Creating a Distribution Model for Small Businesses

Cameron Cobb (50568054), Braden Jackson (50552413), Yuji Kikuchi (50526578),

and Dilynn Strickland (50391107)

College of Engineering & Computer Science, Arkansas State University

CS 4143: Java Application Development

Dr. Christos Grecos

April 28, 2023

Abstract

The business world is expanding rapidly, and the number of businesses that open multiple locations in a general area is increasing with it. With this, the need to also distribute goods between different locations of one business also becomes more apparent. A great example of such is Shadrachs, a company founded and run out of Jonesboro, Arkansas. This company has six locations that sell their products in Jonesboro alone, as well as an additional two in Paragould, Arkansas. Companies like these are in need of reliable distribution softwares, without the pricing and extra-features that come with large-bundle packages from corporations, which they won't use. The proposed project can help solve this issue.

The general goal of this project is to provide these services: a location manager, a distribution manager, and a warehouse manager service. The Location manager should be capable of performing several tasks. These could include updating their inventory, generating a report of their current inventory, and making requests for additional inventory from the company warehouse. The warehouse manager should be able to request inventory reports of specific and all locations, send unrequested inventory to locations, view the warehouse's inventory, and enter new inventory into the system. The warehouse user dashboard should be able to receive notifications detailing order information (such as location, what is in the order, and date the order was placed), be able to generate reports of current orders needing to be fulfilled, clear orders that have been completed, and reject orders that cannot be completed (while also providing a reason for rejection).

The project is to be hosted off of a website, which will utilize a front-end, back-end, and database to be functional. A log-in system will be utilized to ensure the correct people can only access what they are authorized to access. A database will be utilized to store data related to

inventories, users, and other information that will be necessary for the site to work. We will also be utilizing front-end libraries to assist in the styling and creation of a modern, aesthetically pleasing user interface. For the framework, we will be using NodeJS, Express and Redis to manage and run the web-server. Argon2 will be used to help manage passwords for user accounts. Better-Sqlite3 will be used for the database, and JOI will be used for creating schemas for the database. Services like tailwind or bootstrap may be used for the simplification of front-end design, while EJS will be used for fragmenting HTML to be rendered server side. Javascript will be used for the majority of the backend unless otherwise is needed.

Creating a Distribution Model for Small Businesses

Our goal in this project is to create a well-written inventory distribution system that can be used by many businesses that have the correct use for such. We have set out in the right direction and we are on a well thought out timeline to deliver a usable product for our consumers. The tools we are going to use to accomplish this web application are included in this paper along with descriptions of what they are and how we will or have applied them.

Team Contributions

This application was created by the "BroCode" team, consisting of Cameron Cobb (50568054), Braden Jackson (50552413), Yuji Kikuchi (50526578), and Dilynn Strickland (50391107). The work assignments of this project were distributed based on what the team leader, Cameron Cobb, was able to delegate with the entire team. These assignments were also distributed based on each team member's area of expertise. Cameron focused on front-end frameworking, designing and creating the database, merging and integrating front-end development with back-end development written by other members, team communication, and project management. Braden focused on helping solidify the user experience of the application by streamlining certain processes, as well as assisting in the creation of the registration and login portion, and also spearheading the orders functionality. Yuji focused on back-end functionality, such as leading the charge on the registration/login functionality and creating the location, warehouse, and item/inventory systems so that they could be integrated into the front-end. Dilynn also focused on the Items functionality, as well as the initial setup of the application including creating the GitHub repository, initializing the NodeJS framework, adding dependencies, and implementing Tailwind into the application. He also assisted in completing

complicated merges of conflicting branches, tailwind styling, and code review. Dilynn also was a major contributor to fixing difficult bugs throughout the project. While this is generally what each team member specialized in during the course of the project, the reality is that all team members had to help in every portion of the project, even if it was outside of their specialty.

Literature Review

An inventory management application from a developer named GreaterWMS on Github is similar to our project. For instance, the project on Github includes some similar functions such as viewing inventory, order management, listing warehouse information, and managing employees. In addition to that, the layout for the application is similar to our application as well in that it has the menu bar on the left side of the main screen, as well as sharing several of the same functionalities. The functionalities implemented in the inventory management application by GreaterWMS that are different than those implemented in Distro include financial management and customer management, however, Distro does not include these features as functionality because it is outside of the initial scope of the project Distro is built based on the programming languages such as HTML,CSS, Tailwind, and EJS for the front-end part and JavaScript for the back-end. Furthermore, for the database portion, SQLite3 is utilized. Regarding the communication protocol, since our project is created as a Web application, users for the application need to communicate through web browsers.

