Inter IIT Tech
Meet 11.0

GROW SIMPLEE

Final REPORT

SUBMITTED BY
TEAM ID 57

INDEX

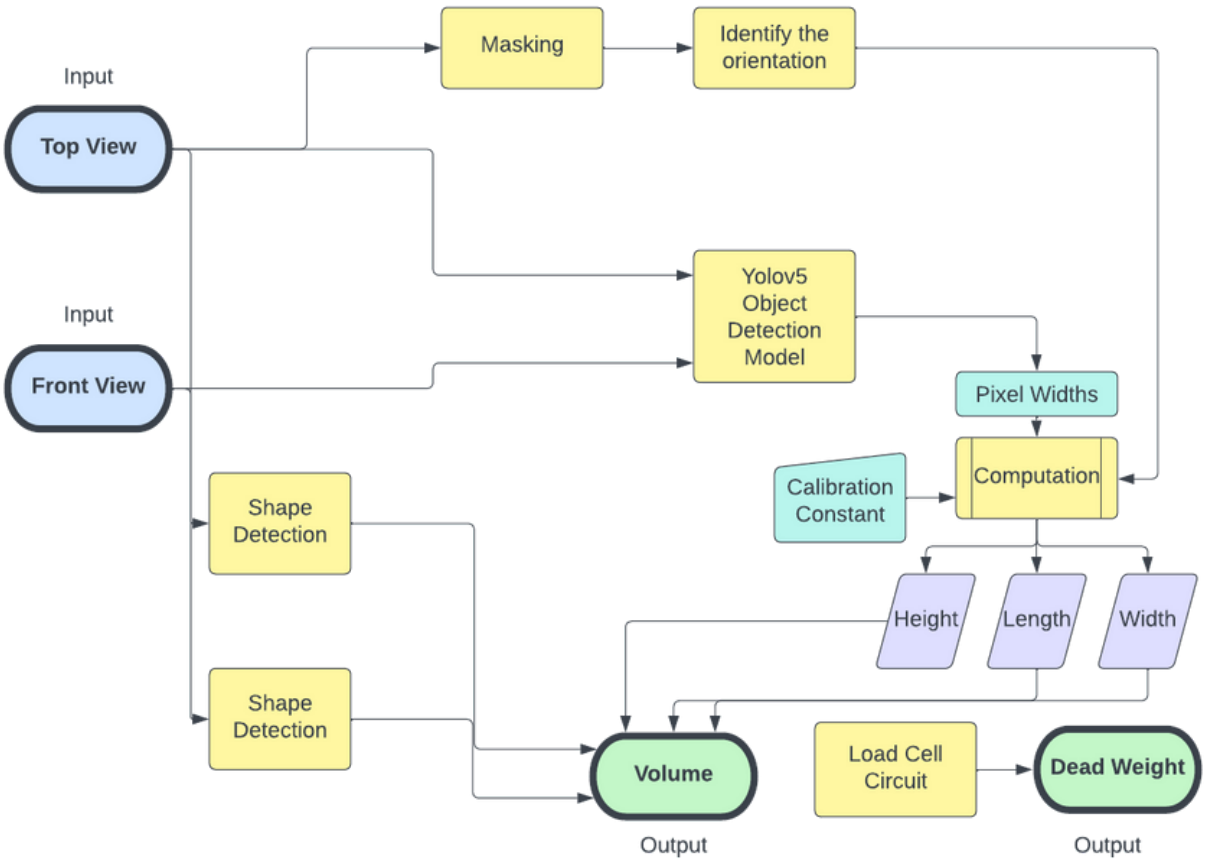
Content of Our Upcoming Strategies

CHALLENGE 1

CHALLENGE 2

CHALLENGE 1

Dead Weight & Volumetric Weight Calculation



Flowchart for the Challenge 1

APPROACH

For the estimations

We placed two cameras on a platform, one at the top to capture the top view of the object and one at the bottom to capture the front view. Previously during the mid-evaluation, we thought of using two cameras at the bottom to capture the right and left view of the object, But to measure the volume accurately in this approach, we had to keep the product parallel to both cameras. So now, to calculate the volume independent of its orientation, we decided to place a camera above it to determine the angle it makes with the base axis.

To detect the object's orientation, we mask the top view image and measure its angle with the surface base axis.

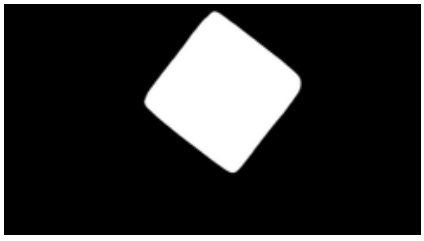
Both the images are fed to the YOLOv5x model for detecting the dimensions, a Deep Learning-based model for object detection, and gives the pixel coordinates of the object —using which we can find the pixel lengths of the bounding box of the product. These measured values are divided by the calibration constant to find the actual dimensions of the bounding box. By using the orientation angle and bounding box lengths, we can see the exact measurements of the product.

Further, we use the shape detection model to predict the three-dimensional shape of the product and use the dimensions accordingly to find the final volume.

MASKING IMAGE

for better angle detection

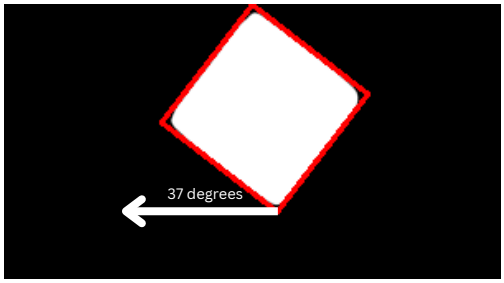
The top view image is masked using the OneFormer model trained on the COCO dataset (large-sized version, Swin backbone). OneFormer is the first multi-task universal image segmentation framework. It must be trained only once with a single adaptable architecture, a single model, and a single dataset to outperform existing specialized models across a semantic, instance, and panoptic segmentation tasks. OneFormer uses a task token to condition the model on the job in focus, making the architecture task-guided for training and task-dynamic for inference, all with a single model.



