

**PROJECT BASED EXPERIENTIAL  
LEARNING PROGRAM (NALAIYA  
THIRAN)**

**INVENTORY MANAGEMENT SYSTEM FOR RETAILERS**

**A PROJECT REPORT**

*Submitted by*

**NAVEENKUMAR (1116190801104)**

**RAVISURYA (1116190801134)**

**SANJAI (1116190801145)**

**SANJAY (1116190801147)**

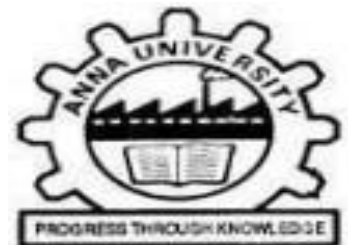
**TEAM ID : PNT2022TMID02538**

**DEPARTMENT OF ELECTRONICS AND  
COMMUNICATION ENGINEERING**

**RAJALAKSHMIENGINEERING COLLEGE**

**(An Autonomous Institution)**

**Thandalam, Chennai-602 105**



**NOVEMBER 2022**

# **Index**

1. Introduction
  - 1.1 Project Overview
  - 1.2 Purpose
2. Literature
3. Objective
  - 3.1 Primary Objective
  - 3.2 Secondary Objective
4. Scope of Application
5. Background Research
6. Requirement Analysis
  - 6.1 IMS Requirement
  - 6.2. User's Requirement
7. Feasibility
  - 7.1 Economic feasibility
  - 7.2 Technical feasibility
8. Ideation
  - 8.1 Brainstorm and idea prioritization
  - 8.2 Problem Statement
  - 8.3 Empathy map
9. Setting up the environment
  - 9.1 Flask setup
  - 9.2 Docker setup
  - 9.3 IBM Cloud Account creation
  - 9.4 Sendgrid Account Creation
  - 9.5 IBM Cloud CLI install
10. Deployment of app in IBM Cloud
  - 10.1 Containerize the application
  - 10.2 Upload image into IBM cloud registry
  - 10.3 Deploy in Kubernetes Cluster
11. Integrating Sendgrid Services
12. Project Design Phase - 1

12.1 Proposed Solution

12.2 Problem Solution fit

12.3 Solution Architecture

## 13. Project Design Phase - 2

13.1 Data Flow Diagrams

13.2 Technology Architecture

13.3 Functional Requirement

13.4 Customer Journey

## 14.Implementation of Web Application

## 15.Project Planning Phase

15.1 Milestone and Activity List

15.2 Sprint Delivery Plan

## 16. Project Development Phase

16.1 Sprint 1

16.2 Sprint 2

16.3 Sprint 3

16.4 Sprint 4

## Conclusion

# **1.INTRODUCTION**

## **1.1 PROJECT OVERVIEW**

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.

## **1.2 PURPOSE**

The project Inventory Management System is a complete desktop based application designed on .Net technology using Visual Studio Software. The main aim of the project is to develop Inventory Management System Model software in which all the information regarding the stock of the organization will be presented. It is an intranet based desktop application which has an admin component to manage the inventory and maintenance of the inventory system. This desktop application is based on the management of stock of an organization. The application contains general organization profile, sales details, Purchase details and the remaining stock that are presented in the organization. There is a provision of updating the inventory also. This application also provides the remaining balance of the stock as well as the details of the balance of transaction. Each new stock is created and entitled with the name and the entry date of that stock and it can also be updated any time when required as per the transaction or the sales is returned in case. Here the login page is created in order to protect the management of the stock of the organization in order to prevent it from the threads and misuse of the inventory.

## **2.LITERATURE SURVEY**

### **1) Research paper on Inventory management system**

**Author** : Punam Khobragade\* , Roshni Selokar\* , Rina Maraskolhe\*

Prof.Manjusha Talmale

**Year** : 2018

**Description** : It is helpful for the businesses that operate hardware stores, where the store owner keeps the records of sales and purchase. This project eliminates the paperwork, human faults, manual delay and speed up process. Inventory Management System will have the ability to track sales and available inventory, telling a store owner when it's time to reorder and how much to purchase.

### **2) A Review of Inventory Management System**

**Author** : Varalakshmi G S1 , Asst Prof. Shivaleela S2

**Year** : 2021

**Description** : Inventory management system is a web application that focuses on inventory and sales clearance. This web application has logical tools for evaluating ideal inventory levels and selecting the appropriate replenishment strategies automatically. This technique eliminates the risk of stock-outs of fast-moving goods by minimizing delays.

### **3) An Online Based Inventory Management System Implementation In Printing Business**

**Author** : 1 Rafat Ara, 2 Md. Abdur Rahim

**Year** : 2018

**Description** : To describe inventory management system software which is used effectively in printing business. In this system the manual activities of printing business are altered into computerized systems. The system originates various

essential reports automatically. It reduces paperwork, human errors as well as speeds up the whole system.

#### 4) Study Of Smart Inventory Management System Based On The Internet OF THINGS (IOT)

Author : Souvik Paul\*, Atrayee Chatterjee, Digbijay Guha

Year : 2019

Description : Great advantages compared to the traditional mode, and we expect good prospects for its development. Inventory Management is a key area for customer service and cost optimization in any manufacturing setup.

#### 5) Simulation of inventory management systems in retail stores: A case study

Author : Puppala Sridhar a , C.R. Vishnu b , R Sridharan

Year : 2021

Description : High requirement of managing and controlling the inventory with appropriate policies to elevate the organization's performance. In fact, a proper system has to be implemented for monitoring customer demand. This system will, in turn, assist in maintaining the right level of inventory. In this direction, the present research focuses on a retail store and explores a solution for an inventory-related problem experienced by the firm

### **3.Objective**

#### **3.1 Primary objective**

The primary objectives of the project are mentioned below:

- To fulfill the requirement for achieving the Bachelor's degree
- To know the fundamentals of the python flask and Visual Studio

#### **3.2 Secondary objective**

The secondary objectives of this project are mentioned below:

- To develop an application that deals with the day to day requirement of any production organization
- To develop the easy management of the inventory
- To handle the inventory details like sales details, purchase details and balance stock details.
- To provide competitive advantage to the organization.
- To provide detailed information about the stock balance.
- To make the stock manageable and simplify the use of inventory in the organization



#### **4. Scope of the Application**

Inventory Management System (IMS) is targeted to the small or medium organization which doesn't have many warehouses i.e. only to those organizations that have single power of authority.

Some of the scope are:

- Only one person is responsible in assigning the details or records
- It is security driven.
- Godown can be added as per the requirement

#### **5. Background Research**

We started research by identifying the need of IMS in the organization. Initially we bounded our research to find the general reasons that emerged the needs of the Inventory Management System. We used different techniques to collect the data that can clearly give us the overall image of the application. The techniques we used were interviews with the developers, visiting online websites that are presented as the templates and visiting some organization to see their IMS application.

Basically the following factors forced us to develop IMS application:

- Cost and affordability
- Lack of stock management.
- Effective flow of stock transfer and management.
- Difficulty in monitoring stock management.

## **6. Requirement Analysis**

We collected a number of requirements for the project from our primitive research, website visits, and interviews with the concerned personnel and their experiences regarding the concepts of its development. We have even visited some organizations in Kathmandu valley and analyzed its importance and tried to develop the project by fulfilling all the weaknesses that were found in the application. We then decided to build the same type of application with different logic flow and new language which will be suitable for the small organization.

### **6.1 IMS Requirement**

The goal for the application is to manage the inventory management function of the organization. Once it is automated all the functions can be effectively managed and the organization can achieve the competitive advantage.

Business requirement are discussed in the Scope section, with the following additional details:

- Helps to search the specific product and remaining stock.
- Details information about the product sales and purchase.
- Brief Information of the organization today's status in terms of news, number of present inventory as per the date entered.
- It helps to identify the total presented inventory in the company.
- To know the balance and details of sales distributed on a specific date.
- There is proper transaction management of inventory.
- All transactions have a specific entry date along with quantity and rate.
- Only admin can login in the page.

## 6.2. User's Requirement

User requirement are categorized by the user type

Admin

- Able to create new godown along with date.
- Able to edit the entry as per entry.
- Able to add, modify and delete the stock entry.

Inventory management

- Able to check the stock available.
- Able to check the balance payment.
- Able to view the remaining sales stock.

## **7. Feasibility**

### 7.1 Economic Feasibility

The system is estimated to be economically affordable. The system is a medium scale desktop application and has an affordable price. The benefits include increased efficiency, effectiveness, and better performance. Comparing the cost and benefits the system is found to be economically feasible.

### 7.2 Technical Feasibility

Development of the system requires tools like:

- IBM Cloud Object Storage
- Python-Flask
- IBM DB2
- IBM Container Registry
- Docker
- Kubernetes
- Visual Studio Code


## **8. Ideation and proposed solution**

### **8.1 Brainstorm & Idea Prioritization Template:**

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich number of creative solutions. Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

Reference:<https://app.mural.co/t/kongu5284/m/kongu5284/1664427092304/f23a60c8d85de8369099eaf9cccd3b648d29bca?sender=u592162ba670be7259c850082>

#### **Step-1: Team Gathering, Collaboration and Select the Problem Statement**

<div>Template</div> <div></div> <div><b>Brainstorm &amp; idea prioritization</b></div> <div>Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.</div> <div><div>🕒 10 minutes to prepare</div><div>🕒 1 hour to collaborate</div><div>👤 2-8 people recommended</div></div> <div><a href="#">Share template feedback</a></div>	<div><div>➔</div><div><b>Before you collaborate</b></div><div>A little bit of preparation goes a long way with this session. Here's what you need to do to get going.</div><div>🕒 10 minutes</div></div> <div><div>➔</div><div><b>Team gathering</b></div><div>Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.</div></div> <div><div>➔</div><div><b>Set the goal</b></div><div>Think about the problem you'll be focusing on solving in the brainstorming session.</div></div> <div><div>➔</div><div><b>Learn how to use the facilitation tools</b></div><div>Use the Facilitation Superpowers to run a happy and productive session.</div><div><a href="#">Open article</a> ➔</div></div>	<div><div>➔</div><div><b>problem statement</b></div><div>What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.</div><div>🕒 5 minutes</div></div> <div><div><b>PROBLEM</b></div><div>To create an inventory management system that will allow retailers to satisfy demand from customers without running out of inventory or holding too much on hand.</div></div>
--	---	--

## Step-2: Brainstorm, Idea Listing and Grouping

### Brainstorm

Write down any ideas that come to mind that address your problem statement.

15 minutes

**Tip**

You can write a sticky note and use the hand function to stick it back to the screen.

**James P**

**James M**

**Jeffrey H**

**John J**

### Group ideas

Take turns sharing your ideas while clustering similar or related ideas as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

10 minutes

**Tip**

And a handy-dandy tip to make sure you're ready to go: make sure you have a sticky note for each cluster. You can use the hand function to stick it back to the screen.