System Requirements

Our application, Distro, is a hyper focused ordering app that makes it easier for several locations under the same company order from a centralized or network of warehouses. When you

navigate through our owner registration page followed by logging in, you will find useful tools as an owner of the company. Adding employees under specific locations, adding items that can be added as inventory to warehouses, and being able to place an order from a specific location from a warehouse are what the system is expected to do. Requirements include being an owner of a business that requires a warehouse that stores items that the business requires to operate. The owner does not have to have multiple locations and warehouses; one location and one warehouse suffices to be able to take advantage of our system. The owner does not have to have any employees, although we found that without them it is hard to run both locations and warehouses at once.

High-Level Interface Description

Distro consists of two different interfaces. The public-facing informatic portion of the website, and the user interface portion of the website. The public-facing website contains a horizontal, top-of page toolbar that contains links to several other pages that contain information about the project. These links include the “Home”, “About”, “Contact”, and “Our Team” pages, as well as links to Register for and Log In to the service. This navbar also changes based on if the user is logged in already or not. If the user is logged in, the navbar will display “Dashboard” and “Sign Out” instead of “Login” and “Register” respectively. This informatic portion of the website is purely meant to be used as an informational resource for prospective users, and provides no additional functionality to the service outside this.

Once logged into the service, the user is sent to a dashboard, which contains almost all of the user interface view. This dashboard has a vertical navbar on the left side of the page, and a horizontal navbar on the top of the page. The top horizontal navbar is used to navigate to the

Home page (otherwise known as the Landing Page), or the Dashboard home. The left vertical navbar contains most of the navigation for the rest of the dashboard, with the rest being located within pages of the dashboard. This left vertical navbar includes links to the “Overview” (otherwise known as the Dashboard Home), “Inventory”, “Warehouse”, “Locations”, “Orders”, and “Accounts” pages. The Overview page is where the user lands once they have logged into the website. This page, or panel, shows the User’s name, their role in the company, the company they are assigned to, and the main address of that company. All users and views are able to access the Dashboard overview panel.

Panels become role-restricted after the Overview panel to help suit the user experience. The team opted to not show unavailable options at all, as opposed to some services that will simply “grey-out” options that aren’t available to the logged-in user.

For Owners and Admins, the Accounts Panel, Warehouse Panel, Orders Panel, and Location Panel are available. For Warehouse managers, only the Warehouse and Orders panel are visible. For Location managers, only the Inventory and Orders panel are visible.

Object Oriented Aspects

While the creation of objects was not entirely necessary for the Distro project, they were still used throughout the codebase thanks to NodeJS and the utilization of Javascript objects. While any method of creating a web-application consists of requests and responses, NodeJS formats its requests and responses as objects, so that they can be easily accessed by the developer, and easily added to. This is also built on when using EJS, as almost all of the project pages are passed an object containing information about the user, called the “session”. This session, if it exists, tells each page if the user is logged in, what their role is, and other basic

account information. New information can also be inserted into the request object as well, such as what page the user is loading, or what locations/warehouses are associated with the account. This allows each page to be customized to the user's experience. Specifically, this allows the dashboard to be customized so that the dashboard EJS files can recognize the user's role to show them the proper information. Javascript objects are also used in the project to pass simple bits of information between EJS pages and partials. For instance, on the home-screen, the EJS will pass the NodeJS session within a Javascript object to the header partial. If the session exists, the partial renders a "sign-out" and "dashboard" button onto the navbar, however if the session does not exist, this means that the user is not logged in, and it renders a "Register" and "Login" button instead.

Data Structures and Algorithms

The app will consist of multiple libraries that use a variety of data structures. Starting with the data, with the Model-View-Controller design pattern used in this web application, the Models are the connection with the database. When querying the database, the result will be returned in array structures which are a linear data structure. Of course, when querying, the result should return a single row if used correctly but using it to return several items of the same category could be desired. This will return a keyed array which, if the accessor is known, could result in a Big O notation of O(1). Otherwise, we will parse the data to display, which will result in O(n). Further, we will be using Argon2 for password hashing. This is a library in the Node Package Manager that uses a hash algorithm to create password safety when storing and pulling from the database. Redis, another library from npm, uses a dictionary to take care of our session

needs. Dictionaries behave like a keyed vector where, in this case, the key is the session id and the value is the session information.

