**Deliverable 1**

# Ice Cream Tracker Project Notebook

Neapolitan Solutions

Wednesday, September 22nd, 2021

## Section 2: Team Members:



Tyler Callison, Project Manager:



Tyler Padgett, Test Manager:



Dylan James, Design Manager,

Software Manager:



Drew Meade, Configuration Manager:



Mario Inzunza-Grivel, QA Manager:



Cole Kincaid, Requirements Manager:

## Section 3: Project Plan Contents

* Development Environment:
  + Operating System: Windows, MacOS (web application)
  + Language: Javascript, React.js
  + Database Application: Firebase
  + Rationale: Previous Experience, Easy to pick up, modular.
* ICSM Common Case:
  + Architected agile software development
    1. Rationale: small, simple, custom application for a small business, criticality to the business is medium
* Other Methodologies/Techniques:
  + Scrum/Sprints
    1. Rationale: Easy to keep track of information and allows for specific deadlines of specific tasks

## Sections 4 and 5: Basis of Development Estimate

4a) Size Estimate, using Application Point Sizing:

**Using object point counting** {client} [server]

* + Inventory Management System: 1 screen, [1 data table]
  + Order System:
    - Customer Profiles: 1 screen + edit modal, [1 data table]
    - Orders: 4 screens + create, edit modals
      * Products available screen, [1 data table,]
      * Individual product info screen (utilizes products available table) (may be unnecessary),
      * Order input/Order confirmation screen (maybe utilizes both customer info and current cart data tables), and
      * Order/Invoice/Truck tracking screen + [1 data table]
  + Trouble Tickets:
    - 1 screen for customers + create modal + {1 data table}
    - 1 to see all tickets + [1 data table]
    - 1 to view ticket + [1 data table]
  + Changelog: 1 screen, + [1 data table]
  + TOTAL: ~9 Screens, [6 data tables] + {2 data tables}

4b) Schedule:

* + [Project Schedule](https://docs.google.com/spreadsheets/d/1QeU2sPi6dW89G62PEPwvEtTfkLfFjAGA5nVIOWRuGJE/edit?usp=sharing)

5) Labour Hours Estimate:

* 160 total man hours (40 per 3rd level task)

5c) Defects

* Requirements defects: Requirement 44 did not account for incoming shipments of new stock.

## Section 6: Risk identification

Previous risks:

1. Requirement analysis:
   1. While the chosen lifecycle model allows for some flexibility regarding the requirements. Icetrack’s core functionality must be well understood to write a consistent project notebook and code base.

Current risks:

1. Developer inexperience:
   1. React is a new framework for a significant portion of the development team.
   2. Firebase database system is a new tool for the development team.
2. Sprint schedule:
   1. Project schedule has already seen significant schedule adjustments in early stages of development.

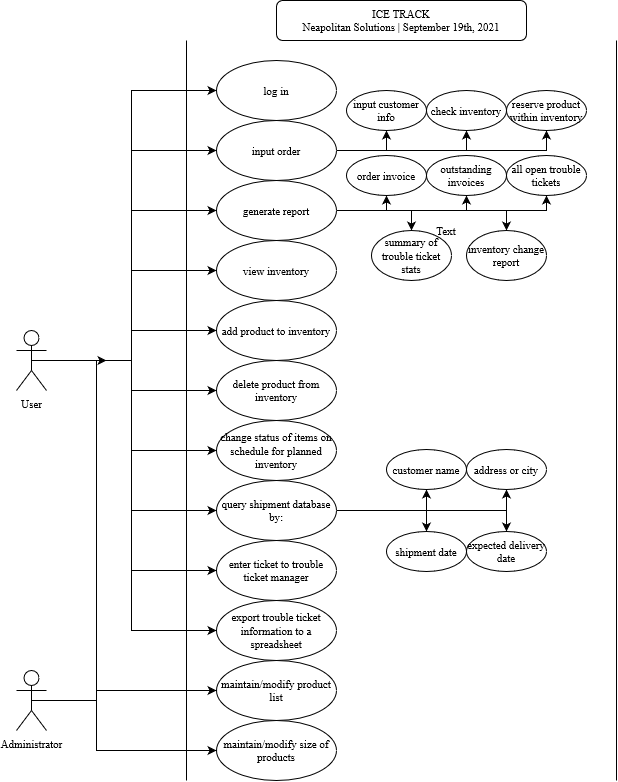
## Section 7: Requirements

[*Requirement Database using Confluence by Atlassian*](https://tylercallison.atlassian.net/l/c/gGgRLwhP)

Ice Track is a web application in development that strives to solve shipment tracking and inventory management within Tom and Adam’s Ice Cream Company. The Ice Track software system is broken down into four key subsystems; order entry, inventory management, shipment tracking, and trouble ticket management.

The order entry subsystem helps by automating the entry of orders into the Companies order database. It will also store the data of customers so that customers can be associated with all of their orders which will allow for easier management of present and past orders from customers. The inventory management subsystem’s job is to track the ice cream inventory for Tom and Adam’s Ice Cream Company by allowing users to add, reserve, and remove products from their inventory. The shipment tracking subsystem keeps track of the status of all shipments within the database including information such as order date, number of boxes within the shipment, if the order has shipped yet, its expected delivery date, its actual delivery date, and its method of delivery. The last of the four systems is in charge of dealing with all the problems that can arise throughout the ordering, inventory management, and tracking that occurs within the system. Both reported internal production/order fulfillment/shipping problems and reported customer-facing problems are taken care of by the Trouble Ticket Management subsystem and the subsystem allows for easy resolvement of issues. These four subsystems are intertwined to deliver a software system that allows for a seamless overall management of Tom and Adam’s Ice Cream Company’s shipments and inventory.

The Ice Track software system will be able to handle all of the order, customer, and issue tracking data. It will also be able to handle multiple users simultaneously, making sure that whenever a change is made in the system all instances are updated/insync. **When performing actions on the application it will respond within 5 seconds**. The system will also be able to restore user information to its previous state if the user wants to revert previous changes.



Total of 60 requirements after 1st analysis

* + General Requirements (22 total requirements going into top level design)
    - Functional (13 functional requirements)
      * Gui design is based on a system of multiple movable and resizable windows
      * roles/privileges shall be implemented to limit the actions that users can make as well as limiting the data they can access
      * All transactions shall be logged and labeled with the user’s name as well as the date and time of the action
      * If the user idles for 5 minutes the system will auto logout the user
      * ***Validations of Inputs***
      * ***Data Structure***
      * Shall allow for users to search for names within the system using the [standard wildcard character](https://apus.libanswers.com/faq/2235)s
      * User shall be able to print any display presented and shall also be able to previewed before the user prints (basically control + p)
        1. Printable reports shall be available to the user as a whole as well as in parts
      * Users shall be provided with an on-line help function
        1. Shall be context sensitive
        2. Provide detailed answers to most questions
        3. Includes help for:

Overall system functions

Transcription descriptions

Screen/ window descriptions

Data fields

* + - * 1. Module/Screen based help (hints/helpful tips on the functions and such on each module or screen)
        2. While the cursor hovers over a data field the user will be given the requirements of said field (ex: if field is mandatory)
      * System shall be menu driven
        1. Standard movement within the system will be screen to screen without having to return to a main menu
        2. User shall be able to access any module with ease via minimal sub-menuing
      * If error occurs when entering information users will be prompted with error messages
        1. Reason for rejection of info
        2. Refers to the minimum requirements for that specific entry field
      * Users shall be able to return to already processed information and are able to add optional information as well as make modifications to prior information
        1. Shall not allow the user to modify previous information that create inconsistent conditions

Ex: “changing the ship address after the product has already shipped”

* + - * At any time, users shall be able to cancel actions before the completion of that action
    - Non-Functional (9 non-functional requirements)
      * Application should respond within 5 seconds
      * System should be able to support multiple users, simultaneously
      * Database must be accessible over lan, but we are thinking of using firebase cloud based solution
      * Security should be [object-based](https://www.ibm.com/docs/en/cognos-analytics/11.1.0?topic=security-object)
      * System shall allow for users to input hyphenated names in any name section(first,middle,last)
      * information maintained by the system shall only have to be entered once
      * Must be modular and allow be additions, upgrades or replacements of modules in future builds
      * **Processing Block Until Error Corrected**
      * The system shall be able to restore user information to its previous state after an action is taken (backtracking/before making a change the whole system is backed up?)
  + Order Entry Requirements (10 total requirements going into top level design)
    - Functional (7 functional requirements)
      * Shall maintain customer information
        1. Customer name
        2. Shipping address
        3. Billing address
        4. Customer status (prefered, ok, shaky)
      * Products list and size of products shall be able to be maintained/modified by an administrator
      * System shall check inventory to determine the availability of an item
        1. If the amount in not available then a message will display saying out of stock and when it is expected to be back in stock
      * System shall reserve the desired inventory
        1. If there is insufficient stock then the user will be able to send a message to the trouble ticket management subsystem, notifying that an order is not able to be taken because a certain item is out of stock
      * User shall be able to print out a copy of the order (invoice) that can be sent to the customer (maybe set up some sort of email thing, keep it paperless?)
      * User shall be able to enter into the system the dates of when payments for orders were received
      * User shall be able to print out a report that contains all outstanding invoices
        1. In order by customer name
        2. Including

