Software Requirement Specification

foodLogistics: Bringing Farms to Families

Michael Solwold, Anahi Antonio, Hanifa Barry,
Luis Magana, and Bryce Harmsen for Taban Cosmos

December 6, 2019

Abstract

The software defined here, foodLogistics, is a food delivery management service intended to serve existing agricultural markets in South Sudan and surrounding areas, linking vendors and drivers, handling driver selection optimization, facilitating driver payment, and providing a portal for vendors to purchase goods from farmers. Three personas use the foodLogistics app: The farmer, the driver, and the vendor. The farmer uses the product to post food that is available for sale. The driver accepts delivery jobs based on availability and vendor requests. The vendor purchases food and requests drivers.

Contents

1 Introduction

1.1 Purpose

The software requirements specification, or SRS, provides a detailed overview of the project referred to as foodLogistics. The SRS is a comprehensive view of each module for the development team, advisors, and our client. It will assist in the maintenance of a consistent scope of the proposed software from the design stages through the deployment of the product.

Changes to the scope must be reflected in this document and the underlying descriptions. This revision process will involve all team members as well as our adviser and client. If a feature or behavior is not contained in this document, it will be considered erroneous and will not be supported.

1.2 Scope

The foodLogistics software is a safe and efficient tool that connects farmers to vendors located in South Sudan and the surrounding areas. Currently, vendors lack a reliable service for obtaining fresh produce. The goal of this application is to solve the logistical problems of finding and transporting food produce to vendors from farmers in rural areas. To accomplish this, the application will utilize contracted drivers to deliver purchased goods to vendors from the farmers.

There is a focus on three distinct user groups: Farmers, Drivers, and Vendors. Farmers will post their goods at a central marketplace that vendors will browse. If a vendor wishes to purchase goods, they will do so from the marketplace page. Once purchased, the app will search for the optimal driver to pick up the goods from the farmer and deliver them to the vendor.

Each user group will have an account specifically tailored to their needs. Vendor accounts will not be allowed to post goods to be purchased and, similarly, farmer accounts cannot purchase goods from the marketplace. Creating these distinct behaviors will ensure that the application has a clearly defined purpose and the roles will not be confused when building the structure of the software.

Drivers will have the most specific role of the three user groups. Drivers will be limited to delivering products from farmers to vendors. Drivers will be offered jobs when selected by the driver optimization algorithm, a key feature of the application. Drivers will have the ability to accept or decline delivery jobs, giving drivers flexibility over their activity. Once a job is accepted, the driver will be given explicit directions to their destinations. A job will be considered complete when both the vendor and driver confirm the delivery of the products.

1.3 Definitions & Acronyms

- Definitions
  - **Farmer** - Farmers produce good that are purchased by vendors and delivered by drivers.
  - **Driver** - Drivers deliver the goods from the farmers to the vendors
  - **Vendor** - A vendor could be either an individual that represents themselves or a group or a group of people that represents a vendor.
  - **Marketplace** - The Marketplace the main page in the application that holds all active listings of goods that are accessible to the user. This page will be dynamic to constraints specified by

both farmers and vendors as well as filters applied as additional constraints by vendors that are searching for goods to purchase.

- **Tech Stack** - The software components that will interact with each other to create the foodLogistics software.
- **Document Object Model (DOM)** - A platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.
- **Virtual DOM** - The DOM that is unique to the user. In many cases the user will interact with the client side in ways that do not require any syncing with the database. React allows for a modular and easy to use DOM that will assist this.
- **Real DOM** - The DOM that all users see. These are static values and views that only change when a user submits a change.
- **Middleware** - The portion of the tech stack that is in charge of translating data for the other side of the stack to use.

- Acronyms & Abbreviations
  - **App** - Application: A piece of software installed on a mobile device
  - **ETA** - Estimated Time of Arrival
  - **DOM** - Document Object Model
  - **SQL** - Structured Query Language

1.4 References

- **Client: Taban Cosmos** - Mr. Cosmos provides expertise in the area of managing projects as well as creating logistics solutions. He has offered himself as a resource for building the foundations for the application that is being developed.
- **React Native** - `https://reactjs.org`
- **Node.js** - `https://nodejs.org/en/`
- **IEEE.** - IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

1.5 Overview

This Software Requirements Specification (SRS) contains three major sections: the introduction, the overall description, and the specific requirements. The introduction covers the purpose, scope, and language of the SRS, setting the tone for the proceeding technical details and providing a point of reference for the reader. The overall description provides a high-level explanation of the full scope of the product, setting a general framework that the specific requirements expand upon. The specific requirements contain the most concrete explanation of the foodLogistics software, connecting the high-level design of the implementable functionality.

