

**INVENTORY MANAGEMENT SYSTEM**

**FOR RETAILERS**

**NALAIYA THIRAN PROJECT BASED LEARNING ON**

**PROFESSIONAL READINESS FOR INNOVATION, EMPLOYABILITY AND ENTREPRENEURSHIP**

**A PROJECT REPORT**

|  |  |
| --- | --- |
| S R BALAJI | 410119106006 |
| M DEVENDHIRAN | 410119106011 |
| K GOPINATHAN | 410119106016 |
| V GUNA | 410119106017 |

**BACHELOR OF ENGINEERING IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**ADHI COLLEGE OF ENGINEERING AND TECHOLOGY**

Approved by AICTE, New Delhi, Permanent Affiliation Status by **Anna University, Chennai**– 600 026

**NOVEMBER 2022**

***Adhi College of Engineering And Technology***

Approved by AICTE, New Delhi, Permanent Affiliation Status by **Anna University, Chennai**– 600 026

# INTERNAL MENTOR: Mrs. AB. BANUMITHRA

Associate Professor

Department of Electronics and Communication Engineering Adhi College of Engineering and Technology,

Kanchipuram – 631 631

# INDUSTRY MENTOR Mr.VASUDEVA HANUSH

IBM

ABSTRACT

The advancements of computing technologies have resulted to cheaper mobile devices and access to various mobile applications in the market. These developments have also opened opportunities for micro and small enterprises to streamline their business processes without investing heavily on enterprise software licenses and hardware resources. This paper explores the convergence of a mobile point of sales (POS) application and a cloud computing inventory management system. However, one of the major problems in implementing a cloud computing system is the Internet speed in the Philippines.

Latency issues can greatly affect the productivity of micro and small businesses since majority of them are engaged in wholesale and retail trade. This can be solved by synchronizing the data coming from the native mobile POS through batch processing with the data of the inventory management system on the cloud in situations where unavailability of, limited, or slow network connection makes it impossible to perform real-time update of data. This paper will also discuss the architecture, data synchronization method, and the challenges and growth prospects of mobile and cloud computing.

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **TITLE** | **PAGE NO** |
|  | **ABSTRACT** |  |
| **1** | **INTRODUCTION** | **01** |
| **2** | **OBJECTIVE** | **02** |
| **3** | **IDEATION PHASE** | **03** |
| 3.1 Literature Survey |
| 3.2 Empathy Map |
| 3.3 Ideation |
| 3.4 Brainstorming |
| **4** | **PROJECT DESIGN PHASE 1** | **06** |
| 4.1 Proposed Solution |
| 4.2 Problem Solution Fit |
| 4.3 Solution Architecture |
| **5** | **PROJECT DESIGN PHASE 2** | **09** |
| 5.1 Customer Journey Map |
| 5.2 Solution Requirements |
| 5.3 Data Flow Diagrams |
| 5.4 Technology Stack |
| **6** | **PROJECT PLANNING PHASE** | **17** |
| 6.1 Prepare Milestone and Activity List |
| 6.2 Sprint Delivery Plan |
| **7** | **PROJECT DEVELOPMENT PHASE** | **20** |
| 7.1 Project Development - Delivery of Sprint - 1 |
| 7.2 Project Development - Delivery of Sprint - 2 |
| 7.3 Project Development - Delivery of Sprint - 3 |
| 7.4 Project Development - Delivery of Sprint - 4 |
| **8** | **CONCLUSION** | **30** |
| **9** | **REFERENCES** | **31** |

**INVENTORY MANAGEMENT SYTEM FOR RETAILERS**

1. **INTRODUCTION**

Retail inventory management is the process of ensuring you carry merchandise that shoppers want, with neither too little nor too much on hand. By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply.

In practice, effective retail inventory management results in lower costs and a better understanding of sales patterns. Retail inventory management tools and methods give retailers more information on which to run their businesses. Applications have been developed to help retailers track and manage stocks related to their own products. The System will ask retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application.

Once retailers successfully log in to the application they can update their inventory details, also users will be able to add new stock by submitting essential details related to the stock. They can view details of the current inventory. The System will automatically send an email alert to the retailers if there is no stock found in their accounts.  So that they can order new stock.

# OBJECTIVE:

The objective of the project is to provide an efficient inventory control whose main functionality apart from calculating the inventory include predicting the requirement for the next order and also there is a “Special Occasion” then accordingly the manager selects the particular occasion and extra requirements is added to the next issuing order to the vendors which need to be approved by the manager. The project also aims to keep track of the shelf lie of resources.

The success criteria depend on

* The accuracy in maintaining the inventory levels
* The accuracy in updating the inventory.
* Each of user when it comes to updating inventory levels

# 3. IDEATION PHASE

* 1. **3.1 LITERATURE SURVEY:**

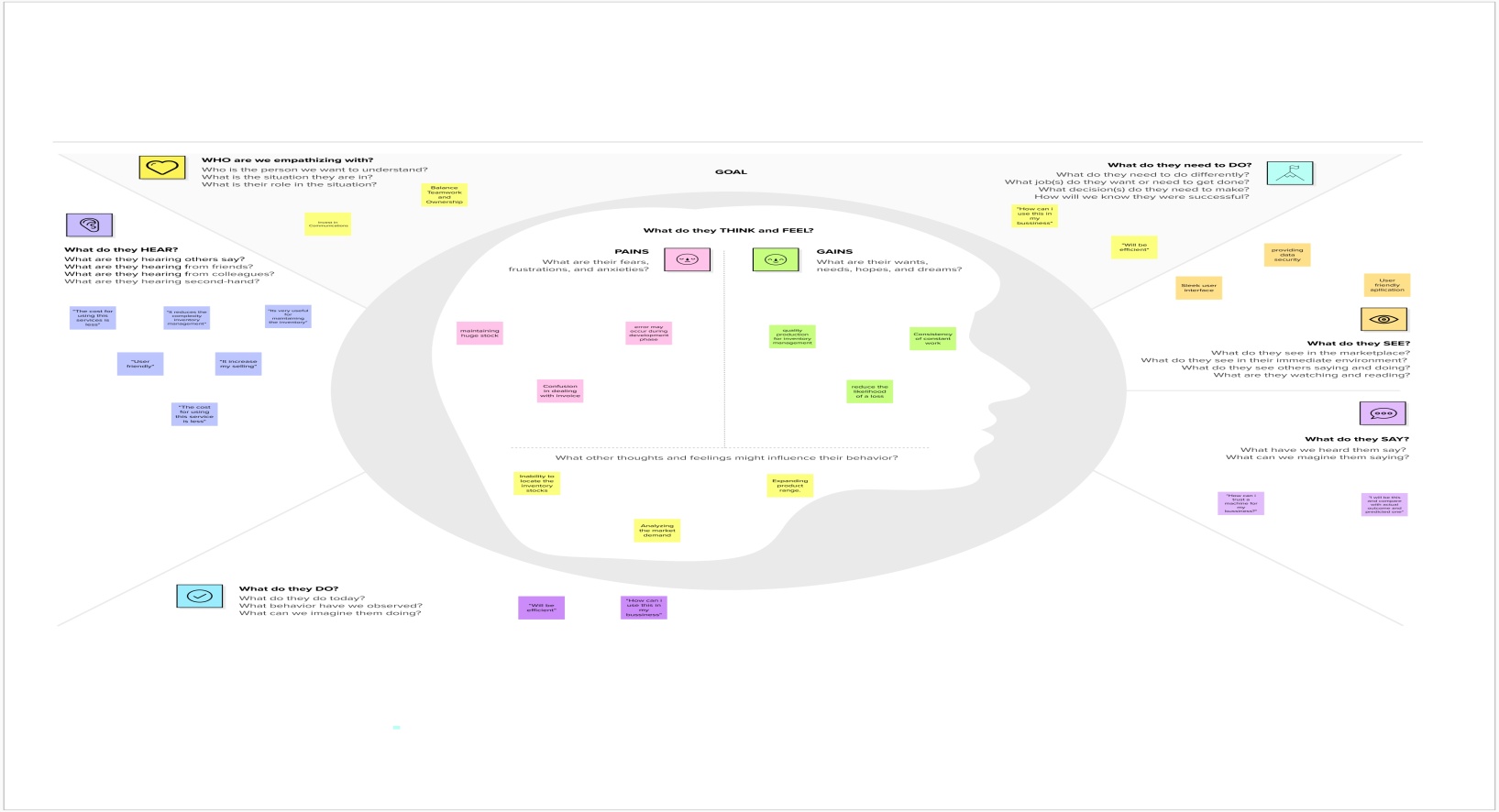
The advancements of computing technologies have resulted to cheaper mobile devices and access to various mobile applications in the market. These developments have also opened opportunities for micro and small enterprises to streamline their business processes without investing heavily on enterprise software licenses and hardware resources. This paper explores the convergence of a mobile point of sales (POS) application and a cloud computing inventory management system. However, one of the major problems in implementing a cloud computing system is the Internet speed in the Philippines.