Amount on each invoice

Date payment is due

Number of days till overdue

* + - Non-Functional (3 functional requirements)
      * System Order limitations
        1. Prefered customers have no limit on the order size
        2. Ok customers are limited to $3000 of goods
        3. Shaky customers are limited to $500 of goods
      * Order Form layout
        1. Multiple entries (item lines) including

Flavor

Size

Quantity

Expected ship date

Cost of item line

* + - * 1. Displays

Customer name

Shipping address

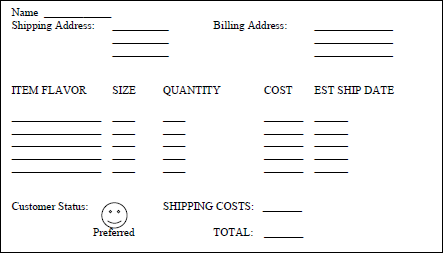
Billing address

Type of shipping that is desired by the customer

Desired receipt date

Shipping cost

Total cost of order

* + - * 
        1. Order form^
      * Cancellation/modification of an order that has shipped/partially shipped will result in the customer being billed for all of the shipping charges
  + Inventory Management Requirements (11 total requirements going into top level design)
    - Functional (11 functional requirements)
      * Inventory tracking
        1. Shall track by flavor and size
        2. Shall maintain a list of sizes that are supported by Tom and Adam’s Ice Cream Company
      * Shall have a list of all available inventory
      * Shall have a list of all inventory that is present but reserved for orders that have not been shipped
      * Shall maintain a schedule for the incoming planned inventory products
      * User shall be able to add products to the inventory
      * User shall be able to change the status of items that on the schedule of planned inventory from “planned” to “actual” inventory
      * User shall be able to delete products from the inventory
      * System shall maintain a log of keeps track of what user adds or deletes from the inventory with each transaction
      * System shall maintain information on why an item was removed from the current inventory
        1. Shipped (including the order number)
        2. Defective product
        3. Product Spoilage
      * System shall send a notification to the trouble ticket management system each time an product is removed from the current inventory due to defect or spoilage
      * User shall be able to generate a report that shows the changes to the ice cream inventory over a specified period of time
        1. Week
        2. Month
        3. Year
    - Non-Functional (0 non-functional requirements)
  + Shipment Tracking Requirements (9 total requirements going into top level design)
    - Functional (4 functional requirements)
      * System shall track the status of all shipments for Tom and Adam’s Ice Cream
        1. Date order was shipped
        2. Number of boxes associated with order
        3. If the full order or partial order was shipped
        4. Expected delivery date
        5. Actual date order was delivered
        6. Method of delivery
      * User shall be allowed to query shipment/s by
        1. Customer name
        2. Destination

Address or city

* + - * 1. Shipment date
        2. Expected delivery date
      * System shall send notification to Trouble Ticket Management System when a package is reported lost or damaged
      * User shall be able to print out shipping status information reports that include
        1. All active orders ready to ship
        2. All active orders in the system that are in the shipment process but are not delivered yet
        3. Summary of all product lost or damaged during a prespecified time period
    - Non-Functional (5 non-functional requirements)
      * Shipping methods shall be a user-maintained list including methods based on shipment location
      * Authorized Shipping Vendors shall be a user-maintained listed that includes
        1. Types shipping
        2. Geographic region
        3. Shipping rate
        4. Vendor rating

Preferred

Ok

Poor

* + - * System shall maintain information on lost or damaged items including
        1. Identification of shipment packages
        2. Problem with the package
        3. The shipping vendor
        4. Cost of lost or damaged package
      * System shall maintain shipping information on reshipped products due to being lost or damaged
      * System shall keep shipping status information on shipments from at least the last 30 days after delivery is complete or payment is received, whichever is later
  + Trouble Ticket Management Requirements (8 total requirements going into top level design)
    - Functional (6 functional requirements)
      * System shall accept tickets from the order entry, inventory management and shipment tracking system and adds them to the problem tracking database
      * User shall be able to enter trouble ticket information manually into the problem tracking database
      * User shall be able to query the problem tracking database by
        1. Customer name
        2. Type of problem
        3. Problem status
        4. Date problem reported
        5. Date problem resolved
      * User shall be able to request a report that summarizes all open trouble tickets by problem type, date reported, or date closed
      * User shall be able to request a statistical summary report, over a specified time, of
        1. # of problems reported
        2. Average time to close
        3. Average # of open problems per day
        4. Average # of problems being worked on per day
      * User shall be able to export trouble ticket information to a format that can be imported to a spreadsheet for further analysis and reporting
    - Non-Functional (2 non-functional requirements)
      * Maintain information on both
        1. Internal production/order fulfillment/shipping problems

Date problem reported

Source: Name of user reporting problem or name of user associated with process reporting the problem

Date problem detected

Date problem resolved

Type of problem

Description of problem

Problem status (open, being worked, closed)

