is no option to go to the stores page this way	X
Searching for an address	Auto Completes an address with nearby addresses	Auto Completes an address with nearby addresses	✓
Selecting a nearby address	Recenters the map to that location and shows distance to all stores from there	Recenters the map to that location and shows distance to all stores from there	✓
Select a store from the list on the left	Redirects to that stores page	Redirects to that stores page	✓
Ctrl + scroll	Zooms in and out of the map	Zooms in and out of the map	✓
Scrolling the mouse wheel	Scrolls down the page	Scrolls down the page	✓

5. User profile page

Test	Expected Result	Seen Result	Passed?
After account creation what is seen	Username, email, and phone if entered	Username, email, and phone if entered	✓
After placing an order	Last order time updates	Nothing happens	✗
After placing an order	Address is added into the page	Nothing happens	✗
Changing account information	There is an option to change email, phone number, and primary address	There are no options for anything	✓

6. Using Paypal

Test	Expected Result	Seen Result	Passed?
Selecting the paypal button	Redirects you to paypals secure API	Redirects you to paypals secure API	✓
Upon successful transaction	Redirects to the delivery page	Redirects to the delivery page	✓
Selecting paypal with the cost being 0	Attempts to open Paypal then closes it	Attempts to open Paypal then closes it	✓

7. Store Page

Test	Expected Result	Seen Result	Passed?
Searching for items	As text is entered, the resulting items containing the same string are show	As text is entered, the resulting items containing the same string are show	✓

Hitting the minus button when at 0	Nothing happens	Nothing happens	✓
Hitting the plus or minus button	Adds one or subtracts one accordingly	Adds one or subtracts one accordingly	✓
Add to Cart	Resets the quantity to 0 after successfully sending the information to the cart	Resets the quantity to 0 after successfully sending the information to the cart	✓
Displays the stores info on the page	Yes	Yes	✓

8. Checkout Page

Test	Expected Result	Seen Result	Passed?
Multiple stores are in the shopping cart	Has a button for each store to switch between	Has a button for each store to switch between	✓
Selecting plus or minus on the items	Overlays the screen with a pending wheel and then deletes or adds one to the cart	Overlays the screen with a pending wheel and then deletes or adds one to the cart	✓
Zeros out an item	Upon reloading the item is gone	Upon reloading the item is gone	✓
Store no longer has items being ordered from it	The store is deleted off the checkout page	Store remains there with an empty cart	X
Numbers always show to the hundredths place	No	No	X

9. Delivery Page

Test	Expected Result	Seen Result	Passed?
Looking at the map	Shows the location of the driver and each store as well as user delivery address and time for delivery	Shows the location of the driver and each store as well as user delivery address and time for delivery	✓
Order status	Order status updates on a 100 second period for sake of showing an example	Order status updates on a 100 second period for sake of showing an example	✓
Addresses are labeled in alphabetical order according to order	Driver is A and each store is the next letter all the way to the user address	Driver is A but each item is given its own letter instead of the entire store	✗
Upon order Delivered	Items are deleted from the cart	Items are deleted from the cart	✗

Future Improvements

This project has the core functionality up and running, but is not flawless and has a while to go before being able to be shown to the public and used as a proper grocery delivery service as it is intended to be. There are two primary portions in which this project has future work being bug/unintended behavior fixes and the addition of more functionality. Starting with the bug fixes, we see the majority on the checkout, store page, and user profile page. The checkout page does not have the stores deleted once they have no more items inside of them which is something we need to tackle. For the store page, if we redirect to an improper store string, the page will error out as we do not properly handle the unknown string. On the user profile page, there are no updates for any of the profile sections making it very useless as of now. There are other unintended behaviors throughout the website but the above are the more major ones at play.

Secondly, there are a swathe of features we want to add as a team to better improve the website's usage. First off for the orders, the ability to have an order history, save orders, and reorder the same items again with a few simple clicks. This will let the user track their usage as well as easily reorder their favorite items with little to no hassle. Next, we want to rework the coloring of the website as it does not feel a consistent theme all the way through which leads to a more jarring experience on the eyes compared to what it should be like. We also want to be in better partnership with

our clients posting their items onto the stores by creating a client portal which will give them control over the items on their store page and stylize it to their liking.