Latency issues can greatly affect the productivity of micro and small businesses since majority of them are engaged in wholesale and retail trade. This can be solved by synchronizing the data coming from the native mobile POS through batch processing with the data of the inventory management system on the cloud in situations where unavailability of, limited, or slow network connection makes it impossible to perform real-time update of

data. This paper will also discuss the architecture, data synchronization method, and the challenges and growth prospects of mobile and cloud computing.

**3.2Empathy Map:**

­­­­­­­­



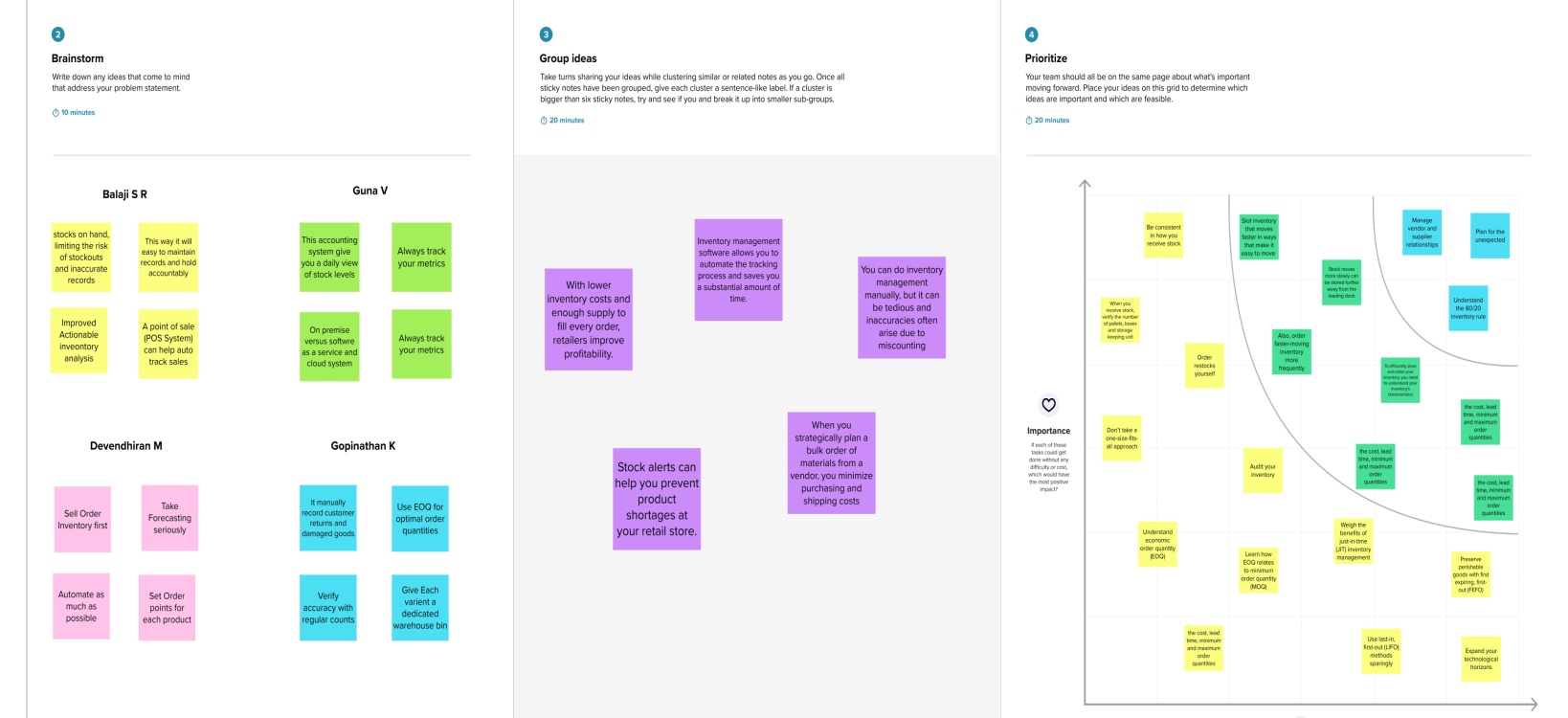
**Fig 1 Empathy Map**

3.3 **Ideation (Inventory Problem Statement):**



**Fig 2 Ideation Phase**

**3.4 Brainstorming:**



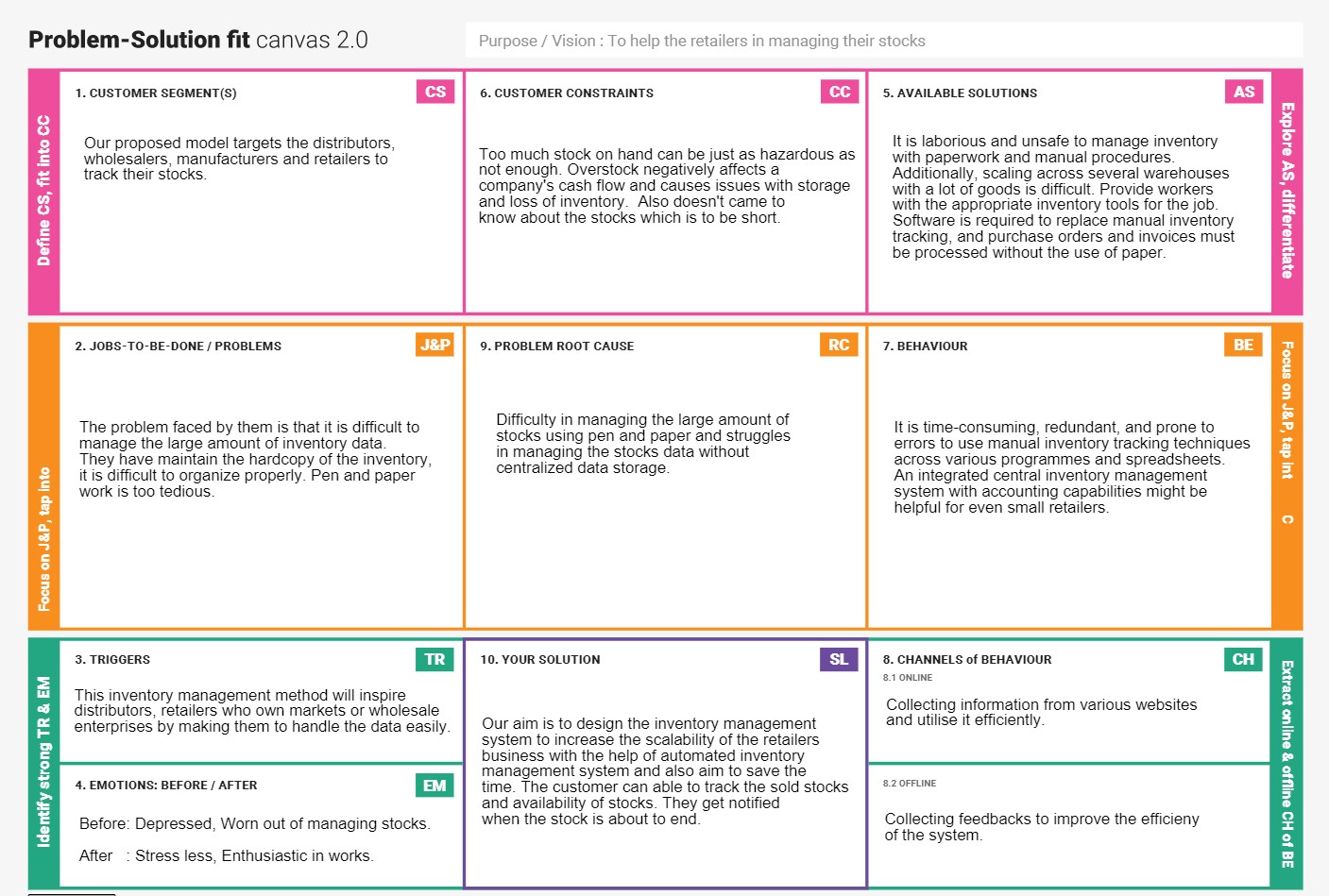
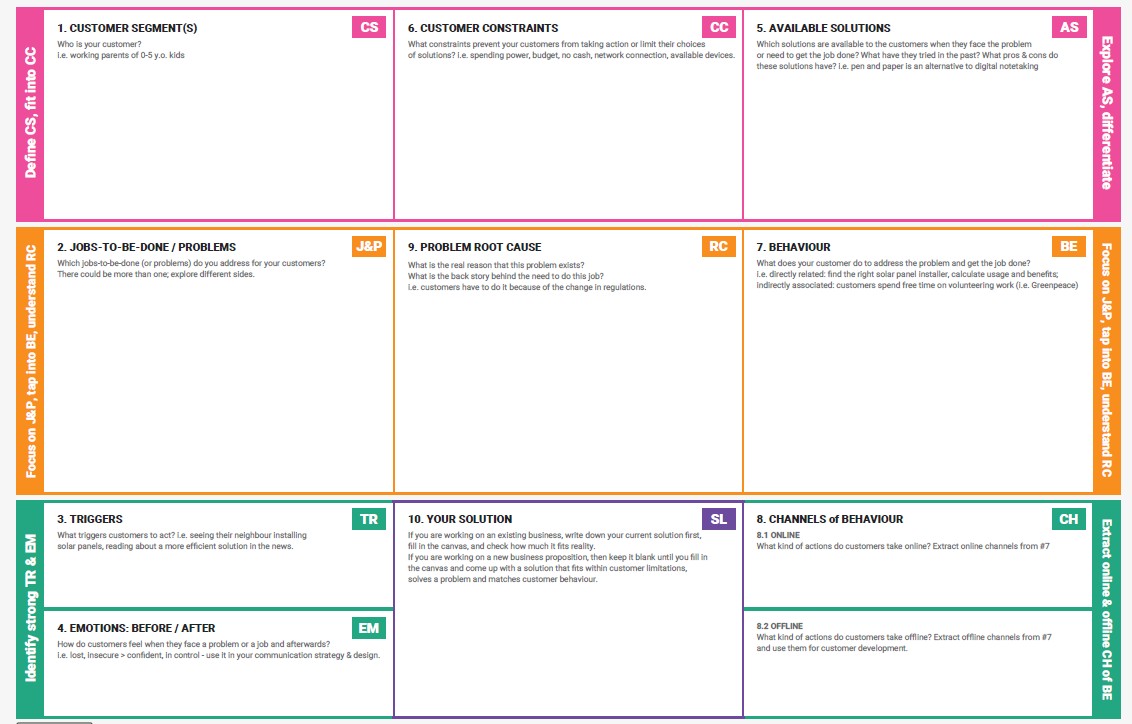
**Fig 3 Brainstroming**

# 4. PROJECT DESIGN PHASE 1

**4.1 Proposed Solution:**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Parameter** | **Description** |
|  | Problem Statement (Problem to be solved) | * The problem faced by the company is they do not have any systematic system to record and keep their inventory data. |
|  | Idea / Solution description | * The right inventory management platform can automate process, improve inventory practices and enhance customer experiences. |
|  | Novelty / Uniqueness | * Track inventory across multiple locations, automatically manage reorder points, forecast demand and plan production and distribution. |
|  | Social Impact / Customer Satisfaction | * Inventory management helps you mange the customer experience when it comes to product returns. * An inventory management system can track important data concerning returned items and giving you the option to maintain additional inventory levels that mirror your return rates. |
|  | Business Model (Revenue Model) | * Balance demand and supply, integrate financial and operational planning, and link high-level strategic plans with mid and long-term business plans. |
|  | Scalability of the Solution | * To increase the scalability of your business, you should use an automated inventory management system for inventory tracking. |