Problem resolution

* + - * 1. Customer-facing problems

Date problem reported

Source: Name of customer reporting problem

Date problem detected

Date problem resolved

Type of problem

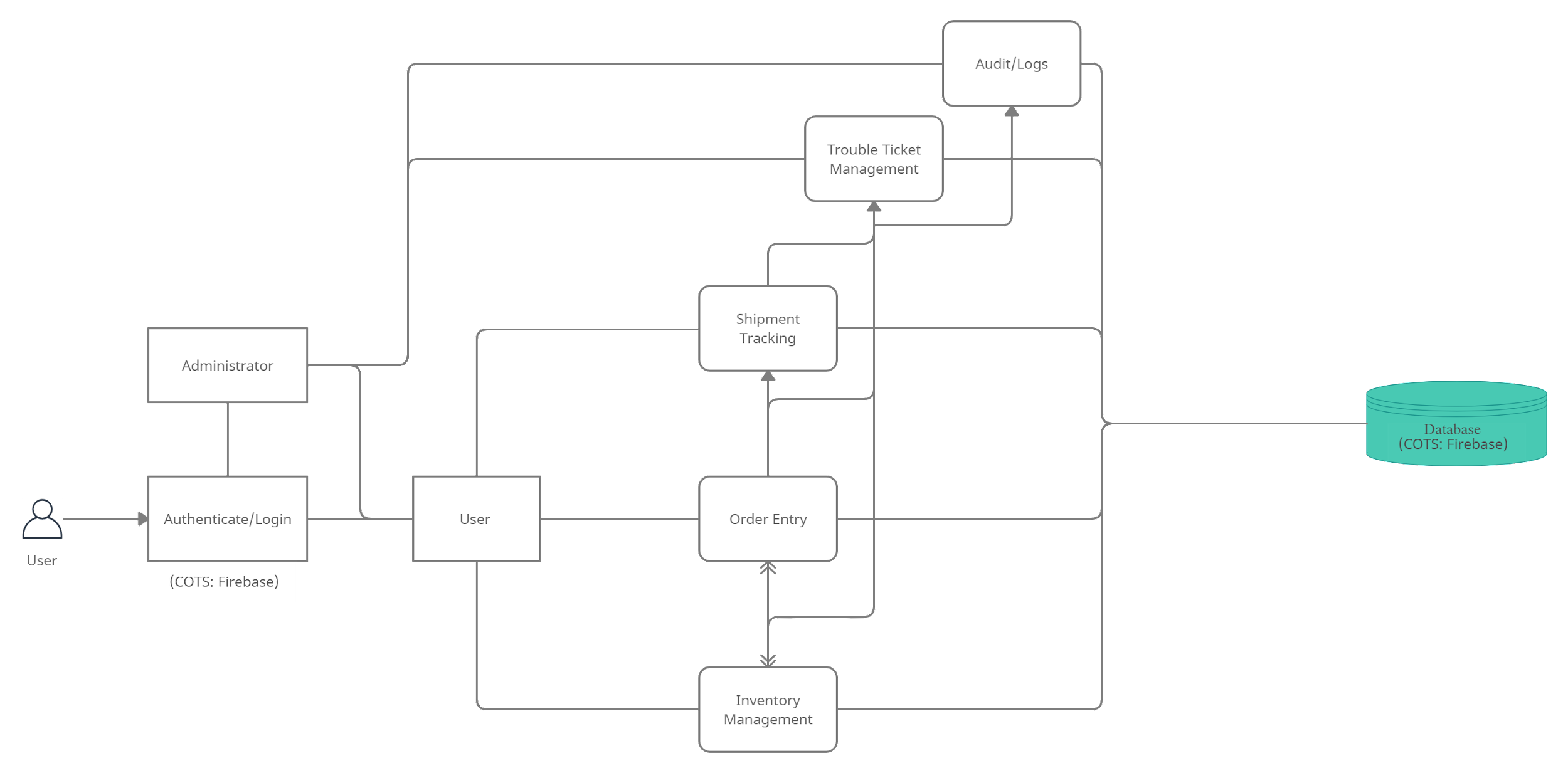
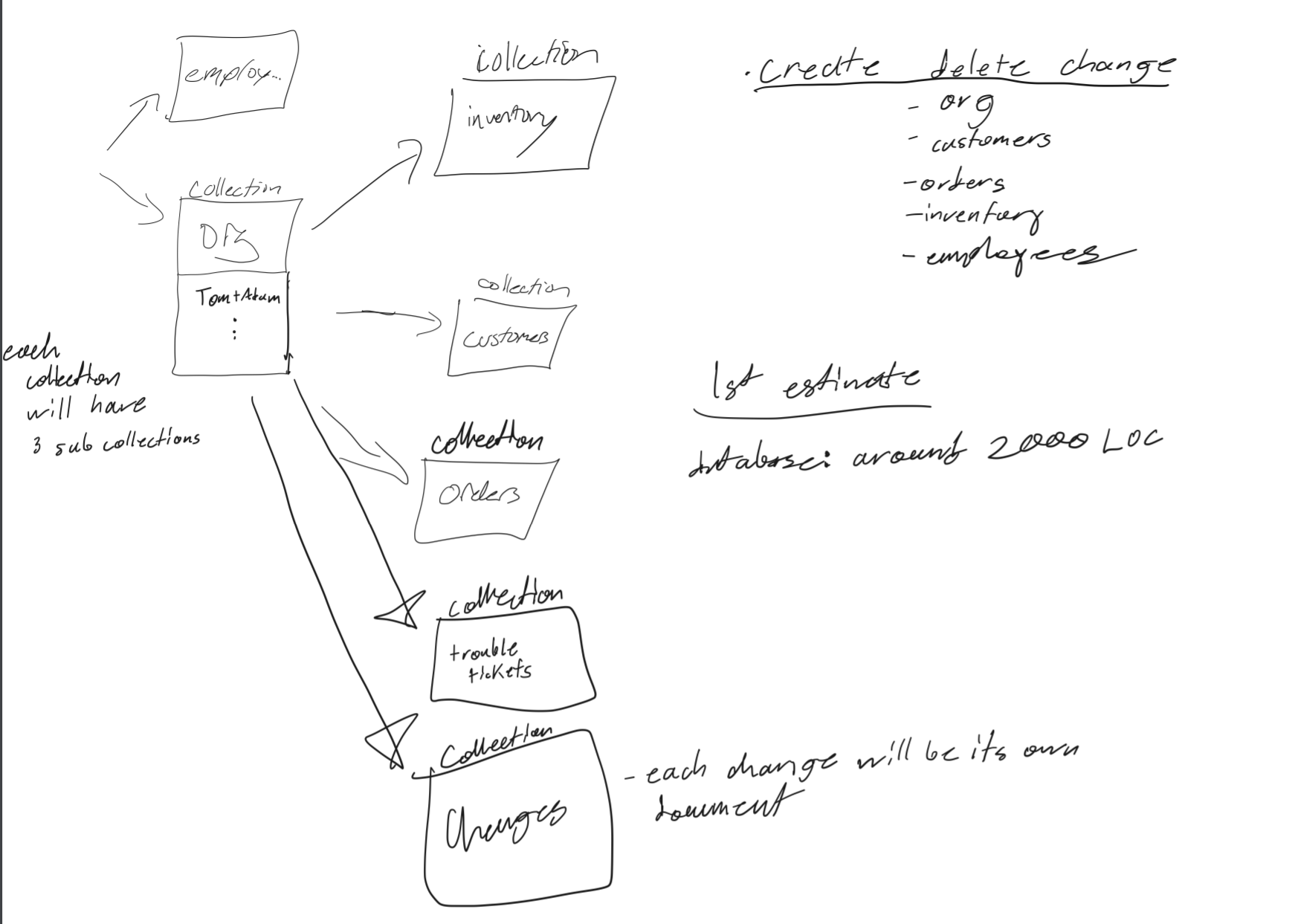
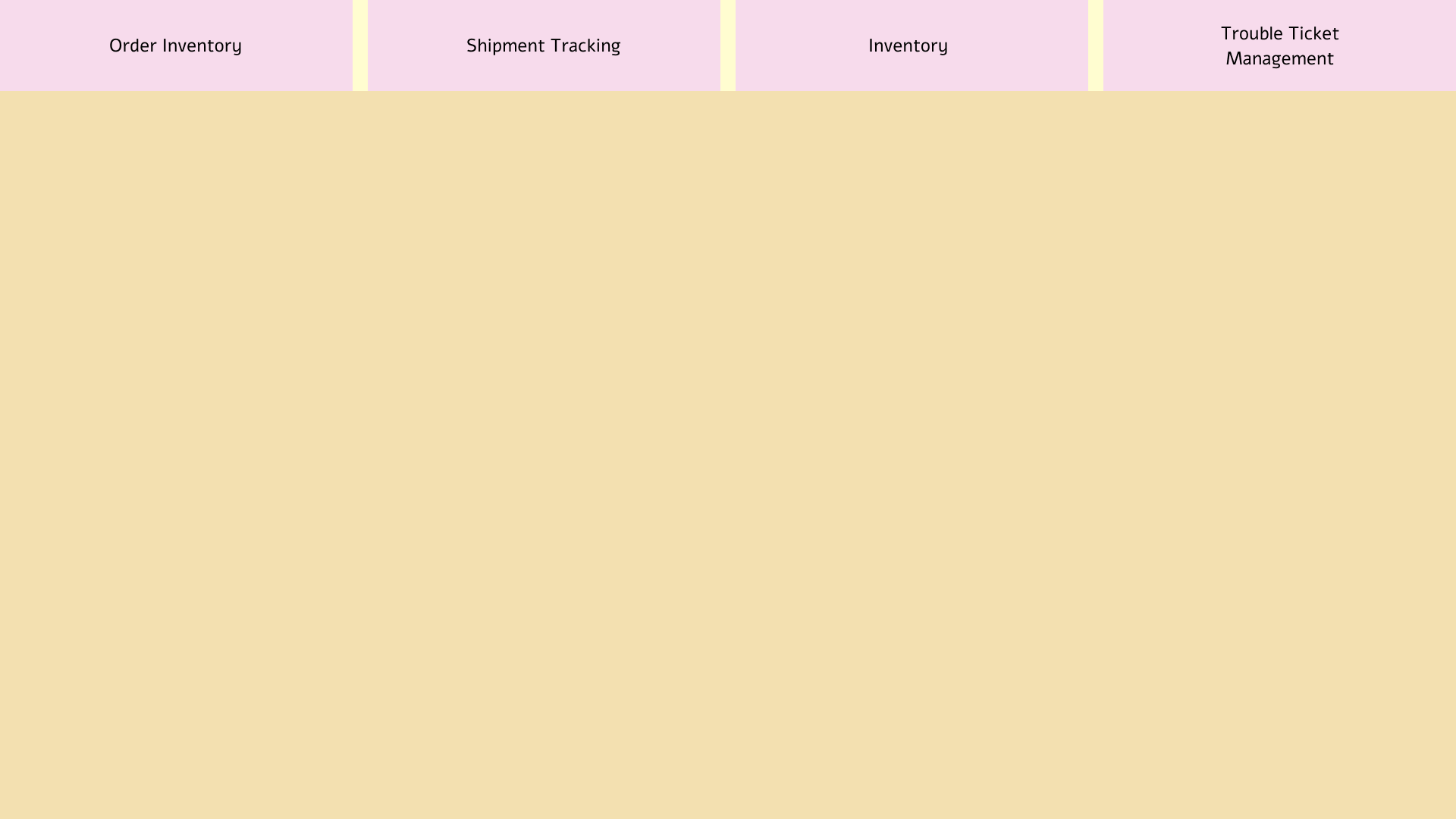
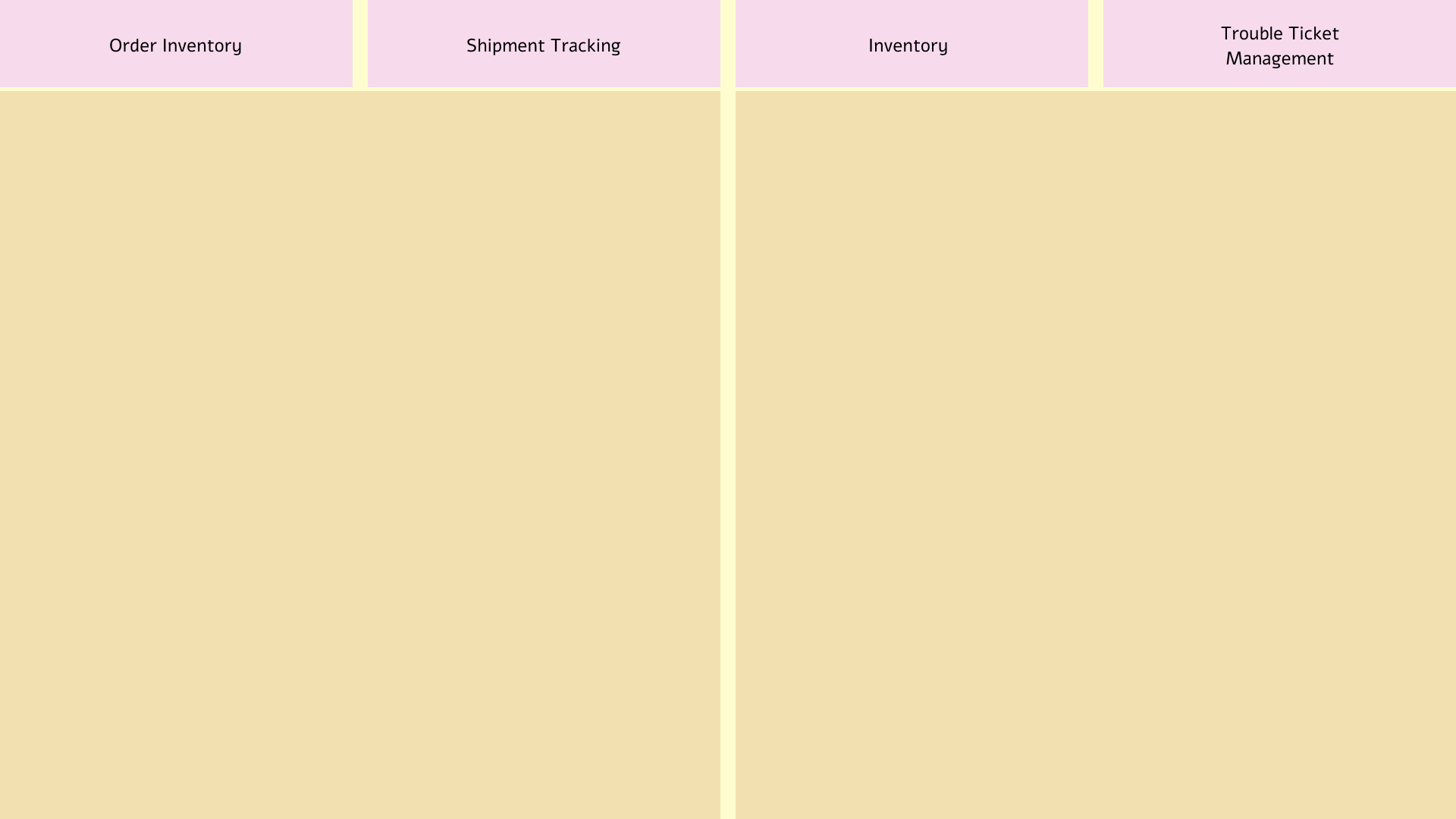
Description of problem

