TATA CONSULTANCY SERVICES

INFRAMIND 3

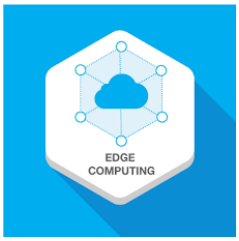
InfraMind

TEAM:

HackCode

Team Details:

Participant Name	CT Number	Role(Team Leader/Member)	Bachelors Discipline	Expected Year Of Passing	Gender
SURAJ SINGH	CT20182379444	Team Leader	Comp Science	2020	M
TEJASWINI M R	CT20182379433	Team Member	Comp Science	2020	F



EDGE COMPUTING SOLUTION

VIDEO PRESENTATION LINK : https://youtu.be/HWO1-MHQl00

GITHUB LINK : https://github.com/suraj038/TCS_INFRAMIND3

Running the Application:

download link : !git clone https://github.com/suraj038/TCS_INFRAMIND3.git

extract zip file

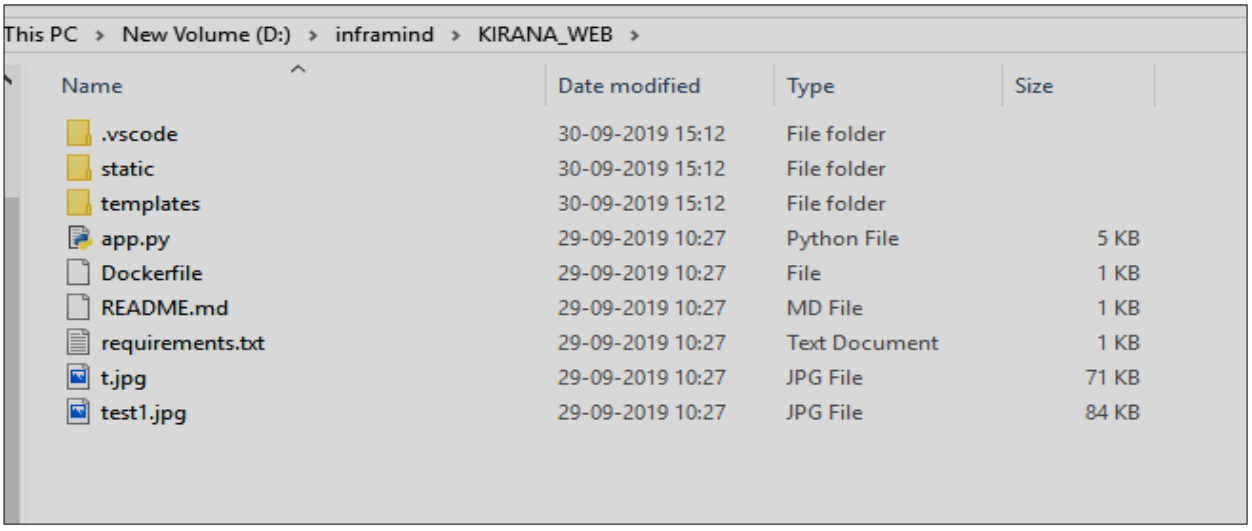
cd TCS_INFRAMIND3-master

Installing Requirements:

pip install requirements.txt

1. Create a directory with name “KIRANA_WEB”.
2. Run the command “python app.py” in the KIRANA_WEB directory using CMD.
3. After running app.py copy the url and paste it in web browser to run website.

Directory Structure:



KIRANA_WEB/app.py

```
#!/usr/bin/env python
from flask import Flask, render_template, Response, request, jsonify
import cv2
import os
from time import sleep
import json
from watson_developer_cloud import VisualRecognitionV3
from werkzeug.utils import secure_filename

app = Flask(__name__)
video = cv2.VideoCapture(0)

list_product = {}
result = []
flag = 0
count = '0'

price = {'Bottle601':10, 'Bottle602':20, 'Bottle603' : 30 ,
        'Chips401': 10, 'Chips402' : 20, 'Chips403' : 30,
        'Choc301': 5, 'Choc302': 100, 'Choc303' : 300,
        'Drink201' : 20, 'Drink202': 45, 'Drink203':90,
        'Shamp501' : 2, 'Shamp502' : 50, 'Shamp503' : 200,
        'Tooth101' : 20 , 'Tooth102' : 45, 'Tooth103': 150}

def clear():
    global count
    global list_product
    global result
    global flag
    global price

    count = '0'
    list_product = {}
    result = []
    flag = 0

    return 'data cleared'

def item_data_price():

    global count
    global list_product
    global result
    global flag
    global count
    global price

    visual_recognition = VisualRecognitionV3(
        '2018-03-19',
```

```
        iam_apikey='MhOB0X6MTF24-rx2QNl-eCqAPxV_9EX05KdPtBZigq0j')

    with open('test1.jpg', 'rb') as images_file:
        classes = visual_recognition.classify(
            images_file,
            threshold='0.6',
            classifier_ids='DefaultCustomModel_1778410211').get_result()
    try:
        print(classes['images'][0]['classifiers'][0]['classes'][0]['class'])
        data = classes['images'][0]['classifiers'][0]['classes'][0]['class']
        res = data.split('_')
        item_id = res[0]
        weight = res[1]
        item_name,item_size = res[2].split()

        for product_res in result:
            if product_res['item_id'] == item_id:
                print('comes here')
                product_res['item_id'] = item_id
                product_res['weight'] = weight
                product_res['item_name'] = item_name
                product_res['item_size'] = item_size
                product_res['price'] = price[item_id]
                product_res['quantity'] = product_res['quantity'] + 1
                flag = 1
        if flag == 0:
            list_product['item_id'] = item_id
            list_product['weight'] = weight
            list_product['item_name'] = item_name
            list_product['item_size'] = item_size
            list_product['price'] = price[item_id]
            list_product['quantity'] = 1
            flag = 0
            result.append(list_product)
            list_product = {}
        print(result)
        flag = 0

    except:
        print('item not found')
    count = int(count) + 1
    count = str(count)

    return result

@app.route('/')
def index():
    """Video streaming home page."""
    return render_template('index.html')

def gen():
    """Video streaming generator function."""
    while True:
        rval, frame = video.read()
```

```
        # print(rval,frame)
        cv2.imwrite('t.jpg', frame)
        yield (b'--frame\r\n'
               b'Content-
Type: image/jpeg\r\n\r\n' + open('t.jpg', 'rb').read() + b'\r\n')

@app.route('/video_feed')
def video_feed():
    """Video streaming route. Put this in the src attribute of an img tag."""
    # print(Response(gen(),mimetype='multipart/x-mixed-replace; boundary=frame'))
    return Response(gen(),mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/btn_new_bill',methods=['GET', 'POST'])
def btn_new_bill():
    print('comes hre')
    if request.method == 'POST':
        data = clear()
        print(data)

    return data

@app.route('/item_data', methods=['GET', 'POST'])
def item_data():

    if request.method == 'POST':
        print('comes hre')
        rval, frame = video.read()
        cv2.imwrite(filename='test1.jpg', img=frame)
        print("Image saved!")
        # Get the file from post request

        res = item_data_price()

    return jsonify(result=res)

if __name__ == '__main__':
    app.run(debug=True)
```