2 Overall Description

2.1 Product Perspective

foodLogistics is a standalone app that addresses a specific problem. While there is a possibility that there are some logistics systems put in place by certain vendors, foodLogistics does not intend to integrate into these established systems as this is strictly a standalone product.

The design of the application will draw inspiration from existing products such as GrubHub and UberEats. The app will provide a seamless experience for vendors that are looking to purchase goods by providing one central page referred to as the Marketplace.

2.1.1 System Interfaces

To keep the data in the app up to date, the app will refresh the information displayed by querying a PostgreSQL instance. This refresh will happen when the application is opened and when the user manually pulls the page down. Additionally, the app will query the database as the user scrolls through the marketplace, populating the screen with listings.

Allowing for the information to be populated dynamically will reduce the weight of the application as well as the amount of data that is used by it. In many cases, users will not need all available information from the marketplace making a full query unnecessary.

Account information will be fetched when a successful login has been made and will only update on request or when a change is made to the account. For more information on logins see the security section below.

2.1.2 User Interfaces

For the sake of simplicity, the user interfaces for each user group will be mostly similar. For granular explanations of each group and their interfaces see the User Groups section below.

As stated above, the core of the application is focused around the Marketplace. Because of this, the main page will be home to that system and any adjacent pages will exist as assistants to that functionality. The Marketplace will have scrollable, sortable, and filterable listings of all products that are available for purchase. This list is dynamic, updating the available items on request when the user pulls down on the screen.

Adjacent to the marketplace, the app will have a page dedicated to user activity. This page will have three distinct states,- one for each user group. Farmers will be given access to a list of all sales; drivers will be given access to all of their deliveries; vendors will be given access to all of their purchases. This list will be kept up-to-date and it must clearly communicate the status of each entry as well as the information associated with it.

The third page in the application will hold a map that is maintained by OpenStreetMaps. This map will display information that is relevant to the user group that is viewing it at the time. This map will be built for navigation or searching nearby farmers and drivers.

The fourth and final page will be dedicated to the user account. This page will hold user account

information, options for updating information, and application settings. This page is simple and deals exclusively with the specific user's experience.

With all three pages, customization options will be limited. This will reduce overall complexity of the application as well as reduce the likelihood of conflicts with device integration.

2.1.3 Software Interfaces

The tech stack used in the application will use five distinct pieces of software. These are listed below in order of proximity to the user interface (Top to Bottom).

- **React Native**
  - Link: `https://facebook.github.io/react-native/`
  - Version: 0.61
  - Source: Facebook Open Source
  - License: MIT License
  - Purpose: React is a JavaScript library that will function as the front-end of the application. React uses states and props to provide a Virtual DOM to the user that is tailored to their experience. When the user wishes to submit some information, React will update the Real DOM
  - Documentation: `https://facebook.github.io/react-native/docs/getting-started`
- **Express**
  - Link: `https://expressjs.com`
  - Version: 4.16.4
  - Source: Open Source
  - License: MIT License
  - Purpose: Express is a route manager that sits on top of Nodejs. When information is pushed from the front end of the application to the middle-ware, express works to efficiently route the information to the relevant Nodejs functions.
  - Documentation: `https://expressjs.com/en/5x/api.html`
- **Node.js**
  - Link: `https://nodejs.org/en/`
  - Version: 13.2.0
  - Source: Open Source
  - License: MIT License
  - Purpose: Nodejs is the core of the middle-ware as it handles events triggered by requests from the front end. Nodejs provides the communication between the front-end and back-end that creates a seamless user experience.
  - Documentation: `https://nodejs.org/dist/latest-v12.x/docs/api/`
- **Node-Prostgres**
  - Link: `https://node-postgres.com`

- Version: 7.14.0
- Source: Open Source
- License: MIT License
- Purpose: Node-Postgres contains various tools that improve the interaction between the middleware and the PostgreSQL database.
- Documentation: `https://node-postgres.com`

- **PostgreSQL**
  - Link: `https://www.postgresql.org`
  - Version: 12.1
  - Source: Open Source
  - License: MIT License
  - Purpose: PostgreSQL is an Open-Source Relational Database that will manage all of the data produced by the application.
  - Documentation: `https://www.postgresql.org/docs/12/index.html`

