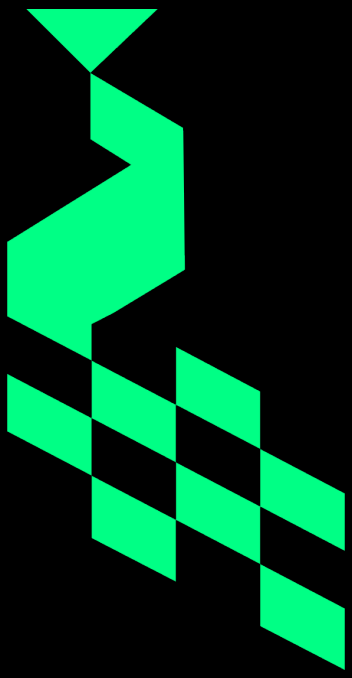
Grow Simplee's Route Planning for Optimized on Time Delivery

End Evaluation Report

Team 53



Inter IIT Tech Meet 11.0

Contents

1 Introduction

The modular nature of the given problem statement allowed us to partition it into three tasks, namely, the volume and weight scanning tool, the route optimization algorithm, and the application interface. Each of these tasks are independent of the other (up until the point of integration, when all three components will have to be merged into a final product).

One observation we made with regard to the route optimization algorithm was that a perfect, one-hundred percent optimal solution might not exist. Hence, we thought it wise to take up a comparative technique and test multiple approaches against one another. We realized there would always be room for optimization and that different approaches may give different results in varying scenarios.

2 Volume and Weight measuring tool

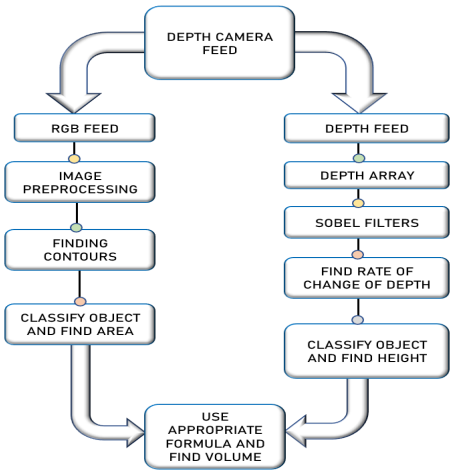
2.1 Volume Estimation Method

Volume Estimation in a contact-less manner with sensors and subsequent processing involves different techniques; prominent ones include image and depth data processing. Our approach to this is based on RGB image processing using an RGB-D camera. The RGB-D camera is a type of depth camera that provides RGB(Red-Green-Blue) and D(Depth) data in real-time. The Depth information is retrievable through a depth map created by the camera.

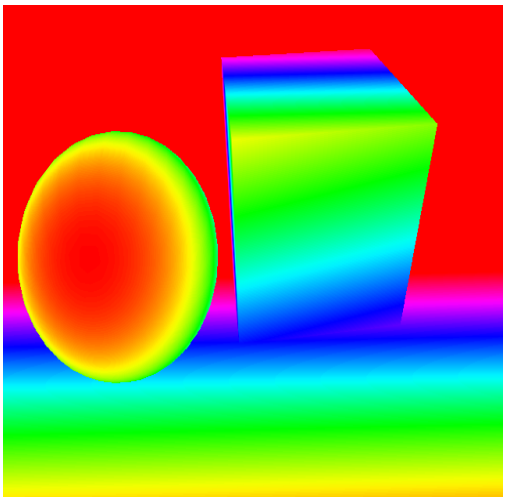
First, we elevated the RGBD camera to a height of 65 cm, setting its field of view to include the item's top surface. Using the Pyrealsense2 package (which is related to the Realsense camera), the RGB feed from the RGBD camera is then taken. Using the OpenCV library, the feed is then transformed to grayscale, further blurred using GaussianBlur, and finally dilated for improved detection and to save computation. We used contours with the findContours function to detect objects.To prevent noises, we put a threshold to the contour region. The standard shapes of the triangle, rectangle, and circle were used to categorise contours that met the criterion based on number of vertices.

Depth data gives us the height of that contoured object from the camera, to find actual height we subtract that height from the heigth of the camera. Using the create depth mask function, which selects the data of the depth frame by analysing the minimum and maximum depth distance, depth data from the depth feed was muted. We used the masked depth data to determine the average gradient of the depth values using the Sobel function to distinguish between common forms like cube, sphere, cylinder, and prism.

