

PROJECT REPORT

Date	14.11.2022
Team ID	PNT2022TMID02970
Project Name	Inventory Management System For Retailers
Team Members	Pradeep Sriram T (727719EUIT115) Nithyananthan K V (727719EUIT110) Prithiv T S(727719EUIT120) Mohamed Shiham (727719EUIT093)

I Introduction

1.1.1.1 Project Overview

1.1.1.2 Purpose

II Literature Survey

2.1 Existing Problem

2.2 Problem Statement Definition

III Ideation & Proposed Solution

3.1 Empathy Map

3.2 Ideation & Brainstorming

3.3 Proposed Solution

3.4 Problem Solution Fit

IV Requirement Analysis

4.1 Functional Requirements

4.2 Non-Functional Requirements

V Project Design

5.1 Data Flow Diagram

5.2 Solution & Technical Architecture

5.3 User Stories

VI Project Planning and Scheduling

6.1 Sprint planning and Estimation

6.2 Milestone And Activities

VII Coding and Solutioning

7.1 Feature I

7.2 Feature II

7.3 Code

VIII Result

IX Advantages and Disadvantages

X Conclusion

XI Future Scope

XII References

1. INTRODUCTION

1.1 Project Overview

The objective of this system is to manage the items in an inventory such as tracking orders, placing orders to other suppliers and checking the items in the inventory. The system allows the admin to maintain the items in the inventory.

Whenever the item levels go low, the system places an order to the supplier. The supplier gets the notification of these orders as soon as they are placed and can send the items to the inventory. There are two login pages each for the admin and supplier.

The software has been developed using the most powerful and secured backend Python and IBM Cloud for the databases and most widely accepted frontend JavaScript with HTML and CSS coding

1.2 Purpose

The primary purpose of inventory management is to ensure there is enough goods or materials to meet demand without creating overstock, or excess inventory

Retail management refers to the process of helping customers find products in your store. It includes everything from increasing your customer pool to how products are presented, and how you fulfill a customer's needs. A good store manager helps customers leave the store with a smile.

2. LITERATURE SURVEY

2.1 Existing problem

- The problem faced by the company is they do not have any systematic system to record and keep their inventory data. It is difficult for the admin to record the inventory data quickly and safely because they only keep it in the logbook and not properly organized.
- Good planning and sales forecast before setting optimal inventory levels, appropriate inventory management requires close coordination between the areas of sales, purchasing and finance.

2.2 Problem Statement Definition

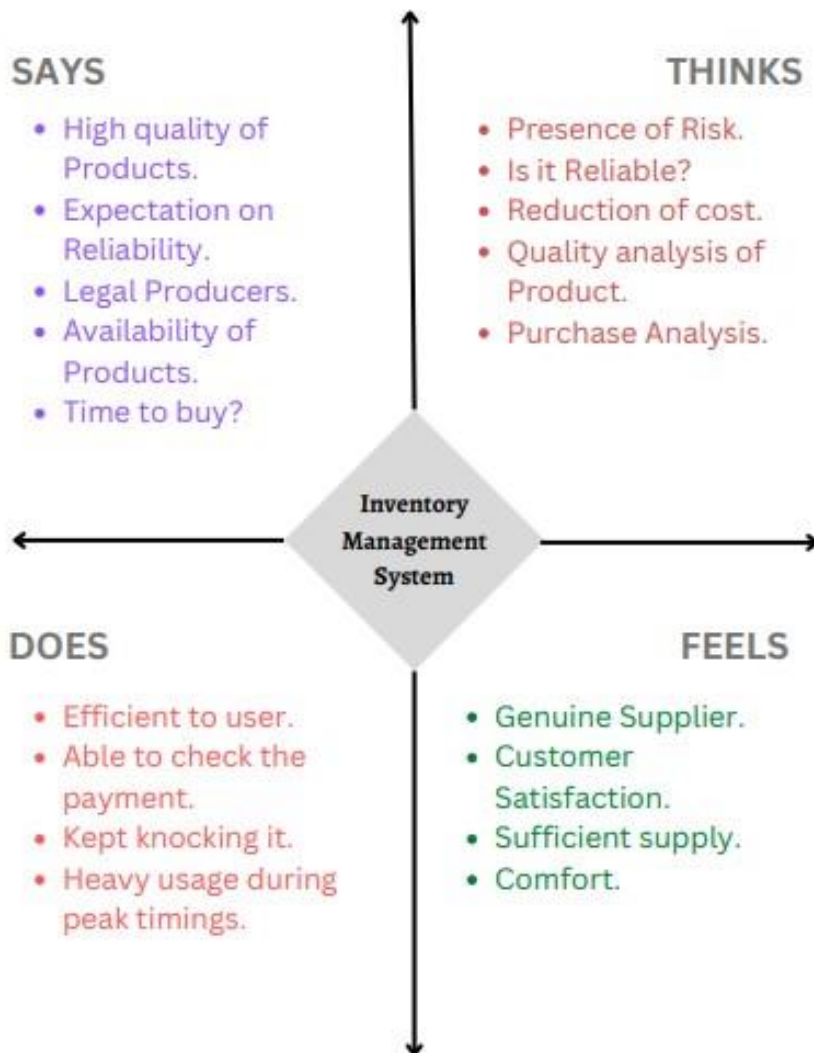
Retail inventory management works by creating systems to log products, receive them into inventory, track changes when sales occur, manage the flow of goods from purchasing to final sale and check stock counts.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes. It is a useful tool to help teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