Additionally, we want to create a better store page as well which means adding categories and a list of popular items near the top. This will make it much easier for the user to navigate rather than just seeing a list of items and having to scan through all of them not knowing the categories they belong to. Also, we will add the ability to see nutritional info for each item upon selecting the item. As a society, we are becoming more and more health conscious so this is a must have on any product trying to sell consumables. We want to add a search radius option on the map so you are not met with such a huge list of stores but a more refined list for you. Finally, we want to add a search bar to the map so that the user can easily find their favorite stores over and over. This is not a complete list of future improvements, but what we are focusing on first. As we hear more back from users, we will adapt our website accordingly to make it the best possible user experience as can be.

Conclusion

All together, the project went very well and a lot was learned. There are definitely areas of improvement and many of things we would probably do differently if given the chance to start over. The team worked very diligently to get the project completed as early as possible and Donald Turner did a very good job on keeping everyone on track and focused during the entire project.

With that being said, everyone else also did their parts in making sure their work was done and done well. Corners weren't being cut and problems were being found as early as can be and adjustments to these problems were made accordingly.

Everyone was consistently making meetings and providing their input. Sometimes daily meetings were a bit longer than intended but most of the time when they were, very good work was being done that definitely saved time on the project later on.

As shown in the **Future Improvements** section, there are still things that need to be resolved to be confident in saying that the webapp is truly ready to be deployed to the public. Although the webapp works great, there will continue to be future improvements to be made on it that can continue to enhance the user experience.

Appendices

I. Software Requirements Specifications

Groceries

Software Engineering Software Requirements Specification (SRS) Document

*Donald Turner
Osman Halwani
Luke Doukakis
Ameer Abdallah*

<http://crazybutterflies.site/>

February 29th, 2021

Contents

1. Introduction	25
2. Background	25
3. Functional Requirements	25
4. Non Functional Requirements	29
5. Approach	31
6. Scope	32
7. Intended Audience	32
8. How our app will be used	33
9. Signatures	33

1. Introduction

The purpose of this document is to provide a detailed description about the requirements needed to develop a grocery delivery web application. The document will serve as a resource to the Crazy Butterflies development team, comprised of Donald Turner, Luke Doukakis, Ameer Abdallah, and Osman Halwani, to provide an interface for the developers to draw upon to see the project in the big picture, as well as help organize and implement every part of it. Throughout this document, we will provide a general description of the project, the functional and non-functional requirements for the application, a list of other relevant application attributes, application usage scenarios, use case diagrams, as well as an updated schedule.

2. Background

We are a team of programmers from California State Polytechnic University, Pomona who have experience in a multitude of different fields. Osman, a self taught programmer, found himself interested in computer science after he started building computers as a child. Osman's favorite programming language is Java which carries over well to this project with our databases as OOP is similar to tables. Luke, a quick witted individual, found himself interested in computer science after enrolling in a class in highschool. Luke's favorite thing to work with is Unity making games which carries over well as he knows what the end user wants to see in a functioning product aesthetically. Ameer, a homegrown computer scientist, uses his self taught skills to learn new and exciting computer science topics. Ameer's favorite thing about programming low level data manipulation which is perfect for managing the millions of entries we will be storing in our database. Donald, the CEO, uses his leadership skills to keep the team on a good pace. Donald's favorite thing about programming is working in VR which carries over to the project in that he can create a more immersive environment for the user.