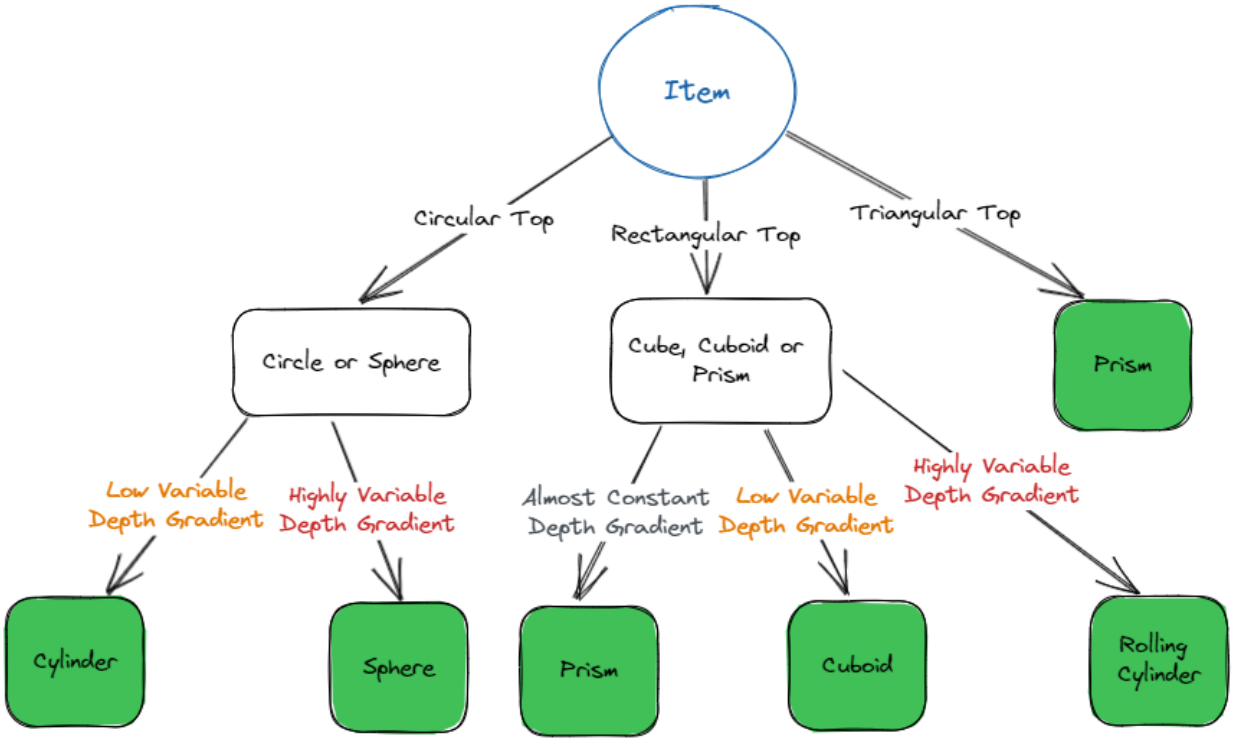
For triangle shaped top its likely to be prism. For rectangle shaped top, if the gradient is relatively low then it was classified as cube/cuboid further if gradient was between 0.3 to 4 then its rolling prism and higher than 5 its rolling cylinder. For circle shaped top, if the gradient is quite low then it was classified as cylinder further if it was higher than 0.3 its sphere. We got the values of area of contours(top side area) in pixel square form, to convert that into effective area we scaled it by a factor that depends on the height of the camera fixed. We applied it using known values of test shape. After categorising as shapes we applied standard formulae for calculating volume.



(a) Flow chart for finding the Volumetric weight and the SKU of the object



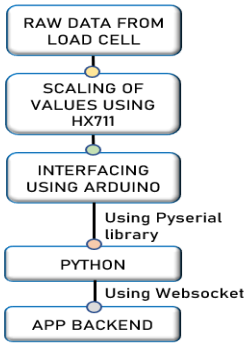
(b) A sample Depth feed of the camera



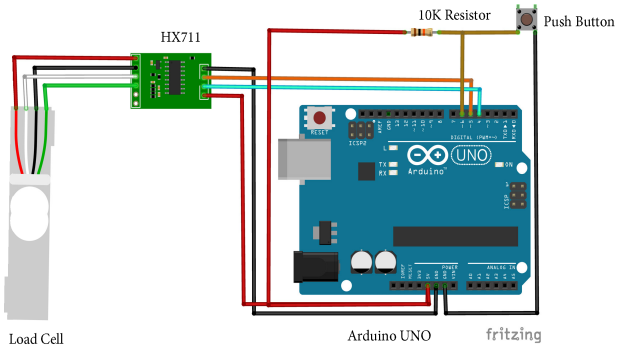
(c) Decision tree for classifying the shame of the given item