**Research and Ideation**

**Preparation**

**Implementation**

**Testing and Feedback**

## Step-3: Idea Prioritization

### Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

**Importance**

**Feasibility**

**Tip**

Remember that ideas that are high importance but low feasibility are not necessarily the best ideas to pursue.

### After you collaborate

You can export the mural as an image or PDF to share with members of your company who might find it helpful.

#### Quick add-ons

- Share the mural**  
Share a view link to the mural with stakeholders to keep them in the loop about the outcomes of the process.
- Export the mural**  
Export a copy of the mural as a PNG or PDF to share to emails, social media, or save it to your drive.

#### Keep moving forward

- Strategy blueprint**  
Define the components of a new idea or strategy.  
[Open the template](#)
- Customer experience journey map**  
Understand customer needs, motivations, and behaviors for an experience.  
[Open the template](#)
- Strengths, weaknesses, opportunities & threats**  
Identify strengths, weaknesses, opportunities, and threats (SWOT) to develop a plan.  
[Open the template](#)

[Share template feedback](#)

## **8.2 Problem Statement**

The two basic inventory decisions that managers face are:

- How much additional inventory to order or produce
- When to order or produce it Although it is possible to consider these two

decisions separately, they are so closely related that a simultaneous solution is usually necessary.

Typically, the objective is to minimize total inventory costs. Total inventory costs typically include holding, ordering, shortage, and purchasing costs. In a continuous review system, managers continuously monitor the inventory position. Whenever the inventory position falls at or below a level  $R$ , called the reorder point, the manager orders  $Q$  units, called the order quantity. (Notice that the reorder decision is based on the inventory position including orders and not the inventory level. If managers used the inventory level, they would place orders continuously as the inventory level fell below  $R$  until they received the order.) When you receive the order after the lead-time, the inventory level jumps from zero to  $Q$ , and the cycle repeats. In inventory systems, demand is usually uncertain, and the lead-time can also vary. To avoid shortages, managers often maintain a safety stock. In such situations, it is not clear what order quantities and reorder points will minimize expected total inventory cost. Simulation models can address this question. In this example, demand is uncertain and is Poisson distributed with a mean of 100 units per week. Thus, the expected annual demand is 5,200 units.

Note: For large values of the rate parameter,  $\lambda$ , the Poisson distribution is approximately normal. Thus, this assumption is tantamount to saying that the demand is normally distributed with a mean of 100 and standard deviation of  $\sqrt{100}$ .

The Poisson distribution is discrete, thus eliminating the need to round off normally distributed random variates. Additional relationships that hold for the inventory system are:

- Each order costs \$50 and the holding cost is \$0.20 per unit per week (\$10.40 for one year).
- Every unfilled demand is lost and costs the firm \$100 in lost profit.

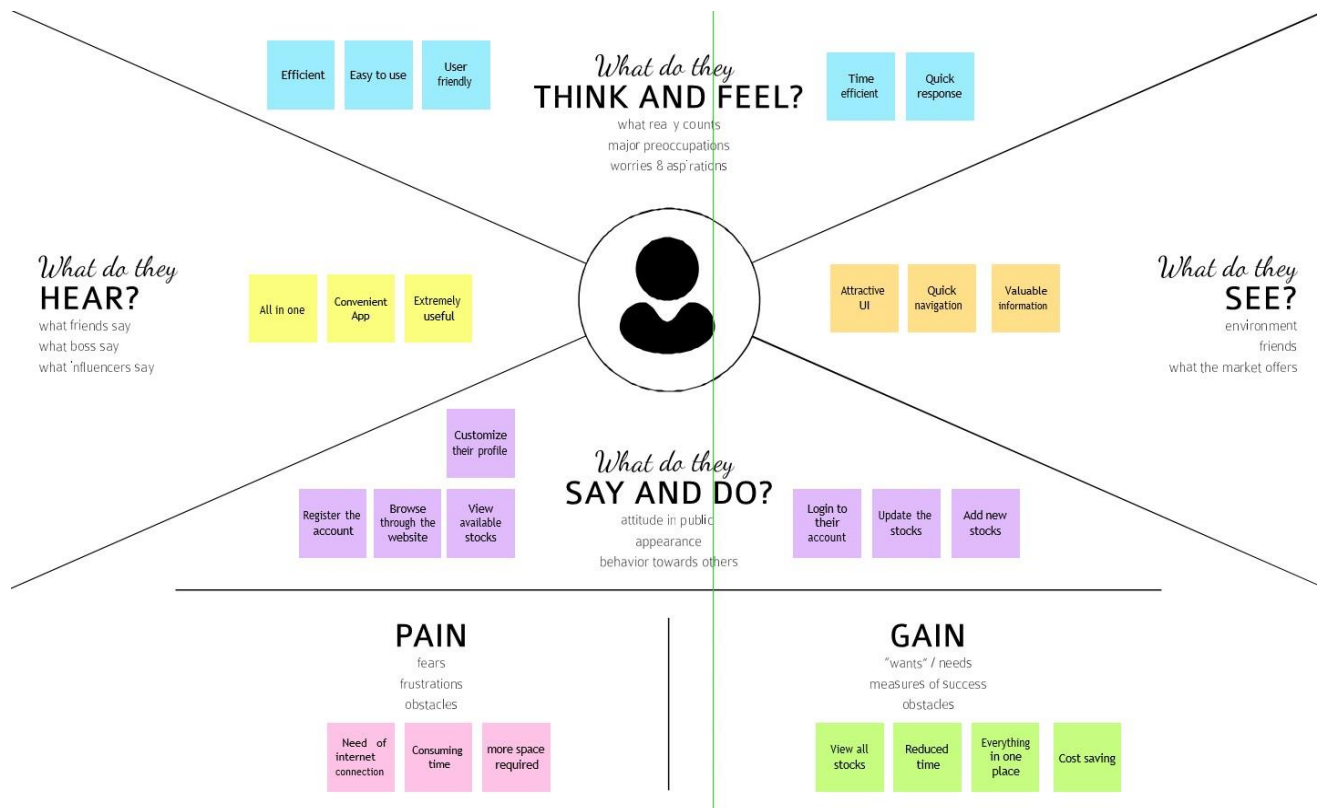
- The time between placing an order and receiving the order is 2 weeks.

Therefore, the expected demand during lead-time is 200 units. Orders are placed at the end of the week, and received at the beginning of the week.

The traditional economic order quantity (EOQ) model suggests an order quantity: For the EOQ policy, the reorder point should equal the lead-time demand; that is, place an order when the inventory position falls to 200 units. If the lead-time demand is exactly 200 units, the order will arrive when the inventory level reaches zero. However, if demand fluctuates about a mean of 200 units, shortages will occur approximately half the time. Because of the high shortage costs, the manager would use either a larger reorder point, a larger order quantity, or both. In either case, the manager will carry more inventory on average, which will result in a lower total shortage cost but a higher total holding cost. A higher order quantity lets the manager order less frequently, thus incurring lower total ordering costs. However, the appropriate choice is not clear. Simulation can test various reorder point/order quantity policies.