- **OpenStreetMap API**
  - Link: `https://wiki.openstreetmap.org/wiki/API`
  - Version: 0.6
  - Source: Open Source
  - License: ODbL License
  - Purpose: OpenStreetMap API will be used to provide the mapping data for our application.
  - Documentation: `https://wiki.openstreetmap.org/wiki/API_v0.6`

2.2 Product Functions

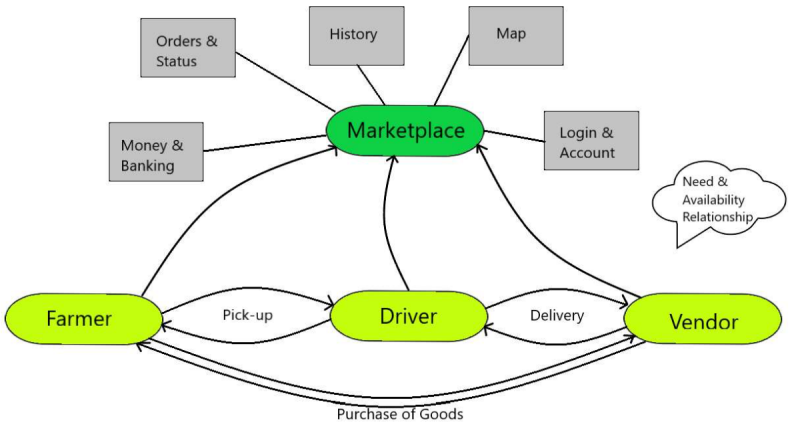

Figure 1: High Level Interaction of the Users

2.2.1 Farmer Class

Farmers are individuals that have goods to sell. Farmer Accounts are allowed to list items that Vendors purchase. Farmers should be required to provide proof of authorized service. If a farmer is not approved they should not be allowed to list items. If a farmer produces products that are not up to standard, vendors should be able to report the farmer for review. If a farmer has multiple reports, their approved status should be put up for review. For each listing, values should be checked against similar listings. If values are not within the relative bounds they should be flagged for review.

2.2.2 Driver Class

Drivers deliver goods from Farmers to Vendors. An algorithm will find drivers that are eligible to deliver the goods and give them the option to accept or decline the job. Drivers should be required to submit a drivers license for review. If they do not submit a valid license their account should not be approved. A driver should be notified of every job that is available to them. If a job times out the driver should be told and warned that they will be made inactive if jobs continue to time out.

2.2.3 Vendor Class

Vendors are individuals or groups that purchase goods from the farmers. There may be many people that operate under one account. If a user attempts to purchase an item it should be verified at the time of purchase. If the item is no longer available, the application should produce an error and notify the user. If a purchase does not successfully communicate with the server the user should be notified of the purchase failure. The system should not allow double purchases. The server should verify if it has received duplicate purchases and throw an error if it has.

2.3 User Characteristics

Users of this application are assumed to have knowledge of food delivery systems. Technical knowledge is less relevant than domain knowledge of agricultural logistics. The goal of the app is to provide a streamlined system, improving the efficiency of the user's current tasks.

2.4 Constraints

The core constraint of a successful launch is a stable connection to a cellular provider. The app heavily relies on connecting to a database to populate the app with relevant information. If this connection does not exist, the majority of the app will lose functionality.

Currently, there is no intention to directly integrate a wallet system, so security concerns are fairly low. To prevent fraudulent behavior,there will be need to verify user activity. These types of systems will be made obvious during development and will be actively documented and patched.

The server solution chosen involves a low-cost subscription to Render.com, hosting a relatively low-memory, high-latency cloud virtual machine. As the software gains users, this low-cost subscription will inevitably require an upgrade to a more expensive subscription, scaling the machine to match the needs of the user base.

2.5 Assumptions and Dependencies

This application is a proof of concept. Therefore, the app will be developed for currently supported Android and iOS operating systems only. It will also be assumed that a stable internet connection is available to all users. Modifications to fit poor connections will be made post-development if necessary.

The app assumes that the immediate users will have the ability to read and speak English. Any translation will be handled post-development.