Level of Concurrency

A well configured web server could be utilized to split tasks to each of its threads. However, this may be a dream for this project due to time and monetary constraints. For the sake of the idea, an explanation is in order. Beginning with the specifics of our live web server, a droplet purchased monthly from DigitalOcean, a total of around 50 or 60 users will be able to use the app concurrently. This is the lowest purchase price that they provide. This is resizeable, of course for more money a month, so the number of threads increases as you pay to resize. The number of concurrent users will increase in direct correlation to the money we provide to DigitalOcean for their services. We believe that the tasks will be parallelized in a first come, first serve fashion where threads will be allocated to the tasks that are requested right when they are requested. This will be up to the machine that we rent.

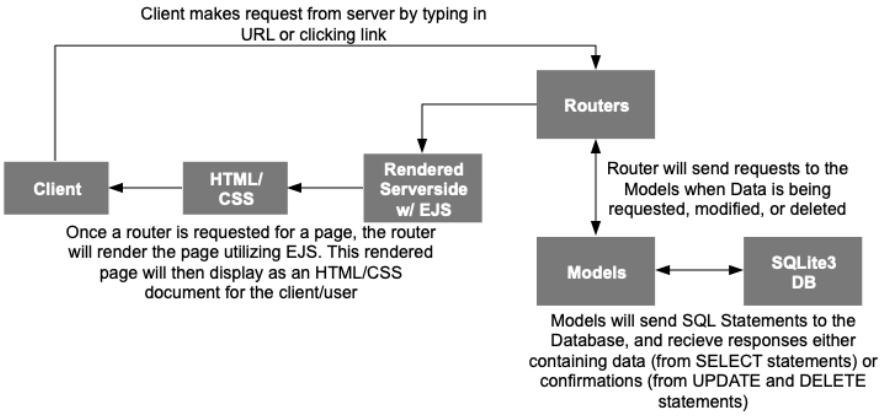
Task Dependencies

As you can see in *Figure 1*, a client will request to see the web page and will be responded with seeing the result of HTML/CSS after being rendered by the server using EJS. When the client performs another action, like clicking a button, it will send a request to the server which will execute the programs in its models. The models will eventually send queries to the SQLite3 database to gather data. Within SQLite3, there is a scheduling event that takes place to prevent errors like overwriting the database. There is a write-lock mechanism implemented inside of SQLite3 when transactions concurrently occur in the database. This mechanism will

allow one transaction to be made at a time, which means only one request can change the database at a time. This is important for data to not be overwritten by a transaction or interfered by another transaction. After a transaction in the database, a response will be made with that data back to the router which will update and render the HTML web page. The client will then see that updated page that their requested action caused.

Figure 1

Top-Level Design of Application showing Task Dependencies



Note: Arrows represent ways in which data is passed between the client(user) and components of the application.

Software Design

The design of the software, in this case, the design of a web application, is being expressed through diagrams and wireframes in most cases. Many examples of these diagrams are found throughout this document. Specifically, *figure 3* was a diagram made prior to the beginning of development on the application. Other diagrams were created during the early stages of development to help clarify dataflows and user permissions, such as *figure 2* and *figure*

9. Additionally, a rough wireframe was also made of the application prior to development to help guide overall development by helping the team understand how the application will be used. An example of this is *figure 5, figure 6, figure 7* and *figure 8.* Two softwares were utilized in the creation of these diagrams and wireframes: Flowchart Designer by Zhang Guangjian and Adobe Illustrator. Flowchart Designer was used to create all diagrams, while Adobe Illustrator was utilized in the creation of the wireframe examples of the application's frontend.

Software Methodology

Our software methodology can be described as a hybrid of waterfall and agile development methods. Our process is partially a waterfall method because of the way the requirement analysis was performed first, and how most of the system design was completed first. In the future, the implementation and testing of designs will begin, which then turns into an agile development methodology. Going forward, the development flow will consist of redesigning according to problems encountered to the design, develop part of the design, deploy the implementation, and then review the entire design again. If only one methodology had to be picke, the waterfall development methodology would suit the project better because how the project was largely planned ahead of development. Going forward, each piece will be developed according to the design and then tested before moving on the next part of the project.