## 8.3

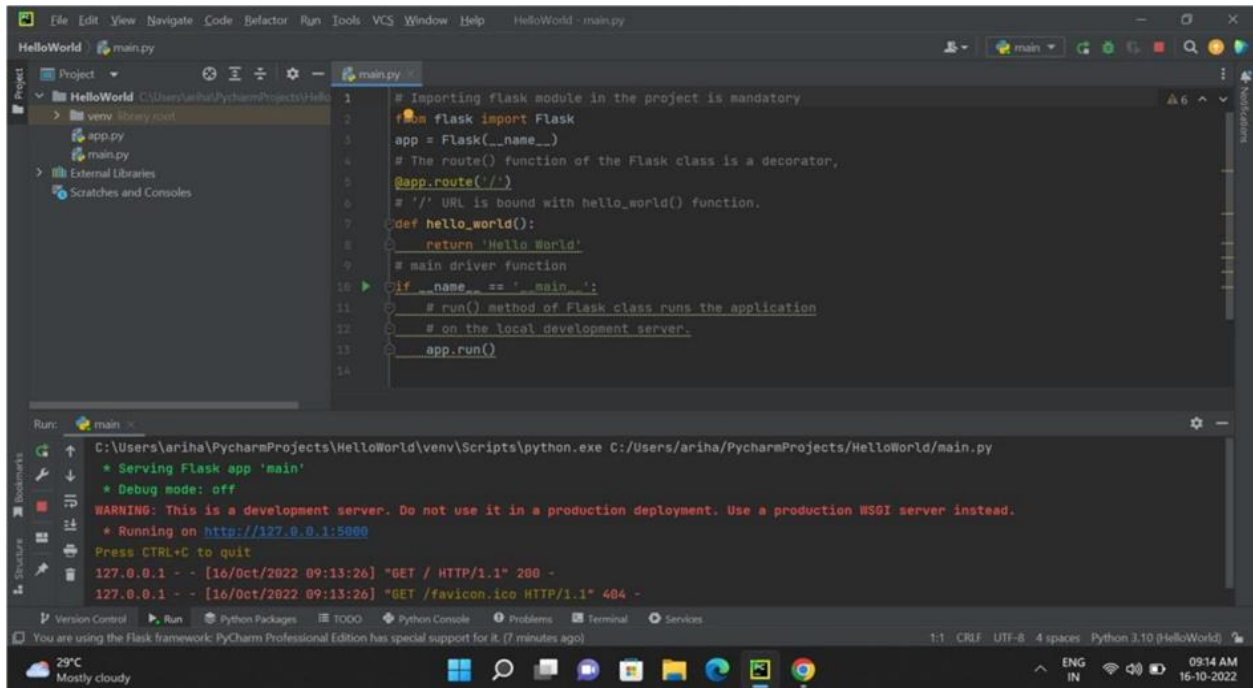
## Empathy Map





## 9. Setting up application environment

### 9.1 Flask Setup

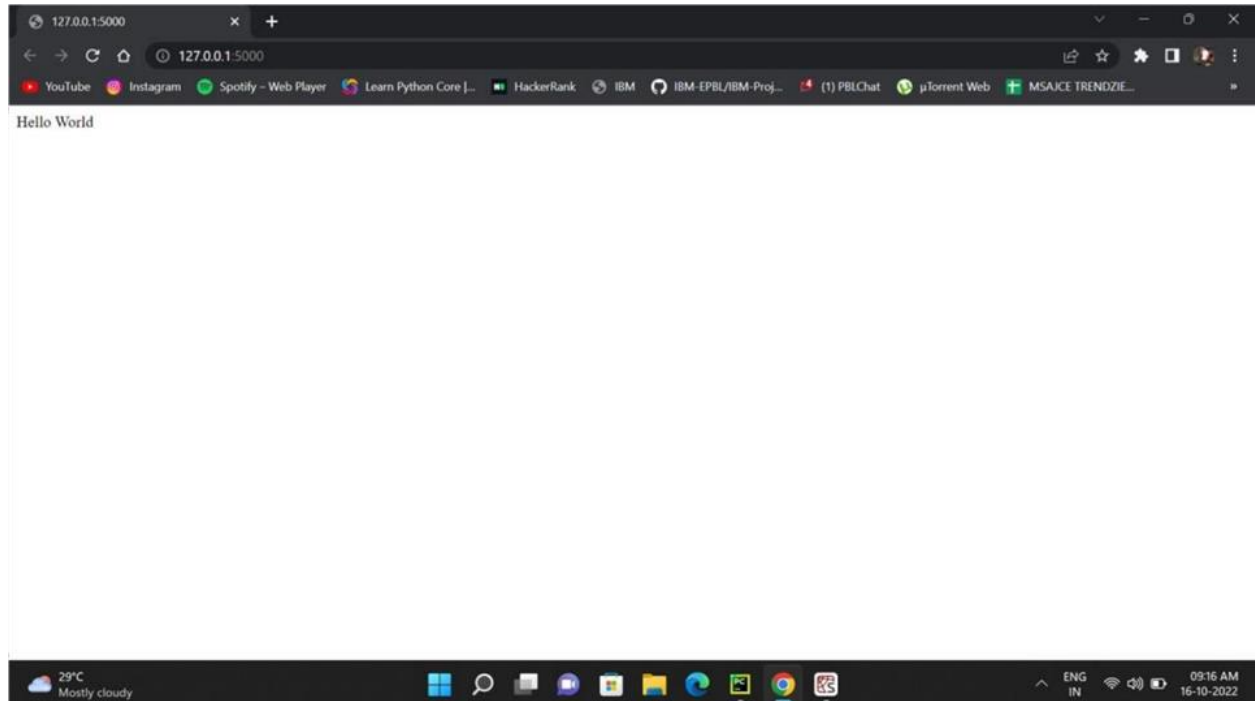


The screenshot shows the PyCharm IDE with a project named 'HelloWorld'. The main.py file contains the following code:

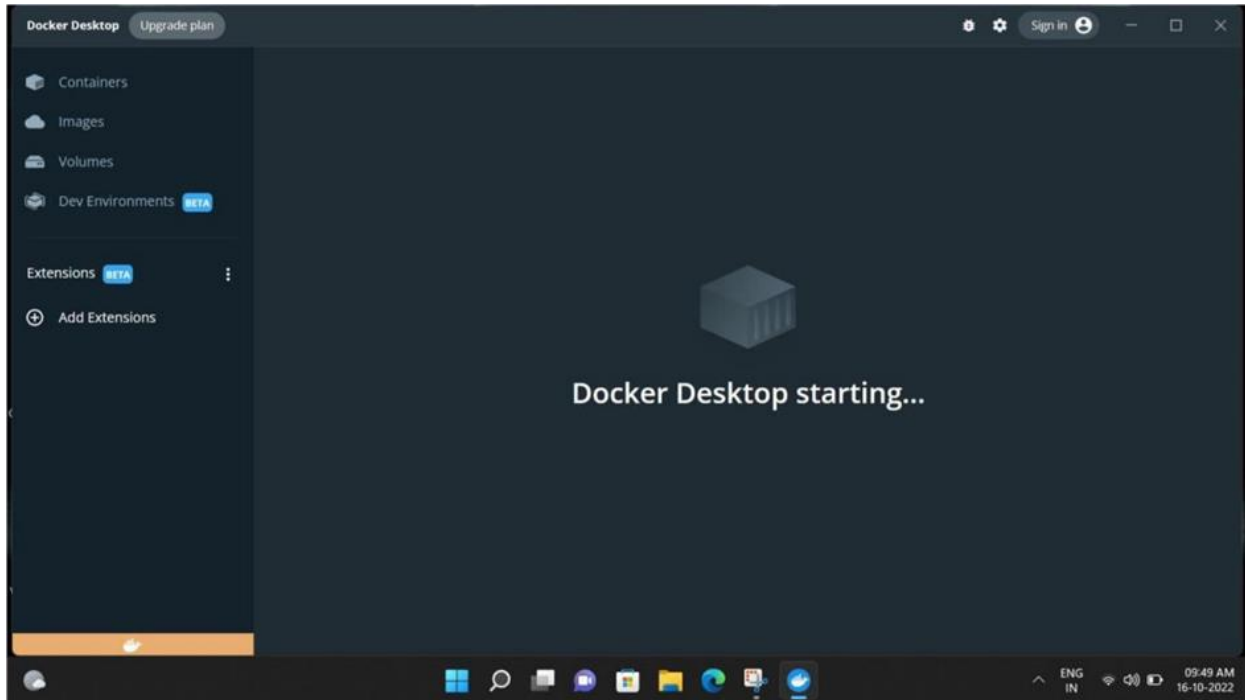
```
1 # Importing flask module in the project is mandatory
2 from flask import Flask
3 app = Flask(__name__)
4 # The route() function of the Flask class is a decorator,
5 # '/' URL is bound with hello_world() function.
6 @app.route('/')
7 def hello_world():
8     return 'Hello World'
9 # main driver function
10 if __name__ == '__main__':
11     # run() method of Flask class runs the application
12     # on the local development server.
13     app.run()
```

The Run console at the bottom shows the following output:

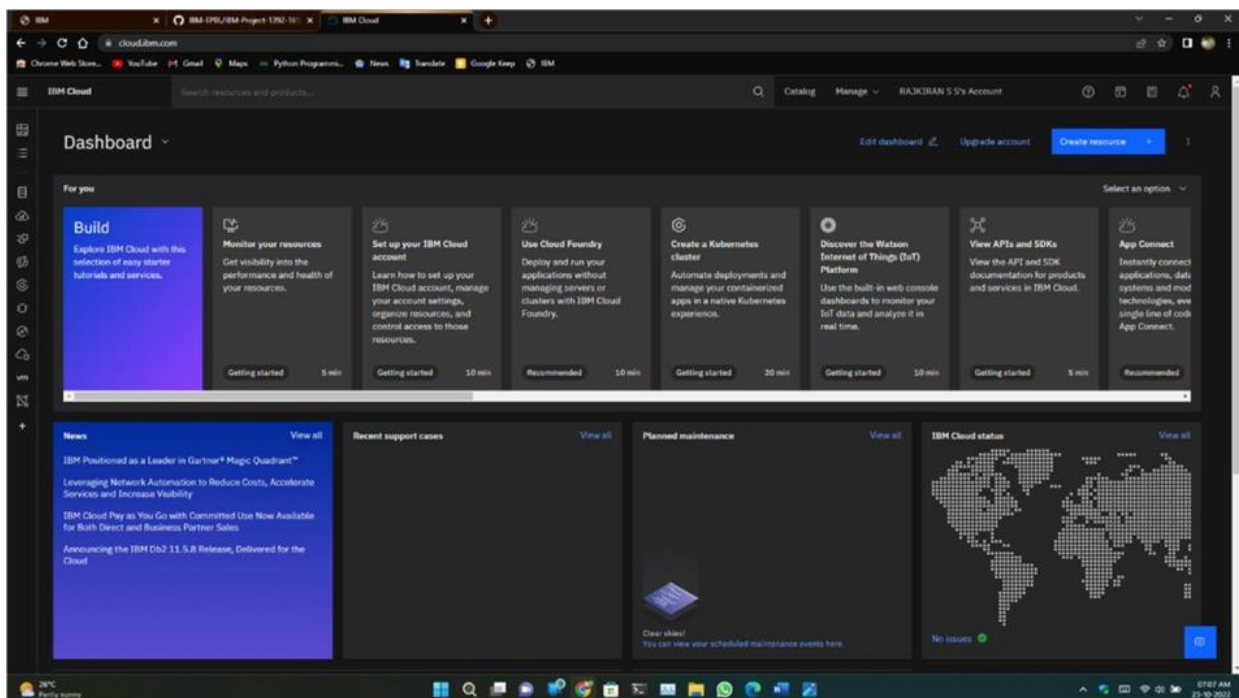
```
Run: main
C:\Users\arisha\PycharmProjects\HelloWorld\venv\Scripts\python.exe C:\Users\arisha\PycharmProjects\HelloWorld/main.py
* Serving Flask app 'main'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [16/Oct/2022 09:13:26] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [16/Oct/2022 09:13:26] "GET /favicon.ico HTTP/1.1" 404 -
```