2.2 Weight Estimation Method

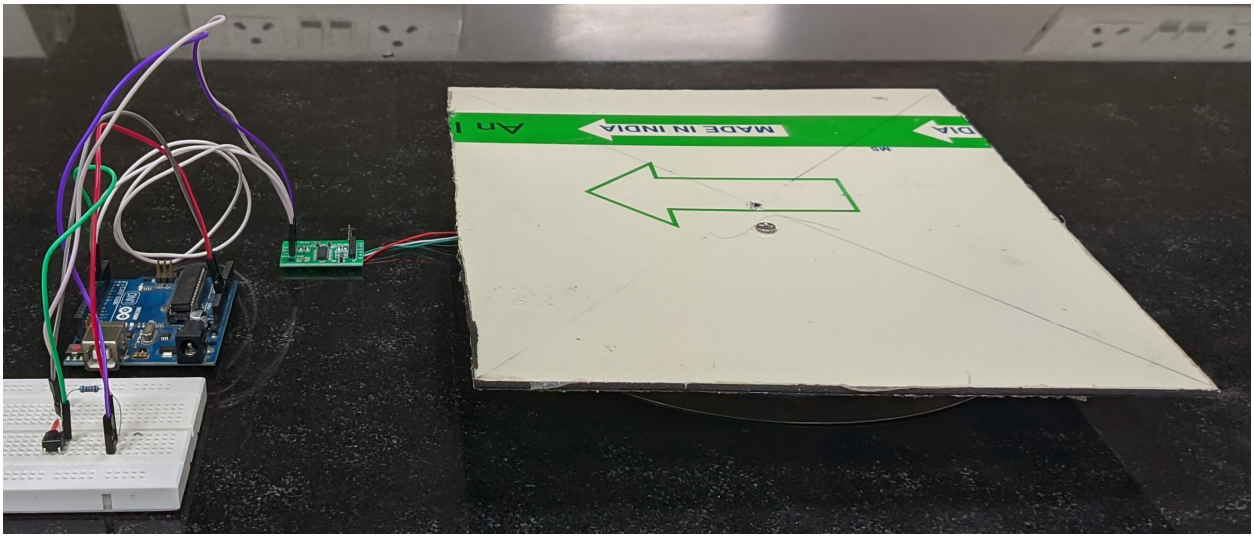
Using a load cell is the standard measurement method in any weight-measuring device. Here, we, too, use the same in the form of a strain gauge-based load cell. The load cell has four terminals and a maximum load limit of 20kg. The way a load cell works is that when an object is placed on it, the elastic membrane of the strain gauge deflects and creates a strain at those locations where the load is placed. There are four such load cells connected in a Wheatstone bridge-type connection. The weight is proportional to the imbalance voltage created at the two midpoint nodes. Since these values are very short-ranged, the HX711 amplifier module scales them up to levels readable by Arduino. These values have 2 decimal point precision. Apart from scaling, the HX711 also powers the load cell circuitry. The HX711 module is interfaced with Arduino and the serial data is communicated from Arduino to Python using 'pyserial' library and then further sent to our backend using web socket, which logs the weight value. The communication from Arduino to PC is done using the USB cable. At the backend side, to avoid the excess flow of serial data, we sleep python code for 0.08 seconds and dynamically keep only 50 or fewer values in our array at any point. In addition to measuring, we kept a Tare feature that will reset the value to 0 and facilitate calibration successive weighing of objects.



(a) Flow chart for finding the dead weight of the object



(b) Internal mechanism of the Load cell



(c) Load Cell used for measuring weight

3 Route Optimization Algorithm

The whole algorithm is divided into 3 parts:

1. Clustering Algorithms
2. Genetic Algorithm
3. Fitness Algorithms

4 Clustering Algorithms

We have implemented a total of 3 Clustering algorithms. Each algorithm takes array of delivery locations as input and outputs K number of clusters. We run all the three algorithms and use the best out of all 3, i.e. the clusters in which the sum of distance is minimum. Before implementing these algorithms, we will be calculating the actual distances between of the delivery locations using Geo APIs. Clustering these points in the beginning would avoid the possibility of far clusters and the clustered locations would be close to each other.

Following are the 3 Algorithms we used.

4.1 Density Based Scanning Algorithm:

We start with a delivery location as a random centroid and fix epsilon to be some radius and check if it has the least MinNbrs neighbours within that radius. If it has that many neighbours we add that cluster. We repeat this process until all the points have been visited once. The points that are not assigned to the clusters are called noise points.

After this process, if some noise points are left out, we assign them to the nearest cluster. This algorithm is based on actual distances (travelled by road) and not the Euclidean ones. This clustering algorithm is faster than K-means.