Inventory Management System For Retailers



3.2 Ideation & Brainstorming

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 amount 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.

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



Step-2: Brainstorm, Idea Listing and Grouping

2

Brainstorm

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

10 minutes

TIP

You can select a sticky note a pencil (or stick) to pin to your drawing!

Pradeep Sriram T

Product	Product	Product
Product	Product	Product
Product	Product	Product
Product	Product	Product

Prithevi T S

Product	Product	Product
Product	Product	Product
Product	Product	Product
Product	Product	Product

Nithyananthan K V

Product	Product	Product
Product	Product	Product
Product	Product	Product
Product	Product	Product

Muhammed Shihab S

Product	Product	Product
Product	Product	Product
Product	Product	Product
Product	Product	Product

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes 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.

20 minutes

TIP

Add contextual tags to sticky notes to make it easier to find, organize, separate, and categorize important ideas as they're added to your mural.

Product Inventory

Product	Product	Product
Product	Product	Product
Product	Product	Product
Product	Product	Product
Product	Product	Product

Customer Manager

Product	Product	Product
Product	Product	Product
Product	Product	Product
Product	Product	Product
Product	Product	Product

Supplier Manager

Product	Product
Product	Product
Product	Product

Sales Product

Product	Product
Product	Product
Product	Product

Account Statistics

Product	Product
Product	Product
Product	Product

Expansion of products

Product	Product
Product	Product
Product	Product

Step-3: Idea Prioritization

4

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

+

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 session.

Export the mural

Export a copy of the mural as a PNG or PDF to attach to emails, include in slides, or save in your drive.

Keep moving forward

Strategy blueprint

Outline the components of a new idea or strategy.

Open the template →

Customer experience journey map

Unleash customer needs, motivations, and obstacles 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

9

3.3 Proposed Solution

The system customizes and only shows recommended jobs based on the user's skill set and preferences (Using graphql api)

Similarly, the same recommendation system helps provide job applicant recommendations to the job recruiters to find the most eligible candidates for their firm.