**4.2. Problem Solution Fit:**



**Fig 4 Problem Solution Fit**

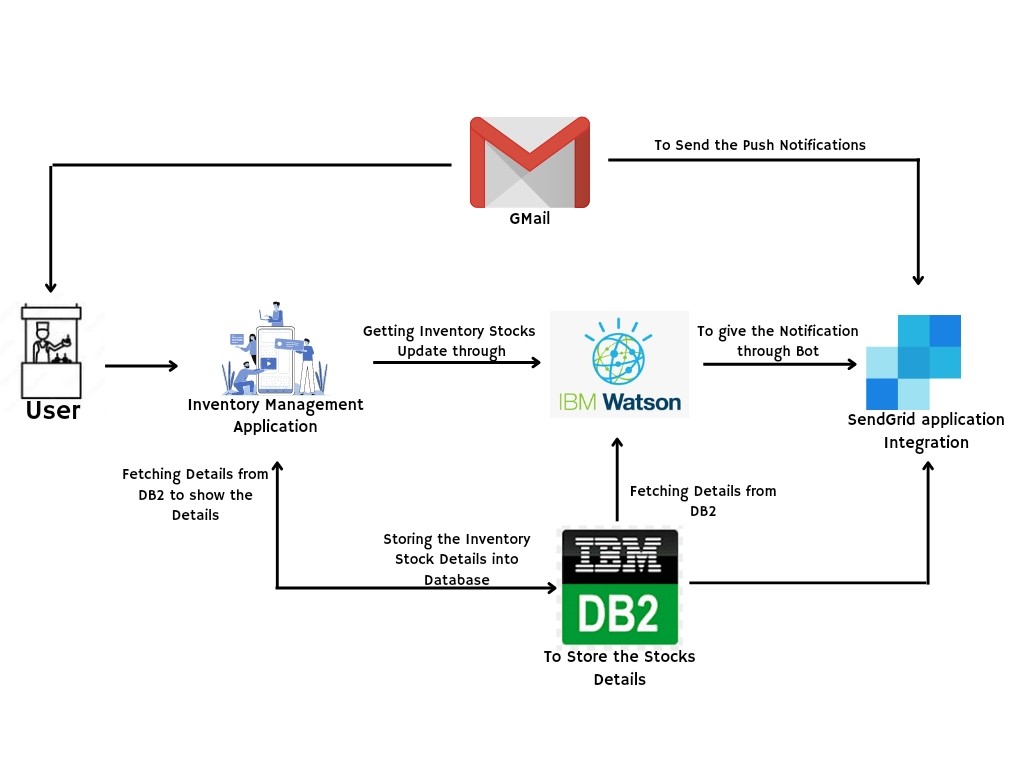
**4.3. Solution Architecture**

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

* Find the best tech solution to solve existing business problems.
* Describe the structure, characteristics, behaviour, and other aspects of the software to project stakeholders.
* Define features, development phases, and solution requirements.
* Provide specifications according to which the solution is defined, managed, and delivered.

# Example - Solution Architecture Diagram:

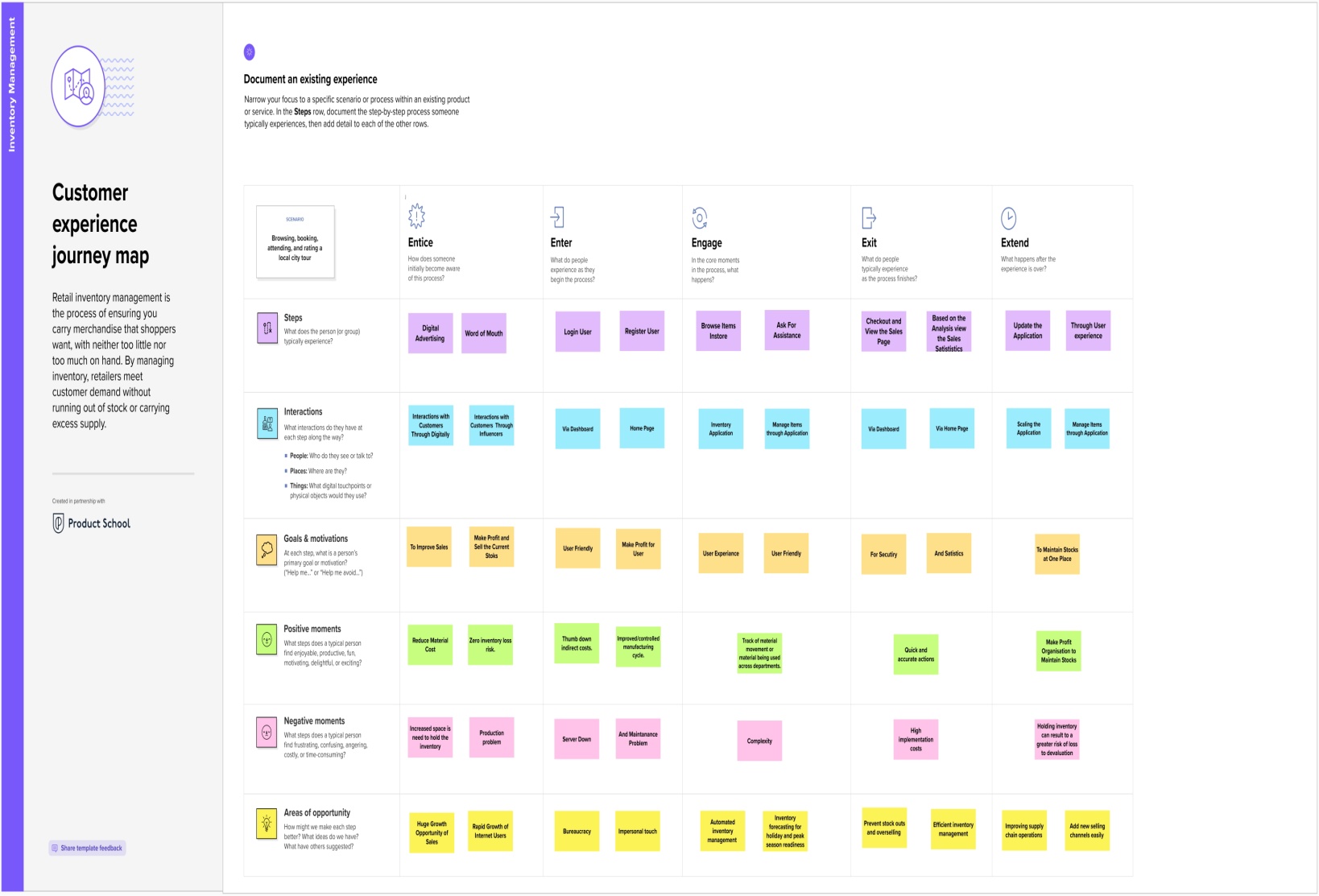
**Inventory Management System For Retailers**