Architectural High Level Design

An illustration of the high-level design of this application can be found in *figure 1* above. In this application, the client using our services will ultimately see the results of HTML/CSS. The HTML/CSS is created in our server using Embedded JavaScript Templating (EJS), which

allows the use of javascripts alongside HTML and CSS, this is helpful for creating conditional variations of webpages. Our server receives requests from routers and sends responses back to them. When a request is received from a router it will queries from the corresponding Models if data from the DB is needed. Then the Models will then execute queries that are needed to respond back to the requests. The queries are sent to our SQLite3 database from the Model, and once data is gathered from that query, a response will then be sent back to the router that will allow the client to see the web page that the response allowed them to see after rendering that web page with the appropriate html.

Internal Module Level Design

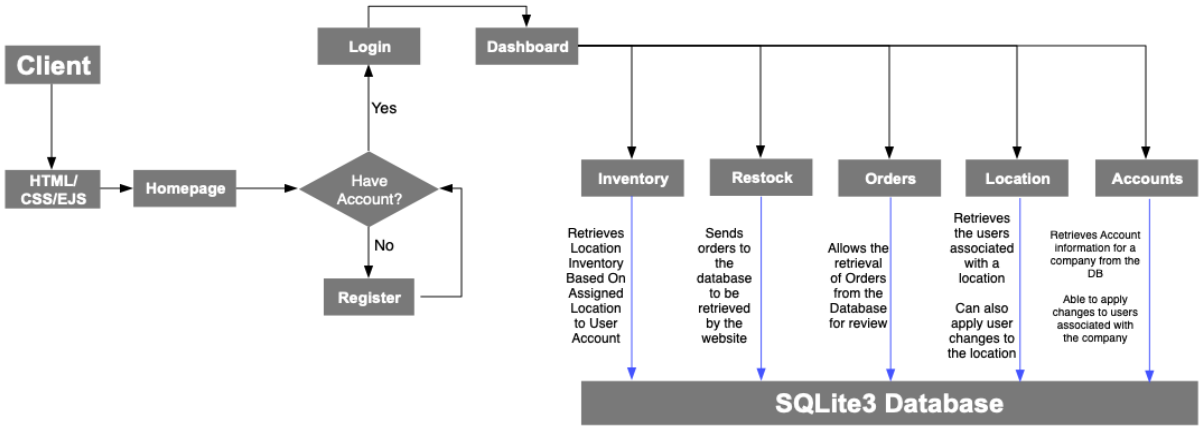
An illustration of the internal module level design of this application can be found in *Figure 2*. Of course, when a client first opens our webpage, they will first see the homepage of our server that EJS renders. In that homepage, the user will have information to look at, and you will come across a registration and login button. If the user does not have an account they will click register to be sent to a registration page. Then the user will be allowed to send a form and request your business to be registered by the Admins after fact checking the user's background information. When the user login, it will check if the user is registered with the correct name and password, and it will check if your name is under a registered business before you are sent to the appropriate dashboard.

The user will have different views on the dashboard according to their role, which will either be admin, owner, location manager, or warehouse user. According to the user's role and location on a their account, they can view that location's inventory. Owners and warehouse users will have a restock view in their dashboard, which allows them to send orders to the database to

add new items or refill them in their warehouse inventories. Location managers, warehouse users, and owners will be able to see the orders view in their dashboard, which allows the retrieval of orders from the database for review. Only owners and admins can use the location view, which allows the user to view, create, or delete locations and show users associated with locations. Owners can only see locations related to their company, while admins can see every company's locations. Only owners and admins have an account view in their dashboards. Owners can add accounts to their company, remove accounts from their company, or transfer accounts between locations. Admins are able to edit the data of any account. All information and action taken from inventory, restock, orders, location, and accounts views are taken from or sent to the database.

Figure 2

Internal Module Level Design.



Note: Blue lines represent database queries/modifications.

Level of Reusability in Design Patterns

Many parts of the application are created in module manners. For instance, all tables have space for their own unique ID, allowing for additional tables to be added in the future that can

more easily access pre-existing tables. Furthermore, because the front-end is utilizing EJS templating, various "partials" can be created for the project and called into the HTML for each page. Examples of this include creating partials of the Header and Footer of each page. Another use of these partials is pre-made body portions of the webpage, such as a "401: Unauthorized" insertion for the page. The use of Tailwind and CSS also allow the creation of classes for frontend aesthetics. These classes can be reused throughout the application for the consistency of design.