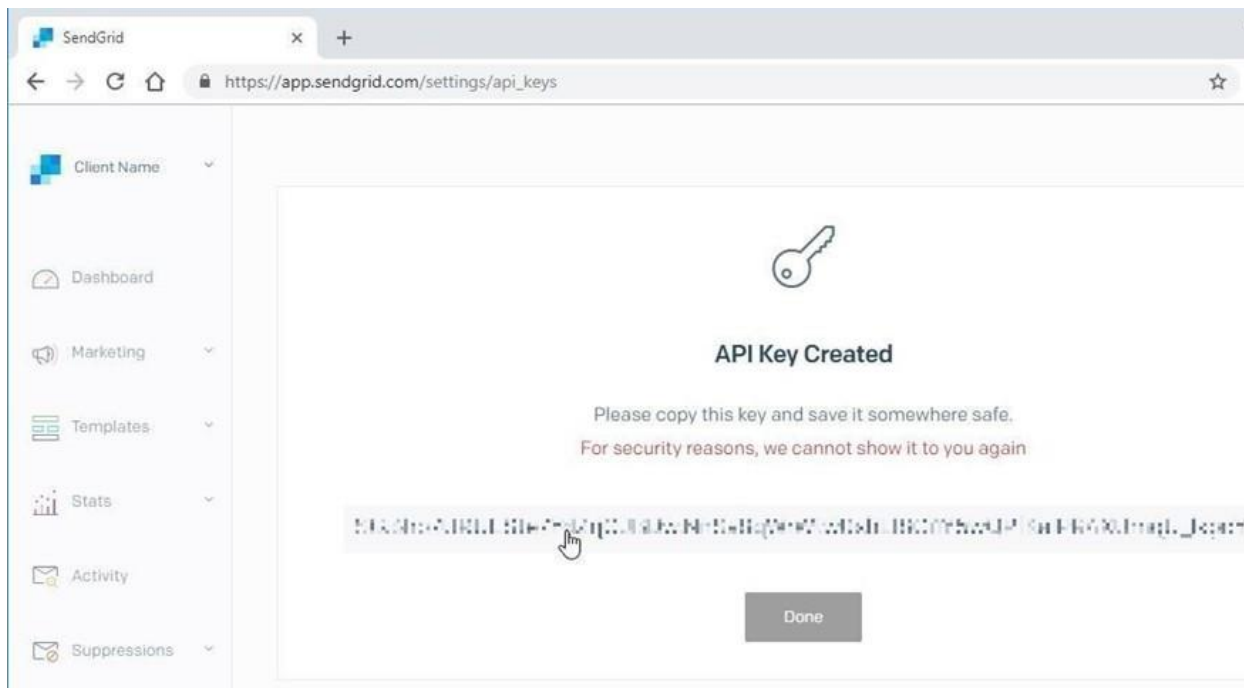
## 9.2 Docker Setup



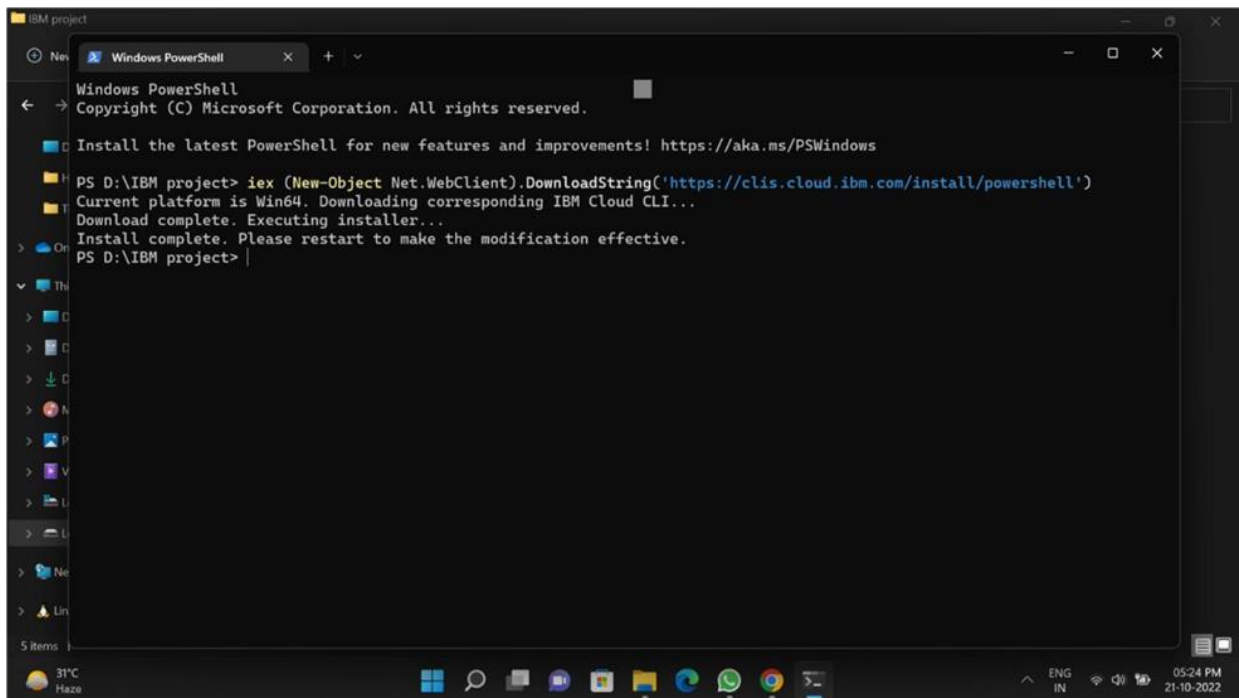
## 9.3 Cloud Account Creation



## 9.4 Sendgrid Creation



## 9.5 IBM Cloud CLI Creation



## 10. Deployment of app in IBM Cloud

### 10.1 Containerize the App-Docker image creation

The screenshot shows the Docker Desktop interface. On the left is a sidebar with navigation options: Containers, Images, Volumes, Dev Environments (beta), Extensions (beta), and Add Extensions. The main panel is titled 'Images on disk' and shows a list of local Docker images. At the top, it indicates 'Last refresh: 1 minute ago', '6 Images', '607.06 MB total size', and '0 Bytes / 607.06 MB in use'. There is a 'Clean up' button. Below the title, there are tabs for 'LOCAL' and 'REMOTE REPOSITORIES'. A search bar is present. A checkbox 'In use only' is checked. The table lists the following images:

NAME	TAG	IMAGE ID	CREATED	SIZE
centos	latest	5d0da3dc9764	about 1 year ago	231.27 MB
httpd	2.4.48-alpine	5a4f32436238	about 1 year ago	54.82 MB
nginx	latest	76c69feac34e	13 days ago	141.78 MB
nginx	stable-alpine	ec84f916d1ec	18 days ago	23.56 MB
ubuntu	latest	cdb68b455a14	13 days ago	77.79 MB
ubuntu	<none>	216c552ea5ba	about 1 month ago	77.84 MB

At the bottom of the Docker Desktop window, system information is shown: RAM 1.80GB, CPU 0.00%, Connected to Hub, and version v4.12.0. The Windows taskbar is visible at the very bottom.

### 10.2 Upload image to IBM Container Registry

```
PS C:\Users\HP> docker tag hello-world icr.io/0034ns/helloworld
PS C:\Users\HP> docker push icr.io/0034ns/helloworld
Using default tag: latest
The push refers to repository [icr.io/0034ns/helloworld]
e07ee1baac5f: Pushed
latest: digest: sha256:f54a58bc1aac5ea1a25d796ae155dc228b3f0e11d046ae276b39c4bf2f13d8c4 size: 525
```

### 10.3 Deploy in Kubernetes Cluster

The screenshot shows the 'mycluster-free' Kubernetes dashboard. The top bar indicates 'Preparing master, workers...' and 'Expires in 30 days'. There are buttons for 'Add tags', 'Help', 'Kubernetes dashboard', and 'Actions...'. The left sidebar has 'Overview', 'Worker nodes', 'Worker pools', and 'DevOps (New)'. The main panel shows a table of worker nodes with the following data:

Name	Status	Worker pool	Zone	Private IP	Public IP	Version
000000c9	Deployed	default	Milan 01	10.144.213.108	169.51.206.6	1.24.7_1543

At the bottom, it shows 'Items per page: 25' and '1-1 of 1 item'. The Windows taskbar is visible at the bottom.

## **11. Integrating Sendgrid Services**

#Integrating Sendgrid Services

import os

import json

from sendgrid import SendGridAPIClient

from sendgrid.helpers.mail import \*

# NOTE: you will need move this file to the root

# directory of this project to execute properly.

def build\_hello\_email():

## Send a Single Email to a Single Recipient

message = Mail(from\_email=From('from@example.com', 'Example From Name'),

to\_emails=To('to@example.com', 'Example To Name'),

subject=Subject('Sending with SendGrid is Fun'),

plain\_text\_content=PlainTextContent('and easy to do anywhere, even with Python'),

html\_content=HtmlContent('<strong>and easy to do anywhere, even with Python</strong>'))

try:

print(json.dumps(message.get(), sort\_keys=True, indent=4))

return message.get()

except SendGridException as e:

print(e.message)

mock\_personalization = Personalization()

personalization\_dict = get\_mock\_personalization\_dict()

for cc\_addr in personalization\_dict['cc\_list']:

mock\_personalization.add\_to(cc\_addr)

for bcc\_addr in personalization\_dict['bcc\_list']:



```

mock_pers['headers'] = [Header("X-Test", "test"),
                        Header("X-Mock", "true")]

mock_pers['substitutions'] = [Substitution("%name%", "Example User"),
                              Substitution("%city%", "Denver")]

mock_pers['custom_args'] = [CustomArg("user_id", "343"),
                             CustomArg("type", "marketing")]

mock_pers['send_at'] = 1443636843
return mock_pers

def build_multiple_emails_personalized():
    # Note that the domain for all From email addresses must match

    message = Mail(from_email=From('from@example.com', 'Example From Name'),
                   subject=Subject('Sending with SendGrid is Fun'),
                   plain_text_content=PlainTextContent('and easy to do anywhere, even with Python'),
                   html_content=HtmlContent('<strong>and easy to do anywhere, even with Python</strong>'))

    mock_personalization = Personalization()
    mock_personalization.add_to(To('test@example.com', 'Example User 1'))
    mock_personalization.add_cc(Cc('test1@example.com', 'Example User 2'))
    message.add_personalization(mock_personalization)

    mock_personalization_2 = Personalization()
    mock_personalization_2.add_to(To('test2@example.com', 'Example User 3'))
    mock_personalization_2.set_from(From('from@example.com', 'Example From Name 2'))
    mock_personalization_2.add_bcc(Bcc('test3@example.com', 'Example User 4'))
    message.add_personalization(mock_personalization_2)

    try:
        print(json.dumps(message.get(), sort_keys=True, indent=4))
        return message.get()

    except SendGridException as e:

```

```
print(e.message)
```

```
return message
```

```
def build_attachment1():
```

```
    """Build attachment mock. Make sure your content is base64 encoded before passing into
    attachment.content.
```

```
    Another example:
```

```
    https://github.com/sendgrid/sendgrid-python/blob/HEAD/use_cases/attachment.md"""
```

```
    attachment = Attachment()
```

```
    attachment.file_content = ("TG9yZW0gaXBzdW0gZG9sb3Igc2l0IGFtZXQsIGNvbnNlI"
                                "Y3RldHVyIGFkaXBpc2NpbmcgZWxpdC4gQ3JhcyBwdW12")
```

```
    attachment.file_type = "application/pdf"
```

```
    attachment.file_name = "balance_001.pdf"
```

```
    attachment.disposition = "attachment"
```

```
    attachment.content_id = "Balance Sheet"
```

```
    return attachment
```

```
def build_attachment2():
```

```
    """Build attachment mock."""
```

```
    attachment = Attachment()
```

```
    attachment.file_content = "BwdW"
```

```
    attachment.file_type = "image/png"
```

```
    attachment.file_name = "banner.png"
```

```
    attachment.disposition = "inline"
```

```
    attachment.content_id = "Banner"
```

```
    return attachment
```

```
def build_kitchen_sink():
```

```
    """All settings set"""
```

```
    from sendgrid.helpers.mail import (
```

```
        Mail, From, To, Cc, Bcc, Subject, PlainTextContent,
```

```
        HtmlContent, SendGridException, Substitution,
```

```
        Header, CustomArg, SendAt, Content, MimeType, Attachment,
```

```
        FileName, FileContent, FileType, Disposition, ContentId,
```



```

TemplateId, Section, ReplyTo, Category, BatchId, Asm,
GroupId, GroupsToDisplay, IpPoolName, MailSettings,
BccSettings, BccSettingsEmail, BypassListManagement,
FooterSettings, FooterText, FooterHtml, SandBoxMode,
SpamCheck, SpamThreshold, SpamUrl, TrackingSettings,
ClickTracking, SubscriptionTracking, SubscriptionText,
SubscriptionHtml, SubscriptionSubstitutionTag,
OpenTracking, OpenTrackingSubstitutionTag, Ganalytics,
UtmSource, UtmMedium, UtmTerm, UtmContent, UtmCampaign)
import time
import datetime

message = Mail()

# Define Personalizations

message.to = To('test1@sendgrid.com', 'Example User1', p=0)
message.to = [
    To('test2@sendgrid.com', 'Example User2', p=0),
    To('test3@sendgrid.com', 'Example User3', p=0)
]

message.cc = Cc('test4@example.com', 'Example User4', p=0)
message.cc = [
    Cc('test5@example.com', 'Example User5', p=0),
    Cc('test6@example.com', 'Example User6', p=0)
]

message.bcc = Bcc('test7@example.com', 'Example User7', p=0)
message.bcc = [
    Bcc('test8@example.com', 'Example User8', p=0),
    Bcc('test9@example.com', 'Example User9', p=0)
]

message.subject = Subject('Sending with SendGrid is Fun 0', p=0)

message.header = Header('X-Test1', 'Test1', p=0)