2.6 Priority List

Critical - Included in Release

- System:
  - Connection to PostgreSQL database
  - The cloud server will serve data through secure HTTP calls to the client
  - Users can view the map and see relevant information
  - Successfully orders request a driver
  - Driver phone number is provided to vendor and farmer throughout an active delivery
  - Have the marketplace display the food produce that is being sold
- All users:
  - Account creation
  - All four tabs (main, activity, map, account)
  - Functional map
  - Clean interface
- Farmers:
  - Create food items
  - Update quantities
  - Set prices
  - Track sales
- Drivers:
  - Set activity status
  - Accept or decline jobs
  - Provided with delivery details and locations
- Vendors:
  - Fill cart
  - Purchase goods

High - Included in release

- System:
  - Optimize the algorithm that assigns drivers to orders
  - Push notifications are sent for each order status update
- All users
  - Offline storage
  - Filter listings
- Farmers
  - Set range of service
- Drivers
  - Set status to inactive after three missed Jobs
  - Use the map to display a route to their destination
- Vendors
  - Track location of drivers that are delivering their goods

Medium - Included in release if time allows

- All users
  - Customize interface
- Farmers
  - Create rolling listings (listings that allow multiple purchases assuming adequate stock)
- Drivers
  - Driver can progress order status
- Vendors
  - Track purchases
  - Change location for delivery

Low - Added post-development

- All Users
  - Download activity
  - Financial transactions will be managed first-hand by the server
  - The software will use Blockchain among other safety requirements to implement a secure money exchange handler

  - Users can message each other within the mobile app
  - The marketplace view will be populated with applicable popularity postings for each user type
- Vendors
  - Pre-ordering items from farmers

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The user interface will be fairly sterile. Because the main functionality is very specific, having an app that is streamlined will assist the users in avoiding any major complications. Below is a flowchart that was provided by our client, detailing the intended user experience.

Figure 2: Application Flow

All three user groups will have a similar interface. There are four tabs that hold individual functionality.

- Main - functions as the main page for all user groups
- Activity - displays relevant user activity
- Map - uses location data to provide user specific information
- Account - holds account information, settings, and application preferences

Below, each tab is explained in depth.

3.1.1.1 Main Page Interface

No Account This screen will hold generic information about the app. Similar to an about page, this main page will display different functionalities and cover the benefits of creating an account on the application. There are to be no live features of the application contained in this screen.

Farmer Farmers should have access to all of their active listings in the main page. There will be information on the specific product, cost, time since listing, number of interactions or purchases, etc. For every active listing there will also be an option to edit the post that, when submitted, will update the database and push the changes to all users.

At the top of the page there will be a button that will display the *Create Listing* page. This page is explained below.

When a vendor purchases their goods, farmers will have access to an *Active Sales* page. This page is explained below.

Driver At the top of the page, drivers will have the ability to toggle their current status from Active to Inactive. If they are inactive, they will not be given any new jobs to accept. If they are active the screen will populate with jobs they have been assigned.

The jobs will populate from top to bottom and have a five-minute timer for each job. The driver will have the ability to see the details of the job and accept or decline it based on their preferences. If the user has not accepted or declined the job at the end of the five minutes the job will auto decline and be assigned to a new user.

If a driver has three consecutive jobs time out, the app will alert them and automatically set their status to inactive to avoid any potential delays.

When on a job, this page will display information about the current job they are on as well as provide clear instruction that the map page is active and can be used for navigation. On the current job, there will be an option to cancel the job before they pick up the goods to be delivered. During this time the status of the job will be *Pending Pickup*. Once they and the farmer have confirmed pickup of the goods, the status of the job will change to *Delivery In-Progress*, the cancel job button will fade out, and they will now be liable for any loss of goods.

Once the job has been completed, the job will join the list of completed jobs that, by default, will be sorted by completion date.

Vendor Vendors will have access to a screen that is populated with listings created by Farmers. Users will only see listings that they are eligible for. Eligibility is determined by their specified range and driver availability. If there are no active drivers that can deliver the products to them, the app will not display the listing.

When a driver has ordered goods, they will have access to an *Active Orders* page. This page is described below.

3.1.1.2 Activity Page Interface

This page will hold relevant activity for each user account. For each user, there will be two states that the page can have. Our client wishes to have some offline functionality and having access to a limited view of their activity when offline is a desired feature.

At the top of each page there will be an option to filter down on specific activities. Those filters are explained in more depth in each section.

No Account The Activity tab will be greyed out to indicate that there is no relevant content for the user.

Farmer This page will hold all sales made by the farmer. At the top of the page there will be specific filters that can be added on the list to allow for quick look-ups of specific sales.