2.KIRANA_WEB/static/js/main_19.js

```
 $(document).ready(function () {

     // Predict
     $('#btn-predict').click(function () {

         // Show loading animation
        console.log('comes here');
         var i = 1;
         var j = 2;

         var final_cost = 0;
```

```
        // Make prediction by calling api /predict
        $.ajax({
            type: 'POST',
            url: '/item_data',
            data: 'data_click',
            contentType: false,
            cache: false,
            processData: false,
            async: true,
            success: function (resp) {
                // Get and display the result
                console.log('comes here');
                console.log(resp.result);

                $("td").remove();
                $("#total").remove();

                for( i=0; i<resp.result.length; i++)
                {
                    var total_price = resp.result[i]['price'] * resp.result[i]['quant
ity'];
                    final_cost = Number(final_cost + total_price);
                    $("#item_data").append("<tr><td scope='row' >" + Number(i+1) + "<
/td><td>" + resp.result[i]['item_id'] + "</td><td>" + resp.result[i]['item_name'] +  "
" + resp.result[i]['item_size'] + "</td>" + "<td>" + resp.result[i]['quantity'] + "</td
><td>" + resp.result[i]['price'] + "</td><td>" + total_price + "</td></tr>");

                }
                $("#total_price").append("<th id='total'>Total Cost :" + Number(final
_cost) + "</th>");
            },
        });
    });
 });
```

3.KIRANA_WEB/static/js/new_bill_3.js

```
$(document).ready(function () {

    // Predict
    $('#btn_new_bill').click(function () {

        // Show loading animation
        console.log('comes here');

        // Make prediction by calling api /predict
        $.ajax({
            type: 'POST',
            url: '/btn_new_bill',
            data: 'data_click',
            contentType: false,
```

```
            cache: false,
            processData: false,
            async: true,
            success: function (resp) {
                // Get and display the result
                $("td").remove();
                $("#total").remove();
                console.log('comes here');
                console.log(resp);
            },
        });
    });
});
```

4.KIRANA_WEB \templates\index.html:

```
<html>
    <head>
        <title>Kirana Product Billing</title>
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-
bootstrap/4.3.1/css/bootstrap.min.css">
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.10.0-11/css/all.min.css">
        <script src="https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"></script>
      </head>
      <style>
        body {
            background-
image: url("https://s3.envato.com/files/240683905/images/flat.png");
                background-repeat:no-repeat;
               background-size:cover;
        }
        table#t01 {
        width: 100%;
        background-color:darkgray;
        }
        </style>
    <body>

        <div class="jumbotron" style="text-align: center">
            <h1>KIRANA  PRODUCT  BILLING </h1>
        </div>

        <div class="container" style="text-align:center">
            <form id="upload-file" method="post" enctype="multipart/form-data">
                <!-
- <input type="file" name="image" id="imageUpload" accept=".png, .jpg, .jpeg"> -->
                <img src="{{ url_for('video_feed') }}">
            </form>
            <br><br>
            <div style="text-align: center">
```

```
                <button id="btn-predict" class="btn btn-primary btn-
lg">Predict!</button>
                <button id="btn_new_bill" class="btn btn-primary btn-
lg">New Bill</button>
            </div>
            <br><br><br>
            <table  id="t01" class="table">
                <thead class="thead-dark">
                  <tr>
                    <th scope="col">Sr. No</th>
                    <th scope="col">Product ID</th>
                    <th scope="col">Name of Product</th>
                    <th scope="col">Quantity</th>
                    <th scope="col">Cost of Product</th>
                    <th scope="col">Final Cost of Product</th>
                  </tr>
                </thead>
                <tbody style="font-weight:900" color="Red" id="item_data">
                </tbody>
                <thead class="thead-dark" id="total_price">
                </thead>
              </table>

              <br><br><br><br><br><br><br><br><br>
        </div>

    <script src="{{ url_for('static', filename='js/main_19.js') }}" type="text/javascri
pt"></script>
    <script src="{{ url_for('static', filename='js/new_bill_3.js') }}" type="text/javas
cript"></script>
    <!-
- <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.1/jquery.slim.min.js">
</script> -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.15.0/umd/popper.min
.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/twitter-
bootstrap/4.3.1/js/bootstrap.min.js"></script>
    </body>
</html>
```