ANGLE CALCULATION

for base axis

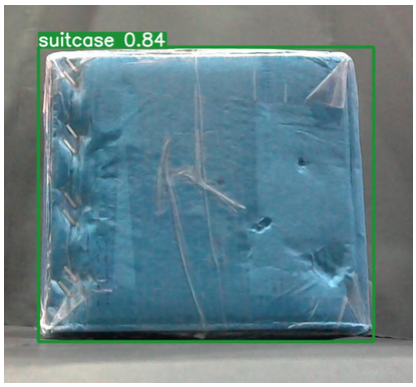
- Read the image
- Convert to grayscale
- Set Threshold
- Get outer contour
- Get minAreaRect points and angle from the outer contour
- Get vertices of the rotated rectangle
- Draw the rotated rectangle
- Correct the angle as needed
- Print the angle and use it for calibration of length & width



BOUNDING BOX LENGTHS

Getting the dimensions

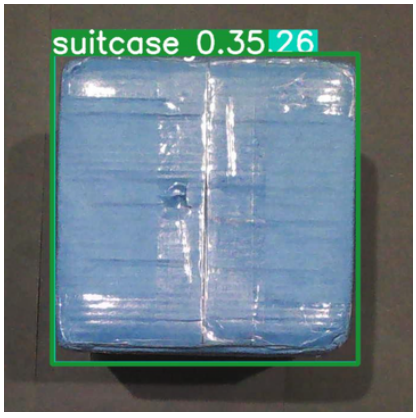
Front View



Yolov5 Output

```
[tensor([[4.05067e+02, 3.06876e+02, 9.70177e+02, 8.03018e+02, 8.38463e-01, 2.80000e+01]], device='cuda:0')]
```

Top View



Yolov5 Output

```
tensor([7.47162e+02, 1.21858e+02, 1.25828e+03, 6.40990e+02, 3.54875e-01, 2.80000e+01], device='cuda:0')
```

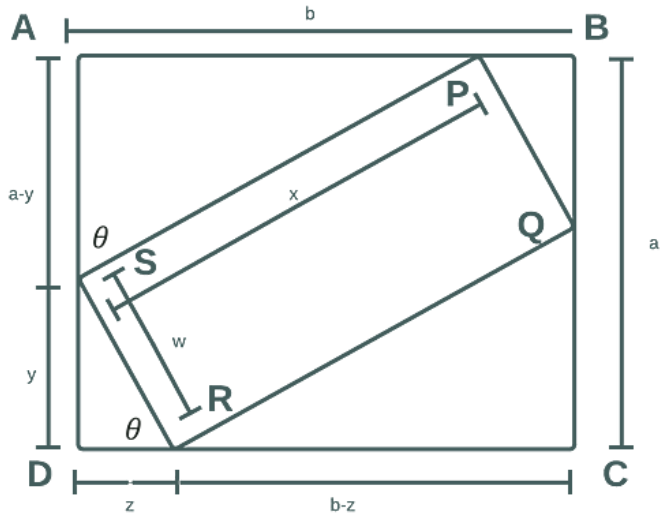
YOLO is an abbreviation for the term 'You Only Look Once.' This is an algorithm that detects and recognizes various objects in an image. Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images.
YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects.

This means that prediction in the entire image is made in a single algorithm run. CNN is used to predict various class probabilities and bounding boxes simultaneously.

Along with the bounding box, it returns the pixel coordinates of the detected object; using these coordinates, we find the pixel widths. We can further derive the original dimensions using a calibration constant that depends on the distance and angle of the camera from the object.

Length of a side = No. of pixels covered by the length/calibration constant

ABCD: Bounding Box
PQRS: Actual Object



CALCULATING ACTUAL DIM'S

Using bounding box lengths & orientation angle

AB=DC Length of Bounding Box
BC=AD Width of Bounding Box
PQ=RS Actual Object Length
PS = RQ Actual Object Width

θ Orientation Angle

From diagram,

$$\tan\theta = \frac{y}{z} = \frac{b-z}{a-y}$$

$$y^2 + z^2 = w^2 \qquad \text{(using Pythagoras theorem)}$$

$$(a-y).y = (b-z).z$$

$$(a-y)^2 + (b-z)^2 = x^2$$

$$a^2 + y^2 - 2ay + b^2 + z^2 - 2bz = x^2$$

$$a^2 + b^2 + w^2 - 2(ay + bz) = x^2$$

And,

$$(b-z) = x\sin\theta$$

$$(a-y) = x\cos\theta$$

So,

$$y = w\sin\theta$$

$$z = w\cos\theta$$

Now,

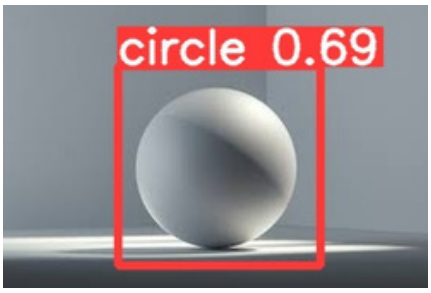
$$w = \frac{a - x\cos\theta}{\sin\theta}$$

$$(a - x\cos\theta)^2 + (b - x\sin\theta)^2 = x^2$$

$$a^2 + b^2 = 2x(a\cos\theta + b\sin\theta)$$

And,

$$x = \frac{a^2 + b^2}{2(a\cos\theta + b\sin\theta)}$$



SHAPE DETECTION

Package type

- Using the top and front view images, we detect the shapes seen through them to predict the 3-dimensional object placed. The model uses a pre-trained YOLO algorithm on common 2-dimensional shapes like squares, rectangles, triangles, and circles.
- If the shape detected by both images is square, then the object is cubical.
- If both of them are rectangular, then the object placed is cuboidal.
- If both images are circular, then the object is spherical.
- If one is a rectangle and the other is a circle, then the object is a cylinder.
- If one is a triangle and the other is either a square or rectangle, then the object is a prism.
- In all other irregularly shaped objects, we use the dimensions of the bounding box to determine the volume (l*b*h)

RESULTS & FEATURES

Strengths of approach

- Our full model is automated and can be run using a single run.
- The model is accurate to point and can determine the volume irrespective of its orientation on the platform.
- The whole model is cost-efficient and accurately predicts the volumetric weight and dead weight without using expensive depth sensors.
- Errorneuos outliers are detected and flagged

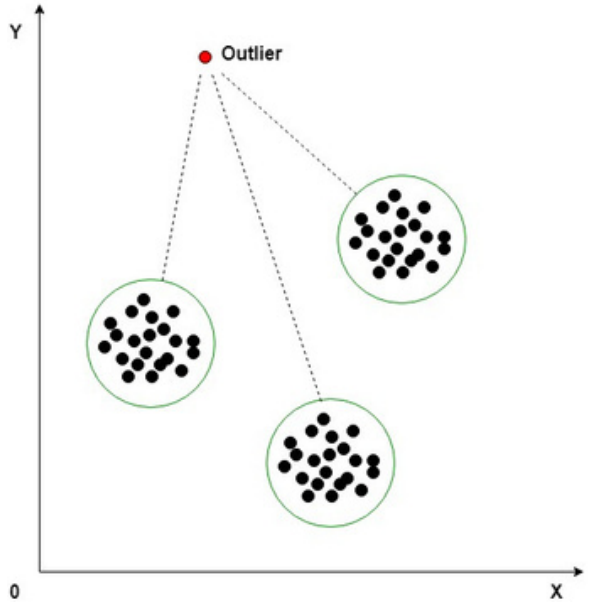
ERRONEOUS DETECTION

Getting the outliers

To detect erroneous objects, we can use clustering-based machine learning models. The clustering will be based on weight, volumetric weight, product, and product id. Clustering-based outlier detection methods assume that the standard data objects belong to large and dense clusters. In contrast, outliers belong to small or sparse clusters or do not belong to any clusters. Clustering-based approaches detect outliers by extracting the relationship between Objects and clusters. An object is an outlier if

- Does not belong to any cluster.
- A significant distance between the sample point and the cluster's center.
- If the object is in a sparse cluster.