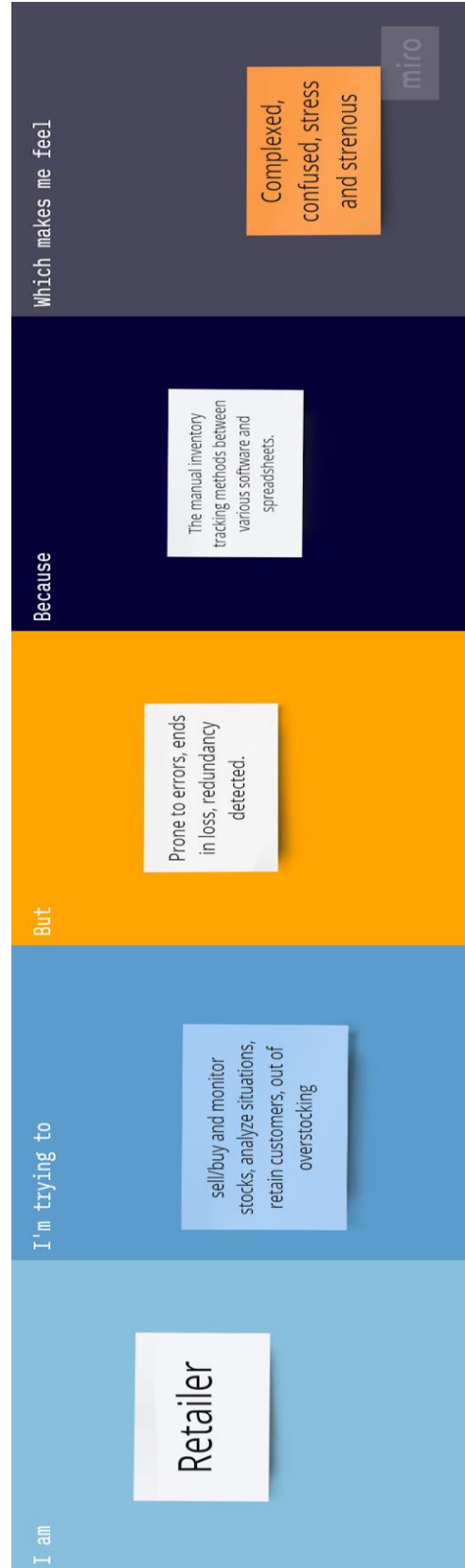
All important data - job seeker's and hoster's personal information needs to be also stored safely and securely. Using a sql database is the most easiest, safest and convenient way possible.

Data needs to also be private in some cases like when information is shared with the host while applying for a job.