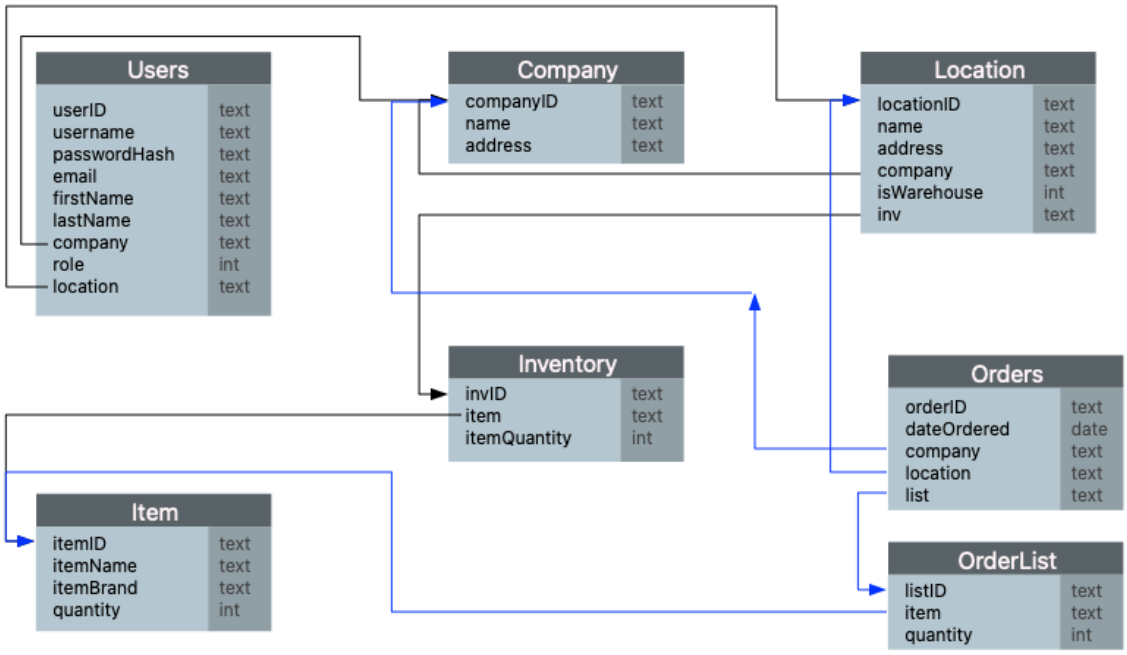
Database

The starting-schema for the database utilized in this project is seen in *figure 3*. This diagram contains the essentials for this database: the Users, Company, Location, Item, Inventory, Orders, and OrderList tables. Each of these tables contain their own unique IDs for easy identification. In practice, some of these are numerical in order, such as Item IDs, while others have unique IDs, such as users. Further into development, other factors will have unique IDs also, such as companies, locations, and warehouses. Tables within the schema, also contain references (otherwise known as Foreign Keys) to other tables. An example of this is how the Inventory table references the Location table, which allows an inventory to be identified based on what location it may belong to. In technicality, every company's location/warehouse inventory is stored in the same table (the Inventory table) with the item being corresponding to its location via its invID. This system works similarly to how the Orders and OrderList tables work. Each order has a unique ID, and a listID. The unique ID is assigned to the order itself, while the listID is used to reference the Order ID. This method of database management was utilized to help reduce columns in a table when listing items. Without this methodology, values

may be repeated more often throughout multiple columns in the database. Changes may also be made to the database throughout the remainder of the project to accommodate unexpected needs and changes that may occur. Database statements are made utilizing SQLite statements that are predefined in the Models table. Additionally, EJS and Better-SQlite3 have measures that can be enacted to prevent malicious use of the application's database, such as SQL injection; these measures will be utilized within the application.

Figure 3

OrderDistribution.db Schema



Note: The black lines represent standard foreign keys, while the blue lines represent foreign keys related to orders specifically. This was done to help visualize the different lines when crossing within the diagram.