4.2 Edge Weighted K-Means:

Based on the locations, we start with K random clusters/centroids. In each iteration, we calculate the distance of each point from the centroids of previous iteration and calculate new centroids which have the minimum distance from the other points in the cluster. This goes on for a few iterations and accuracy improves upto a certain saturation value.

Similar to the previous algorithm, we have calculated the actual distances (travelled by road) and not Euclidean distances based on Latitude and Longitude. In each iteration, the centroid selected is a point which is already a delivery location. This is done to avoid calculating the actual distances of centroid to each point in every iteration. This clustering takes greater time in comparison to the previous algorithm, but provides better results.

4.3 Clustering Based on Area of Location:

Clustering the locations based on area, for example, colonies. Riders will be assigned nearer colonies so that they travel within the same area and minimise the distance as well as the time travelled. This clustering is highly fast.

5 Genetic Algorithm

By using clustering algorithms, we will be making K clusters to be assigned to R riders such that $R > K$., i.e. more than 1 rider maybe assigned to one cluster as after clustering it may happen that some cluster gets much more deliver points for a single riders to be delivered so we will be making K cluster and will be deciding the no of riders to be assigned to each cluster.. This will be based upon the volume of items, weights and bag capacities to achieve optimal results. Once we have clustered the delivery locations, we need to assign each rider the delivery locations with proper ordering. Genetic Algorithm takes the arrays of clusters of locations in the sorted order of EDD as the input and outputs the desired and optimised results.

Genetic algorithm consists of 3 steps:

1. Fitness Value :- The fitness value for the problem is the no of successful deliveries in the final ordering of the delivery points into riders, and the total distance travelled by all the riders. The cost function for this algorithm is called the Fitness Value. Larger the value better the results. For calculating the fitness value we will run some algorithms on the ordering to assign them to the riders in an optimal manner.
2. Generate permutations :- In this step, we need to generate some permutations of the initial ordering of the delivery points and calculate the fitness value of those permuations and store them in a list. The more permutations we are able to generate the better the final results will be.
3. Crossover :- In this step, we will take the list of the permutations that we generated in the last step and select two permutations at random and find the crossover of those two. Crossover of two permutations means that we will take some substring of the first permutation and merge it with the second permutation in a manner that the final permutation is also a permutation. After this, we will also calculate the fitness value of this permutation and add this in to the list.

   After all these steps, we will be left with a list of the permutations and with the fitness value of all of them, as a result we will take the permutation with the highest fitness value i.e. with most no of successful deliveries and least total distance travelled. From this we will be assigning the delivery points to the riders according to the algorithms used for calculating the fitness value and this will be our final result.

6 Fitness Algorithm

We use 3 Fitness algorithms to calculate and optimise our results, and choose the best out of the 3 results.

6.1 Dynamic Programming based on Location and Riders

1. Sort all the items in ascending order of Expected Date of Delivery (EDD).
2. Commence a loop to iterate over all the items 'i' sorted in ascending order according to the Expected Date of Delivery (EDD).
3. Commence a loop to iterate over all the riders 'j'.
4. Now, we use a two-state dynamic programming model defined as :

   dp[i][j] = the number of items successfully delivered till now, for item 'i', rider 'j'

   The state signifies that if item 'i' is assigned to rider 'j', then dp[i][j] is the maximum number of successful deliveries that are achievable preceding item 'i'.
5. For each state (i,j), we store three arrays accounting for the remaining bag capacity, the distance traveled by each rider and the rider's current location.
6. Commence a loop 'l' from 1 to (i-1) to calculate the current state of dp[i][j]
7. Commence a loop over all riders 'k' who delivered the l^{th} item.
8. Now for the state dp[i][j] we assume that the last successful delivery made was that of item 'l' and by rider 'k'.
9. Check if it is possible for rider 'j' to deliver item 'i'

   if possible,

   $$dp[i][j] = max(dp[i][j], dp[l][k] + 1) \tag{1}$$

   else,

   $$dp[i][j] = max(dp[i][j], dp[l][k]) \tag{2}$$

10. For each iteration of the innermost loop we update the remaining bag capacity, the distance traveled by each rider and the rider's current location considering the delivery of item 'i' by rider 'j'.

11. After iterating through all riders, we will get a value of 'j' for which dp[i][j] is the maximum. If multiple j have the same value, then choose the one for which the sum of the total distance traveled by all the riders is minimum. Thus, we will get a final order of delivery items for each rider.
12. To get the final answer, we store the rider 'j' who delivered item 'i' for all states dp[i][j].