```

```
message.header = Header('X-Test2', 'Test2', p=0)
message.header = [
    Header('X-Test3', 'Test3', p=0),
    Header('X-Test4', 'Test4', p=0)
]
```

```
message.substitution = Substitution('%name1%', 'Example Name 1', p=0)
message.substitution = Substitution('%city1%', 'Example City 1', p=0)
message.substitution = [
    Substitution('%name2%', 'Example Name 2', p=0),
    Substitution('%city2%', 'Example City 2', p=0)
]
```

```
message.custom_arg = CustomArg('marketing1', 'true', p=0)
message.custom_arg = CustomArg('transactional1', 'false', p=0)
message.custom_arg = [
    CustomArg('marketing2', 'false', p=0),
    CustomArg('transactional2', 'true', p=0)
]
```

```
message.send_at = SendAt(1461775051, p=0)
```

```
message.to = To('test10@example.com', 'Example User10', p=1)
message.to = [
    To('test11@example.com', 'Example User11', p=1),
    To('test12@example.com', 'Example User12', p=1)
]
```

```
message.cc = Cc('test13@example.com', 'Example User13', p=1)
message.cc = [
    Cc('test14@example.com', 'Example User14', p=1),
    Cc('test15@example.com', 'Example User15', p=1)
]
```

```
message.bcc = Bcc('test16@example.com', 'Example User16', p=1)
message.bcc = [
    Bcc('test17@example.com', 'Example User17', p=1),

```

```

    Bcc('test18@example.com', 'Example User18', p=1)
]

message.header = Header('X-Test5', 'Test5', p=1)
message.header = Header('X-Test6', 'Test6', p=1)
message.header = [
    Header('X-Test7', 'Test7', p=1),
    Header('X-Test8', 'Test8', p=1)
]

message.substitution = Substitution('%name3%', 'Example Name 3', p=1)
message.substitution = Substitution('%city3%', 'Example City 3', p=1)
message.substitution = [
    Substitution('%name4%', 'Example Name 4', p=1),
    Substitution('%city4%', 'Example City 4', p=1)
]

message.custom_arg = CustomArg('marketing3', 'true', p=1)
message.custom_arg = CustomArg('transactional3', 'false', p=1)
message.custom_arg = [
    CustomArg('marketing4', 'false', p=1),
    CustomArg('transactional4', 'true', p=1)
]

message.send_at = SendAt(1461775052, p=1)

message.subject = Subject('Sending with SendGrid is Fun 1', p=1)

# The values below this comment are global to entire message

message.from_email = From('help@twilio.com', 'Twilio SendGrid')

message.reply_to = ReplyTo('help_reply@twilio.com', 'Twilio SendGrid Reply')

message.subject = Subject('Sending with SendGrid is Fun 2')

message.content = Content(MimeType.text, 'and easy to do anywhere, even with Python')

```

```
message.content = Content(MimeType.html, '<strong>and easy to do anywhere, even with  
Python</strong>')
```

```
message.content = [  
    Content('text/calendar', 'Party Time!!'),  
    Content('text/custom', 'Party Time 2!!')  
]
```

```
message.attachment = Attachment(FileContent('base64 encoded content 1'),  
                                FileName('balance_001.pdf'),  
                                FileType('application/pdf'),  
                                Disposition('attachment'),  
                                ContentId('Content ID 1'))
```

```
message.attachment = [  
    Attachment(FileContent('base64 encoded content 2'),  
                FileName('banner.png'),  
                FileType('image/png'),  
                Disposition('inline'),  
                ContentId('Content ID 2')),  
    Attachment(FileContent('base64 encoded content 3'),  
                FileName('banner2.png'),  
                FileType('image/png'),  
                Disposition('inline'),  
                ContentId('Content ID 3'))  
]
```

```
message.template_id = TemplateId('13b8f94f-bcae-4ec6-b752-70d6cb59f932')
```

```
message.section = Section('%section1%', 'Substitution for Section 1 Tag')  
message.section = [  
    Section('%section2%', 'Substitution for Section 2 Tag'),  
    Section('%section3%', 'Substitution for Section 3 Tag')  
]
```

```
message.header = Header('X-Test9', 'Test9')  
message.header = Header('X-Test10', 'Test10')  
message.header = [  
    Header('X-Test11', 'Test11'),
```

```

    Header('X-Test12', 'Test12')
]

message.category = Category('Category 1')
message.category = Category('Category 2')
message.category = [
    Category('Category 1'),
    Category('Category 2')
]

message.custom_arg = CustomArg('marketing5', 'false')
message.custom_arg = CustomArg('transactional5', 'true')
message.custom_arg = [
    CustomArg('marketing6', 'true'),
    CustomArg('transactional6', 'false')
]

message.send_at = SendAt(1461775053)

message.batch_id = BatchId("HkJ5yLYULb7Rj8GKSx7u025ouWVIMgAi")

message.asm = Asm(GroupId(1), GroupsToDisplay([1,2,3,4]))

message.ip_pool_name = IpPoolName("IP Pool Name")

mail_settings = MailSettings()
mail_settings.bcc_settings = BccSettings(False, BccSettingsTo("bcc@twilio.com"))
mail_settings.bypass_list_management = BypassListManagement(False)
mail_settings.footer_settings = FooterSettings(True, FooterText("w00t"),
FooterHtml("<string>w00t!<strong>"))
mail_settings.sandbox_mode = SandBoxMode(True)
mail_settings.spam_check = SpamCheck(True, SpamThreshold(5), SpamUrl("https://example.com"))
message.mail_settings = mail_settings

tracking_settings = TrackingSettings()
tracking_settings.click_tracking = ClickTracking(True, False)

```

```

tracking_settings.open_tracking = OpenTracking(True,
OpenTrackingSubstitutionTag("open_tracking"))
tracking_settings.subscription_tracking = SubscriptionTracking(
    True,
    SubscriptionText("Goodbye"),
    SubscriptionHtml("<strong>Goodbye!</strong>"),
    SubscriptionSubstitutionTag("unsubscribe"))
tracking_settings.ganalytics = Ganalytics(
    True,
    UtmSource("utm_source"),
    UtmMedium("utm_medium"),
    UtmTerm("utm_term"),
    UtmContent("utm_content"),
    UtmCampaign("utm_campaign"))
message.tracking_settings = tracking_settings

return message

```

```

def send_multiple_emails_personalized():

```

```

    # Assumes you set your environment variable:
    #

```

<https://github.com/sendgrid/sendgrid-python/blob/HEAD/TROUBLESHOOTING.md#environment-variables-and-your-sendgrid-api-key>

```

    message = build_multiple_emails_personalized()
    sendgrid_client = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
    response = sendgrid_client.send(message=message)
    print(response.status_code)
    print(response.body)
    print(response.headers)

```

```

def send_hello_email():

```

```

    # Assumes you set your environment variable:
    #

```

<https://github.com/sendgrid/sendgrid-python/blob/HEAD/TROUBLESHOOTING.md#environment-variables-and-your-sendgrid-api-key>

```

    message = build_hello_email()
    sendgrid_client = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))

```

```
response = sendgrid_client.send(message=message)
print(response.status_code)
print(response.body)
print(response.headers)
```

```
def send_kitchen_sink():
```

```
    # Assumes you set your environment variable:
```

```
    #
```

```
https://github.com/sendgrid/sendgrid-python/blob/HEAD/TROUBLESHOOTING.md#environment-variables-and-your-sendgrid-api-key
```

```
    message = build_kitchen_sink()
```

```
    sendgrid_client = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
```

```
    response = sendgrid_client.send(message=message)
```

```
    print(response.status_code)
```

```
    print(response.body)
```

```
    print(response.headers)
```

## **12. Project Design Phase -1**

### **12.1 Proposed Solution**

Inventory management is a challenging problem in supply chain management. The problem faced by the company is that they do not have any system to keep track of inventory data. It is difficult for the retailer to record the inventory data. Every inventory stock manager's main problem is keeping track of how much stock is purchased and how much stock is spent. A tool or system to aid inventory management would be a beneficial tool in this area. Inventory management refers to managing the quantity, quality, location and transportation of various products utilized in manufacturing by various industrial organizations or in sales by various retailers. Usually, Inventory Management systems are limited and fixed to a selected range of items and cannot be modified and extended based on the customer's needs. The Inventory Management System focuses on making it expandable and usable easily by the end user and with constant customer support to alter the use. Unlike other software that provides similar functionalities, Inventory Management System focuses on making it easier by adding details of various other entities that are a part of the organization.

#### **Problem Statement:**

- The retailer's one of the challenging problems is that they face a lot of issues in keeping track of the inventory.
- The retailers must know the expiry date of the products so as to avoid wastage of products and loss of money.
- The retailers should also keep track of the fast-moving goods and also the dead stocks to avoid going out of stocks for fast moving goods and also surplus incoming of dead goods.
- Due to problems, The customers lose satisfaction and reliability over the retailers as there is a high chance the product is not delivered on time or it may be a very old one (in case of dead goods).



### Idea / Solution description:

- This proposed system/app will keep track of the details of every incoming and outgoing goods.
- The system will notify or alert the retailer over the expiry date of the products.
- The availability of stocks of each product is kept in track and the retailer is notified when it goes below the threshold limit.
- All the customers will have their own account on the app which they can use to buy products from the retailers.
- Each customer can see the details of retailers available in their zone, check for product availability and order their product.
- Both the retailers and the customers can track the order easily with this application.

### Novelty / Uniqueness:

- Notifications will be sent to the retailers if any product that the customers have been looking for is not available so that the product can be stocked up soon.
- Also retailers will be notified about the dead stocks in the inventory so that they could stop stocking them since customers are not preferring them.
- Notification will be sent to the customers who buy certain products regularly when the new arrivals are stocked up.

- Notifications are sent to the customer for the products in their wishlist to inform them that the product is available or about any discounts on those products.
- Exclusive discounts and offers are given for regular customers to keep them engaged with the store regularly.