Online When online, the activity page will contain sales made. The page will hold only the most recent sales when initially loaded and will dynamically load additional sales on demand.

Offline Farmers will have a limited view of orders that have been completed in the past month. This list will be maintained inside of the application and contain only the most relevant information. A main concern is storage space so limiting the information that is stored for offline access is imperative.

Filters Farmers will be allowed to filter all, but not limited to, the following items.

- Date
- Items
- Price

Driver

Online When drivers are online, they will have a full list of all jobs they have completed with information about distance, amount earned, amount of time the job took, etc. This page will work to keep track of the completed jobs for the drivers to use as a reference. If a driver wishes to interact with a specific job they will see detailed information about the job including ratings and any notes that may have been left.

Offline Drivers will have a limited view of their completed jobs over the past month. It the offline section will show mileage, total time of the job, pickup location and drop off location, and amount of money earned for the job.

Filters Drivers will be allowed to filter all, but not limited to, the following items.

- Date

- Mileage
- Amount Earned

Vendor

Online Vendors will have access to all of the completed purchases. This page will store information about who they purchased from, the amount of the purchase, what they purchased, who delivered the items, and how long the order took to be fulfilled.

Offline Vendors will have a limited view of their completed purchases over the past month. This view will contain much of the same information as their online page but it will only have purchases in the past month.

Filters Vendors will be allowed to filter all, but not limited to, the following items.

- Date
- Items
- Amount Paid
- Farmer they purchased from

3.1.1.3 Map Page Interface

The map tab will use OpenStreetMap API to display specific information for each user group. When the user is offline, this tab will be greyed out.

No Account The Map Tab will be greyed out to indicate that there is no relevant content for the user.

Farmer Farmers will have a map that shows their area of service. This will give them a visual to help them decide if they should increase or decrease their range of service.

Driver Drivers will have a map that shows the location of all farmers they may get orders from. Once they have accepted a job, the map will give them directions to their destination.

Vendor Vendors will be able to use the map to see eligible farmers in their area. When they have purchased goods, the map will display the location of the driver and the ETA of the delivery.

3.1.1.4 Account & Settings Page

All three user groups will have a similar interface inside of the Account tab. This tab will contain buttons that have a specific action. If a user is not logged in they will see a login screen that they can use to sign into their account or create a new account.

Account Information This button will allow the user to view their account information. Inside of this window the user will have the ability to edit certain pieces of account information after pressing an edit options button. Information types are expanded on in the User Classes section below.

User Settings User Settings refers to all settings that are relevant to the application functionality. While there are to be few customization options as explained above, this window will allow for the addition of options if requested. Examples of possible settings include: Light and Dark Mode, low data mode, font size, etc.

Logout Signs the user out of their account.

Additionally, there are a few additional screens that have critical roles but are accessed through features in the main tabs.

3.1.1.5 Children Pages

Create Listing: Farmers This page will allow farmers to create new listings of items. This page can be pre-filled with information from previous orders to ease the creation of duplicate listings.

Active Orders: Vendor Vendors will have access to a list of orders they have placed but have not been delivered yet. It will show if the items are pending pickup or in transit. When the items are in transit this page will show who is delivering it, their eta, and a link to the map page to see their location.

Create Account: All Users This page will be used to create an account. After a username, email, password, and phone number are entered the user will choose an account type. They may choose Farmer, Driver, or Vendor. Once the user has chosen an account type, the page will populate with account specific information for the user to fill.

Status: All Users This page will display the jobs that are in-progress. If a Farmer has an order that is pending delivery, a driver has a job that is in progress, or a vendor has an order that has not been delivered it will be displayed in this page.

3.1.2 Hardware Interfaces

Our tech stack is the following:

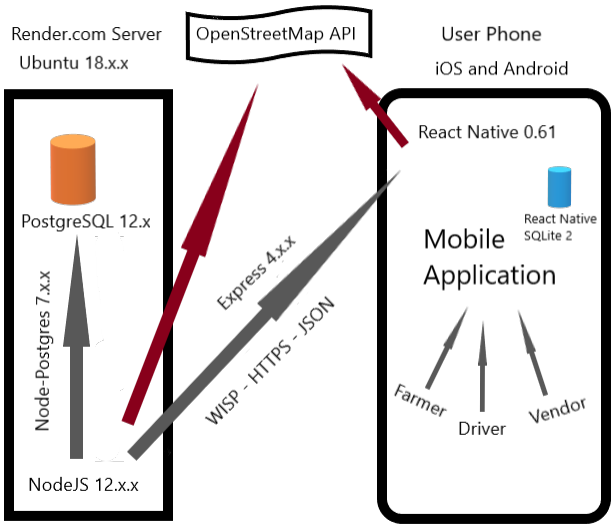

Figure 3: Interaction of the Tech Stack

The foodLogistics software will interact with a Linux-based cloud virtual machine at Render.com. This machine will host the server. Both PostgreSQL and NodeJS are operable in a Linux environment. The cloud server will be interacted with by the system administrator through SSH secure keys. The users will interact with the server through the foodLogistics mobile application. This mobile application will be hosted client-side using React Native, calling to the cloud server using HTTP protocols. For security, the server will provide an SSL certificate, allowing an HTTPS connection. React Native is a cross-platform framework that compiles into native code for the user device, simplifying phone hardware interaction. The two limiting hardware factors here are the 1 GB memory constraint of the server (unless more is purchased), and the user phone operating system minimum requirement. The client has not provided a lower bound for backward compatibility, so the development team has deemed Android 4 and iPhone 5 to be a reasonable lower bound. The software may encapsulate more of the phone market below these bounds but is not guaranteed to reach below these bounds.

3.1.3 Software Interfaces

We will be using react on the front end of the app. It works perfectly for our uses because it is focused on user states and letting the user interact with a virtual DOM before committing anything to the Real DOM.

React flows really well into a NodeJS middleware and Express only improves that interaction. Express should route traffic to the most efficient functions and take some load off of the NodeJS software.

NodeJS will send traffic to a PostgreSQL based database with help from the Node-Prostgres library. Node-Postgres will be used to improve security and limit any issues with high load on the database.

This tech stack should be integrated in a way that is easy to replicate to allow for expansion post

development.

3.1.4 Communications Interfaces

The foodLogistics mobile application will rely on WISP for internet connection from the client to the server. This connection will use HTTP protocol. For security of financial and personal data, clients will connect to the server via HTTPS. The software will also require a third-party API interaction with OpenStreetMap to run the maps services within the mobile app. Communication between users resides mostly in location sharing and push notifications of order status updates.

3.2 User Classes

Listed below are each user group's functions, including interactions with other user groups.

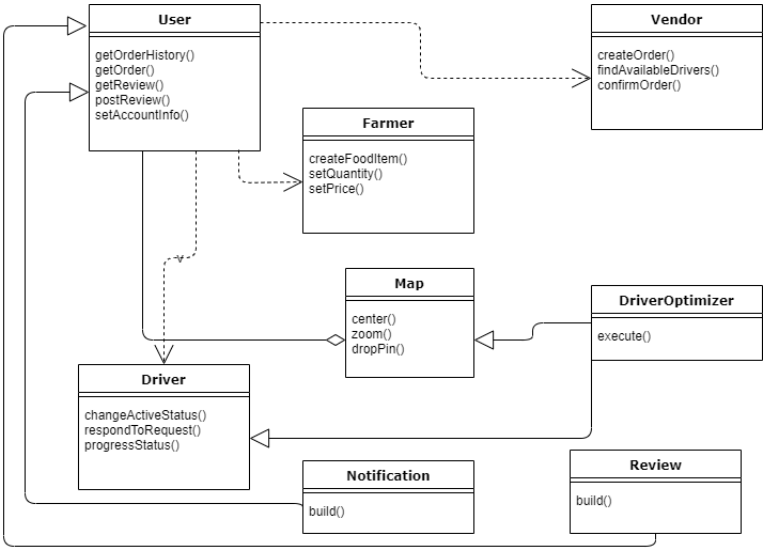

Figure 4: Low Level Functionality of the Users

3.2.1 User Class 1: Farmer

Farm Registration The farmer must provide details of the farmer's name and location of farm site.

Driver interaction scenarios

Receive Pick-up Notification The farmer receives a push notification when an order has been submitted for food from the vendor.

Prepare Pick-up Along with the push notification, the farmer is provided order details including food items, quantity, and driver ETA.

Tracking Delivery Upon pick-up of the order, the farmer will be provided access to a map pin that demarcates the current position of the driver throughout the delivery.

Delivery Notification The farmer receives a push notification when the order has been successfully delivered.