**Fig 5, Solution Architecture**

**5 . PROJECT DESIGN PHASE 2**

* 1. **­5.1 Customer Journey**



**Fig 6 Customer Journey**

5.2 **Solution Requirements**

**Functional Requirements**

Following are the functional requirements of the proposed solution.

|  |  |  |
| --- | --- | --- |
| **FR No.** | **Functional Requirement (Epic)** | **Sub Requirement (Story / Sub-Task)** |
| FR-1 | **User Registration** | Registration through Form  Registration through Gmail |
| FR-2 | **User Confirmation** | Confirmation via Email |
| FR-3 | **Business regulations** | Many needs may fit under this category |
| FR-4 | **Product management** | Easily track product information  Quickly produce reports for single or multiple sold products |
| FR-5 | **Audit Monitoring** | The technique of tracking crucial data is known as audit tracking |
| FR-6 | **Historical Data** | Specify the amount of storage you need to handle this expansion |

**­­**

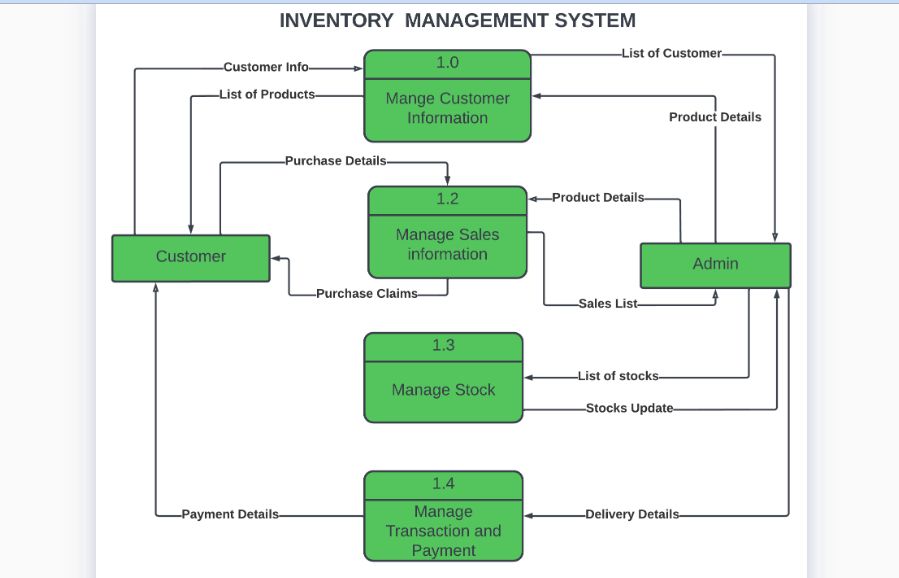
**Non-functional Requirements:**

Following are the non-functional requirements of the proposed solution.

|  |  |  |
| --- | --- | --- |
| **FR No.** | **Non-Functional Requirement** | **Description** |
| NFR-1 | **Usability** | * User-friendly and overall satisfaction of the user while using this website. * Backups for database are available. |
| NFR-2 | **Security** | * The security requirements deal with the primary security. only authorized users can access the system with user name and password of administrator . |
| NFR-3 | **Reliability** | * The system should use a reliable database management system. * All database updates must be dependable. |
| NFR-4 | **Performance** | * Easy tracking of records and updating can be done . |
| NFR-5 | **Availability** | * The software will be available only to administrator of the organization and the product as well as customer details will be recorded by him. |
| NFR-6 | **Scalability** | * The ability of a system to handle a growing amount of work |
| NFR-7 | **Portability** | * The software is a web based application and is built in PYTHON and SQL so it is platform independent and is independent of operating system. |

**5.3 Data Flow Diagrams:**

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows with an system. A neat and clear DFD can depict the right amount of the system requirement graphically. It show how data enters and leaves the system, what changes the information, and where data is Stored System. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



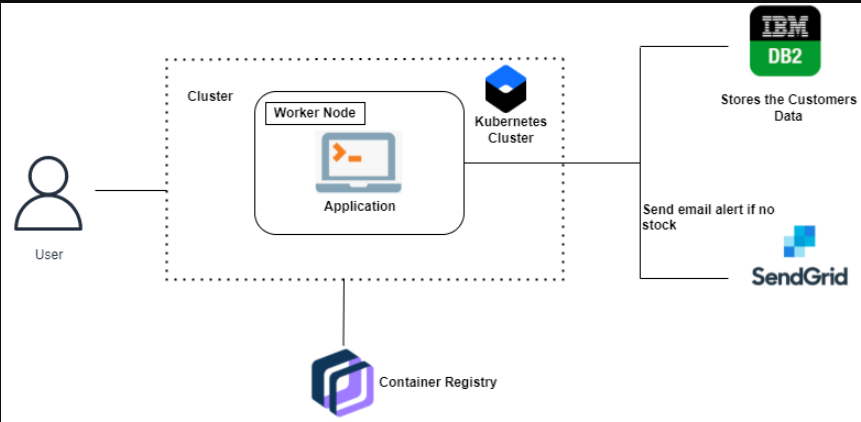
**Fig 7 Data Flow Graphs**

**5.4 Technology Stack:**

**Technical Architecture:**

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table 2

**Order processing during pandemics for offline mode**

****

**Fig 8 Technical Architecture**

**Table-1 : Components & Technologies:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Component** | **Description** | **Technology** |
|  | User Interface | How user interacts with application e.g.  Web UI, Mobile App, Chatbot etc. | HTML, CSS, JavaScript, Pyhton |
|  | Application Logic-1 | Logic for a process in the application | IBM Watson STT service |
|  | Application Logic-2 | Logic for a process in the application | IBM Watson Assistant |
|  | Database | Data Type, Configurations etc. | MySQL. |
|  | Cloud Database | Database Service on Cloud | IBM DB2, IBM Cloudant etc. |
|  | File Storage | File storage requirements | IBM Block Storage or Other Storage Service or Local Filesystem |
|  | External API-1 | Purpose of External API used in the application | SendGrid |
|  | Infrastructure (Server / Cloud) | Application Deployment on Local System / Cloud | Kubernetes, Docker, IBM cloud |

**Table-2: Application Characteristics:**

| **S.No** | **Characteristics** | **Description** | **Technology** |
| --- | --- | --- | --- |
|  | Open-Source Frameworks | Flask is used for interaction and connection with application | Python Flask |
|  | Scalable Architecture | Presenting tier: User interface for login and updating stocks. | HMTL, CSS, Flask, IBM DB2 |
|  | Availability | Availability can be made by using cloud. | Kubernetes, Docker |
|  | Performance | Performance of the application can be improved by adding the containers in cloud DB | Kubernetes, Docker |