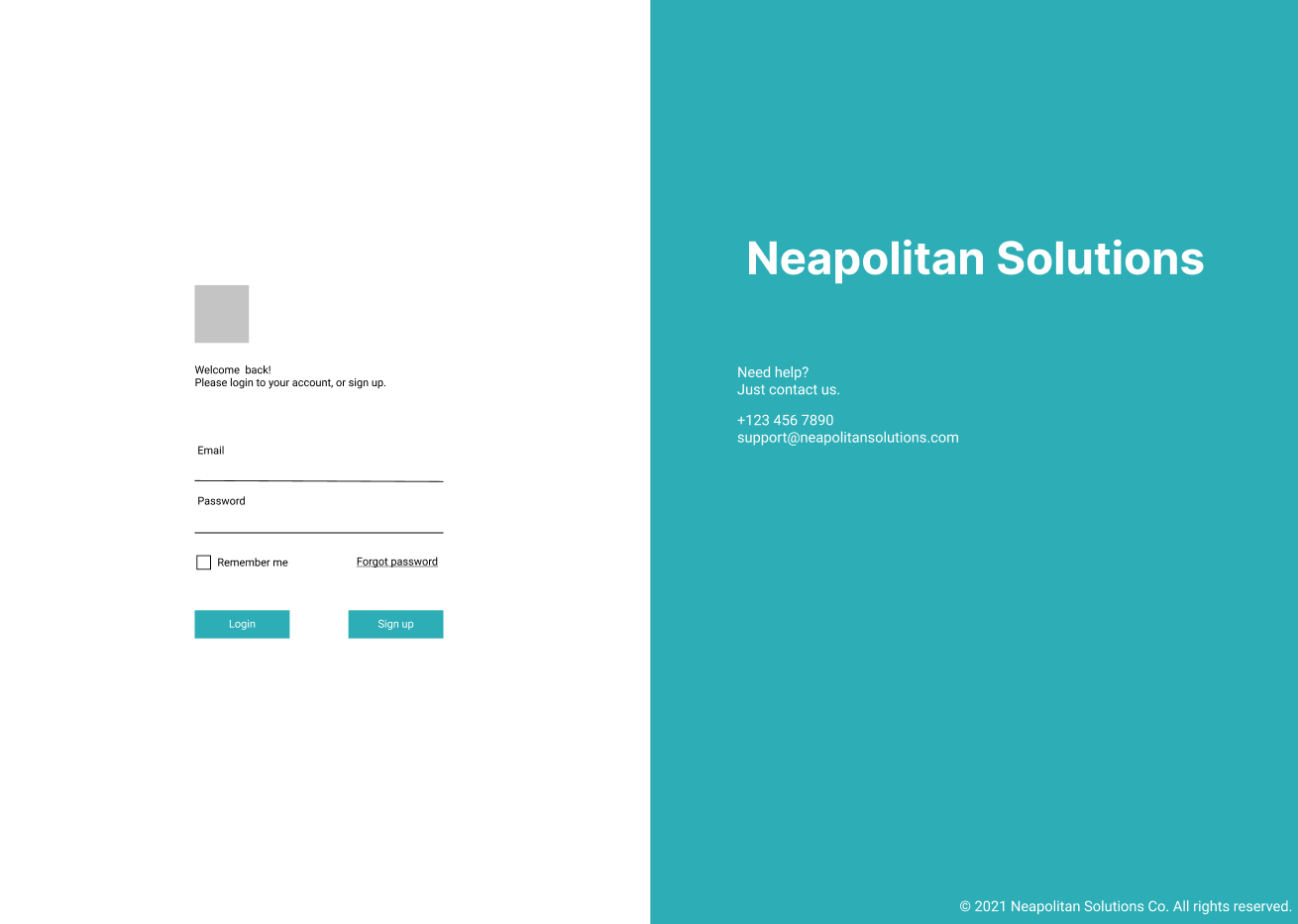
Problem status (open, being worked, closed)