Review The farmer is prompted to provide a star rating and written review of the driver.

Vendor interaction scenarios

Create Available Food Item The farmer creates a new food item or selects a pre-defined food item and marks it as available at the farm site.

Update Quantity for Available Food Item The farmer updates the quantity of available food items when new quantities of the item become available (through harvest or other sources). The software will automatically decrements the farmer's inventory as orders are completed.

Set Price for Food Item The farmer can adjust the price per quantity and select a pre-defined unit of quantity for each available food item. Pre-defined units are kilogram, gram, liter, and each.

3.2.2 User Class 2: Driver

Registration Request The driver requests access to the site as a driver, providing a vehicle make, model, and license plate number. Driver is provided access to the site once vehicle information is confirmed (or driver is denied access when vehicle information is fraudulent).

Set Activity Status The driver sets status as active or inactive through a toggle on the driver's home screen. When the driver is listed as active, the driver will be included in DOA executions and will be offered delivery jobs. When the driver is listed as inactive, the driver will be excluded from DOA executions. Farmer interaction scenarios

Confirm Job The accepts (or rejects) the delivery request. The driver is provided details about the delivery including size of order, farmer details, vendor details and route details. When the driver accepts the delivery, the farmer and vendor are sent push notifications that the order has been assigned a driver and the order status is set as awaiting pick-up.

Pick-up Arrival Upon pick-up of food from the farmer, the driver progresses the status of the order from awaiting pick-up to on its way. This triggers a push notification to the farmer and vendor.

Complete Order & Review Upon delivery of the order, the driver progresses the status of the order from on its way to delivered. This action triggers push notifications to the farmer and vendor of the delivery status update. This action also initiates the review process, prompting the driver to assess the farmer through a star rating and written review.

Vendor interaction scenarios

Confirm Job When the DOA selects a driver for a delivery job, the driver is prompted to either accept or reject the job. The driver is provided with delivery details including order, farmer, vendor, and route details. If the driver accepts the job, the delivery is assigned to the driver, the driver is provided route details through the map interface, and the driver is given access to progress the delivery status of the order.

Confirm Delivery Upon delivery of the order to the vendor, the driver progresses the order to delivered. This prompts automated actions within the software outlined in section 2.8.2.5 above.

Review In addition to the automated actions of section 2.8.2.5, the driver is also prompted to review the vendor through a star rating and written review.

3.2.3 User Class 3: Vendor

User Registration The vendor is prompted to fill out and submit an informational form containing vendor name, location, and payment details when signing up for a vendor account with foodLogistics. Upon verification of vendor details, the vendor is granted an account (or rejected in the case of false information).

Set up payment account As a part of the registration, the vendor enters credit card information to be used for orders made by the vendor. These details are verified before the vendor is granted access to an account.

Farmer interaction scenarios

View Food Availability The vendor views food available from any of the farmers by selecting the farmer through a map pin or farmer menu.

Populate Cart The vendor can see quantities available for each food item and can add up to the available quantity of a food item to the cart.

Purchase Food Once the vendor has selected food from a farmer, the vendor submits the order from that farmer. The vendor is prompted to confirm the order details, including items, quantity and cost.

Review Upon delivery of the order, the vendor is prompted to provide a start rating and written review of the food delivered.

Driver interaction scenarios

Request Driver Upon purchase of an order, a vendor requests a driver. This initiates an execution of the DOA using the farmer and vendor locations as parameters.

Confirm Driver When the DOA returns the prospective driver(s) to the vendor, the vendor can see the driver details (including ratings) and either accept or reject the driver. If the vendor accepts the driver, the delivery job is progressed to awaiting pick-up. If the vendor rejects the driver, the DOA is executed again, excluding the rejected driver from the search.

Track Shipment Along with order progress updates via push notifications, the vendor is provided a map pin demarcating the location of the driver throughout the delivery process.

Delivery Confirmation Upon delivery of the order, the vendor is provided a notification that the order has been marked delivered by the driver.

Review Upon delivery of the order, the vendor is prompted to provide a start rating and written review of the driver.

3.3 Performance Requirements

The DOA will return a list of optimal drivers to the requesting driver through an AJAX call within 5 seconds. The drivers selected by the DOA can be at most 0.8 miles above the optimal available driver's distance from the farm listed in the order.

3.4 Logical Database Requirements

The Database should hold all information relevant to the application. Minimal information will be stored client side for for storage concerns and security. Pooling will be used by Node-Postgres to reduce load on the database when multiple users are attempting to access the same information.