Query Language

There are a variety of query languages that can be used for the database systems of web applications. For instance, XQuery, GraphQL, DMX, etc. Within those query languages, we utilize Structured Query Language (SQL) for the database of our application. Following the content of an article from the Business News Daily, SQL is *a programming language used to communicate with and manipulate databases.* Furthermore, the specification of SQL is described as that SQL is extremely accessible across many platforms as used in various businesses and other organizations and is a very powerful language for its portability, quicker query operations, and easier code implementation. The most important reason for using SQL as the query language for the project is because we will be utilizing the Relational Database Management System (RDBMS) of SQLite3 that can be operated with SQL.

User Interface

The User Interface (UI) for our application will be appearing as the following designs:

Figure 4

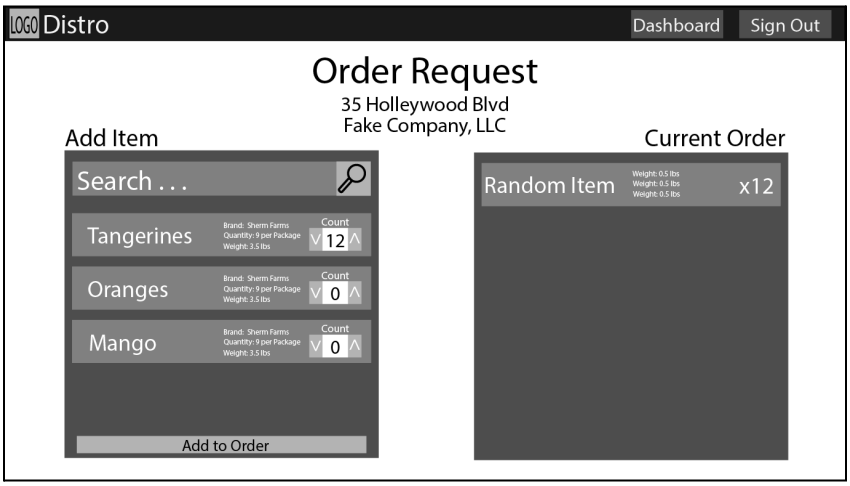
Application Logo, Name, and Motto

Figure 5 & Figure 6

Wireframe of original Dashboard and location panel.

Figure 7

Original Wireframe of Order Request Panel



Graphics

According to a website article from World Wide Web Consortium (W3C), the graphics for an application can be defined as "*Visual representations used on a Web page to enhance or enable the representation of an idea or feeling, in order to reach the Web site users"*, and for the visual representations used for our application Embedded JavaScript Templating (EJS) will be utilized to generate HyperText Markup Language (HTML) and Custom Style Sheets (CSS) files server side. EJS allows the graphics for the application to be controlled in a simpler manner because JavaScript code can be directly written into files that are similar to HTML (called EJS

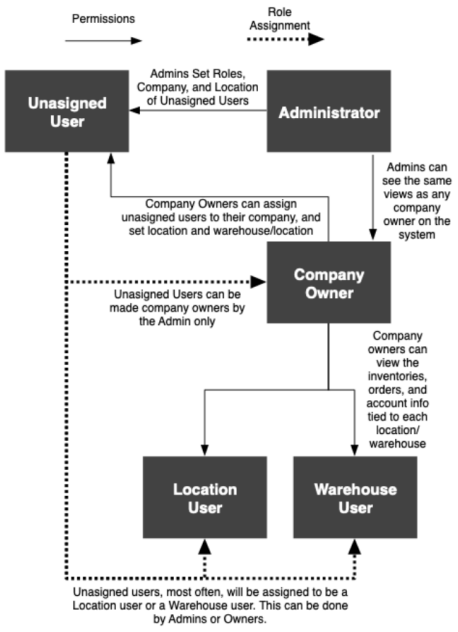
files). This functionality of EJS enables us to code less for generating HTML pages by templating HTML codes with relatively faster compilation and graphics rendering time. The application utilizes TailwindCSS as well. TailwindCSS is a utility-first CSS framework, which means it can be written inline in HTML. It is packed full of compiled CSS elements making it easy to code reactive HTML elements.