Screen shots of “kirana Product billing” website:

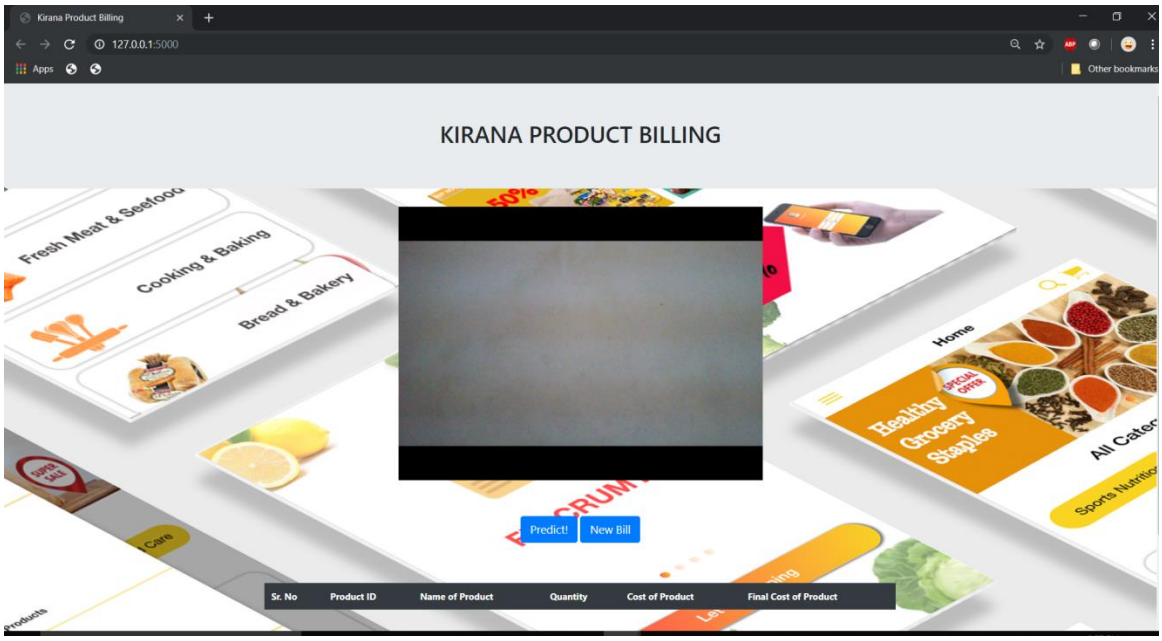


Fig 1. kirana product billing website

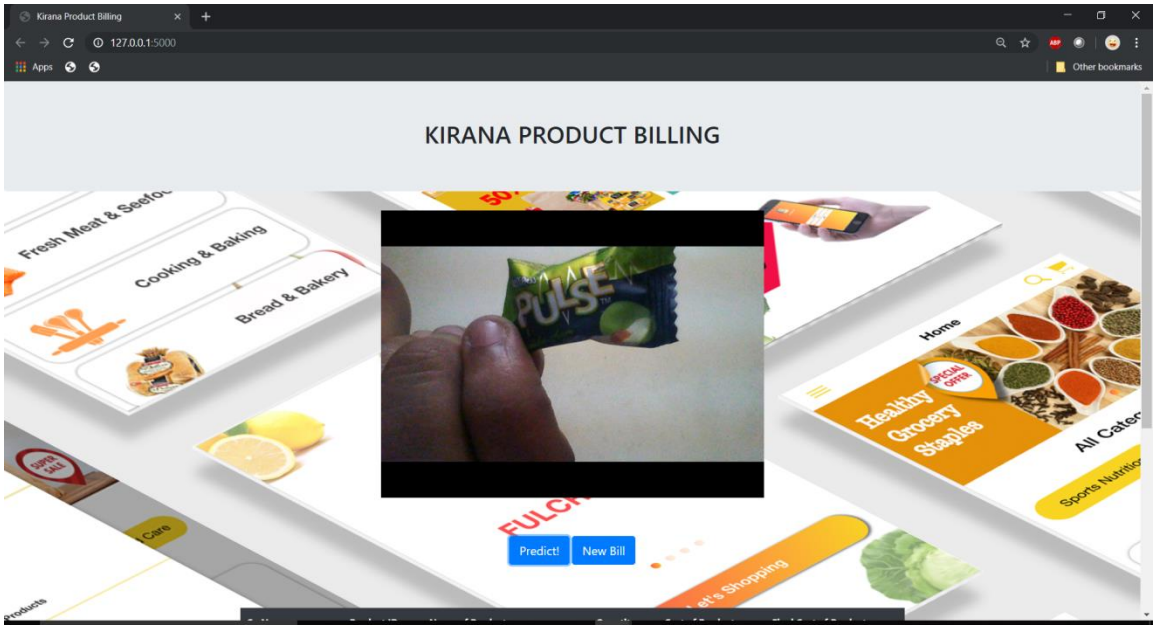