# 6.PROJECT PLANNING PHASE

* 1. **6.1 Prepare Milestone and Activity List:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Functional Requirement (Epic)** | **User Story Number** | **User Story / Task** | **Story Points** | **Priority** | **Team Members** |
| Sprint-1 | Registration,Login Page and Connect wiih IBM db2 | USN-1 | As a user, I can register for the application by entering my email, password, and confirming my password. | 2 | High | Balaji, Devendhiran, Guna,  Gopi |
| Sprint-2 | Dashboard and further functionality Implementaion | USN-2 | As a user, to view the Stocks and current invenoty stock | 1 | High | Balaji, Devendhiran, Guna,  Gopi |
| Sprint-3 | Connect with sendgrid and other third party  Integration | USN-3 | As a user, to receive mails and current stock information through mail | 2 | Low | Balaji, Devendhiran, Guna,  Gopi |
| Sprint-4 | Create a Image in docker and contanirize a  Application | USN-4 | As a user, Can finalise the application and Test it | 2 | High | Balaji, Devendhiran, Guna,  Gopi |
| Sprint-5 | Deploy in IBM cloud Kubernates Cluster | USN-5 | As a user, Depolyment stage and scalling the application | 1 | High | Balaji, Devendhiran, Guna,  Gopi |

* 1. **6. 2 Sprint Delivery Plan:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Total Story Points** | **Duration** | **Sprint Start Date** | **Sprint End Date (Planned)** | **Story Points Completed (as on Planned End Date)** | **Sprint Release Date (Actual)** |
| Sprint-1 | 20 | 6 Days | 24 Oct 2022 | 29 Oct 2022 | 29 Oct 2022 | 29 Oct 2022 |
| Sprint-2 | 20 | 6 Days | 31 Oct 2022 | 05 Nov 2022 | 29 Oct 2022 | 29 Oct 2022 |
| Sprint-3 | 20 | 6 Days | 07 Nov 2022 | 12 Nov 2022 | 29 Oct 2022 | 29 Oct 2022 |
| Sprint-4 | 20 | 6 Days | 14 Nov 2022 | 19 Nov 2022 | 29 Oct 2022 | 29 Oct 2022 |
| Sprint-5 | 20 | 6 Days | 16 Nov 2022 | 19 Nov 2022 | 29 Oct 2022 | 29 Oct 2022 |

# 7. PROJECT DEVELOPMENT PHASE

**Project Development - Delivery of Sprint 1**

* 1. **CODE**:

from crypt import methods  
import re  
from wsgiref import validate  
from flask import Flask, render\_template,redirect,request, session  
import ibm\_db  
  
app = Flask(\_\_name\_\_)  
  
app.secret\_key = 'super secret key'  
app.config['SESSION\_TYPE'] = 'filesystem'  
  
#Database Connection  
conn = ibm\_db.connect("DATABASE=bludb;HOSTNAME = b0aebb68-94fa-46ec-a1fc-1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud;PORT = 31249;SECURITY = SSL;SSLSererCertificate = DigiCertGlobalRootCA.crt;UID = rnj93429;PWD = kzoIIKcwZ05e4tyW","","")  
  
  
# Dashboard Details class  
class dashboardDetails:  
    def \_\_init\_\_(mainDetails,totalStock,currentInventoryStock,lowStocks,nonMovableStocks):  
        mainDetails.totalStock = totalStock["1"]  
        mainDetails.currentInventoryStock = currentInventoryStock["1"]  
        mainDetails.lowStocks = lowStocks["1"]  
        mainDetails.nonMovableStocks = nonMovableStocks["1"]  
         
class stockDetails:  
    def \_\_init\_\_(stockDetails,stockId,stockName,stockQty,costPrice,sellingPrice,status):  
        stockDetails.stockId = stockId  
        stockDetails.stockName = stockName  
        stockDetails.stockQty = stockQty  
        stockDetails.costPrice = costPrice  
        stockDetails.sellingPrice = sellingPrice  
        stockDetails.status = status  
  
  
@app.route("/")  
def home():  
    return render\_template("inventoryHome.html")  
  
@app.route("/register")  
def register():  
    return render\_template("inventoryRegisterUser.html")  
  
@app.route("/RegisterUsr",methods=['GET','POST'])  
def registerUsr():  
    msg = ''  
    if request.method == 'POST':  
        storeName = request.form['storeName']  
        emailId = request.form['emailId']  
        password = request.form['password']  
        country = request.form['country']  
        state = request.form['state']  
        validationQuery = "SELECT \* FROM SHOPDETAILS WHERE EMAILID = ?"  
        validateStmt = ibm\_db.prepare(conn,validationQuery)  
        ibm\_db.bind\_param(validateStmt,1,emailId)  
        ibm\_db.execute(validateStmt)  
        account = ibm\_db.fetch\_assoc(validateStmt)  
        if account:  
            msg = "Account is Already exists !"  
        else:  
            query = "INSERT INTO  SHOPDETAILS (STORENAME,EMAILID,USRPASSWORD,COUNTRY,STATE) VALUES(?,?,?,?,?)"  
            stmt = ibm\_db.prepare(conn,query)  
            ibm\_db.bind\_param(stmt,1,storeName)  
            ibm\_db.bind\_param(stmt,2,emailId)  
            ibm\_db.bind\_param(stmt,3,password)  
            ibm\_db.bind\_param(stmt,4,country)  
            ibm\_db.bind\_param(stmt,5,state)  
            ibm\_db.execute(stmt)  
            msg = "Registration Successfull"  
            return render\_template("inventoryLogin.html",msg = msg)  
    elif request.method == "POST":  
            msg = "Please Fill the form"  
   
    return render\_template("inventoryRegisterUser.html",msg = msg)  
  
@app.route("/login")  
def login():  
    return render\_template("inventoryLogin.html")  
  
@app.route("/loginUsr",methods=["GET","POST"])  
def loginUsr():  
    msg = ''  
    if request.method == 'POST':  
        emailId = request.form['emailId']  
        password = request.form['password']  
        query = "SELECT \* FROM SHOPDETAILS WHERE emailId = ? AND usrpassword = ?"  
        stmt = ibm\_db.prepare(conn,query)  
        ibm\_db.bind\_param(stmt,1,emailId)  
        ibm\_db.bind\_param(stmt,2,password)  
        ibm\_db.execute(stmt)  
        account = ibm\_db.fetch\_assoc(stmt)  
        if account:  
            session['loggedin'] = True  
            session['id']=account["EMAILID"]  
            userid = account['EMAILID']  
            session['emailId'] = account['EMAILID']  
            msg = "Loggin Sucessfull"  
            return redirect("/dashboard")  
        else:  
            msg = "Invalid Email Id or Password !"  
            return render\_template("inventoryLogin.html",msg = msg)  
  