3. Functional Requirements

Criticality Scale

<i>Non-Critical</i>	Non
<i>Less Critical</i>	Less
<i>Critical</i>	Normal
<i>Very Critical</i>	Very
<i>Extremely Critical</i>	Extreme

1. The system will allow the user to create an account in response to a request to create

- an account (i.e: Create Account button).
- a. **Criticality** - Extreme
 - b. **Dependencies** - none
 2. **The system shall allow the users to change their password**
 - a. **Description** - The password can be changed if the user would like to change their password
 - b. **Dependencies** - none
 3. **The system will prompt the user to input account information in response to creating an account**
 - a. **Description** - The account creation process will require a full name, email, password and re-entry of password.
 - b. **Criticality** - Extreme
 - c. **Dependencies** - F1.
 4. **The system will prompt the user to set up other account information in response to creating an account**
 - a. **Description** - Other information includes primary address, addresses credit/debit cards, EBTs, etc.
 - b. **Criticality** - Very
 - c. **Dependencies** - F1.
 5. **The system will allow users to select multiple items from multiple stores to add to their basket in response to adding item to their cart**
 - a. **Description** - Allow the user to search for products from a filtered number of stores. Without filtering, it should automatically choose stores closest to the customer.
 - b. **Criticality** - Very
 - c. **Dependencies** - F1.
 6. **The system will allow the user to checkout multiple items from multiple stores.**
 - a. **Description** - Another attribute of a class are the days of the week and times of the day when that class meets, for example “M W F 10:00am - 11:50am”.
 - b. **Criticality** - Very
 - c. **Technical issues** - This attribute must be generated from either direct user input or data retrieved from a web service or remote database.
 - d. **Risks** - Non-plaintext data could be more complicated to collect, store, and output appropriately.
 - e. **Dependencies** - F1.
 7. **The system will store data of all types in a secure relational database in response to an admin including new data or the user creating a new account**
 - a. **Description** - Examples of data entities that will need to be included are Accounts, Store Items, Available Drivers, etc.

- b. **Criticality** - Extreme
 - c. **Dependencies** - none
- 8. **The system will allow the user to search for items in response to input from the user in a text box**
 - a. **Description** - The user should be able to search for items while by default selecting to search from stores nearest to them first.
 - b. **Criticality** - Extreme
 - c. **Dependencies** - F6.
- 9. **The system will remove items from the user's cart in response to a request to remove an item from their cart**
 - a. **Description** - The user should be able to remove any item from their cart when reviewing their shopping cart before checking out.
 - b. **Criticality** - Very
 - c. **Dependencies** - F1, F4.
- 10. **The system will request a series of information from the user in response to the user checking out items in their basket.**
 - a. **Description** - The system will request the user to add or select an address, then ask for delivery time, specific delivery instructions, a mobile number to contact the user at for updates on the order, and then select or add a payment method.
 - b. **Criticality** - Extreme
 - c. **Dependencies** - F1, F4, F6.
- 11. **The system will be mobile friendly**
 - a. **Description** - The system will be as easy to use on the mobile side as it is in the browser itself.
 - b. **Criticality** - Very
 - c. **Dependencies** - none
- 12. **The system will have a shopping cart called a basket for each user**
 - a. **Description** - The system will keep track of all of the items the user has selected to purchase in the current system
 - b. **Criticality** - Very
 - c. **Dependencies** - F7
- 13. **The system will remember the information entered by a user.**
 - a. **Description** - The system will remember the items in a grocery cart
 - b. **Criticality** - Very
 - c. **Dependencies** - F7, F12
- 14. **The system will be easy to navigate through**
 - a. **Description** - The system will have a clear line of direction in how to order and get groceries delivered
 - b. **Criticality** - Very
 - c. **Dependencies** - none

15. The system will have a map showing nearby grocery stores

- a. **Description** - The system will have a map displaying nearby grocery stores that are connected to the app.
- b. **Criticality** - Very
- c. **Dependencies** - external

16. The system will have the ability to select a store

- a. **Description** - The system will allow the user to select a store on the map
- b. **Criticality** - Very
- c. **Dependencies** - F15

17. The system will store store's inventories in a table

- a. **Description** - The system will store the available inventory of any store in a table
- b. **Criticality** - Very
- c. **Dependencies** - none