Fig 2. Taking picture of object to bill in kirana product billing website

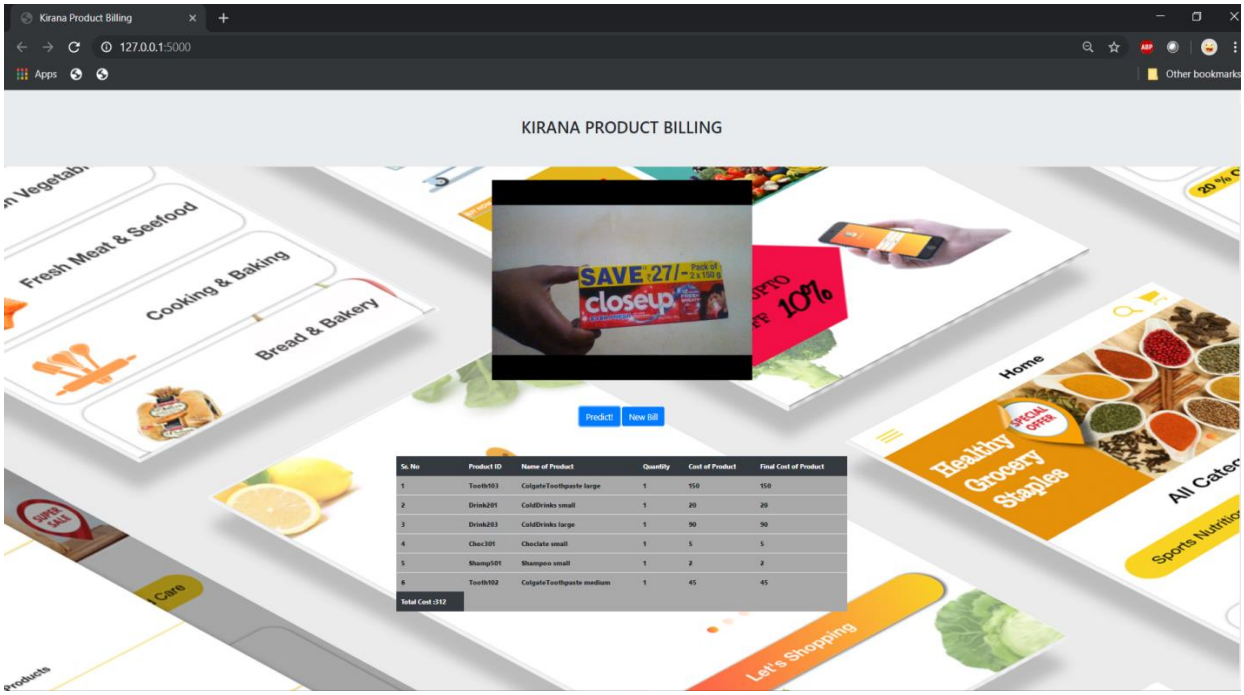


Fig 3. Taking picture of another object and its attribute are displayed in table form