To implement these, we can use algorithms like K-Means or Density-based clustering. But this model can only be trained if given a sample dataset of the volume and weights mapped to their product id.



HARDWARE SETUP

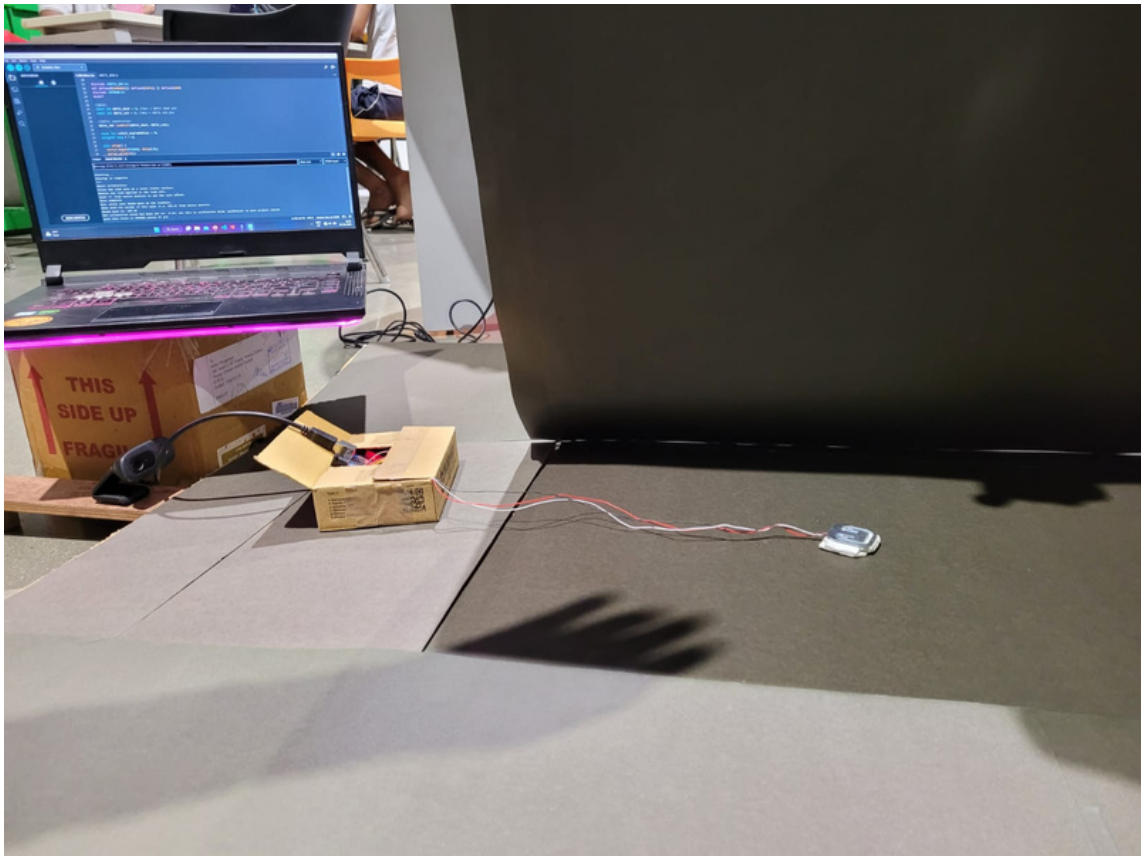
The physical stuff

Features:

- Sturdy for heavy objects
- Compatible for conveyor belt
- Complete automation
- Minimal manual work required
- High accuracy results
- Cost efficient and recyclable

Components:

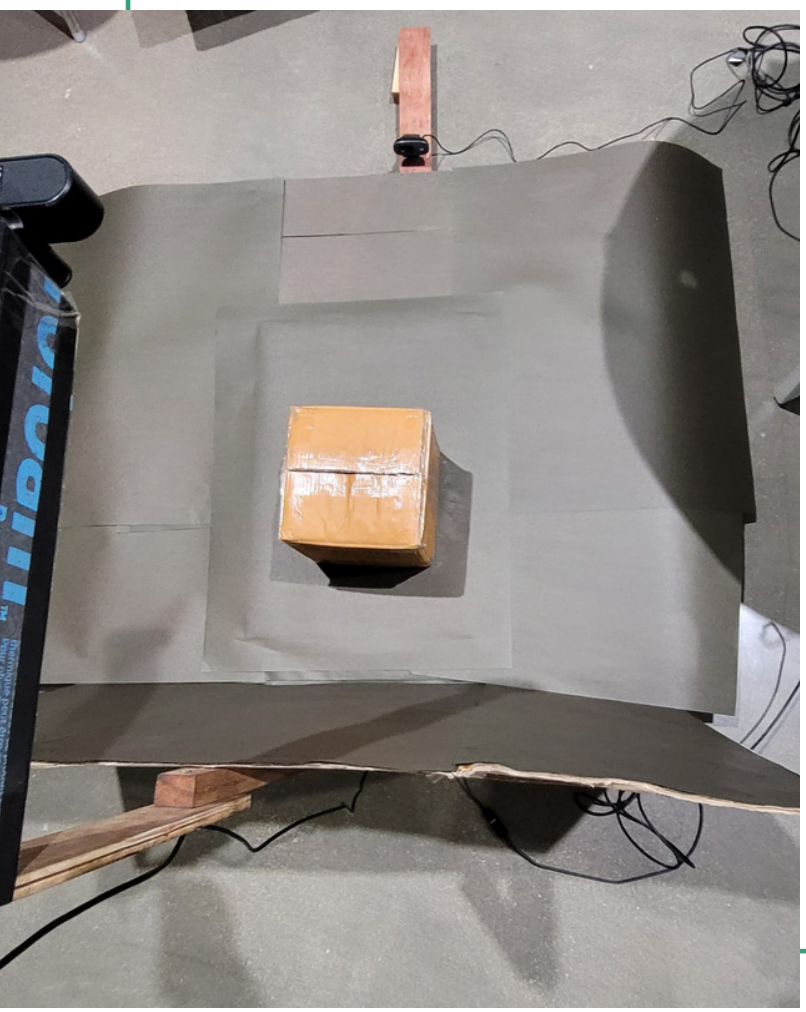
- Front HD camera
- Top FullHD camera
- Black background base
- Load cell circuit
- 135cmx130cmx105cm dimensions
- 5.5kgs setup weight



HARDWARE SETUP
Volume estimation

For getting volume

- Capture image from the top camera. Keeping it in middle using the L-shaped fixture helps to remove angles due to inclination.
- Object angle w.r.t conveyor axis is calculated for calibrating length and breadth using the masked image.
- Black background helps in reducing noise and effects from shadows and bad lighting conditions.
- Front camera takes another image for the object detection .