6.2 Assigning Riders based on Nearest Location and Bag Capacity

1. Commence a loop over a threshold variable *max_possible_distance*, over the range.This threshold variable will be used in a later stage of the algorithm.
2. Commence a loop to iterate over all the items 'i' sorted in ascending order according to the Expected Date of Delivery (EDD).
3. Commence a loop to iterate over all the riders 'j'
4. For an item 'i' and rider 'j', the rider must have sufficient volume available in their bag to satisfy the order of deliveries and pickups preceding item 'i', as well as to add item 'i'. Also, the rider must reach the location of item 'i' before its EDD, and the distance between the current location of the rider and the location of item 'i' must be less than or equal to *max_possible_distance*.
5. Finally, we assign item 'i' to the rider that satisfies the required constraints and delivers the item in minimum time.

6.3 Assigning Riders basesd on Optimisation on EDD, Package Volumes and distance between two consecutive deliveries

1. Iterate through the riders sorted in descending order of bag volume.
2. We start with rider 1 from the hub and move to the closest delivery location from the hub.
3. At every delivery location, we find the next closest delivery location from the current location.
4. We continue in this manner until the rider's bag cannot hold the next closest item.
5. After the rider's bag capacity has been exceeded, the rider returns to the hub, and now rider 2 begins from the hub in the same manner as the previous rider.
6. This process continues over all the riders until all the items have been assigned to some rider.

7 Dynamic addition of Pickup Points

For handling the dynamic addition of pickup items, we will iterate through the order of deliveries assigned to each rider. For each rider's delivery sequence, we will calculate the extra distance/time that will be added if the pickup item is inserted between two deliveries. The bag volume of the rider and the time slot for executing the pick up will also be taken into account. Also, we will try to ensure that after catering to this pickup, the rider is still able to complete the initially assigned deliveries within time and with sufficient bag volume. Finally, we will choose the rider that follows these constraints and for whom the extra distance/time is minimum.

8 Conclusion

Hence the clustering algorithms, genetic algorithms and fitting algorithms work hand in hand to optimize the route for each rider, with on-time delivery and distance travelled being the most important metrics. Clustering solves the issue of performance on far cluster orders , and it will avoid routes that intersect each other, hence multiple riders won't be allocated to the same region.

In order to ensure correctness of our solution, we wrote a function that will test it against all the constraints to certify that none are violated and to calculate the percentage of on-time deliveries.

9 Application Interface

9.1 Tech Stacks

We have used the following Tech Stacks:

1. Database: MongoDB (NoSQL)
2. Backend: FastAPI (Python)
3. Frontend: NextJS

Database:
We have utilised MongoDB for our application, making it horizontally scalable and able to handle the growth of the company. MongoDB's JSON-like structure provides a dynamic schema and performance benefits.

Backend:
The backend was developed using FastAPI, a high-performance server that is highly customisable to meet the needs of developers. We have documented our API using Swagger and ReDoc, allowing other developers to easily understand the code and its routes to contribute to the repository in the future. This will benefit the company if they adopt our code.

Frontend:
The front end was developed using NextJS, a React framework. The UI of the application closely resembles the company's current website, making integration easy. The application used the same font and enough contrast, accessibility and other UI/UX principles, and the application is responsive to all device sizes.

PWA:
The application is a progressive web application, meaning it functions as both a website and a mobile app. Users will be prompted to install the application when opened in a browser. It offers many of the benefits of a native app, including offline caching, push notifications and instant updates without needing a separate app store. This is cost-effective for the company in the long term.

9.2 Users of the apps

The app has three types of users:

1. Item Scanning Personnel
2. Warehouse Mangers
3. Delivery Riders

9.3 Functional Requirements

The item scanners, warehouse managers as well as riders will have to log in using an id and password before obtaining access to the following functionalities:

Item Scanning personnel

1. As items are placed within the scanning tool, data received in the sensors will be fed to the application and will be visible on the web/app interface, along with the inferences regarding the item's information.
2. The scanner's task is to monitor the sensor data and approve the item currently visible on the interface. The item will then be added to the list of all previously scanned items.
3. Scanners may remove any item from this list at any point in time if they wish to do so.
4. Finally, once the final list of scanned items is ready, the scanner will have to submit it via the interface. This action will add the items to the database.

Warehouse Managers

1. The manager's dashboard will display the list of scanned items as well as the information about the riders for

that day.

2. In case of any inconsistencies, the manager will be able to alter the information about any rider or item.
3. Upon clicking the ”Dispatch” button, the routing algorithm will start running in the back-end and will perform optimal clustering of items to riders, as well as finding the optimal route for each rider.
4. Finally, once the final list of scanned items is ready, the scanner will have to submit it via the interface. This action will add the items to the database.
5. On the navigation page, the manager will be able to view the current location of each rider and their projected route. This page will also support dynamic addition and removal of pickup points.