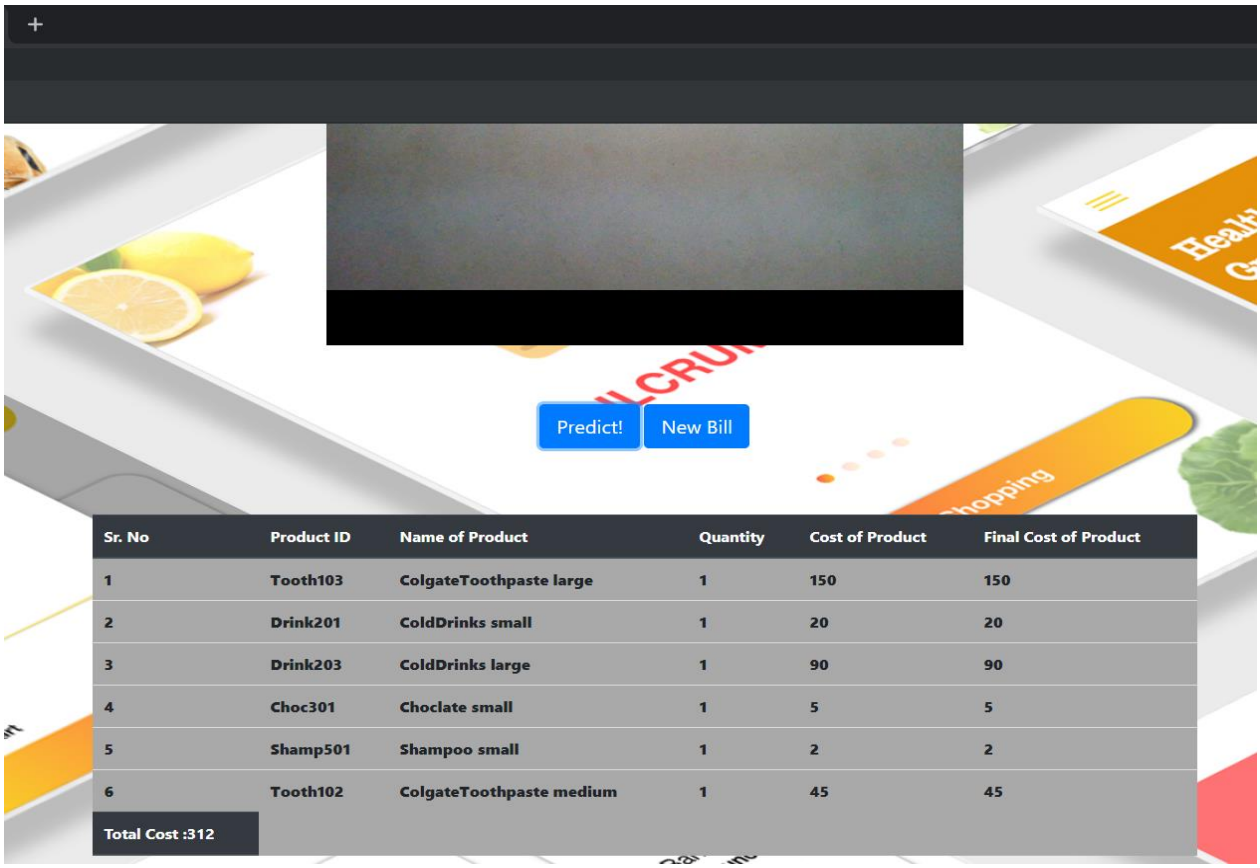


Fig 4. Total cost is getting update as products are added in kirana product billing website

Problem statement:

Build a system that allows automatic detection of product using camera. The detection of product must be with respect to the size of the product, type of product and automatically take the cost of product to make a bill of materials at checkout. This has to be done in real-time without sending the data to cloud for processing as some of these stores can be in remote areas with intermittent connectivity and Kirana do not want customers to wait due to latency issues for connectivity**.**

1.Introduction/Understanding the problem statement:

Excess time and resources required to build or perform updations in an existing application environment creates a problem for the organizations and also is a very tedious task.

Some of the problems faced in the process are:

- The plethora of organizational and human resources being wasted to perform a simple task.
- Stores in remote areas with intermittent connectivity wants customers to wait due to latency issues for connectivity.

How does "Kirana Product Billing" Website solve the above-mentioned problems:

We aim at building a web interface "Kirana Product Billing" which facilitates automatic detection of product using camera and display product attributes like product-id, product-name, quantity, cost etc.. which is achieved by training a machine learning models for particular products.

The idea is to minimize human interface and automate the process so as to increase the efficiency in managing and deploying the edge computing.

The implementation of this website will solve the following problems:

- Reduce the time required to check the cost of particular products.
- Reduce the human work force at billing counter.
- Reduce the time required for billing and calculating total amounts.

2. Details of technology used:

The proposed solution is leveraging many new technologies and framework to provide the user with a quick and seamless user experience.

IBM Watson Visual Recognition Service to train ML model:

The IBM Watson Visual Recognition service uses deep learning algorithms to identify scenes and objects in images that you upload to the service. We can create and train model to identify subjects that suit our needs.

Frontend (Web Portal):

A Web Portal where the user can interact with their configuration using a web browser. The web portal leverages technologies used in web engineering.

HTML:

Hypertext Markup Language is the standard markup language for documents designed to be displayed in a web browser

Bootstrap:

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and JavaScript-based design templates for typography, forms, buttons, navigation and other interface components

Jquery:

jQuery is a JavaScript library designed to simplify HTML DOM tree traversal and manipulation, as well as event handling, CSS animation, and Ajax.

AJAX:

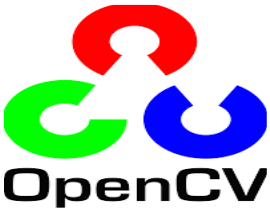
AJAX stands for Asynchronous JavaScript and XML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script

Backend:

Flask:

Flask is a micro web framework written in Python. It is classifie as a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation,or any other components where pre-existing third-party libraries provide common functions.

OpenCV:



OpenCV-Python. OpenCV-Python is a library of Python bindings designed to solve computer vision problems. OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax

3. Required Software /Hardware:

HARDWARE:

- i5 8th Generation Processor
- 8GB RAM
- 1TB Hard Disk
- Monitor
- Camera
- CPU to train model (we used IBM Watson studio)

SOWTWARE:

- Windows 10
- Python language
- Editor (Visual Studio Code)
- A Browser such as Chrome, Edge, Opera to navigate the Web Portal.
- Web Cam

3. Achieved Cost Saving:

REDUCED EXPENSE:

This application can be considered as cost effective, as cost of webcam is quite very low compared to cost of bar code reader machine.

USER FRIENDLY:

It is very easy to handle and also don't require human effort as webcam can be fixed at certain place and customers only need to keep items to be purchased facing barcodes towards webcam.

DURABILITY:

Webcam is more durable and even if some problem occurs, it is easy to fix compared to barcode reader machines.

COLLABORATIVE ENVIRONMENT:

This application can be used in shopping malls and small stores to automate billing process. It can also be used to develop similar application like in library, a book can be assigned to student using image scanning system.