@app.route("/dashboard")  
def dashboard():  
    totStockQuery = "SELECT COUNT(\*) FROM STOCKDETAILS"  
    activeStocksQuery = "SELECT COUNT(\*) FROM STOCKDETAILS WHERE STOCKSTATUS = 'Active'"  
    lowStocksQuery = "SELECT COUNT(\*) FROM STOCKDETAILS WHERE QTYOFSTOCK < 50"  
    nonMovableStockQuery = "SELECT COUNT(\*) FROM WAREHOUSEDETAILS"  
     
    totStock = ibm\_db.prepare(conn,totStockQuery)  
    activeStocks = ibm\_db.prepare(conn,activeStocksQuery)  
    lowStocks = ibm\_db.prepare(conn,lowStocksQuery)  
    nonMovableStocks = ibm\_db.prepare(conn,nonMovableStockQuery)  
     
    ibm\_db.execute(totStock)  
    ibm\_db.execute(activeStocks)  
    ibm\_db.execute(lowStocks)  
    ibm\_db.execute(nonMovableStocks)  
     
    totalStock = ibm\_db.fetch\_assoc(totStock)  
    activeStock = ibm\_db.fetch\_assoc(activeStocks)  
    lowStock = ibm\_db.fetch\_assoc(lowStocks)  
    nonMovableStock = ibm\_db.fetch\_assoc(nonMovableStocks)  
     
    detail = dashboardDetails(totalStock,activeStock,lowStock,nonMovableStock)  
     
    saleDetailsQuery = "SELECT \* FROM SALES"  
    saleDetail = ibm\_db.prepare(conn,saleDetailsQuery)  
    ibm\_db.execute(saleDetail)  
    databaseSales = ibm\_db.fetch\_tuple(saleDetail)  
    sales = []  
    while databaseSales != False:  
        sales.append(databaseSales)  
        databaseSales = ibm\_db.fetch\_tuple(saleDetail)  
         
    return render\_template("inventoryDashboard.html",details = detail,sales = sales)

if \_\_name\_\_ == "\_\_main\_\_":  
    app.run(debug = True)

# 7. 2 Project Development - Delivery of Sprint 2

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="https://unicons.iconscout.com/release/v4.0.0/css/line.css">

<link rel="stylesheet" href="style.css">

<title>Registration Page</title>

</head>

<body>

<div class="contaier">

<div class="forms">

<div class="form login">

<!-- Registration Form -->

<div class="container1">

<div class="form signup">

<div class="tittle1" <h1><b><u>Registration</u></b></h1></div>

<form action="#">

<div class="input-field">

<input type="text" placeholder="Enter your Store name" required>

<i class="uil uil-user"></i>

</div>

<div class="input-field">

<input type="text" placeholder="Enter your email" required>

<i class="uil uil-envelope icon"></i>

</div>

<div class="input-field">

<input type="password" class="password" placeholder="Create a password" required>

<i class="uil uil-lock icon"></i>

<i class="uil uil-eye-slash showHidePw"></i>

</div>

<div class="input-field">

<input type="text" class="text" placeholder="State" required>

<i class="uil uil-map icon"></i>

<!-- <i class="uil uil-eye-slash showHidePw"></i> -->

</div>

<div class="input-field">

<input type="text" class="text" placeholder="Country" required>

<i class="uil uil-map-marker-shield"></i>

</div>

<div class="checkbox-text">

<div class="checkbox-content">

<input type="checkbox" id="termCon">

<label for="termCon" class="text">I accepted all terms and conditions</label>

</div>

</div>

<div class="input-field button">

<input type="button" value="Signup">

</div>

</form>

<div class="login-signup">

<span class="text">Already have a account?

<a href="#" class="text login-link">Login Now</a>

</span>

</div>

</div>

</div>

</div>

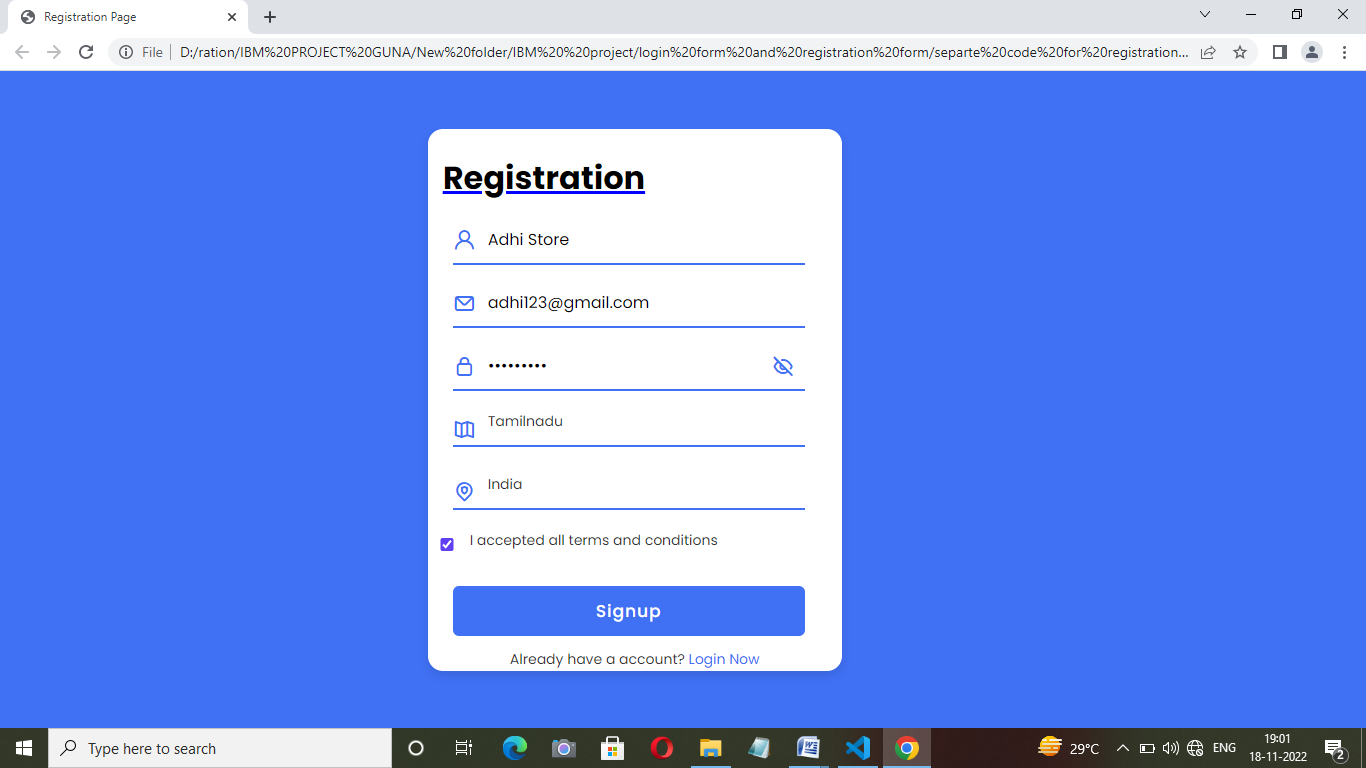
</div>

</div>

</body>

</html>

Project Development - Delivery of Sprint – 2:



* 1. **7.3 Project Development –Delivery of Sprint -3**

@app.route("/stocks")  
def stocks():  
    stockDetailsQuery = "SELECT \* FROM STOCKDETAILS"  
    stockDetail = ibm\_db.prepare(conn,stockDetailsQuery)  
    ibm\_db.execute(stockDetail)  
    databaseStocks = ibm\_db.fetch\_tuple(stockDetail)  
    stocks = []  
    while databaseStocks != False:  
        stocks.append(databaseStocks)  
        databaseStocks = ibm\_db.fetch\_tuple(stockDetail)  
             
    return render\_template("stocks.html",stocks = stocks)  
  
@app.route("/addStocks",methods=["GET","POST"])  
def addStocks():  
    if request.method == "POST":  
        stockName = request.form['stockName']  
        qtyStock = request.form['qtyofstock']  
        costPrice = request.form['costPrice']  
        sellingPrice = request.form['sellingPrice']  
        warehouseid = request.form['warehouseId']  
        addStocksQuery = "INSERT INTO  STOCKDETAILS (STOCKNAME,QTYOFSTOCK,COSTPRICE,SELLINGPRICE,WAREHOUSE) VALUES(?,?,?,?,?)"  
        stmt = ibm\_db.prepare(conn,addStocksQuery)  
        ibm\_db.bind\_param(stmt,1,stockName)  
        ibm\_db.bind\_param(stmt,2,qtyStock)  
        ibm\_db.bind\_param(stmt,3,costPrice)  
        ibm\_db.bind\_param(stmt,4,sellingPrice)  
        ibm\_db.bind\_param(stmt,5,warehouseid)  
        ibm\_db.execute(stmt)  
         
        flash("Stocks Added Sucessfully :) ")  
        return redirect("/stocks")  
  
@app.route("/editStocks",methods = ["GET","POST"])  
def editStocks():  
    if request.method == "POST":  
        stockId = request.form['stockId']  
        stockName = request.form['stockName']  
        qtyStock = request.form['qtyofstock']  
        costPrice = request.form['costPrice']  
        sellingPrice = request.form['sellingPrice']  
        warehouseid = request.form['warehouseId']  
        addStocksQuery = "UPDATE STOCKDETAILS SET STOCKNAME = ?,QTYOFSTOCK = ?,COSTPRICE = ?,SELLINGPRICE = ?,WAREHOUSE = ? WHERE STOCKID = ?"  
        stmt = ibm\_db.prepare(conn,addStocksQuery)  
        ibm\_db.bind\_param(stmt,1,stockName)  
        ibm\_db.bind\_param(stmt,2,qtyStock)  
        ibm\_db.bind\_param(stmt,3,costPrice)  
        ibm\_db.bind\_param(stmt,4,sellingPrice)  
        ibm\_db.bind\_param(stmt,5,warehouseid)  
        ibm\_db.bind\_param(stmt,6,stockId)  
        ibm\_db.execute(stmt)  
        return redirect("/stocks")  
     
    return render\_template("stocks.html")  
  
@app.route("/deleteStocks",methods = ["GET","POST"])  
def deleteStocks():  
    if request.method == "POST":  
        stockId = request.form["stockid"]  
        deleteQuery = "DELETE FROM STOCKDETAILS WHERE STOCKID = ?"  
        stmt = ibm\_db.prepare(conn,deleteQuery)  
        ibm\_db.bind\_param(stmt,1,stockId)  
        ibm\_db.execute(stmt)  
        return redirect("/stocks")  
    else:  
        return redirect("/stocks")  
  
@app.route("/warehouse")  
def inventory():  
    warehouseDetailsQuery = "SELECT \* FROM WAREHOUSEDETAILS"  
    warehouseDetail = ibm\_db.prepare(conn,warehouseDetailsQuery)  
    ibm\_db.execute(warehouseDetail)  
    databaseWarehouse = ibm\_db.fetch\_tuple(warehouseDetail)  
    warehouses = []  
    while databaseWarehouse != False:  
        warehouses.append(databaseWarehouse)  
        databaseWarehouse = ibm\_db.fetch\_tuple(warehouseDetail)  
     
    return  render\_template("warehouse.html",warehouses = warehouses)  
  
@app.route("/addWarehouse",methods = ["GET","POST"])  
def addWarehouse():  
    if request.method == "POST":  
        warehouseName = request.form['warehouseName']  
        warehouseLocation = request.form['warehouseLocation']  
        addWarehouseQuery = "INSERT INTO WAREHOUSEDETAILS (WAREHOUSENAME,WAREHOUSELOCATION) VALUES(?,?)"  
        stmt = ibm\_db.prepare(conn,addWarehouseQuery)  
        ibm\_db.bind\_param(stmt,1,warehouseName)  
        ibm\_db.bind\_param(stmt,2,warehouseLocation)  
        ibm\_db.execute(stmt)  
        return redirect("/warehouse")  
  
@app.route("/viewStocks",methods=["GET","POST"])  
def viewStocks():  
    if request.method == "POST":  
        warehouseId = request.form["warehouseId"]  
        stockDetailsQuery = "SELECT \* FROM STOCKDETAILS WHERE WAREHOUSE = ?"  
        stmt = ibm\_db.prepare(conn,stockDetailsQuery)  
        ibm\_db.bind\_param(stmt,1,warehouseId)  
        ibm\_db.execute(stmt)      
        totalstocks = ibm\_db.fetch\_tuple(stmt)  
        stocks = []  
        while totalstocks != False:  
            stocks.append(totalstocks)  
            totalstocks = ibm\_db.fetch\_tuple(stmt)  
         
        return render\_template("warehouseStocks.html",stocks = stocks)

**8.CONCLUSION**

In summary, the project works is relevancy to the objectives set. Thus, activities of developing the system which is planning and analysis is based on the result retrieved from the interview on observation. Not only that, as this would be the first computerized system that will be used by the store, the functions only focused on solving major problem which is inventory management problem. The interfaces design is also categorized as user friendly due to lack of IT background of the workers which means the system can be handle by people not even from IT background. Due to time constraints, it is not possible for the developer to implement many functions in the system, thus the developer has few future works suggestions for continuation. By having this integration, Store can practice Just-In-Time inventory where the store does not need to hold many stocks which is not a good practice of inventory control.

**REFERENCE**

1. M. Gendron. Business Intelligence and the Cloud. John Wiley & Sons, 2014.
2. G. Lasco. The pathetic state of PH Internet. Inquirer.net, September 2015.<http://opinion.inquirer.net/88125/the-pathetic-state-of-ph-internet>
3. SME STATISTICS: 2013 MSME Statistics. Department of Trade and Industry, MSMED Plan, 2011 – 2016.

<http://www.dti.gov.ph/dti/index.php/resources/sme-resources/sme-statistics>

1. J. Sedivy, T. Barina, I. MOrozan, A. Sandu. MCsync – Distributed, Decentralized Database for Mobile Devices. IEEE.2012, pp.0-5.
2. A. Imam, S. Basri, R. Ahmad. Data Synchronization Between Mobile Devices and Sever-side Databases: A Systematic Literature Review. Journal of Theoretical and Applied Information Technology, Vol.81, No.2. 2015, pp 364-382.

.