Function in a Distributed Environment

The application will be distributed to nodes such as null users, location managers, warehouse managers, owners, and administrators with a unique number for each role as 0, 1, 2, 3, and 4 respectively. Null users (called unassigned users in our system) are the users who have registered for the application and are waiting for roles to be assigned . From here, an Admin or Company Owner user can assign a Null User to a specific role (Company Owners can only assign the user to roles within their company). Null users can be assigned to Location users or Warehouse users by a Company owner, and an Admin can assign a null user to a Company Owner role, in the cases of new companies registering with the service. Location and Warehouse users function in similar manners, as they both handle inventories, however Location managers are capable or filing order requests to warehouses, and Warehouse managers are capable of reviewing and accepting or denying orders. Additionally, warehouse managers can add additional items into the database so that they can be added to their warehouse inventory. Company owners can oversee all inventories, whether it belongs to a warehouse or a location, as well as edit account information for those assigned to their company. Admins can do the same actions as a Company owner, however they are also capable of doing these actions to any company in the system.

Figure 8

Diagram and Table Describing User Roles and Hierarchy.



0	1	2	3	4
Unasigned User	**Location User**	**Warehouse User**	**Company Owner**	**Administrator**
- User just created account - Not assigned to a company or location yet. - Must be assigned by an Admin or Owner	- Can view the Inventory of the assigned location. - Can create orders for the assigned location.	- Can view the inventory of the assigned warehouse - Can view and accept orders recieved from locations	- Can view the inventory of all locations owned by company. - Can view all orders sent within company. - Can change user roles and location/ warehouse assignments. - Can create locations and warehouses.	- Can see all inventories and orders of ALL companies in the system. - Can assign users to companies. - Can create companies locations, and warehouses.

Application Testing Plan

Given that the target audience of this application is for businesses, we spoke with a local business owner who fell within the target market. This owner, Brandon King of IV Kings Coffee, runs a coffee shop with two locations, one of which also operates as a warehouse. When speaking with King, he stated that he would expect to be able to view the inventory of his business locations, the inventory of his warehouse, and be able to create accounts for his employees or locations to utilize to login to the software. He also said that it would be ideal if the users could create orders so that they could more easily transfer items between locations.

With this in mind, the biggest testing points of this application is that it contains the following features, and that they work in an expected manner: a user registration and login/logout system, a dashboard for all users to view once logged in, a manner for business owners to create warehouses and locations within the application, a manner for business owners to create items to be stored within locations and warehouses, a manner for business owners to create accounts for their employees to access the application, a manner that employees can see a view specific to their role in the business, and finally, a method that users can move items between locations and warehouses.

By utilizing the described plan, the Distro application has satisfied almost all requirements. The application is currently capable of registering Owners, who can then create Location and Warehouse users within their own company. This system component was also made more secure than the initial design by having the Owner create accounts internally, rather than allowing users to create their own accounts, and Owners accepting requests to join the company (which posed a potential security risk if the Owner accidentally accepted the wrong user into their company). Additionally, the Owner is capable of creating Locations and Warehouses, and viewing the inventory of each location and warehouse within their company.

There are features missing from the application, as of writing include the removal of items from an inventory (in a situation that does not include item transfer between locations or warehouses, such as purchase, expiration, or damage), the implementation of an “admin” role, the ability to change account information (emails, usernames, passwords), and, finally, the order functionality was not completed.

The removal of items was not implemented due to time constraints, and because it was deemed unnecessary to present a proof-of-concept. This could be implemented fairly easily

given enough time. This is also the reasoning for the implementation of the ability to change account variables (such as name, email, and password) to have not been completed.. The administration role was deemed unnecessary in the current state of the application. This is because there is very little to administer over that the Owner would not just handle themself. Finally, the Order system was not fully completed by the time of writing this report. Efforts will continue to be made after this report is submitted, however. There are several reasons that this was not fully completed, with the largest being that this project was fairly large, and the Orders system was dependent on every other system working consistently. However, because this may be incorporated into the project by time of presentation, the User Guides will be written under the assumption that the Ordering system is functional within the Distro application.

User Guide (Overview)

When you navigate to the Distro homepage, the user will see a brief introduction to our system as well as two buttons, Register and Login. The top navigation bar has six navigation buttons. The picture of our logo when clicked on will navigate the user to our landing page. It is situated at the top left corner of our app. The About page will take you to a page with a small quote from our fearless CEO of what our app set out to accomplish. The Our Team page will bring the user to a page that shows all of our partners photos, names, and job titles. The Contact page will show the user a way to get in contact with us, although it has fake information in it at the moment. Login will take the user to the login page which is standard to most applications where they will be able to fill out the information they have given before to login to the application. If they do not have an account yet, there is a button on the bottom of the login form as well as in the top right of the application. Register will bring them to an owner registration

page where they can then make an account. Upon completion of the registration, they will be sent to the login page to login to the application.