4. Architecture:

A detailed architecture of the solution is stated below

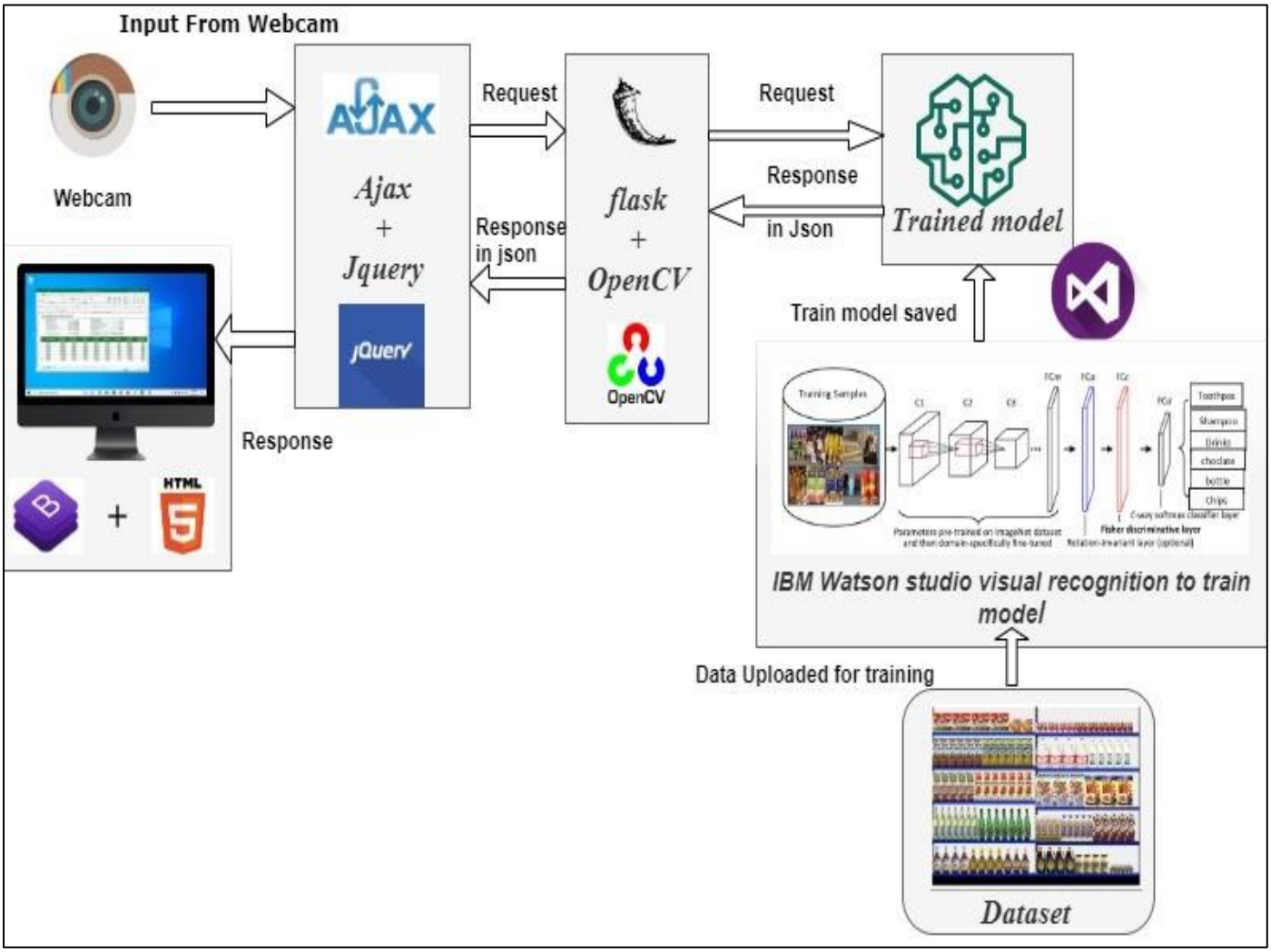
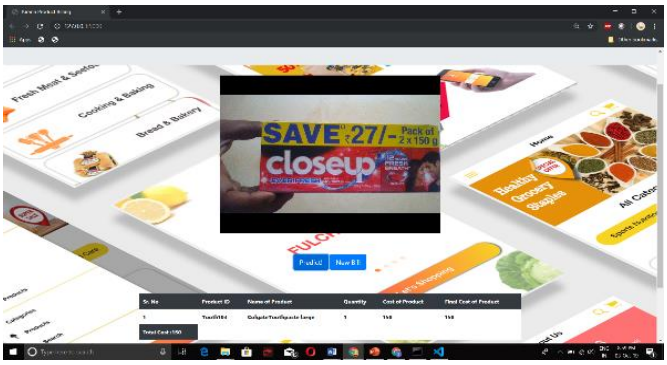


Fig. 5 : High-Level Architecture Diagram

5. Solution Brief Description:

WEBCAM: Take the real time product(datasets) infront of webcam for detection (You can also use desktop/Laptop camera for product detection).



Different Products (datasets) used for train the model for detection:-

1). Water Bottle of different size (small, medium, large),

2). Chips of different size (small, medium),

3). Chocolates of different size (small, medium, large),

4). Cold Drinks of different size (small, medium, large),

5). Shampoo of different size (small, medium, large),

6). Toothpaste of different size (small, medium, large).