18. The system will show a list of items available at the store

- a. **Description** - The system will show a list of the current inventory at the store
- b. **Criticality** - Very
- c. **Dependencies** - F17

19. The system will calculate the time to deliver your shopping cart

- a. **Description** - The system will determine how long it will take all your items to get to you
- b. **Criticality** - Normal
- c. **Dependencies** - none

20. The system will calculate the time to deliver your shopping cart

- a. **Description** - The system will determine how long it will take all your items to get to you
- b. **Criticality** - Normal
- c. **Dependencies** - none

21. The system will show the prices next to each product

- a. **Description** - The system will show the user how much each item they are purchasing costs
- b. **Criticality** - Very
- c. **Dependencies** - F17

22. The system will allow the user to select an item and see a blown up image with nutritional facts

- a. **Description** - The system will show the nutritional facts and see a larger version of the image
- b. **Criticality** - Very
- c. **Dependencies** - F17

4. Non Functional Requirements

1. **The system shall have a nice looking interface**
 - a. **Description** - The user interface for the system will be aesthetically pleasing to the user by keeping a consistent color palette.
 - b. **Dependencies** - 14
2. **The system shall encrypt the passwords**
 - a. **Description** - The passwords that the users enter when trying to log in will be encrypted.
 - b. **Dependencies** - none
3. **The user will be able create an account using “Log in with Google”.**
 - a. **Description** - The user can use “Log in with Google” to create an account using their Google credentials.
 - b. **Dependencies** - none
4. **The system shall allow the users to save their credit card information**
 - a. **Description** - After inputting credit card information the user will be prompted if they would like to save their credit card information
 - b. **Dependencies** - 10
5. **The system shall track the driver on their routes**
 - a. **Description** - The system will keep track of the gps location of the driver to determine where they are
 - b. **Dependencies** - none
6. **The system shall notify the user about delivery updates**
 - a. **Description** - The user will receive updates at various stages of the process letting them know how soon they will receive their groceries.
 - b. **Dependencies** - none
7. **The system shall keep track of the number of times a customer has used our service**
 - a. **Description** - The system will store data for each time a user logs in and uses the service
 - b. **Dependencies** - none
8. **The system shall delete inactive profiles of over a year**
 - a. **Description** - The system will delete inactive profiles in order to keep our system low cost with less data to maintain.
 - b. **Dependencies** - none
9. **The system shall notify you if an item can be found elsewhere cheaper**
 - a. **Description** - The system will let the user know if something in their shopping cart can be found at another store for a cheaper price.
 - b. **Dependencies** - none
10. **The system will reward repeat customers**
 - a. **Description** - The system will check how often a user utilized our service and

offer them rewards for being a repeat user

- b. **Dependencies** - none
- 11. The system shall categorize all groceries into respective categories**
 - a. **Description** - The system will group all vegetables, canned goods, etc into their respective categories
 - b. **Dependencies** - 7
- 12. The system shall limit the amount of “add item to basket” requests to 10 unique items every 15 seconds**
 - a. **Description** - To prevent server overloads and ddos attacks, only 10 unique item requests over 15 seconds will be allowed
 - b. **Dependencies** - 5
- 13. The system shall be able to store up to 100k different users data**
 - a. **Description** - The systems data storage will be large enough to have 100k different users information stored at a single time.
 - b. **Dependencies** - 4
- 14. The system shall be responsive to the users input**
 - a. **Description** - The system shall not delay any longer than 3 second for processes that require reaching out to the server and getting data
 - b. **Dependencies** - none
- 15. The system shall reach out to the nearest driver**
 - a. **Description** - The system will ensure the nearest driver to the store gets asked to pick up the items first before asking people further away to ensure quick deliveries for the customer.
 - b. **Dependencies** - none
- 16. The system shall have protection against password guessing systems like Hydra, a password cracker.**
 - a. **Description** - The system will ensure you can only try 7 login attempts before getting locked out for a certain period of time.
 - b. **Dependencies** - none
- 17. The system shall have stress tests done upon it to ensure its integrity**
 - a. **Description** - The system must be tested to ensure it can handle a high level of traffic at any certain time or a DDOS attack. To test this we will use LOIC or HOIC.
 - b. **Dependencies** - none
- 18. The system shall allow the user to access new items added to inventory within 2 seconds of being added**
 - a. **Description** - The system will ensure that the user does not have to wait longer than 2 seconds to receive the most updated inventory
 - b. **Dependencies** - 8
- 19. The system shall allow the user to override the recommended amount of drivers**