#### Social Impact / Customer Satisfaction:

- One important reason, the customers are highly satisfied with this app is that they won't waste time on the product which is unavailable. They can check availability from the app itself.
- Since the app is automated and it is constantly updating after every purchase the work of keeping track of products is almost NIL for retailers.
- The customer satisfaction is improved reasonably due to the timely service offered to them.
- The money wasted on expired and dead goods is greatly reduced which helps the retailers a lot.

#### Business Model (Revenue Model):

Hereby we can provide a robust and most reliable inventory management system by using:

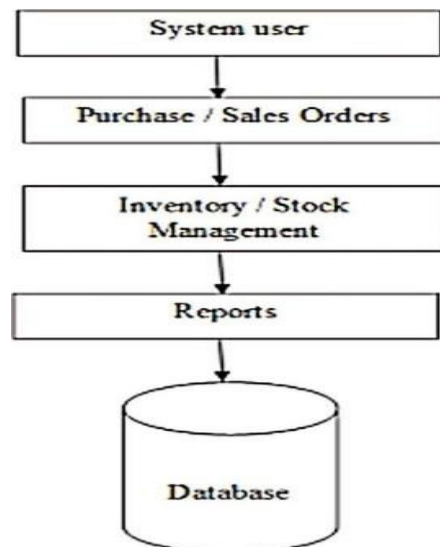
- Can deploy the most appropriate business advertising models.
- Can implement loss preventing strategies with this model.
- This model also ensures all time, anywhere availability of products.
- Usage of discounts/exclusive discounts business strategy for increasing the Customer base

## 12.2 Problem Solution Fit

Define CS, fit into CC	<b>1. CUSTOMER SEGMENT(S)</b> <span>CS</span> <ul style="list-style-type: none"> <li>❖ Customers are retailers, shop owners, business people who are struggling to keep track of their inventory.</li> <li>❖ Due to this issue, they face many issues like: <ul style="list-style-type: none"> <li>✓ Loss due to dead products in the inventory, unavailability of fast moving products, etc.</li> <li>✓ Unnecessary headache due to improper maintenance of inventory.</li> </ul> </li> </ul>	<b>6. CUSTOMER CONSTRAINTS</b> <ul style="list-style-type: none"> <li>❖ Since most of the softwares like these will be a subscription model, the customer must be paying as they use them. This may be against their budget.</li> <li>❖ To use this software the customer must be trained or he must hire a person to do that for him.</li> <li>❖ To deploy this software, the customer must have a powerful device which is compatible with the software.</li> </ul>	<b>5. AVAILABLE SOLUTIONS</b> <span>AS</span> <ul style="list-style-type: none"> <li>❖ <b>Solution:</b> The traditional solution for the inventory management problem is to track the incoming and outgoing goods with a pen and paper.</li> <li>❖ <b>Pros:</b> <ul style="list-style-type: none"> <li>✓ Easy to use</li> <li>✓ Less cost</li> </ul> </li> <li>❖ <b>Cons:</b> <ul style="list-style-type: none"> <li>✓ Error rate is high</li> <li>✓ Manual tracking is a tedious work</li> </ul> </li> </ul>	Explore AS, differentiate
Focus on J&P, tap into BE, understand RC	<b>2. JOBS-TO-BE-DONE / PROBLEMS</b> <span>J&amp;P</span> <ul style="list-style-type: none"> <li>❖ The objective of the software is to make the inventory tracking easier by automating the inventory. Example, the initial stocks information is fed to the software and from there it tracks the details of incoming and outgoing products.</li> <li>❖ This can generate automatic alerts/notifications to help the user in their work. Example, Alert for dead stocks in inventory, Alert for the goods which is to be refilled, Notifications for the user defined conditions like if sales go higher than certain limits etc...</li> <li>❖ Graphical representation of sales is also possible.</li> </ul>	<b>9. PROBLEM ROOT CAUSE</b> <ul style="list-style-type: none"> <li>❖ The primary reason for this problem to exist is the periodic change in demand of the customers.</li> <li>❖ This indirectly affects the inventory as change in customers needs is proportional to the sale of a particular products.</li> <li>❖ This keeping track of inventory effectively helps in managing the dead and fast moving products.</li> </ul>	<b>7. BEHAVIOUR</b> <span>BE</span> <ul style="list-style-type: none"> <li>❖ The customer must find a effective inventory tracking software.</li> <li>❖ He must implement it in his business to streamline his work and make more profit.</li> <li>❖ He must volunteer himself to learn to use the software or be ready to hire a person who can do it for him.</li> </ul>	Focus on J&P, tap into BE, understand RC
Identify strong TR & EM	<b>3. TRIGGERS</b> <ul style="list-style-type: none"> <li>❖ Understanding the fact that using a software to automate inventory system helps him to make more money and also make his work easier. Also seeing other retailers making more money using this software.</li> </ul> <b>4. EMOTIONS: BEFORE / AFTER</b> <b>Before:</b> They feel lost due to losses which occur due to improper management of inventory (Manual pen and paper tracking). <b>After:</b> They feel like success after making increased profits, reducing the mistakes that happen in manual process.	<b>10. YOUR SOLUTION</b> <ul style="list-style-type: none"> <li>✓ Design a flask based Inventory management system application.</li> <li>✓ Enable email based alerts for dead and fast moving products using sendgrid framework.</li> <li>✓ Provide a option for graphical view of sales</li> </ul>	<b>8. CHANNELS of BEHAVIOUR</b> <span>CH</span> <b>8.1 ONLINE</b> Online Inventory trackers which come for free may steal personal information of users and it may also contains a lot of ads. <b>8.2 OFFLINE</b> Manual logs can be maintained. Employees can be hired to maintain the inventory system logs when the business grows.	Extract online & offline CH of BE

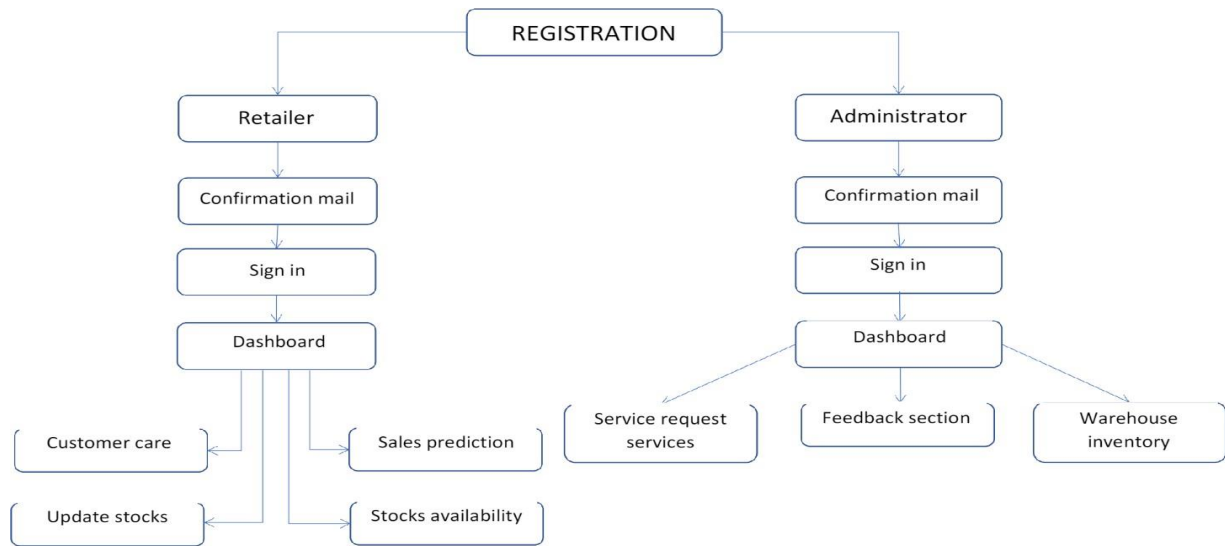
## 12.3 Solution Architecture

There was not an efficient solution available in the many companies during these days. Every process was based on paperwork, human fault rate was high, the process and tracing the inventory losses were not possible, and there were no efficient logging systems. After the computer age, every process started to be integrated into the electronic environment. And now we have qualified technology to implement new solutions to these problems.



## 13. Project Design Phase - 2

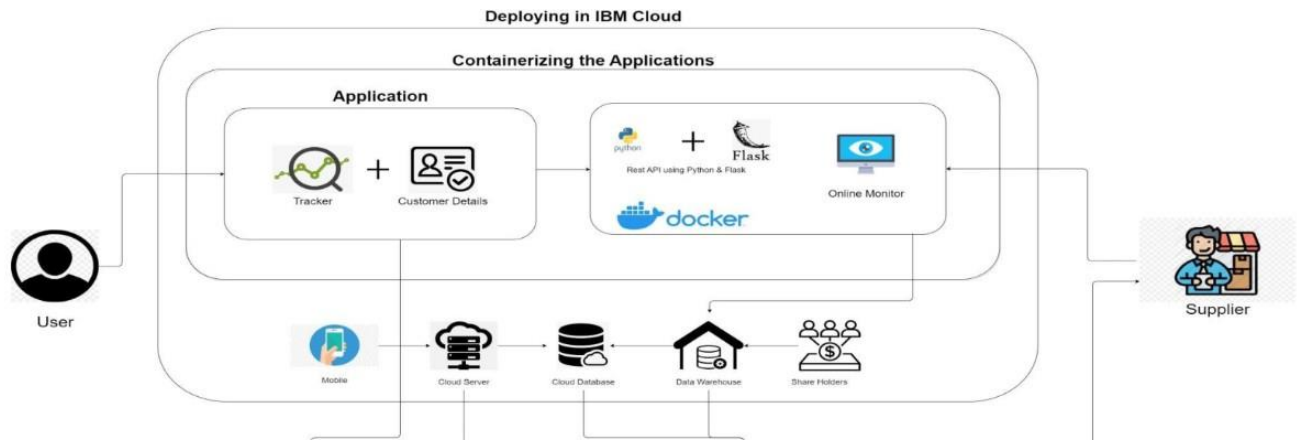
### 13.1 Data Flow Diagrams



#### User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I can login through my E-mail	I can access my account / dashboard	Medium	Sprint-1
	Confirmation	USN-3	As a user, I can receive my confirmation email once I have registered for the application	I can get confirmation email for my account and create an authenticated account.	Medium	Sprint-1
	Login	USN-4	As a user, I can log in to the authorized account by entering the registered email and password	I can login with registered email and password.	High	Sprint-1
	Dashboard	USN-5	As a user, I can view the products that are available currently.	Inventory can be viewed once logged in.	High	Sprint-2
	Stocks update	USN-6	As a user, I can add products which are not available in the inventory and restock the products.	When the products are not available, retailers can restock and update their inventory.	Medium	Sprint-2
	Sales prediction	USN-7	As a user, I can get access to sales prediction tool which can help me to predict better restock management of product.	The sales prediction tool should forecast the sales so tat the users can order properly and retailers can predict the order to sell.	Low	Sprint-3
Administrator	Request for customer care	USN-8	As a user, I am able to request customer care to get in touch with the administrators and enquire the doubts and problems.	Users can contact customer support and get help and service from administrators.	Medium	Sprint-4
	Giving feedback	USN-9	As a user, I am able to send feedback forms reporting any ideas for improving or resolving any issues I am facing to get it resolved.	Users can give feedback of issues or improvements to the administrators.	Medium	Sprint-4