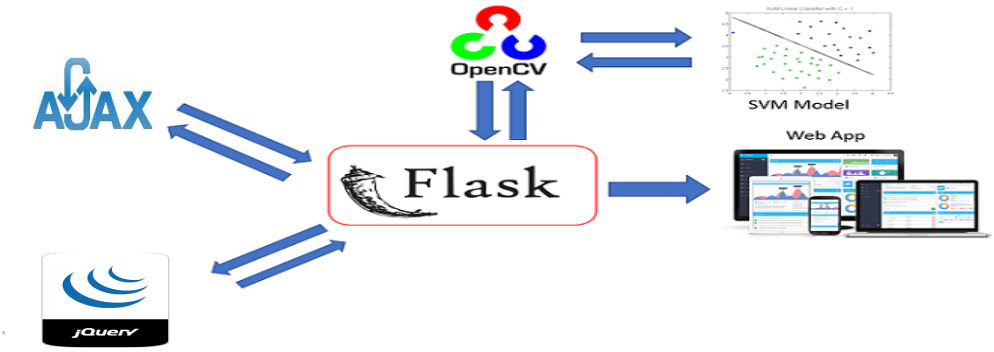
Requirements for Web-Camera Devices:

1. resolution 640×480 or higher;
2. Ability to change the focus (in **MANUAL FOCUS** mode). Usually such web camera devices have a ring around lenses which can be used to adjust the focus. Web camera devices with ability to change focus: Microsoft LifeCam VX-1000, Logitech QuickCam Pro 9000, Labtec Webcam 2200 and any other with manual focus support.
3. Manual focus is required to get sharp image of the object. With autofocus web camera devices usually are not able to take sharp photo of the object.
4. you should put the object with proper distance before the web-camera to capture entire object make sure you have a sharp image and the barcode is not cut on the picture.

AFTER IMAGE IS CAPTURED:

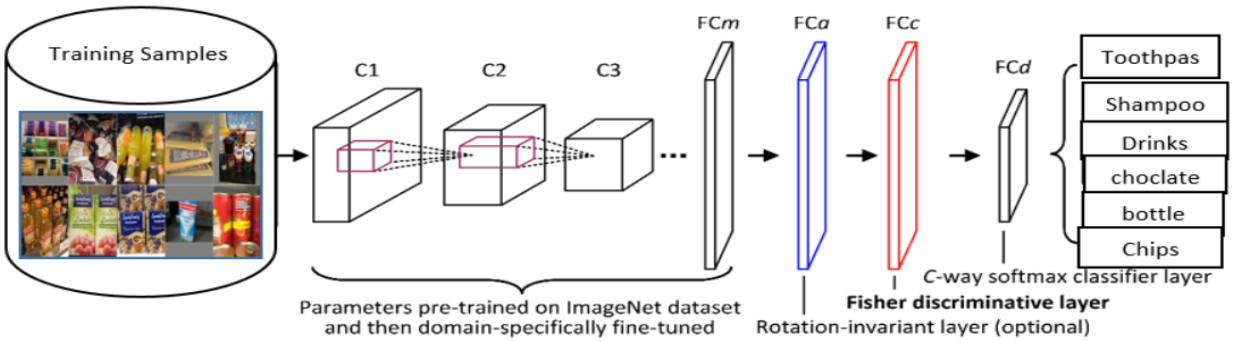
- Input from webcam will be fetched to Ajax & JQuery and will request for the JSON file.

- ***jQuery*** provides several methods for ***AJAX*** functionality. With the ***jQuery AJAX*** methods, we can request text, HTML, XML, or JSON from a remote server using both HTTP Get and HTTP Post - And we can load the external data directly into the selected HTML elements of our web page.
- Ajax and JQuery will request the same with Flask(backend) and OpenCV(open source computer vision).



- Then flask will **request for an API** from Trained model
- Sent image will be compared with model weight, output will be sent in the JSON format to flask it will response to jQuery and ajax with this out put wich will be displayed in the form of table on website.

HOW DATASETS ARE TRAINED?



fig

- Training samples such as Toothpaste, Shampoo, Cold Drinks, Water Bottle etc.., of different size and parameters are sent to CNN model for training.
- The CNN is a type of Deep Neural Networks (DNN) that consists of many layers such as the Conv layers, Pooling layer, Flattening Layer and the fully-connected layer. It mainly used for image classification purposes.

- Once the model is trained, The trained model will send the **Response in JSON** format to Flask
- Flask will send the same Response in JSON format to Ajax & JQuery.

- **AJAX** methods request XML, or JSON from a remote server using both HTTP Get and HTTP Post.

- Once the Ajax Receives the response from flask, it will send the output to our website named **KIRANA PRODUCT BILLING** in the form of table.

- Table contains following attributes :

1. Product ID
2. Name of the product
3. Quantity of the product
4. Cost of the product
5. Final cost of the product which is calculated using formula (cost X quantity)
6. Total cost