- a. **Description** - The system will allow the user to choose how many drivers they want to deliver their goods from multiple stores
 - b. **Dependencies** - none
- 20. The system shall only allow up to 3 different stores per checkout**
- a. **Description** - The system shall not allow the user to order from more than 3 stores at a single time
 - b. **Dependencies** - none

5. Approach

System

The application will run on any device that has a modern browser (ie: edge, firefox, chrome, etc). It will dynamically change the webpage depending on if the user is on mobile or desktop.

Response Time

The application server should be finished with any computations within 4 seconds of the user entering the URL and accessing the site.

Workload

The application must be able to support up to 10,000 users simultaneously.

Scalability

The application should be able to scale up as the demand for Groceries increases.

6. Scope

In terms of the capabilities and maintenance costs of our application, we have defined a scope for the project based on our time and monetary restrictions. Capabilities will include the ability to access locations of both vendors and customers in order to calculate routes and make deliveries, where the route calculation will be handled by Google Maps' API for optimal performance and time efficiency. Also included will be the capability for users to create accounts and store information regarding their purchases, choices and delivery locations. All of this

information will be able to be sent and retrieved from a database, with sensitive information encrypted for the sake of security. We will adjust our data storage/payment plan for the database according to how much data we find that our system uses to satisfy the requirements, though we will have a limit on this due to the fact that we will be paying out of pocket for this data storage. In terms of time restrictions, the goal is for the project to be completed by the end of the 2021 Spring semester, or May 15th.

7. Intended Audience

The intended audience of the application are consumers that go to grocery stores. Rather than physically going to the store we will allow groceries to be delivered to the consumer. We want to appeal to consumers that want the convenience of getting groceries delivered. Almost everybody in the USA goes to grocery stores therefore we want to attract the general population of the USA. We will try to make the service affordable to the average US consumer. More specifically we will target students, the working class, stay at home dads/moms, and businesses. Additionally, we want to target individuals who are more susceptible to illnesses such as COVID-19. By marketing to such individuals we can promote their safety and well being through the user of our app.

The second group of people we want to target are people who are in need of money as we will need drivers to deliver products to people once orders are filled. This will most likely be younger individuals or people just trying to make a quick buck. We will advertise flexible working hours and a manageable amount of weekly hours.

8. How our app will be used

The user will first go to our domain to get into the app. The user will register an account with our application. The user can either be a driver or a consumer. The consumer will find stores that he/she would like to buy groceries from. The consumer adds items to the cart and then a total is generated based on the miles the driver needs to travel. A driver will claim the job and fulfill the order. The driver will go to the stores and buy groceries with the company card. Then he/she will deliver the groceries.

9. Signatures

Name Of Customer: _____
Signature Of Customer: _____

Date: _____
Date: _____

Name Of CEO: Donald Turner
Signature Of CEO: *Donald Turner*

Date: 2/26/2021
Date: 2/26/2021

Name Of Employee: Luke Doukakis
Signature Of CEO: *Luke Doukakis*

Date: 2/26/2021
Date: 2/26/2021

Name Of Employee: Ameer Abdallah
Signature Of CEO: *Ameer Abdallah*

Date: 2/26/2021
Date: 2/26/2021

Name Of Employee: Osman Halwani
Signature Of CEO: *Osman Halwani*

Date: 2/26/2021
Date: 2/26/2021

II. Database Relational Model

The above model shows how the database and models for our website has been setup and will be setup in the future for something like Nutritional Facts which are not currently implemented into the product.