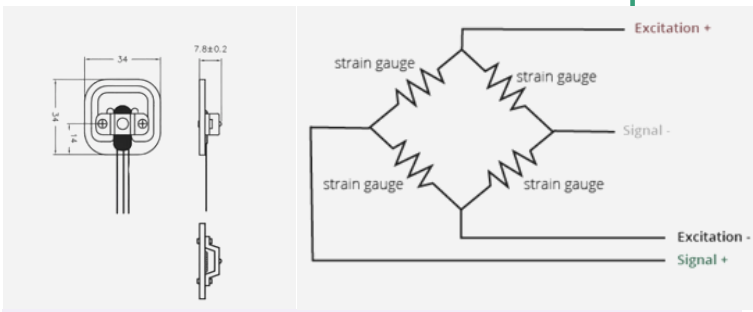


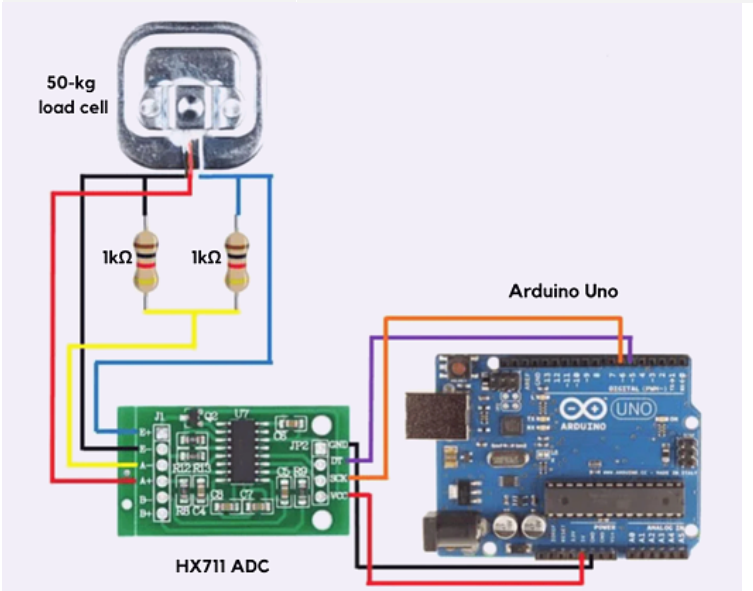
HARDWARE SETUP

Weight estimation

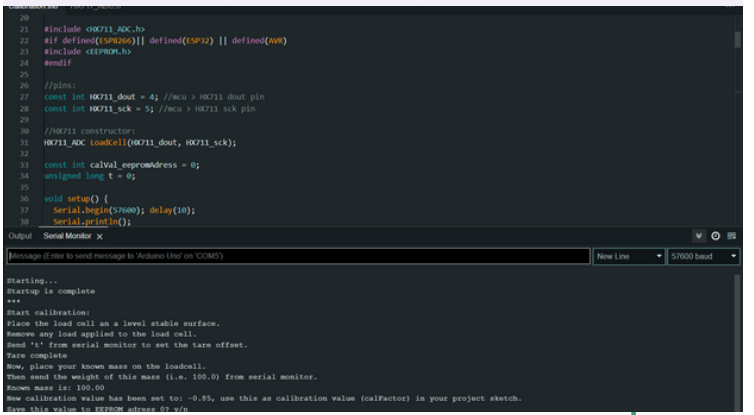
For getting weight

- **50kg-load cell** sensor is used for sensing the weight. This is a **half-bridge load sensor of 1kΩ**, used in weight scales. The working principle is that when the half-bridge is being stretched, the wheatstone bridge reading changes, sending the signal via the red signal wire. This is Alloy Steel wired weighing load cell with high accuracy, simple structure, and simple installation.
- While using this load cell sensor, the correct force is applied to the outer side of the strain E-shaped beam portion of the sensor (i.e., a strain gauge affixed to the intermediate, adhesive coating with white beam arms); and the outer edges to form a shear force in the opposite direction, (i.e., middle strain beam bending) necessary changes can occur under stress, strain beam side by another force should not be a barrier.
- The change in resistance is so small that we need an external amplifier like an **HX711** breakout board to amplify the small changes to readable values that can be read by the Arduino microcontroller or ADC ICs.
- The dual-channel 24 Bit Precision A/D weight **Pressure Sensor Load Cell Amplifier** and ADC HX711 Module is a small breakout board; for the HX711 IC that allows you to easily read load cells for measuring weight. We are able to get very accurate weight measurements after the calibration of the sensor.
- **calibration factor = (reading)/(known weight)**
- The load cell and ADC are connected to the **Arduino microcontroller's GPIO** pins for input-output of the sensor readings and controlling the load cell through code.
- The serial data from the Arduino console measuring the object's weight is averaged out using python script and sent out as CSV file.





Circuit diagram for weight measurement



APPROACHES
Routing Algorithms

APPROACH

- We are using Google OR-Tools to generate a solution for **Capacitated Vehicle Routing Problem with Time Window Constraints(CVPRTW)**.
- Developing upon the work till mid-evaluation, we had two approaches - one Google OR-Tools and the second Sweep Clustering followed by Ant Colonization technique. After running on test cases, the first one performed better, and we went ahead.
- In the capacitated vehicle routing problem with time windows (CVRPTW), a fleet of delivery vehicles with uniform capacity must service customers with known demand and a delivery deadline by which the delivery has to be delivered(EDD). The vehicles start and end their routes at a common depot. The objectives are to minimize the fleet size and assign a sequence of customers to each truck of the fleet **minimizing the total distance traveled** such that all customers are served. The total demand served by each vehicle does not exceed its capacity.
- Google OR-Tools provides **VRP**, **CVRP**, and **VRPTW** as separate modules; we have combined these three to satisfy the constraints of the problem statement. The algorithm considers capacity constraints for each rider and time window constraints for each delivery location (corresponding to the EDD).
- The items are assigned to each rider so that the total volume does not exceed the bag capacity of the rider.
- Each location is assigned a **time window** within which the package has to be delivered.
- Keeping the capacity and time window constraints in mind, the clustering is done, and routes are assigned to each rider with the minimum possible total distance traveled.
- It might be that delivery at all locations can't be done with the current set of riders in the first tour. This is handled by **adding penalties to each delivery location, which considers** the volume of the object, time, and distance required to reach the destination from the hub. This is explained in the next section.

PENALTIES SYSTEM

- The main issue with simple **CVPRTW** is, if all the delivery items won't be able to fit in all the vehicles, the algorithm shows that there is no feasible solution.
- So, to avoid this, we add a term called a penalty. Each location has a penalty associated with it which is added to the total distance traveled if the location is dropped from the current set of routes.
- So when the algorithm minimizes the total distance traveled, the penalty is also minimized which **increases the percentage of total deliveries** and reduces the number of locations that are dropped.
- The penalty for each location is calculated as follows -
  Sum of total distance +
  3 * (distance from the location to hub) +
  2 * (time from location to hub) +
  1 * volume of the delivery item
- The following method is taken into account because the main criteria are for the total distance traveled and then time and capacity, so they are multiplied by 3,2,1 respectively.