Delivery Riders

1. Each rider should be able to view the route they need to take in order to reach the next delivery/pickup point.
2. Riders will also be intimidated about the expected time within which they should reach that location

9.4 Map APIs Used

1. For Geocoding of Addresses : Google Maps Geocoding API
2. For Routing between two locations : Google Maps Directions API, Geoapify Routing API
3. For Map Tiling and Mapping : React Leaflet

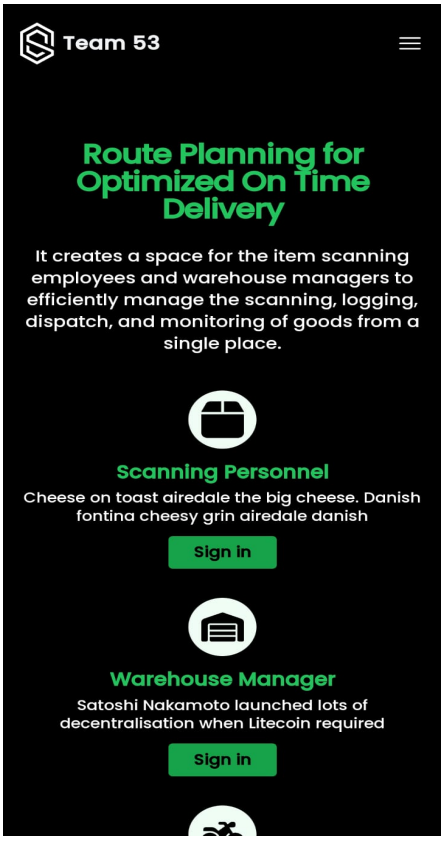
9.5 Future Aspects

Incorporate Customers as users: Initially, the application will be internal for staff use, but it can be expanded and developed to include customers in the future. NextJS provides excellent SEO functionality, allowing the page to rank highly in Google searches.

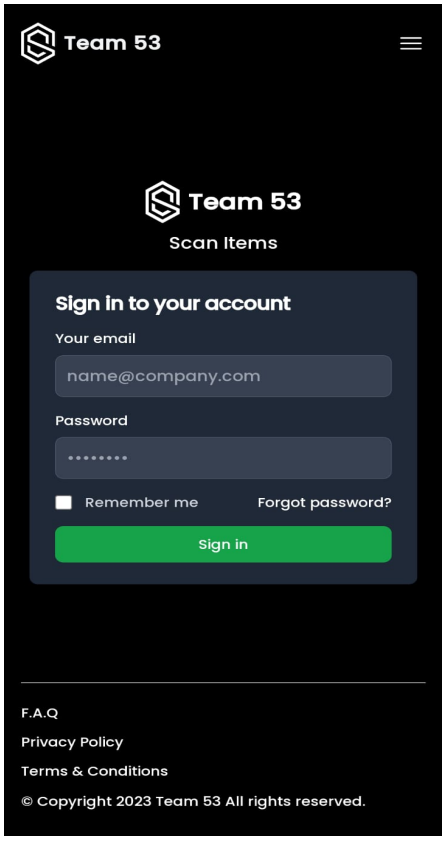
Manual Editing of Rider Routes by Warehouse Manager: This may involve adjusting the order of stops, adding or removing stops, or making other changes to the route to ensure that deliveries are completed efficiently and on time. The manual editing of rider routes can help to optimise delivery processes, improve delivery times, and ensure that customers receive their orders as quickly and efficiently as possible.

Use of Advanced ML algorithms for volume scanning tool: The use of advanced ML algorithms will speed up the volume scanning tool. By incorporating machine learning algorithms, the volume scanning tool can perform these measurements more quickly and accurately. This can lead to increased productivity and reduced errors in the warehouse, as the volume scanning tool can process a larger volume of objects in a shorter amount of time, with less chance of human error.