Problem resolution

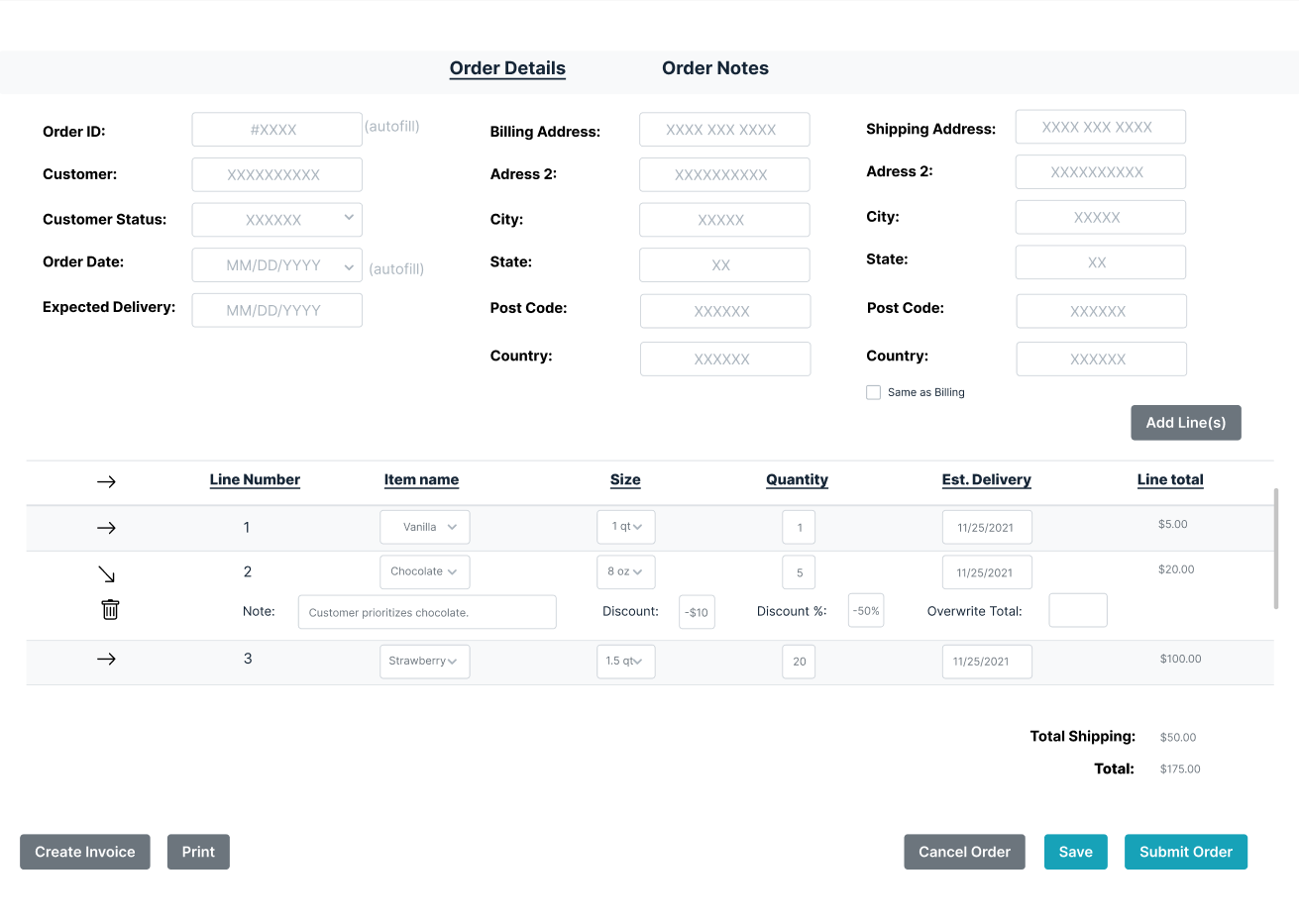
* + - * Typical Internal/Customer Problems shall be a user-maintained list, and -will include typical issues as well as actions to take to resolve them

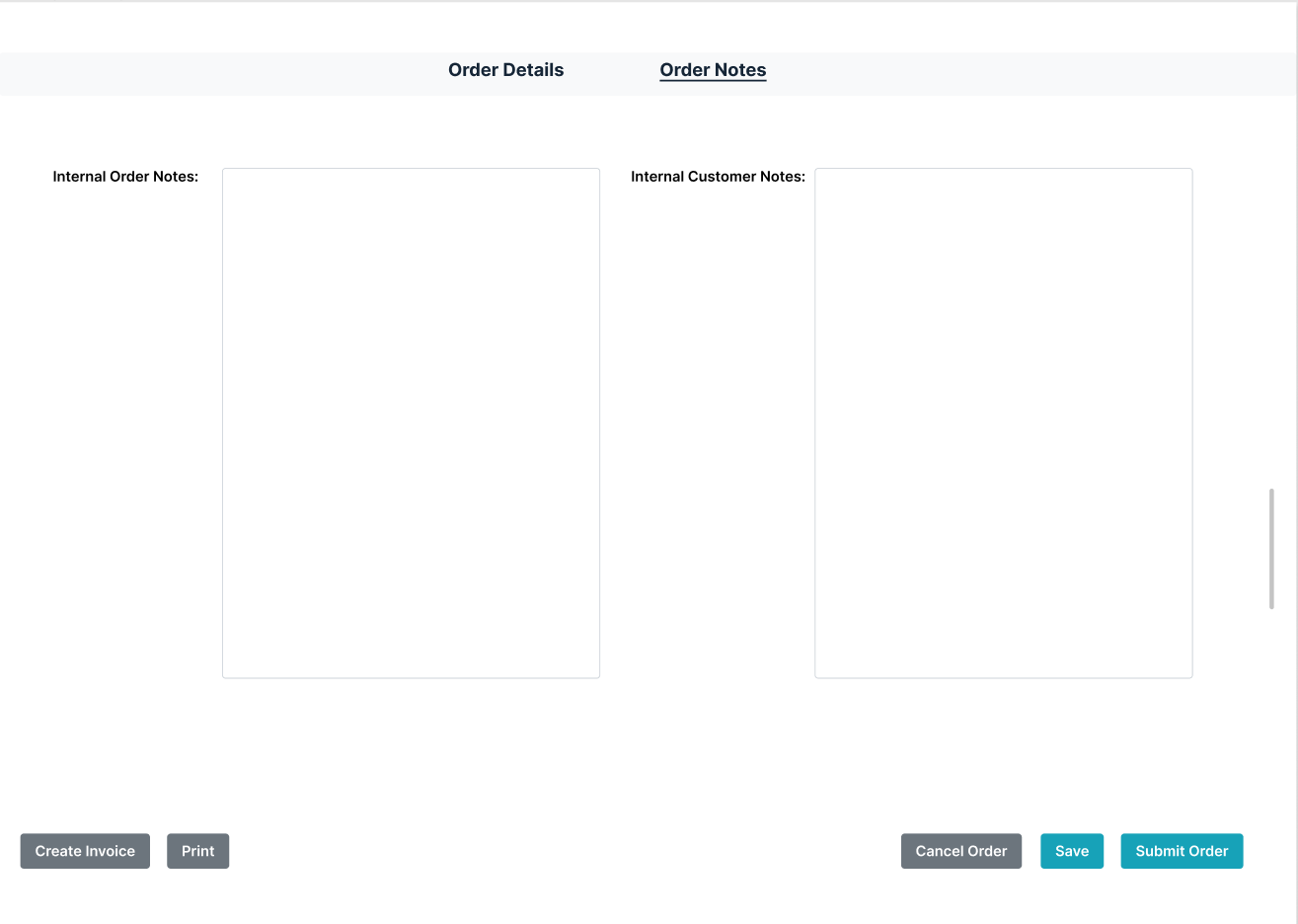
## Section 8: Design

1. Top level architecture design (This will be updated in future deliverables):
   1. Pipes and filter design:
   2. 
2. Alternatives investigated and rationale for selection
   1. Event based is a possible architecture alternative and layered architecture.
   2. We choose a pipe and filter architecture since users will have the ability to modify/delete/commit data from various data sets.
3. Database/data structure design:
   1. One-to-many and many-to-many relational entities depending on operation.
   2. 
4. User interface prototype
   1. The interface includes modular panels that allow for resizing and movability; grid snapping for common arrangements of panels is also included. A top bar that spawns panels or activates the corresponding panel. Or if the user desires, a classic single screen view, and traverse through the screens via the top menu bar.
   2.  
   3. 
5. Detailed Design:
   1. Login Screen:

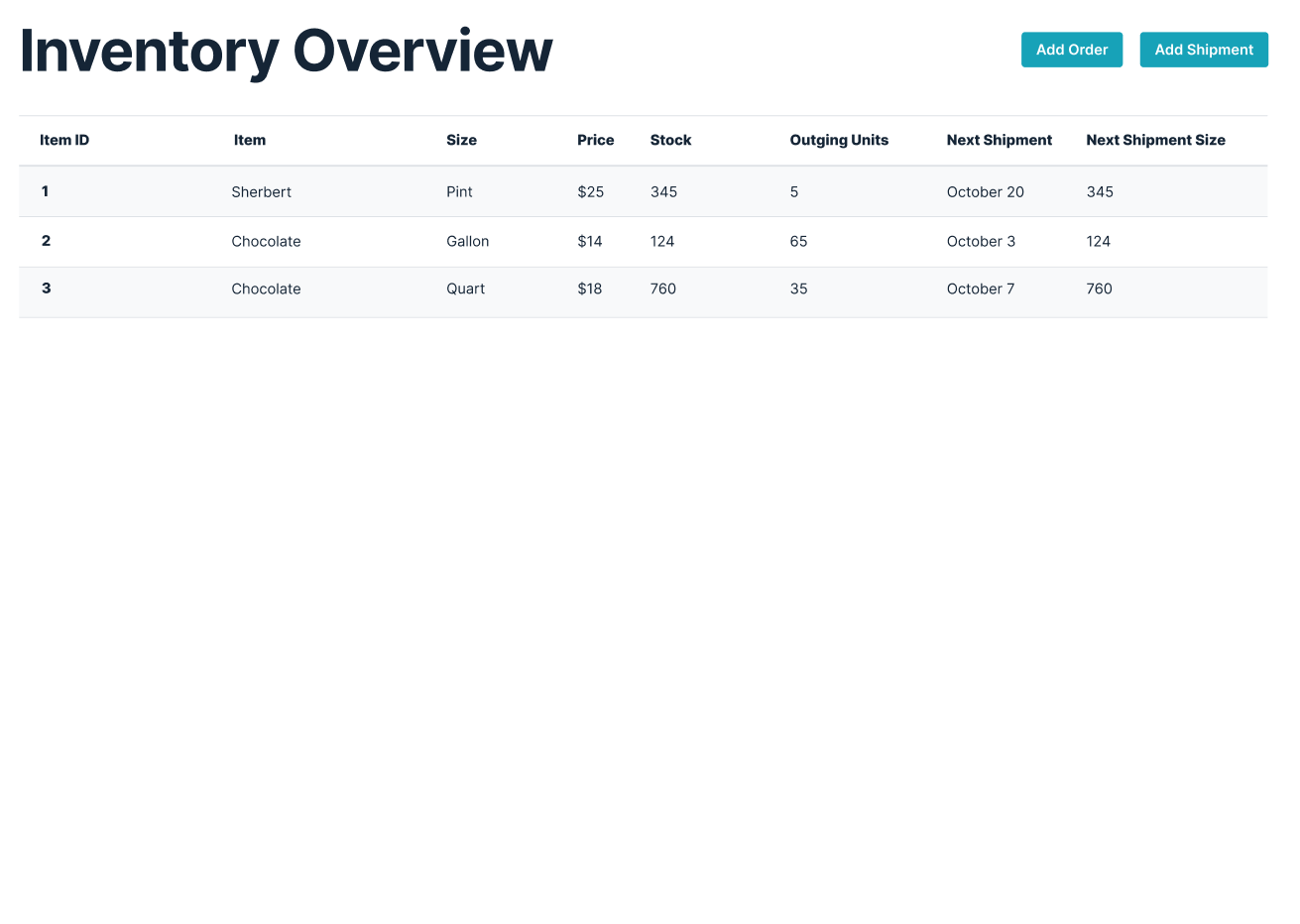


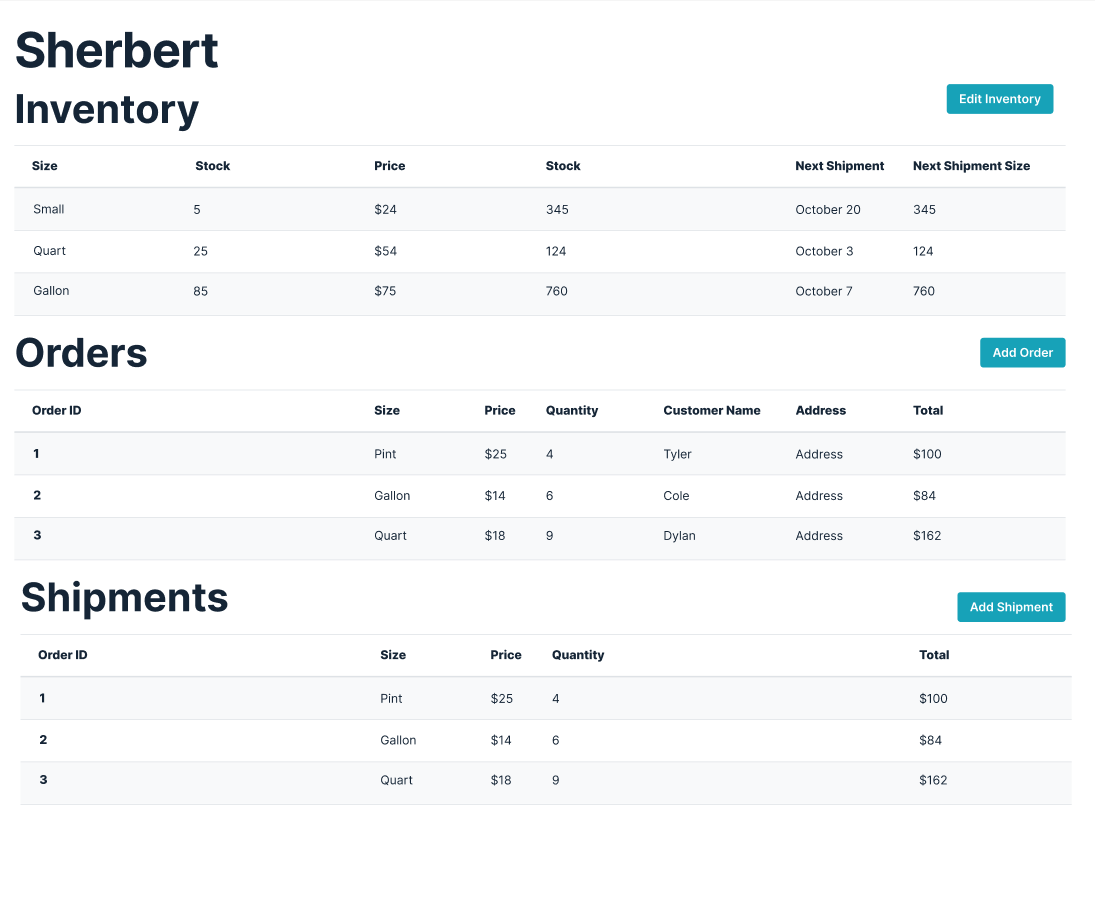
* 1. Order Entry:

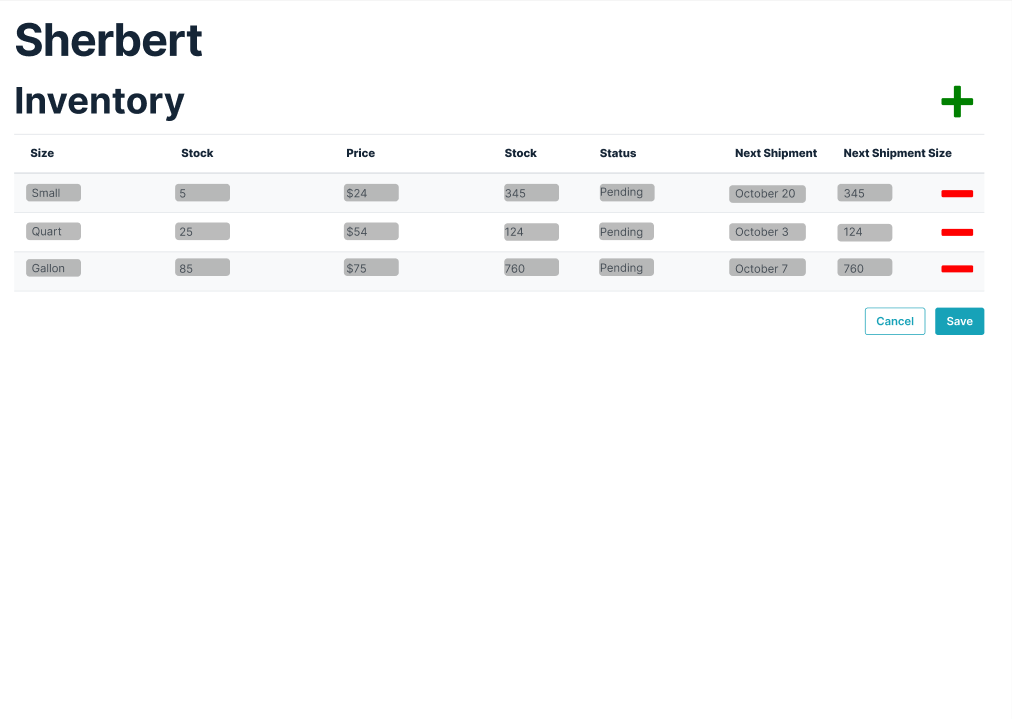




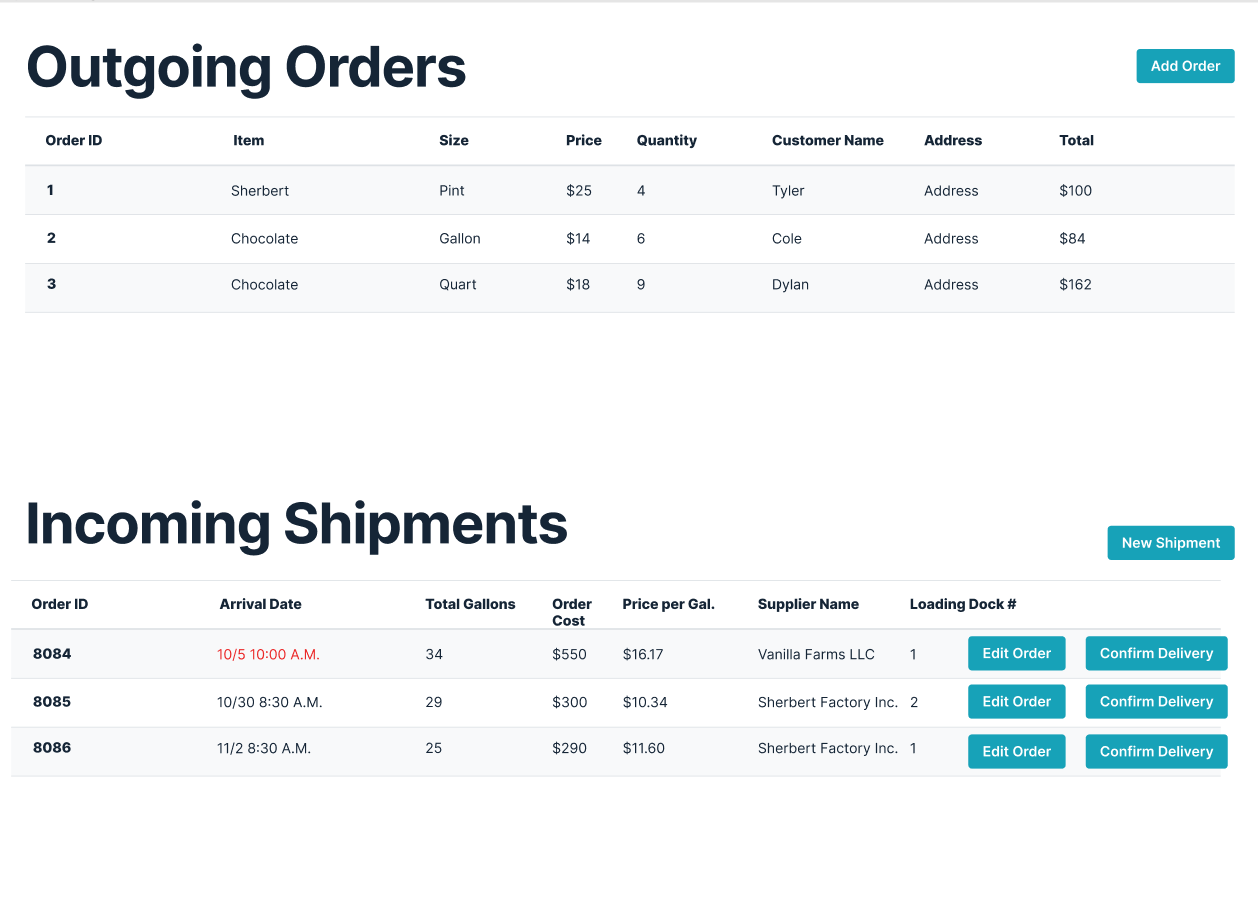
* 1. Inventory:



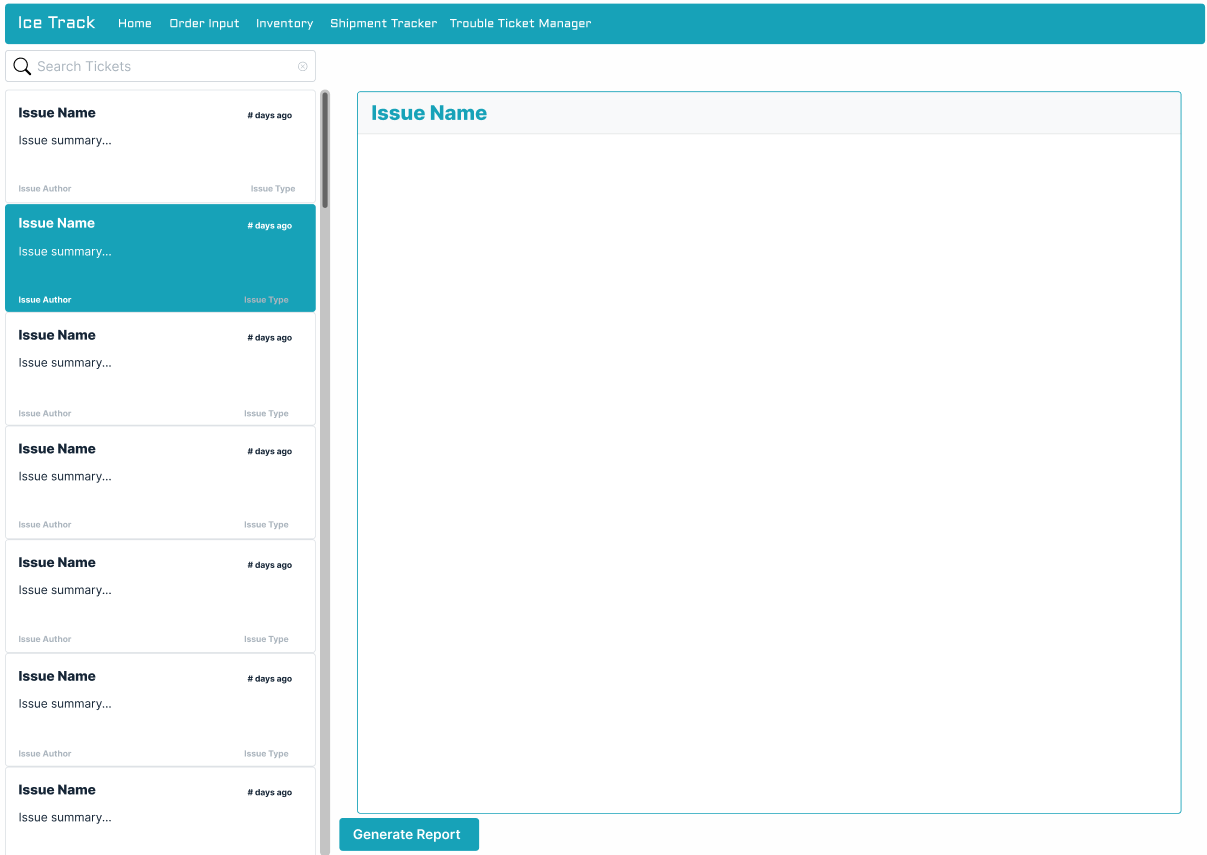




* 1. Shipment Tracking



* 1. Trouble Ticket:



1. Informal notes
   1. We set a default grid layout of important subsystems and the user can customize the view based on their needs (rearrangeable grid view)
   2. Here is the the design link: https://www.figma.com/file/xxo2QabDRJRBiJC8hKRu3N/Ice-Track-UI?node-id=0%3A1
2. Design review inspection notes
   1. Normal employees cannot view trouble ticket management system
   2. Ensure modularity and movable/resizable windows. Although all aspects will be respectively responsive.