CONSTRAINTS

$$Minimize \sum_{i\in N}\sum_{j\in N}\sum_{v\in V} C_{ij}X_{ij}^{v}$$

$$Subject\ to \sum_{v\in V} y_i^v = 1\ ,\ for\ i \in N$$

$$\sum_{i\in N} x_{ij}^v = y_j^v\ ,\ for\ j \in N\ and\ v \in V$$

$$\sum_{j\in N} x_{ij}^v = y_i^v\ ,\ for\ i \in N\ and\ v \in V$$

$$\sum_{i\in N} d_i y_i^v \le Q\ ,\ v \in V$$

$$\sum_{i\in N} x_{i1}^v \le 1\ ,\ v \in V$$

$$\sum_{j\in N} x_{1j}^v \le 1\ ,\ v \in V$$

The first equation states that the total traveling distance of all vehicles is to be minimized. 2nd represents the constraint that each customer must be visited once by one vehicle. It is guaranteed in Eq. (3) and Eq. (4) that each customer is visited and left with the same vehicle, A constraint in Eq. (5) ensures that the total delivery demands of vehicle v do not exceed the vehicle capacity. Eq. (6) and Eq. (7) express that vehicle availability should not be exceeded.

ALTERNATE APPROACHES

- Another approach that we tried was Clustering the locations using Sweep Clustering Algorithm and then using Ant Colony Optimization on each cluster which is explained in the next section. Google OR-Tools performed better and hence we went forward with it.

APPROACHES

Ant Colonization Technique

APPROACH

Ant colony optimisation (ACO) is a meta-heuristic for solving hard combinatorial optimisation problems.This can be used to solve the Travelling Salesman Problem.The algorithm is based on how ants search for food from their colony. In this method, each ant is assigned to a path from colony to food, and while traveling on the path, they release a chemical pheromone on the path. Considering the equal rate of evaporation for all paths, the shortest path will have the highest intensity of the pheromone. Finally, all the ants follow the same route after a certain iteration.

Pseudo-Code

```
INITIALIZE:
  Set the number of ants (n) and iterations (m)
  Initialise the pheromone matrix with random values
  Calculate the heuristic information matrix

FOR each iteration (m):
  FOR each ant (n):
    Initialise the tour list
    SELECT the next city based on pheromone and heuristic information
    ADD the selected city to the tour list
    UPDATE the pheromone matrix
  END FOR
  UPDATE the pheromone levels on the entire trail
  REMEMBER the best tour
END FOR

RETURN the best tour
```

DYNAMIC PICKUP

Adding new point to cluster

- Firstly append the point in both the distance and time matrix.
- VRP(**Vehicle Routing Problem**) - It gives the route connecting all the points within the minimum distance possible.
- Whenever a new dynamic point gets added, we check for the current route of all drivers.
- As the problem also involves **capacity**, for each cluster (i.e; each driver route), check for the location after which the current capacity can fit into the bag (since it is a pickup). Let's call it an allowed point.
- Then for each cluster, run VRP from the allowed point to the hub and store all the routes.
- Calculate the total distance for all the clusters after the new route is appended.
- Sort the minimum distance change between all the clusters in ascending order and store them in the difference list.
- Start checking from index 0 of the difference list and check whether the vehicle is satisfying the given time window; if it satisfies, we add the dynamic point to the cluster which is in index 0 or else continue the process for the next set of clusters.
- After the route is found, we update the vehicle routes and weights of the bag.

Pseudo-Code

```
function add_dynamic_points():
    distance = []
    for j in num_vehicles
        allowed_point = find_allowed_point()

        route = vrp(vehicle_route[allowed_points:])
        distance.append(vehicle_distance)

    difference = vehicle_distance - vehicle_previous_distance
    difference.sort()

    while true:
        check_time_satisfies_for_cluster(i)

     update(vehicle_routes, capacity)
```

NEW ROUTE

Assign new route to vacant driver

- When the route of a driver has finished, then we assign the driver a new route keeping in mind the time of work left in their shift for the day.
- We are going to use the **CVPRTW** again but in this case, the number of vehicles will be **1**. The locations which will be assigned to the driver will be the **remaining set of unvisited points** this is being achieved by assigning the already delivered location's capacity with more than the vehicle capacity. In this way, the locations won't be revisited. The time window for the driver will be a minimum of 5 hours and the remaining hours left for the driver in the day.

BAG CREATION

Arrangement of package in Bags

For each rider, the bag can be filled using the following method - start with the objects from the last delivery and keep stacking up the objects from the earlier deliveries. This way the objects of the earliest delivery location will be at the top of the bag and will be easier for the rider to find and hand them over in minimum time.

GEOCODING

Converting Address to coordinates

- Geocoding - We are using Geocode by Awesome table extension for geocoding. Using selenium we automatically submit the google form containing the address which triggers the geocoding extension and generates the latitude and longitude corresponding to the address.