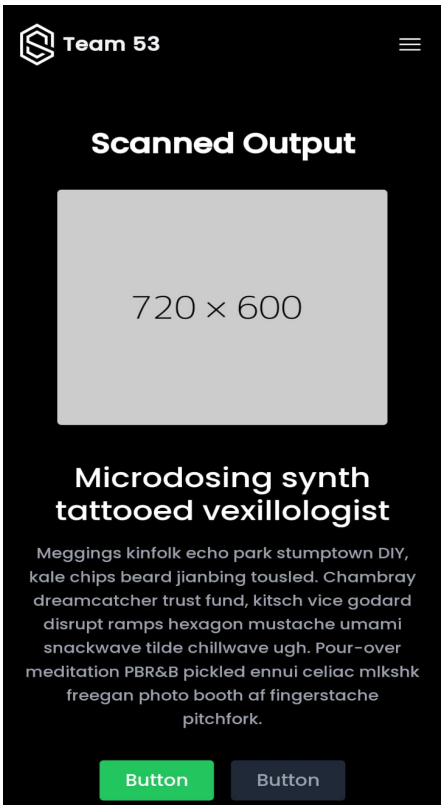
A Appendix : Application Design

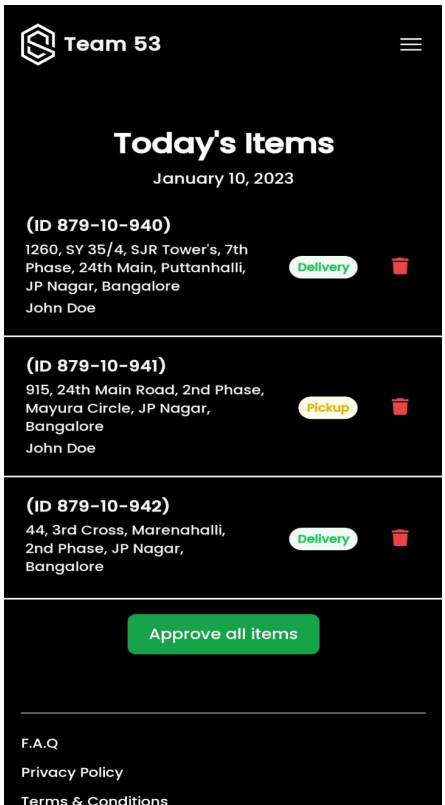


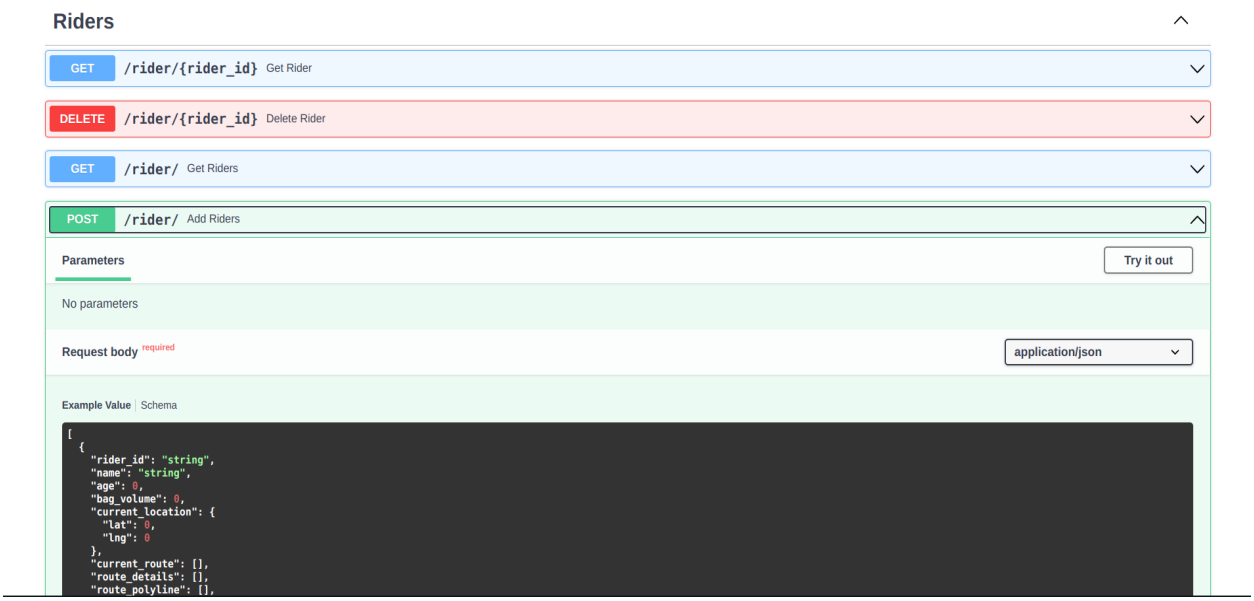
(a) Landing Page