Once logged in, the user is met with a dashboard that has some added visuals that they have not had previously. While the top navigation looks similar to before, we take away the Register, Login, About, Our Team, and Contact pages and just give them a Home and Logout option, which are self explanatory. Added functionality now includes a side navigation bar that houses all of the usable tools provided in the application. Overview greets the user with three cards that contain personalized information, such as name, the role in the company, and the company name and address. All top-screen navigation bars appear the same for all users. The vertical left-side navigation bar is dynamic based on the user's roles. Owners will be able to access the Locations, Warehouses, Orders, and Accounts panels on the dashboard. Location managers and Warehouse managers will each be able to access an Inventory panel, and an orders panel.

User Guide (Inventory)

Only the warehouse manager and the location manager can access the Inventory panel on the dashboard. From this panel, a user can create items from this inventory by clicking on the "Add a New Item" button at the bottom of their inventory screen. Once on the Item Creation screen, the user is prompted with a field to input the Item name, brand, category, and quantity. Once the user has submitted this form, the item will now appear in the location or the warehouse's inventory.

User Guide (Locations)

Only the owner of a company can view the locations panel. This panel shows all locations assigned to the company, and also contains a link to view each inventory of the locations listed. There is also a button at the bottom of the page titled “Create New Location” that takes the user to a form that creates a new location. This form has inputs for Location name and address. After the creation of this location, it will now appear in the Locations list.

User Guide (Warehouse)

Only the owner of a company can view the warehouse panel. This panel shows all warehouses assigned to the company, and also contains a link to view each inventory of the warehouses listed. There is also a button at the bottom of the page titled “Create New Warehouse" that takes the user to a form that creates a new warehouse. This form has inputs for warehouse name and address. After the creation of this warehouse, it will now appear in the Warehouse list.

User Guide (Orders)

Owners and location managers can make orders for items that are inside a warehouse of their choosing. For the owner, there is a drop down list that will show all locations that they will choose, as well as a drop down list of warehouses. The owner will select one of each to make an order for the selected location from the selected warehouse. The location manager will have only their location to choose from and can select any warehouse. There is then a “Create an Order” button that will bring the user to another page where they will select items by increasing quantity. After the user selects how many of each item they want, they can then click on the

“Order” button. The order is then stored in the database and then the user will be brought back to the original order page. The order will be distributed into the location’s storage after it is received.

User Guide (Accounts)

The accounts panel is only accessible by owners, and is used to create new accounts for users within their company. The panel itself lists all of the accounts associated with the company, as well as displaying their role within the company. At the bottom of the list, there is a button titled “Create New Account”, which will direct the user to a form containing inputs for the new user’s username, email, password, first name, and last name. Once the new user is created, the owner will be redirected to the users list, and the new user will be able to log in to their account.

References

A*rgon2*. npm. (n.d.). Retrieved February 21, 2023, from https://www.npmjs.com/package/argon2

Brooks, C. (2023, February 21). *What is SQL?* Retrieved February 21, 2023, from https://www.businessnewsdaily.com/5804-what-is-sql.html

DigitalOcean. (n.d.). Retrieved February 21, 2023, from https://www.digitalocean.com/

GreaterWMS Team. (2021, January 28). *GreaterWMS*. GitHub. Retrieved April 28, 2023, from https://github.com/GreaterWMS/GreaterWMS

Guangjian, Z. (2016, April 23). *Flowchart designer*. Mac App Store. Retrieved February 21, 2023, from https://apps.apple.com/us/app/flowchart-designer/id1107185578?mt=12

Installation. Tailwind CSS. (n.d.). Retrieved February 21, 2023, from https://tailwindcss.com/docs/installation

NPM docs. npm Docs. (n.d.). Retrieved February 21, 2023, from https://docs.npmjs.com/

Redis Documentation. Redis. (n.d.). Retrieved February 21, 2023, from https://redis.io/docs/

W3C. Graphics - W3C. (n.d.). Retrieved February 21, 2023, from https://www.w3.org/standards/webdesign/graphics#:~:text=What%20are%20Graphics%3F,reach%20the%20Web%20site%20user