TIME AND DISTANCE MATRIX

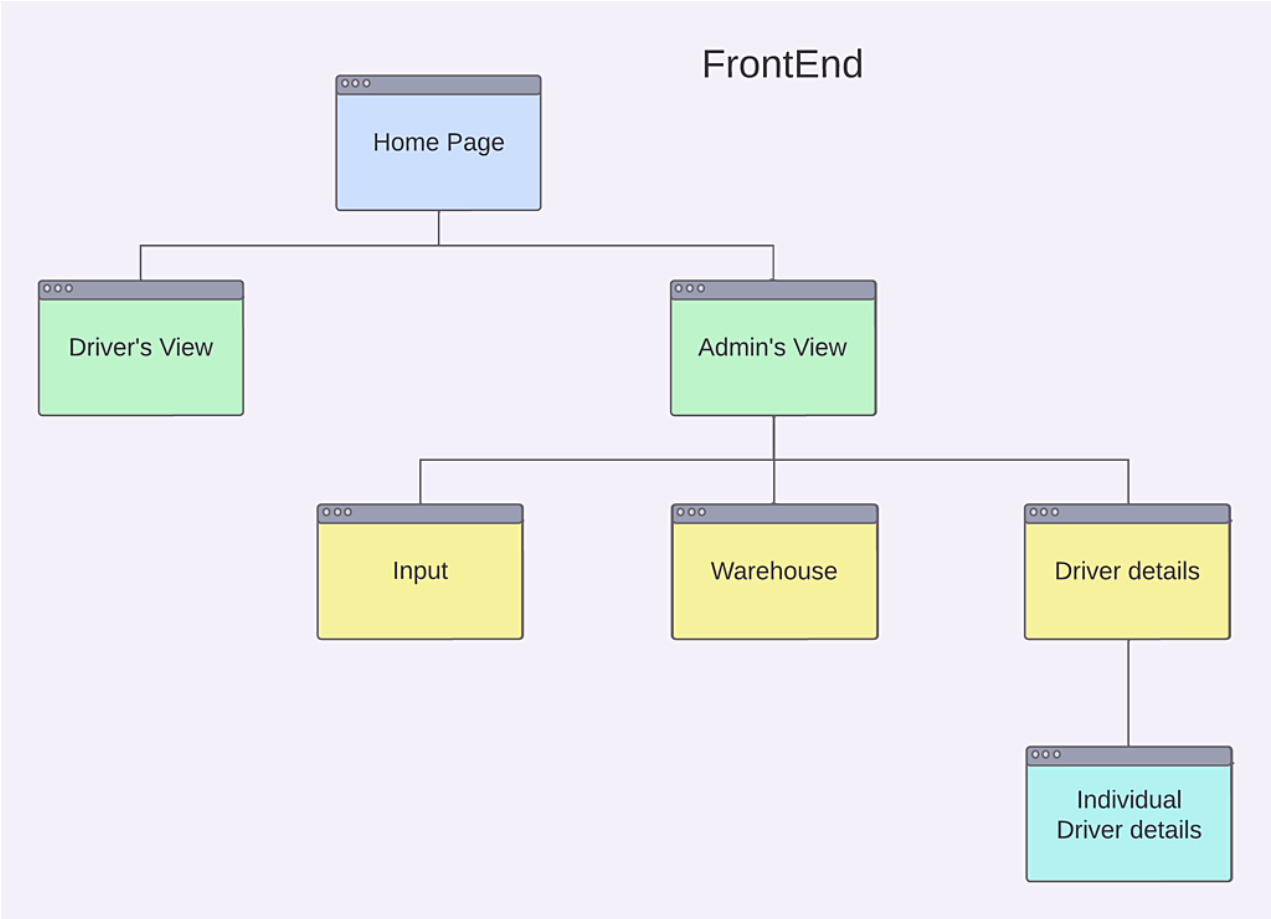
Corresponding distances and times

- After getting the latitude and longitude of each location, we are using the OSRM API to get the distance and time matrices, which are then used by the algorithm for routing.
- Using OSRM API, we can only get a 100x100 matrix at a time. So can fill the complete matrix in chunks of 100x100 at a time.

WEB APPLICATION

A walk through the site and process

We have designed our website keeping some fundamental software engineering principles in mind. Our web application is user-friendly, has a simple and intuitive layout, and is mobile responsive. We have also made sure that the website's loading time is minimal. Along with this, we are using all the available open-source resources to ensure the website is self-sufficient.



FRONT END

The Admin view

An admin has three major tasks, and we have made a page for each to make it more modular and have a clear workflow for him.

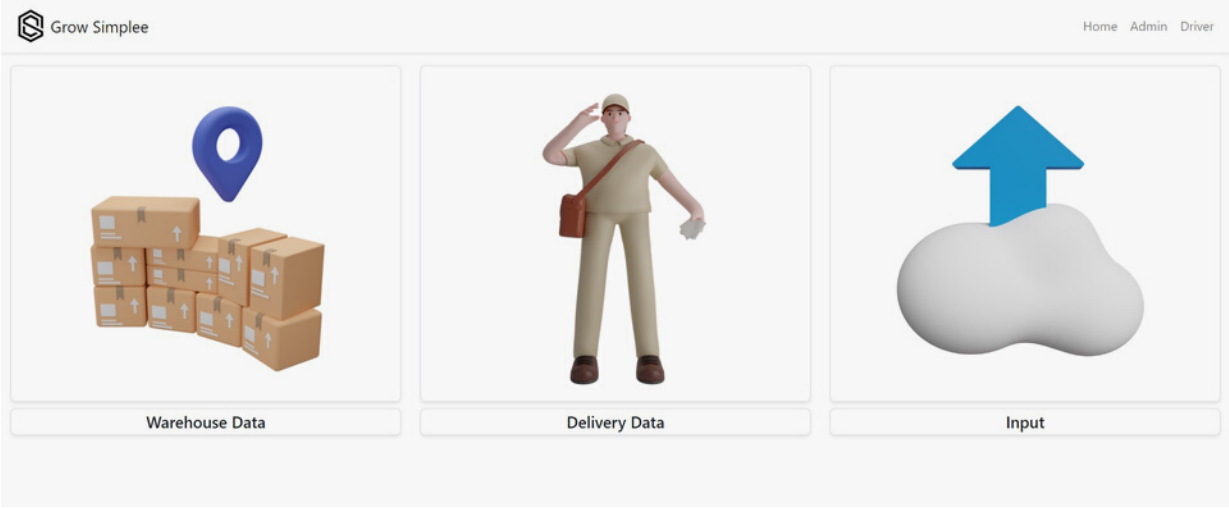
- **Input page:**
  On this page, the admin can take input required for the algorithm. We have provided him with the facility to add an address and tentative volume for the Dynamic Pickup points. We have given him access to feed details of driver and delivery locations. This data collected would then be sent to the backend for the algorithm to run.

- **Driver Details:**
  Here Admin can see a list of all the drivers that are out there on the field for delivery, and he is also given a feature to see the entire route of a particular driver. Admin would also be able to see the statistics of that specific driver, like the number of deliveries he has done and on-time deliveries. Through this, the admin can analyze a particular driver's efficiency, which would help him grow his business.

- **Warehouse Page:**
  Here the admin is shown the entire list of all the delivery items, their status, and their mapping to a particular driver. We have also included the statistics of overall per-hour deliveries, which would help the admin analyze the drivers' general behavior.

Admin view covers all the major tasks handled by Admin like Input, Warehouse details, Driver details