3.4 Problem Solution fit



3.5 Customer Problem Statement



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

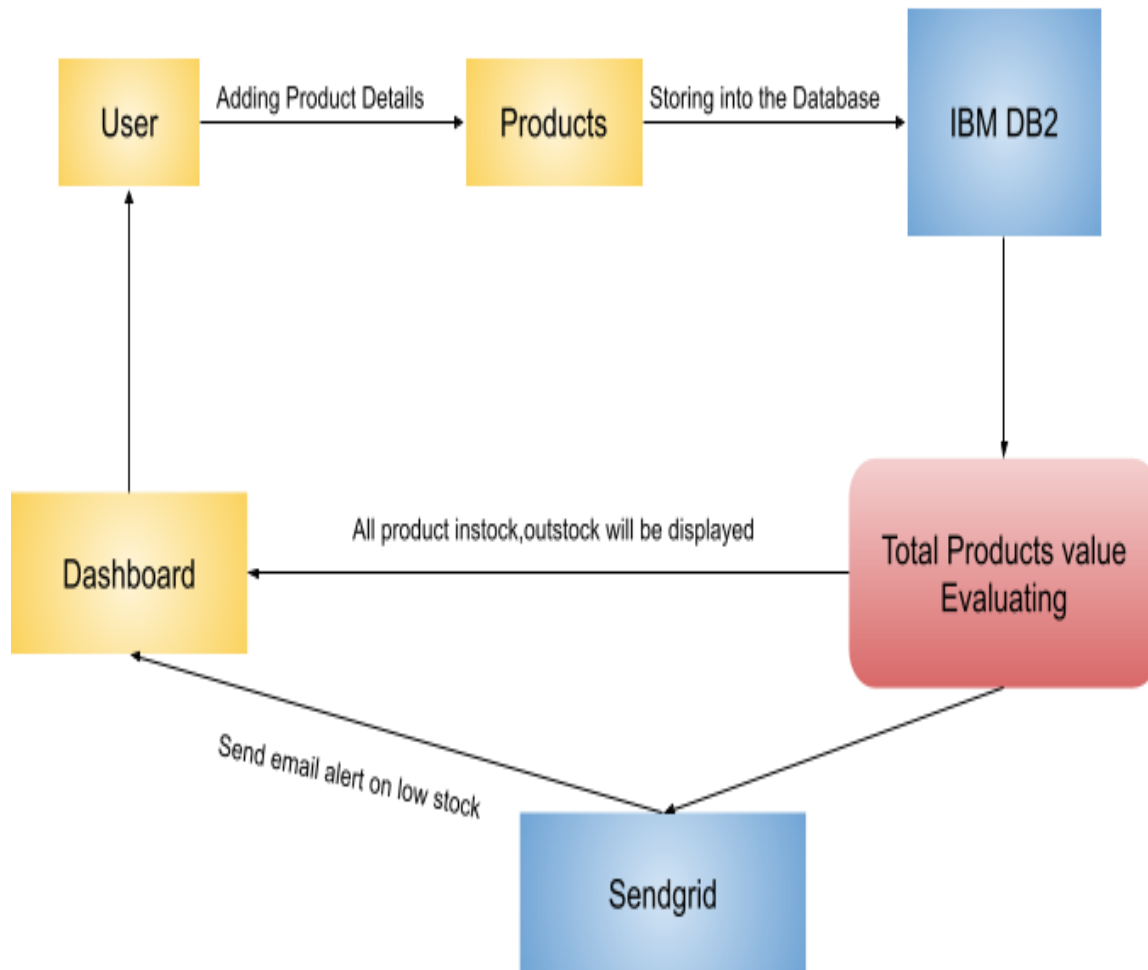
- The System aims at providing an efficient interface to the user for managing of inventory, it shall also provide the user varied options for managing the inventory through various functions at hand. The ingredient levels are continuously monitored based on their usage and are checked for the threshold levels in the inventory and accordingly the user is alerted about low levels of certain ingredients. The design is such that the user does not have to manually update the inventory every time, the System does it for the user.
- The System calculates and predicts the amount of usage for specific set days that are pre-set by the user(admin) , it also alerts the user of an impending action to order ingredients before the specific day set by the user. Therefore the user never has to worry about manually calculating the estimated usage of the ingredients as the System does it for the user.
- The simple interface of the System has functions like adding a recipe, removing or updating the recipe. It also extends to functions such as adding a vendor for an ingredient,, removing the vendor, checking threshold levels, processing orders, altering processed orders etc.