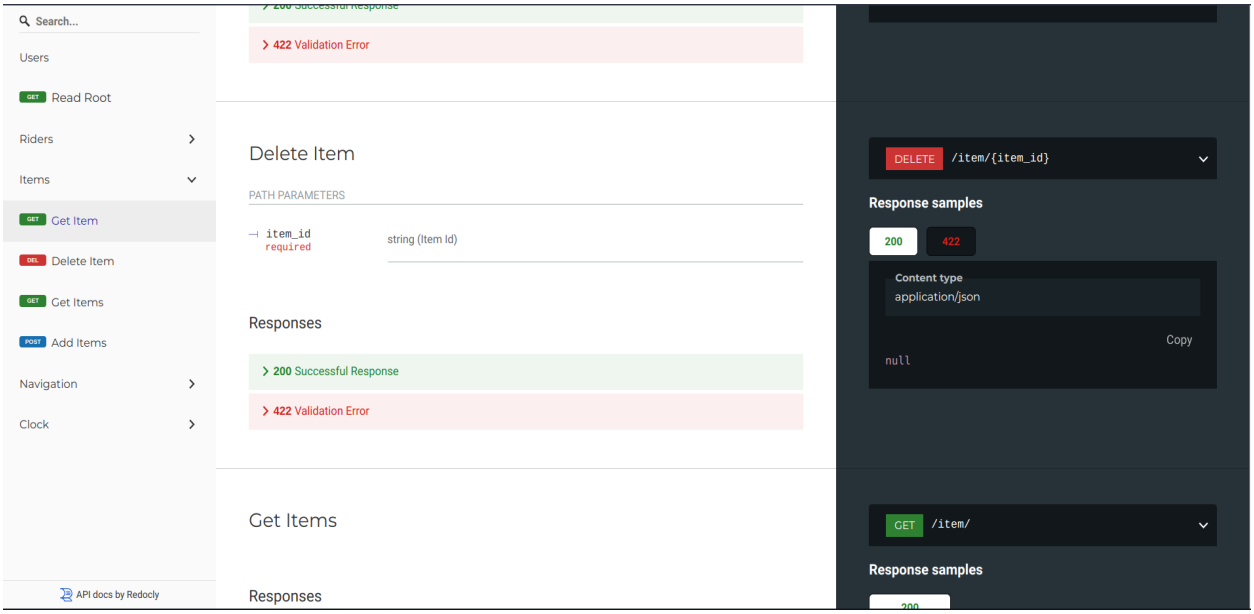
(b) Login Page



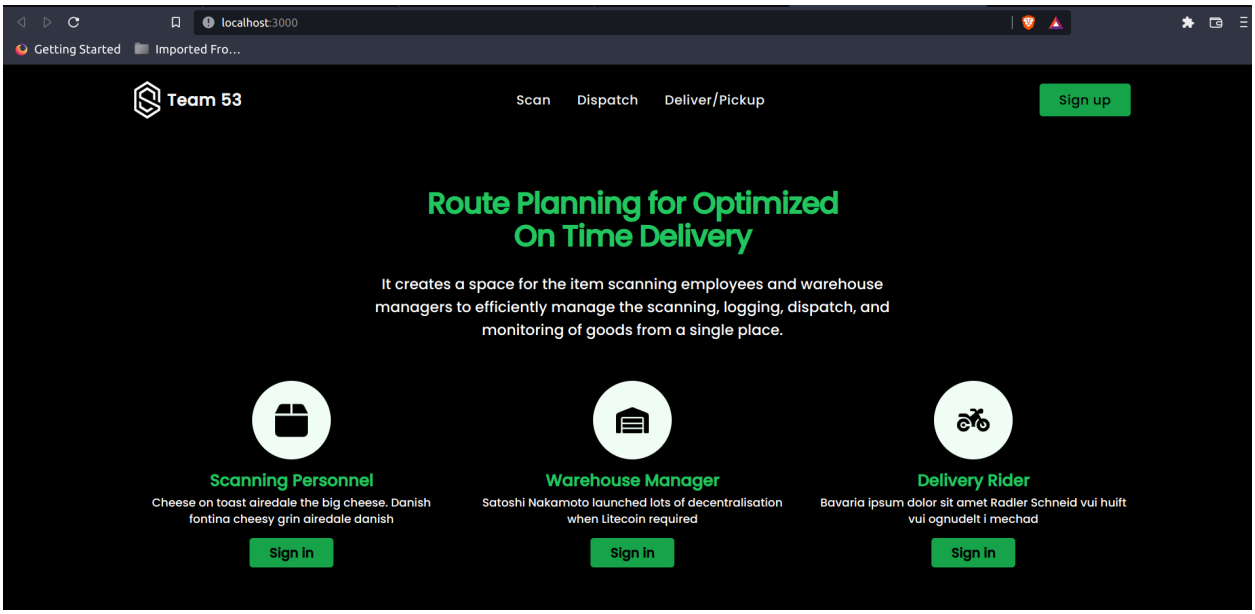




(a) Swagger Documentation for API



(b) ReDoc Documentation for API



(a) PWA: Website as well as a Mobile App