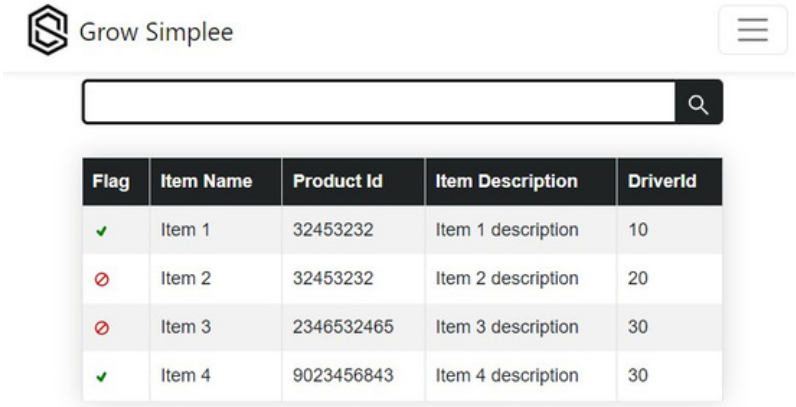
Admin Page



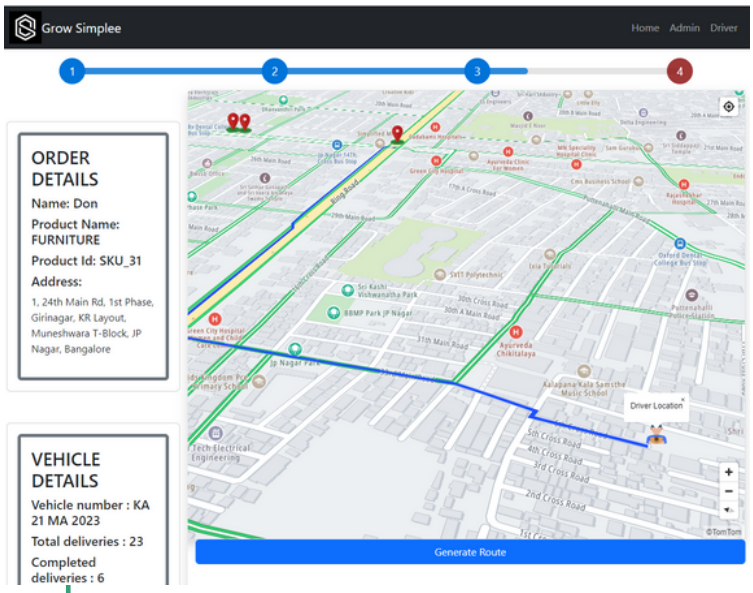
FRONTEND

Driver View

Warehouse Page

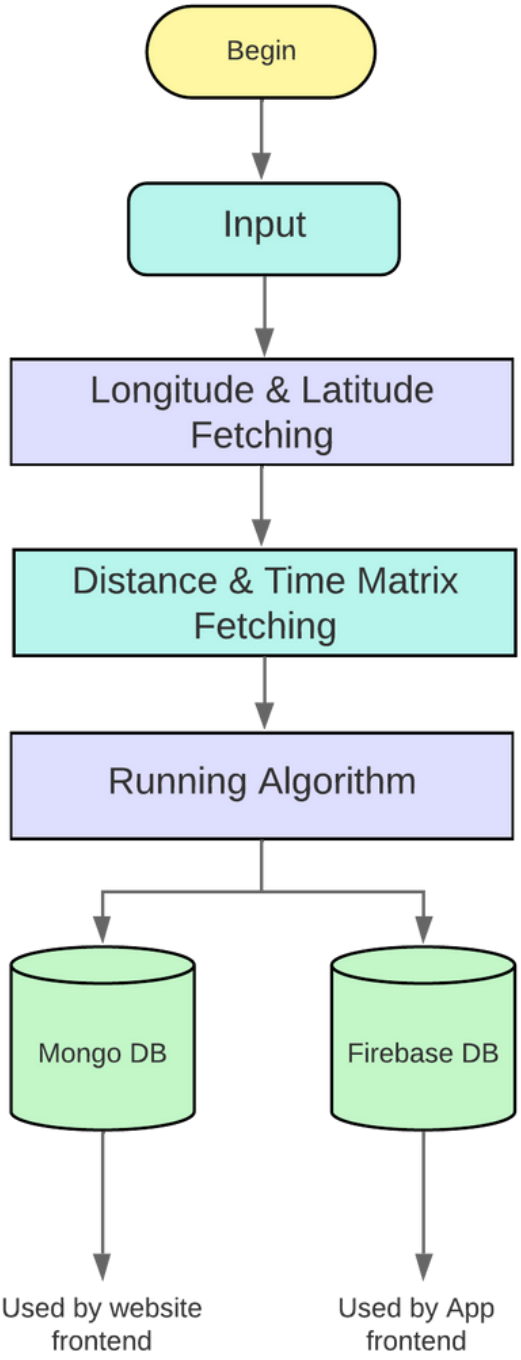


Driver page



A driver would be able to see a map that shows him the route of his immediate next delivery and also points to all the delivery locations allocated to him in one particular ride. He can click on the start delivery button, and it will track his real-time location and would show him the tentative time for him to reach the next delivery location. The driver is also shown the product details that he should know to deliver it. Also, once he reaches the location and the object is delivered, he is supposed to click on the Delivery done button that shall refresh a new route for him which would be the route for his next delivery. He is also allowed to take a break in between for 5 minutes, and the timer would be paused during that duration. For this, once a driver completes his last delivery, he will be re-routed to the warehouse of the company, and this entire trip is considered as one round trip. Throughout the front end, we have used **TOMTOM API** for displaying maps and routes to the user. It detects the driver's real-time location and helps us provide proper instructions to the driver.

We have assumed that we will provide our drivers with Electronic Vehicles (EVs) to optimize the cost spent. There will be a charging station at the warehouse, so the driver won't have to waste time during their delivery on that. With this, we are not only taking care of cost-effectiveness, but also we are taking a step toward an environmentally friendly delivery system. Hence, with this model's help, we can reduce our costs by 88%.



BACKEND

The backside routes

We have used python **Flask** for the backend development of our Website. In the backend, we are first accepting the delivery locations and corresponding delivery items from the Admin Input page. We are also feeding data of all the drivers that would be on duty to deliver the items to these locations. This data is first sent to **HERE API** that helps us fetch the geo-locations of all the delivery addresses. Now with the help of those geo-locations, we are using **OSRM API** to fetch the distance and Time matrix for the same. Now that we have all the required inputs for our algorithm it is then sent to the algorithm and the output of the algorithm is sent to the mongo DB database and Firebase which are then fetched by the UI of the web and mobile application respectively.

We have used two schemas:

The first schema consists of data of the drivers assigned and their locations for delivery.
1. Driver-ID
2. Driver Name
3. Locations- The longitude and latitude of points to be delivered
4. Address-Addresses of the items
5. Product-ID-To help in tracking Item

The second schema is to track the successful deliveries and time taken for delivery