The database will be a relational structure. All orders should be in one table and use foreign keys to join sets. Clearly defined relationships will help keep queries clean and readable which will help debugging and minimize cross contamination of data.

There are two types of databases within the foodLogistics software. The main database is the PostgreSQL database on the server. The secondary database, a React Native SQLite 2 database, exists on each user phone. The main database will be backed up during low traffic once daily. The secondary database will be updated on order completion or on user login.

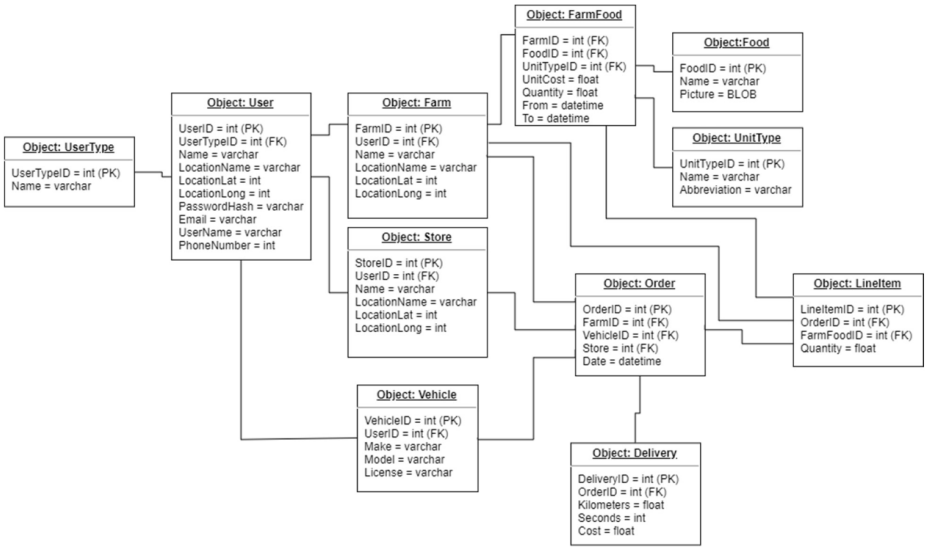

Figure 5: E-R Diagram of the Database

3.5 Software System Attributes

3.5.1 Reliability

The server will be backed up once daily during low traffic hours. This will ensure a higher reliability of the data served by the software. This software heavily depends on a reliable internet connection and will thus be no more reliable of a software than the local internet service providers of the users. To provide a more fault tolerant system, a process manager will be run in conjunction with Node as a watcher. Upon process termination, the process manager will restart Node.

3.5.2 Availability

The mobile application will be available to any user group with reliable internet connection. The server will be hosted through a cloud virtual machine. This virtual machine will serve data through HTTP calls. The software requires the user to own a cell phone meeting the minimum requirements detailed in section 3.2.

3.5.3 Security

Our focus with security is correct password handling as well as preparing for the integration with a third party wallet. The majority of our security breaches would likely stem from SQL Injections. Node-Postgres will help in this area in most cases, but it is still a very real flaw that could exist.

3.5.4 Maintainability

Most of the applications expansion will come from altering the front end of the application. Because of this, building a modular code base should be the focus from the start of development until the end. The team should keep track of any potential issues of maintainability that arise during development and address them in order of priority through the task backlog.

3.5.5 Portability

The server-side code base will be movable and available separately within a git repository, and the database can be migrated through some well-formulated queries. The virtual machine operating system image and its configurations exist within Render.com and are not easily portable. The client side is ultimately portable among mobile devices. This is a feature of React Native and its cross-platform capability. Non-mobile devices are not considered.

3.5.6 Validity Check

The mass majority of the client's requests are valid. One concern is with the chosen server hosting service, Render.com. According to the payment options available at Render.com, only static sites are free for hosting. Otherwise, and in our case, a monthly budget is necessary for pages to be rendered dynamically from a database. The client provided no budget in the requirements. The client should determine a budget or request a different server hosting service.

3.5.7 Consistency Check

The majority of the client's requirements are consistent. The only concern is the client's response to *Should the algorithm be the sole decision maker, or should drivers be able to deny jobs that are requested?*, was yes. These two options cannot both receive an affirmative answer and exist simultaneously. Unless stated otherwise by the client, the development team will move forward with the option that the driver will be able to deny job requests.