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

S.No	Component	Description	Technology
1.	User Interface	How user interacts with application e.g. Web UI, Mobile App, Chatbot, etc.	HTML, CSS, JavaScript, IBM Cloud Object Storage, Python-Flask, Kubernetes, Docker, IBM DB2, IBM Container Registry.
2.	Application Logic	The logic for a process in the application	Python-Flask.
3.	Database	Data Type Configuration etc.	MySQL, etc.
4.	ChatBox	Chatbox for users to access help from a virtual assistant on the application.	IBM Watson Assistant
5.	Cloud Database	Database Service on Cloud	IBM DB2
6.	File Storage	File storage requirements	IBM Cloud Object Storage
7.	App Container	Contain the whole application in a single container.	Docker Container, IBM Container Registry
8.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud Local Server Configuration: port 5000 Cloud Server Configuration :	Local, Cloud Foundry, Kubernetes.
9.	Send Mail	To send emails when low stock is present in the inventory to retailers.	IBM SendGrid

### 13.3 Functional Requirement

#### **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 Registration through Username and Password
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Sign In	Sign in to the application by Gmail, Username and Password.
FR-4	Dashboard	Can view the product details.
FR-5	Ordering	Order required products by putting them in a cart first.
FR-6	Restocking	Ordering more products when the stock is low.

Following are the non-functional requirements,

#### **Non-functional Requirements:**

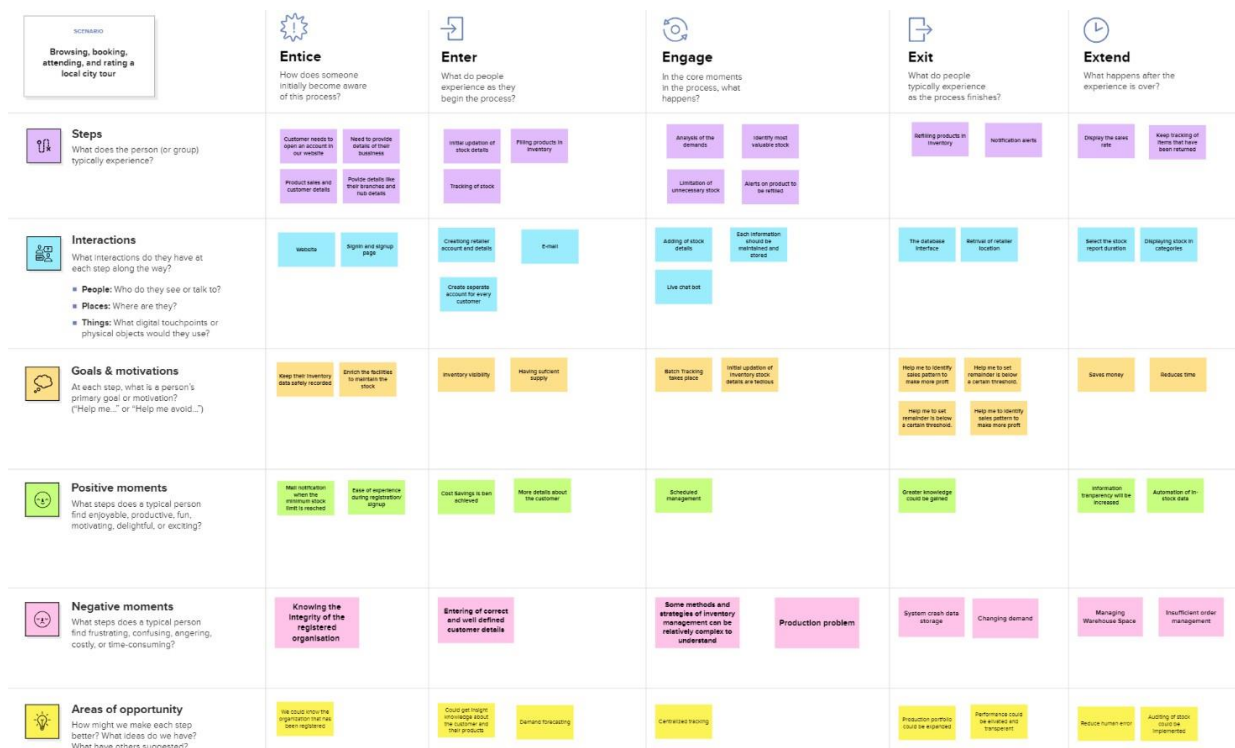
Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	<b>Usability</b>	Designing/Developing the site to be having a learning curve. Having simple and easy to navigate website for users. Attractive looking web-page. Making the site to be responsive for desktops and mobile users.
NFR-2	<b>Security</b>	The security should be strong as to the attackers wont be penetrating to the authorized users account or data. Log in system is used to prove authentication and authorization. Security can be increased by using OTP. Cookies based security system for authentication and improved visiting experience on the site for clients.



NFR-3	<b>Reliability</b>	Should be having the capacity to handle sufficient numbers of users and not be lagging or experiencing any discomfort when browsing when the web-page is busy. Should have minimum errors when executing the programs. Should be available even at the times of calamity.
NFR-4	<b>Performance</b>	The convenience of this is it reduces the time period of searching in aisle, searching for desired product, etc. It reduces costs, saves time, restocking period and predicts the best selling products. This makes the business more productive and profitable by having an organized management system.
NFR-5	<b>Availability</b>	This uses IBM DB2 to ensure high availability of database servers and performances.
NFR-6	<b>Scalability</b>	As DB2 is highly scalable, the coding can be produced and developed efficiently and new features can be introduced easily. Reusing the code can be done to add any new features. IBM Container in Docker registry is used which is highly scaleable.

## 13.4 Customer Journey Map



## 15. Project Planning Phase

### 15.1 Milestone and Activity List

Milestones	Activities	Description
Project Development Phase	Delivery of Sprint – 1,2,3,4	To develop the code and submit the developed code by testing it
Setting up App environment	Create IBM Cloud account	Signup for an IBM Cloud account
	Create flask project	Getting started with Flask to create project
	Install IBM Cloud CLI	Install IBM Command Line Interface
	Docker CLI Installation	Installing Docker CLI on laptop
	Create an account in sendgrid	Create an account in sendgrid. Use the service as email integration to our application for sending emails
Implementing web Application	Create UI to interact with Application	Create UI <ul style="list-style-type: none"><li>• Registration page</li><li>• Login page</li><li>• View products page</li><li>• Add products page</li></ul>
	Create IBM DB2 & connect with python	Create IBM DB2 service in IBM Cloud and connect with python code with DB
Integrating sendgrid service	Sendgrid integration with python	To send emails form the application we need to integrate the Sendgrid service
Developing a chatbot	Building a chatbot and Integrate to application	Build the chatbot and Integrate it to the flask application
Deployment of App in IBM Cloud	Containerize the App	Create a docker image of your application and push it to the IBM container registry
	Upload image to IBM container registry	Upload the image to IBM container registry
	Deploy in kubernetes cluster	Once the image is uploaded to IBM Container registry deploy the image to IBM Kebernetes cluster



Milestones	Activities	Description
Ideation Phase	Literature Survey	Literature survey on the selected project & information gathering
	Empathy Map	Prepare Empathy map to capture the user Pains & Gains, prepare list of problem statement
	Ideation	Organizing the brainstorming session and prioritise the top 3 ideas based on feasibility & Importance
Project Design Phase I	Proposed Solution	Prepare proposed solution document which includes novelty, feasibility of ideas, business model, social impact, Scalability of solution
	Problem Solution Fit	Prepare problem solution fit document
	Solution Architecture	Prepare solution architecture document
Project Design Phase II	Customer Journey	Prepare customer journey map to understand the user interactions & experience with the application
	Functional requirement	Prepare functional & non functional requirement document
	Data Flow Diagram Technology architecture	Prepare Data Flow Diagram and user stories Draw the technology architecture diagram
Project Planning Phase	Milestones & Activity list	Prepare milestones and activity list of the project
	Sprint Delivery Plan	Prepare sprint delivery plan

## 15.2 Sprint Delivery Plans

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	5	High	P.Harika P.Hanishya K.V Yogyaatha Nivedita K.B
Sprint-1		USN-2	As a user, I can register for the application through Gmail	5	High	P.Harika P.Hanishya K.V Yogyaatha Nivedita K.B
Sprint-1	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application	5	Medium	P.Harika P.Hanishya K.V Yogyaatha Nivedita K.B
Sprint-1	Login	USN-4	As a user, I can log into the application by entering email & password	5	High	P.Harika P.Hanishya K.V Yogyaatha Nivedita K.B
Sprint-2	Dashboard	USN-5	As a user, I can view the products which are available	10	High	P.Harika P.Hanishya K.V Yogyaatha Nivedita K.B
Sprint-2	Stocks update	USN-6	As a user, I can add the products which are not available in the inventory and the restock the products to make them available	10	Medium	P.Harika P.Hanishya K.V Yogyaatha Nivedita K.B

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-3	Sales prediction	USN-7	As a user, I can be able to get access to sales report to predict the which product runs out of stocks fast . so it helps me to predict the restock of product	20	High	P.Harika P.Hanishya K.V Yogyaatha Nivedita K.B
Sprint-4	Customer care	USN-8	As a user, I am able to get in touch with customer care to enquire the doubts and problems I have faced	10	Medium	P.Harika P.Hanishya K.V Yogyaatha Nivedita K.B
Sprint-4	Feedback	USN-9	As a user, I am able to send feedback forms to resolve issues and improve the system.	10	Medium	P.Harika P.Hanishya K.V Yogyaatha Nivedita K.B

### Project Tracker, Velocity & Burndown Chart: (4 Marks)

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	05 Nov 2022	20	19 Nov 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	16 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	16 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

**Velocity:**

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

**Sprint 1: 3.33**

**Sprint 2: 3.33**

**Sprint 3: 3.33**

**Sprint 4: 3.33**

## 16. Project Development Phase

### Front end:

#### Index.html

```
<!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="/css/style.css">
  <link rel="stylesheet" href="/css/common.css">
  <title>Home</title>
</head>
<body>
  <header id="home-header">
    <ul>
      <li><a href="/login.html">Login</a></li>
    </ul>
  </header>
  <script src="/js/main.js"></script>
</body>
</html>
```