1. Driver-ID
2. Driver Name
3. Product-ID
4. Time Taken - Time taken for delivery
5. Source
6. Destination

A WALK THROUGH OUR SITE

Seeing the view

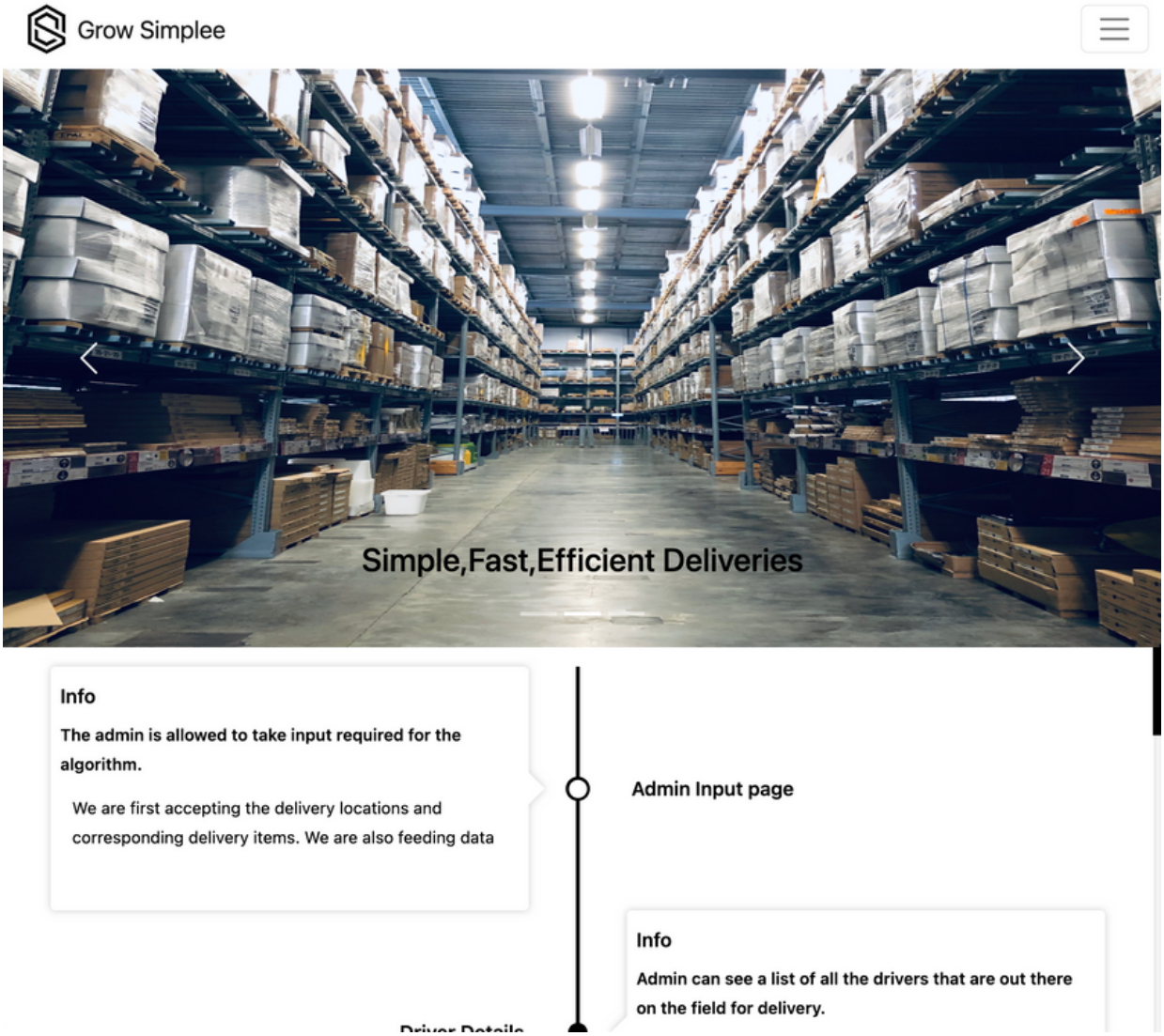
Tech Stack:

Frontend: React JS, Flutter
Data-Base: MongoDB, Fire-Base
Backend: Flask
Latitude & Longitude: HERE API
Distance & Time Matrix: OSRM API
Map & Route UI: TomTom API For Website(For Admin View and Visualization), Google Maps for App(Ensures Driver gets the best route while traveling)

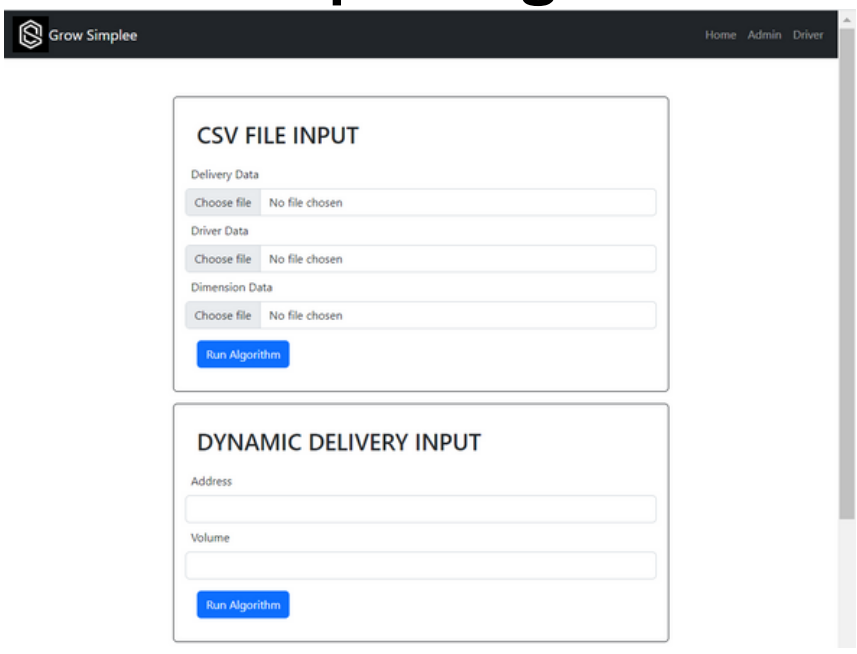
Driver Delivery Schema
productID: list Address: list locations: list driverId: string drivername: string

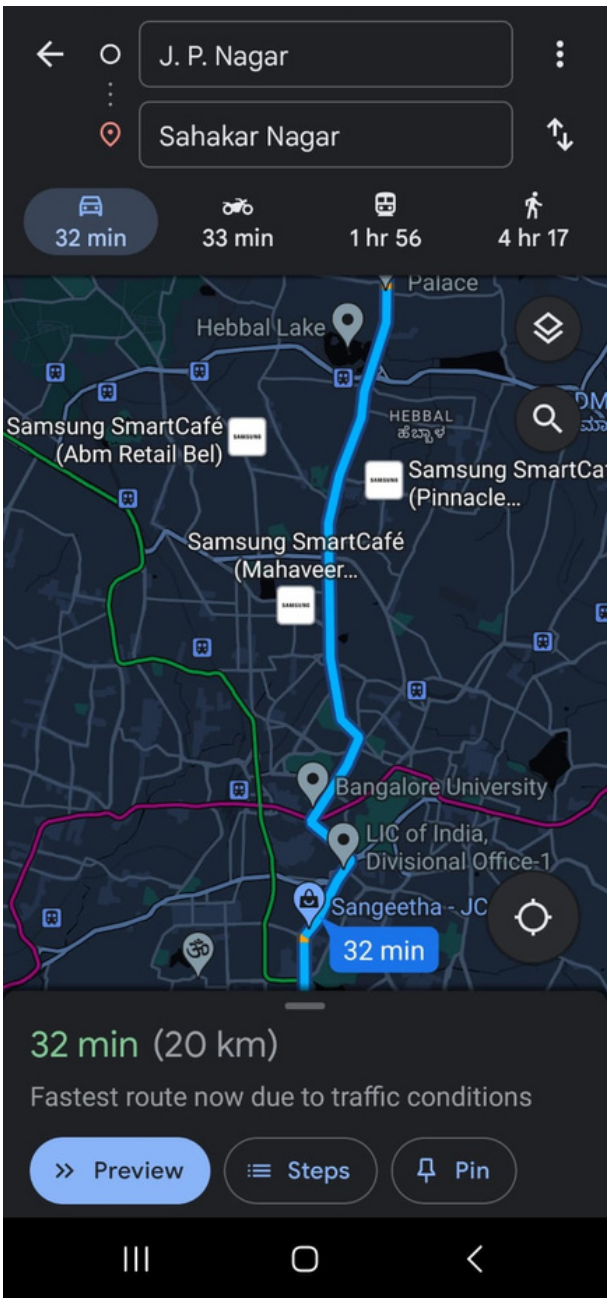
Drivery Journey Schema
TimeTaken: float driverId: string drivername: string distanceTravelled: float src_coord: object dest_coord: object

Home Page



Input Page





APP

Driver friendly app