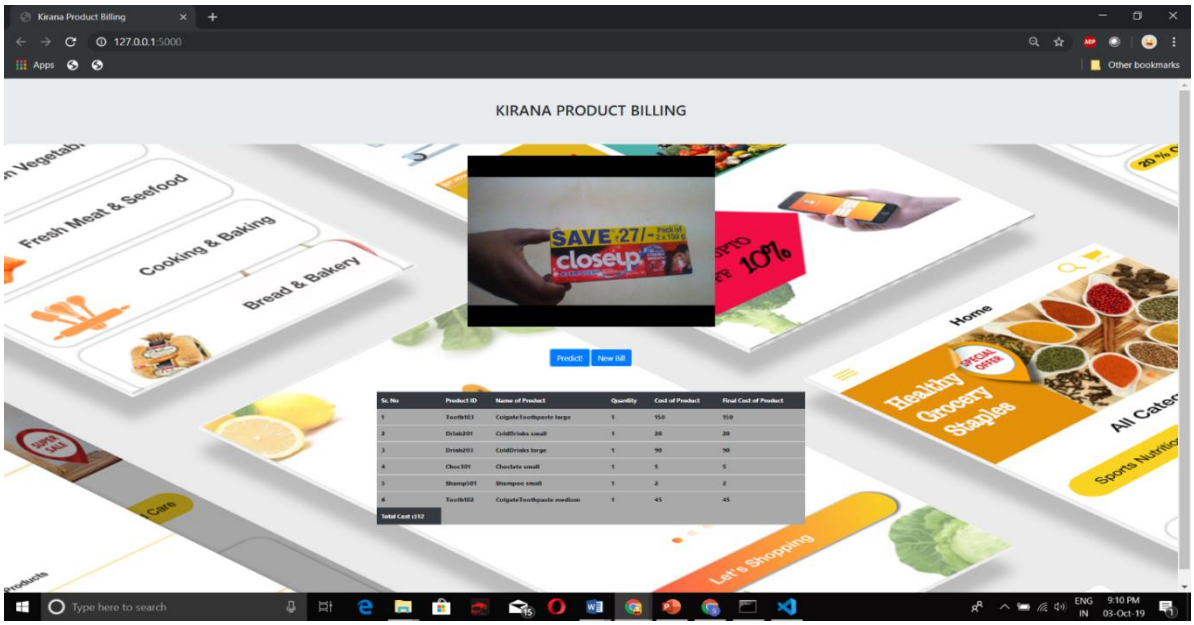


Fig 6

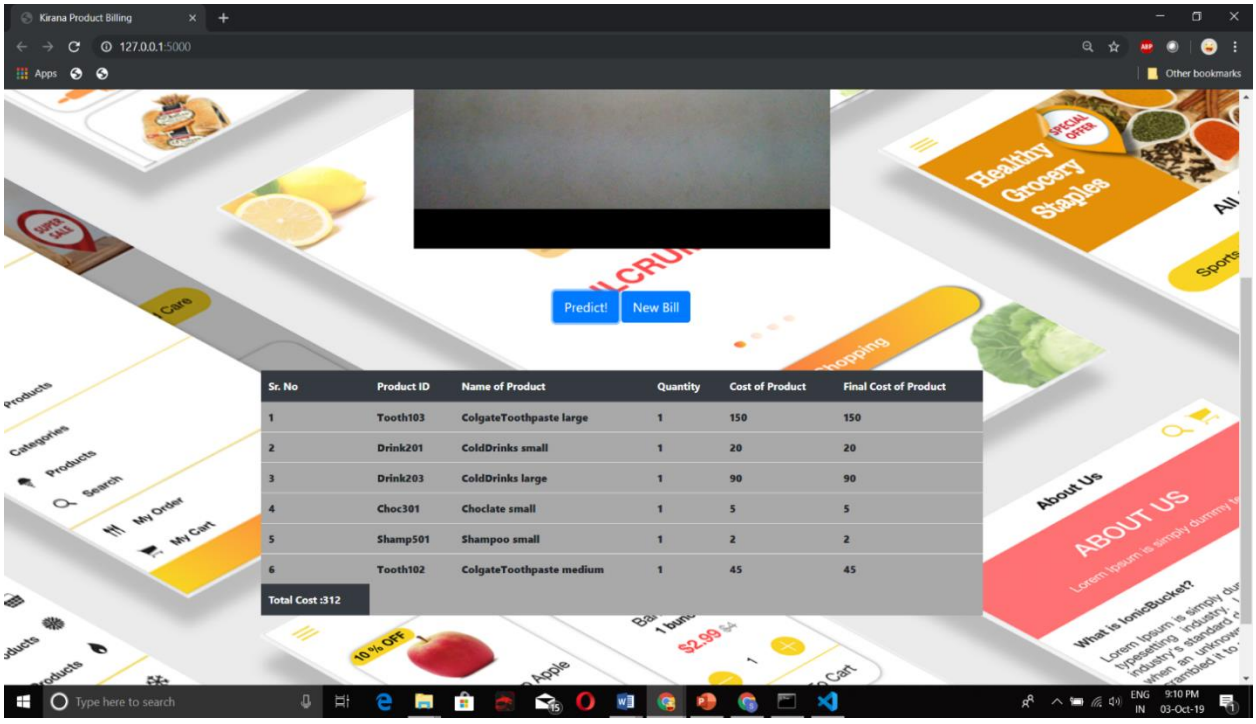


Fig 7

Fig 6 and 7 displaying of object attributes product ID, name of the product, quantity, cost of product, final cost and total cost

6. Scope of Automation:

-General Stores

General stores just by having one computer without having any QR-scanner or any other device they can able to adopt modern billing method through our website.

-Large warehouses

Large warehouses have a huge amount of products to be delivered to it's right address. Before that, each product is being scanned to check the to and from address. With today's existing technology, it takes some time and the delivery get's late. This website reduces the scanning time and delivery will be much quicker.

-medical store

In medical line, some seconds decides life and death of a patient. The major cause of time loss is the long queues at the medical store which prevents the medicals supply reach to the patient in time. This website will help to deliver the medical supply faster then already available QR scanner.

-virtually helps to find the product

Several times one came across a product image online which he has no idea about. our website will help user to find out all about the product virtually by scanning it.

7. Conclusion:

Menial task like provisioning of resources such as human resource, server, networking and hardware components can be really tedious and tiresome which also requires a lot of upfront investment for shop keepers. All of the above tasks slows down the process of billing and takes customer time. These problems can be overcome by leveraging edge computing platorms which can be used to create a scalable, reliable, secure and cost-effcient, speed environment with automation at every step. "Kirana product billing" helps shop keeper for fast calculation and billing of their products and utilize human source for other main tasks.