## Section 9: Code and Test

Code Review Policy

1. Code / Unit Test / Integration Test review/inspection notes (optional\*)

1.1. Code Convention Checks

| **Item number** | **Check description** | **Observation** |
| --- | --- | --- |
| 1.1.1 | Is there an established code convention / style guide? |  |
| 1.1.2 | Does the code respect the project coding conventions and stylings? |  |
| 1.1.3 | Are the variables declared properly commented? |  |
| 1.1.4 | Are all functions, methods, and classes documented? |  |
| 1.1.5 | Are function, method, and class behaviors accurately described in their documentation? |  |
| 1.1.6 | Are a function’s input and output parameters documented in the function header? |  |
| 1.1.7 | Are obscure code blocks commented? |  |
| 1.1.8 | Is commented out code explained? |  |
| 1.1.9. | Are comment blocks consistent in length? |  |
| 1.1.10. | Is each developer creating a unit test for functions and classes they’ve written? |  |

1.2. Error Handling Checks

| **Item number** | **Check description** | **Observation** |
| --- | --- | --- |
| 1.2.1. | Are a function’s edge cases safely handled to prevent runtime errors? |  |
| 1.2.2. | Do error messages describe exactly how the error occurred? |  |
| 1.2.3. | Are resources released in all error paths? |  |
| 1.2.4. | Is the function caller notified with appropriate error code on error? |  |
| 1.2.5. | Has the error handling code been tested for complete test coverage? |  |

1.3. Software Quality Review

| **Item number** | **Check description** | **Observation** |
| --- | --- | --- |
| 1.3.1. | Is the team following a comprehensive software quality model (McCall model, Boehm’s model, ISO/IEC 25010 model)? |  |

1. Unit test plan

2.1. Automated Test Implementation

| **Item number** | **Check description** | **Observation** |
| --- | --- | --- |
| 2.1.1. | Is the UUT stable enough to be subjected to automated tests? |  |
| 2.1.2. | Is a specific Test Automation Framework (Linear Scripting, Test Library Architecture Framework, Data-Driven Testing Framework, Keyword-Driven or Table-Driven Testing Framework, Hybrid Test Automation Framework) implemented for the UUT? If so, is it documented? |  |
| 2.1.2. | Are only repetitive tests automated? |  |
| 2.1.3. | Are regression tests included in automated tests? |  |
| 2.1.4. | Are tasks that require setup of a known environment included in automated tests? |  |
| 2.1.5. | Can the additional time required to develop and verify an automated test be recouped? |  |

2.2. Test Cases management

| **Item number** | **Check description** | **Observation** |
| --- | --- | --- |
| 2.2.1. | Are the test cases documented in an accompanying artifact? |  |
| 2.2.2. | Are Test Cases simple and transparent? |  |
| 2.2.3. | Are Test Cases created with End User in mind? |  |
| 2.2.4. | Is Test Case repetition avoided? |  |
| 2.2.5. | Were Test Cases developed by relying on the UUT documentation? |  |
| 2.2.6. | Do a UUT’s Test Cases check all software requirements specified in the UUT’s documentation? |  |
| 2.2.7. | Are Test Cases assigned an ID to ease defect tracking at a later stage? |  |
| 2.2.8. | Do all test cases return the Test Environment to the pretest state, if applicable? |  |
| 2.2.9. | Are test cases assigned a risk with an associated contingency plan? |  |

1. Unit test/retest results

| **Item number** | **Check description** | **Observations** |
| --- | --- | --- |
| 3.1.1. | Has the unit test data required for each test been identified? |  |
| 3.1.2. | Have the expected unit test outputs been identified? |  |
| 3.1.3. | Has the team specified how the test data is identified, maintained, and version-controlled? |  |
| 3.1.4. | Have the procedures for reporting errors, test results, and reworking and retesting programs been documented? |  |
| 3.1.5. | Are the unit test’s preconditions set up? |  |
| 3.1.6. | Is there a procedure in place for reporting errors, test results, and reworking and retesting programs? |  |
| 3.1.7. | Has complete Test Coverage been achieved for a UUT using the documented unit tests? |  |
| 3.1.8. | Is there a defect management policy in place? |  |
| 3.1.9. | Are errors assigned different severity levels (Critical, High, Medium, Low, Info)? |  |
| 3.1.10. | When a change is being migrated to the test environment, has the tet manager been notified in advance? |  |
| 3.1.10. |  |  |

1. Integration test plan

| **Item number** | **Check description** | **Observations** |
| --- | --- | --- |
| 4.1.1. | Are the interface test’s preconditions set up? |  |
| 4.1.2. | Has complete Test Coverage been achieved for a UUT using the documented unit tests? |  |
| 4.1.3. | Has the integration strategy (big-band integration, top-down integration, bottom up integration, hybrid integration) been taken into account when designing the interface test plan? |  |
| 4.1.4. | Have all module relationships been tested? |  |
| 4.1.5. | Have the modules under test already been unit tested? |  |

1. Integration test/retest results

| **Item number** | **Check description** | **Observations** |
| --- | --- | --- |
| 5.1.1. | Has the interface test data required for each test been identified? |  |
| 5.1.2. | Have the expected interface test outputs been identified? |  |
| 5.1.3. | Does the interface testing validate the data at the field level? |  |
| 5.1.4. | Has the team specified how the interface test data is identified, maintained, and version-controlled? |  |
| 5.1.5. | Have the procedures for reporting errors, test results, and reworking and retesting programs been documented? |  |
| 5.1.6. | Is there a procedure in place for reporting errors, test results, and reworking and retesting programs? |  |
| 5.1.7. | Are errors assigned different severity levels (Critical, High, Medium, Low, Info)? |  |

## Section 10: Risk Analysis

**Risk status / Areas needing further analysis / questions still not answered and plans for getting risks / issues resolved.**

* + Risk status
    - Need to not overload the server by making smart database calls
      * Do NOT fetch the same data more than once unless absolutely necessary to ensure up to date
      * Get once, process, save
    - Git branches and codebase
      * NEVER PUSH DIRECTLY TO MASTER OR DEV
        1. Only make PRs to get approved by rest of team (Monday code reviews)
      * Always work on a separate branch for the current ticket unless the tickets are subtasks
    - We are using Google services for authentication
      * If Google gets breached there is a possibility our data is compromised as well
  + Areas needing further analysis
    - Database security / rules / roles
      * Database structure finalization to prevent unauthorized access to data
  + Questions still not answered and plans for getting risks / issues resolved
    - Plan for getting risks / issues resolved
      * Bring up at next meeting and in discord to discuss plan of action
        1. If can be resolved in discord no need to meet
        2. Else we meet as a group and brainstorm how to solve the issue at hand
    - Questions still not answered
      * Deployments and deployment schedule
        1. When/how often will we deploy master to Firebase hosting?
      * Assignment/ticket schedule

## Section 11: Independent System Test

Independent System Test:

* + Test Plan
    - Create a testing environment that is separate from the main development environment
      * Does the program compile and run? What are the errors or warnings, if any? Troubleshoot any warnings or errors and continue from there.
        1. Let group know of any cases where code will not work
      * Create testing branches pulled from the development branch. This testing branch is what I use to test methods BEFORE pushing to the main branch.
      * Create a separate, possibly local/offline, testing database. This allows the manipulation of data without running the risk of messing up the main data in a main database.
      * Create two/duplicate pull requests (PR), one for the dev branch and one for the main branch. Review the PR going to dev, and if that one is good, merge the PR in dev and main. This allows for more control as to what goes into the main branch, and prevents the other changes in the DEV branch from merging to main when they are not ready or working.
    - Backend
      * Database
        1. Test the efficiency of the database structure. Add, remove and edit multiple forms of data to see how well the database handles it.
        2. Record the reads/writes to the database after each functionality check. This allows me to see how efficient the functions and methods of the program are. The more efficient it is, the less time and money it takes.
      * Methods/Functions test
        1. If possible, test each function individually and monitor the output. If the output is correct, and formatted correctly, record the results. If not, record the error and troubleshoot.
        2. Then, run the functions as a whole. Make sure that each function that relies on another function works as intended. Record the outputs.
        3. Make sure the functions are efficient. How long are they taking to execute? How can we minimize the time it takes for them to execute?
    - Frontend
      * How well does the front end flow? Does it look professional or sloppy? Is it intuitive? Is it complicated or simple?
      * Test the responsiveness of the front end. Test for multiple screen sizes and make sure that nothing overlaps incorrectly and make sure that each container is still visible.
  + Results
    - Have a document of each method and the expected output and input. Compare the actual output with the expected output.
    - Open issues on BitBucket that need to be attended to.

## Section 12: Timesheet

Timesheet

* + Form
    - <https://forms.gle/URQvCbDD9DUDoJRt9>
  + Responses
    - [Timesheet](https://docs.google.com/spreadsheets/d/1ySu6__y8mtiMQMnJt5g-4wuSD63RR8tAYpSXdWYxRcY/edit?usp=sharing)