## Login.html

```
<!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="/css/common.css" />
    <link rel="stylesheet" href="/css/account-forms.css" />
    <title>Login</title>
  </head>
  <body>
    <form class="acc-form">
      <h3>Login</h3>
      <div>
        <label for="email">Email
          <span class="required-input">*</span>
        </label>
        <input type="email" name="email" id="email" class="acc-input" required />
      </div>
      <div>
        <label for="password">Password
          <span class="required-input">*</span>
        </label>
        <input type="password" name="password" id="password" class="acc-input" required />
      </div>
      <div>
        <button type="submit" id="login-btn" class="btn acc-submit-btn">Login</button>
        <p class="text-center">Not registered?
          <a href="/register.html">Create an account</a>
        </p>
      </div>
    </form>
  </body>
</html>
```

## Register.html

```
<!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="/css/common.css" />
  <link rel="stylesheet" href="/css/account-forms.css" />
  <title>Register</title>
</head>
<body>
  <form class="acc-form">
    <h3>Register</h3>
    <div>
      <label for="name">Name
        <span class="required-input">*</span>
      </label>
      <input type="text" name="name" id="name" class="acc-input" required />
    </div>
    <div>
      <label for="email">Email
        <span class="required-input">*</span>
      </label>
      <input type="email" name="email" id="email" class="acc-input" required />
    </div>
    <div>
      <label for="password">Password
        <span class="required-input">*</span>
      </label>
      <input type="password" name="password" id="password" class="acc-input" required />
    </div>
    <div>
      <label for="re-password">Re-password
        <span class="required-input">*</span>
      </label>
      <input type="re-password" name="re-password" id="re-password" class="acc-input" required />
    </div>
  </form>
</body>
</html>
```

```

</div>
<div>
  <label for="address">Address</label>
  <textarea id="address" class="acc-input" rows="3"></textarea>
</div>
<div>
  <button type="submit" id="register-btn" class="btn acc-submit-btn">Register</button>
  <p class="text-center">Already registered?
    <a href="/login.html">Login</a>
  </p>
</div>
</form>
</body>
</html>

```

## CSS files:

### Account-forms.css

```

body {
  background-image: linear-gradient(
    to right top,
    #7081c9,
    #282682,
    #0630eaeb,
    #282682,
    #7081c9
  );
  display: grid;
  place-items: center;
}

.acc-form {
  width: min(90%, 500px);
  margin: 2rem 0;
  padding: 3rem 2.5rem;
  display: flex;
  flex-direction: column;
  background-color: var(--primary);
}

```

```
    box-shadow: 3px 3px 15px #00000020, -3px -2px 30px #00000020;
}
```

```
.acc-form > * {
    margin: 0.75rem 0;
}
```

```
.acc-input {
    width: 100%;
    padding: 0.75rem 1rem;
    outline: none;
    border: 2px solid #888;
    margin-top: 0.5rem;
    margin-bottom: 0.5rem;
}
```

```
.acc-input:focus {
    border-color: var(--accent);
}
```

```
.acc-submit-btn {
    width: 100%;
    margin: 1rem 0 2rem;
    padding: 0.75rem 1rem;
    font-weight: bold;
}
```

```
.acc-form a {
    color: var(--accent);
    font-weight: bold;
}
```

```
.acc-form a:hover,
.acc-form a:focus {
    color: var(--dark-accent);
    text-decoration: underline;
}
```

```
}
```

```
.acc-form > h3 {  
  text-transform: uppercase;  
  color: var(--accent);  
  font-size: 1.8rem;  
  text-align: center;  
}
```

```
.required-input {  
  color: var(--accent);  
  font-weight: bold;  
  margin-left: 0.25rem;  
}
```

```
textarea.acc-input {  
  resize: none;  
}
```

## Common.css

```
@import url("https://fonts.googleapis.com/css2?family=Source+Sans+Pro&display=swap");
```

```
*,  
*::before,  
*::after {  
  margin: 0;  
  padding: 0;  
  box-sizing: border-box;  
}
```

```
::root {  
  --primary: #fff;  
  --secondary: #000;  
  --accent: #281996;  
  --dark-accent: #281996;  
}
```

```
body {
```



```
    min-height: 100vh;
    font-size: 1.2rem;
    font-family: "Source Sans Pro", sans-serif;
}
```

```
li {
    list-style: none;
}
```

```
a {
    text-decoration: none;
    color: inherit;
}
```

```
input,
button,
textarea {
    font: inherit;
}
```

```
.btn {
    color: var(--primary);
    background-color: var(--accent);
    border: none;
    cursor: pointer;
    transition: transform 0.25s ease, background-color 0.25s ease;
}
```

```
.btn:hover {
    background-color: var(--dark-ascent);
    transform: skew(2deg);
}
```

```
.text-center {
    text-align: center;
}
```

## Backend:

### Requirements:

alembic==1.8.1  
click==8.1.3  
dnspython==2.2.1  
email-validator==1.3.0  
Flask==2.2.2  
Flask-SQLAlchemy==3.0.2  
Flask-WTF==1.0.1  
greenlet==1.1.3.post0  
ibm-db==3.1.3  
ibm-db-sa==0.3.8  
idna==3.4  
itsdangerous==2.1.2  
Jinja2==3.1.2  
Mako==1.2.3  
MarkupSafe==2.1.1  
python-dotenv==0.21.0  
SQLAlchemy==1.4.42  
Werkzeug==2.2.2  
WTForms==3.0.1

## Wsgi.py

```
from application import create_app
```

```
app = create_app()
```

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

## Config.py

```
from dotenv import load_dotenv  
load_dotenv()
```

```
import os
```

```
basedir = os.path.abspath(os.path.dirname(__file__))
```

```
CONFIG_OBJECT_NAME = os.getenv('CONFIG_OBJECT_NAME', 'config.DevelopmentConfig')
```

```
class Config():
```

```
    SECRET_KEY = os.urandom(24)
```

```
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

```
    API_VERSION_PREFIX = os.getenv('API_VERSION_PREFIX', '/api/v1/')
```

```
class DevelopmentConfig(Config):
```

```
    SQLALCHEMY_ECHO = True
```

```
    SQLALCHEMY_DATABASE_URI = f'sqlite:///{'os.path.join(basedir, "app.db")}'
```

```
class ProductionConfig(Config):
```

```
    SQLALCHEMY_DATABASE_URI = os.getenv('DB_URL')
```

## Lib

### Response.py

```
from flask import jsonify
```

```
class Response():
```

```
    @staticmethod
```

```
    def success(data):
```

```
        response = {
```

```
            'status': 'success',
```

```
            'data': data
```

```
        }
```

```
        return jsonify(response), 200
```

```
    @staticmethod
```

```
    def error(data, error_code):
```

```
        response = {
```

```
            'status': 'error',
```

```
            'code': error_code,
```

```
            'data': data
```

```
        }
```

```
        return jsonify(response), error_code
```

## Application

### \_\_init\_\_.py

```
import ibm_db_sa
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from config import CONFIG_OBJECT_NAME

db = SQLAlchemy()

def create_app():
    app = Flask(__name__)
    app.config.from_object(CONFIG_OBJECT_NAME)
    db.init_app(app)

    from application.auth.routes import auth_blueprint

    app.register_blueprint(auth_blueprint, url_prefix=app.config['API_VERSION_PREFIX'])

    with app.app_context():
        db.create_all()

    return app
```

## Auth

### Controllers.py

```
from application import db
from application.auth.forms import RegistrationForm, LoginForm
from application.auth.models import Retailer
from lib.response import Response

def register():
    form = RegistrationForm()
    if form.validate():
        retailer_data = {}
        retailer_data['name'] = form.name.data
        retailer_data['email'] = form.email.data
```

```

retailer_data['address'] = form.address.data
retailer = Retailer(**retailer_data)
retailer.set_password(form.password.data)
db.session.add(retailer)
db.session.commit()
return Response.success(data=retailer.id)
return Response.error(data=form.errors, error_code=500)

```

```

def login():
    form = LoginForm()
    if form.validate():
        retailer = Retailer.query.filter_by(name=form.name.data).first()
        if retailer is None or not retailer.check_password(form.password.data):
            return Response.error(data='login error', error_code=500)
        return Response.success(data='login success', error_code=500)
    return Response.error(data=form.errors, error_code=500)

```

## Forms.py

```

from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired, Email, EqualTo, ValidationError
from application.auth.models import Retailer

```

```

class RegistrationForm(FlaskForm):
    name = StringField('Name', validators=[DataRequired()])
    email = StringField('Email', validators=[DataRequired(), Email()])
    address = StringField('Address', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    repeat_password = PasswordField('Repeat Password', validators=[DataRequired(), EqualTo('password')])

    def validate_username(self, name):
        user = Retailer.query.filter_by(name=name.data).first()
        if user is not None:
            raise ValidationError('Please use a different username.')

    def validate_email(self, email):
        user = Retailer.query.filter_by(email=email.data).first()

```

```
if user is not None:
    raise ValidationError('Please use a different email address.')
```

```
class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
```

## Models.py

```
from email.policy import default
from application import db
from werkzeug.security import generate_password_hash, check_password_hash
```

```
class Retailer(db.Model):
    __tablename__ = 'retailers'
    id = db.Column(db.Integer, primary_key=True, nullable=False, autoincrement=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), index=True, unique=True, nullable=False)
    address = db.Column(db.String(300), nullable=True)
    password_hash = db.Column(db.String(200), nullable=False)
    is_active = db.Column(db.Boolean, nullable=True)

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)
```

## Routes.py

```
from flask import Blueprint
from application.auth.controllers import register, login
```

```
auth_blueprint = Blueprint('auth', __name__)
```

```
auth_blueprint.add_url_rule(rule='/register',
                             view_func=register,
                             endpoint='register',
                             methods=['POST'])
```

```
auth_blueprint.add_url_rule(rule='/login',
```

```
view_func=login,  
endpoint='login',  
methods=['POST'])
```

## Docker file

FROM python:3.10-slim-buster

WORKDIR /app

COPY requirements.txt requirements.txt

RUN apt update

RUN apt install build-essential -y

RUN pip3 install --upgrade pip

RUN pip3 install -r requirements.txt

COPY ..

CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]

## Demo video:

[https://drive.google.com/file/d/1UgH3P0Tldx9kbbk1OvzNuu0w5pn-7qIZu/view?usp=share link](https://drive.google.com/file/d/1UgH3P0Tldx9kbbk1OvzNuu0w5pn-7qIZu/view?usp=share_link)

## Github link:

<https://github.com/IBM-EPBL/IBM-Project-30594-1660149805>

## Conclusion

To conclude, Inventory Management System is a simple application basically suitable for small organizations. It has basic items which are used for the small organization. Our team is successful in making the application where we can update, insert and delete the item as per the requirement. This application also provides a simple report on daily basis to know the daily sales and purchase details. This application matches for small organization where there small limited if godwin's. Through it has some limitations, our team strongly believes that the implementation of this system will surely benefit the organization