4.2 Non-Functional requirements

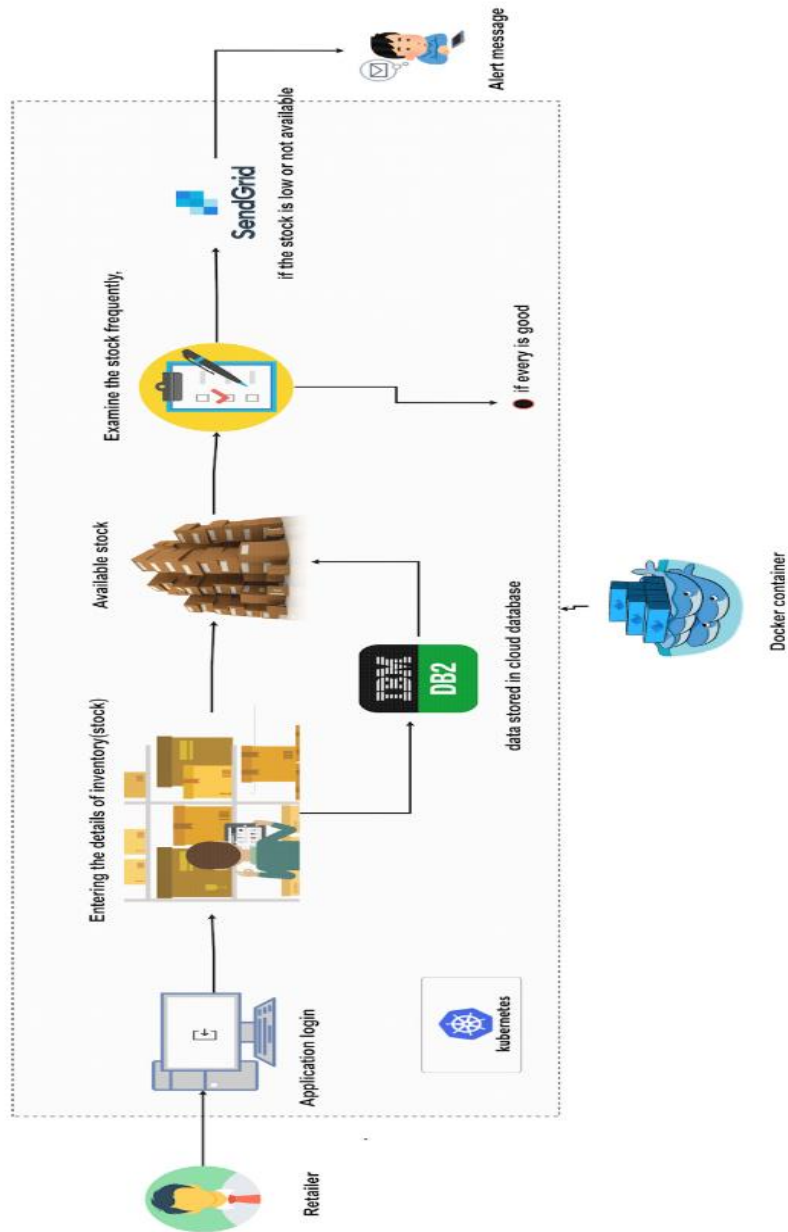
- The system must not lag, because the workers using it don't have down-time to wait for it to complete an action.
- The system must complete updating the databases, adding of recipe, ingredient, vendor and occasions successfully every time the user requests such a process.
- All the functions of the system must be available to the user every time the system is turned on.
- The calculations performed by the system must comply according to the norms set by the user and should not vary unless explicitly changed by the user
- The System must give accurate inventory status to the user continuously. Any inaccuracies are taken care by the regular confirming of the actual levels with the levels displayed in the system.
- The System must successfully add any recipe, ingredients, vendors or special occasions given by the user and provide estimations and inventory status in relevance with the newly updated entities.

5. PROJECT DESIGN

5.1 Data Flow Diagrams



5.2 Solution & Technical Architecture



5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Retailer	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 will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	Medium	Sprint-1
	Login	USN-3	As a user, I can log into the application by entering email & password	I can access my account /dashboard	High	Sprint-1
	Dashboard	USN-4	As a user, I can view the stock list and suppliers list	Once I log in to the system, I can be able to view the stocks	Medium	Sprint-1
	Items	USN-5	As a user, I can add the items.	I can create a new type of item	High	Sprint-2
		USN-6	As a user, I can see the items	I can be able to see the items that can be added to the inventory	Low	Sprint-2

	Inventory	USN-7	As a user, I can add the items to inventory.	I can add items to the inventory with quantity	High	Sprint-2
		USN-8	As a user, I can see the items in the inventory.	I can see the inventory items with quantity	Low	Sprint-2
	Indication	USN-9	As a user, I can be able to receive indication	I receive a notification when the stock running low	High	Sprint-3
	Location	USN-10	As a user, I can be able to see items from a particular store location	I can be able to make purchase from a particular location	Medium	Sprint-3
		USN-11	As a user, I can add a new location of my store	I can be able to add new store locations	Medium	Sprint - 3
Customer	Purchase	USN -12	As a customer, I can be able to purchase good from the particular location of the store	I can be able to purchase from the store	High	Sprint - 4
Retailer & Customer	Deployment	USN-13	As a user, I can access the software in the web	I can access the software in web	High	Sprint -4

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	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.	High	Split between all 4
Sprint 1		USN-2	As a user, I will receive confirmation email once I have registered for the application	Medium	Split between all 4
Sprint 1	Login	USN-3	As a user, I can log into the application by entering email & password	High	Nithyananthan & Prithiv
Sprint 2	Items	USN-5	As a user, I can add the items.	High	Mohamed Shiham
Sprint 2		USN-6	As a user, I can see the items	Low	Nithyananthan
Sprint 2	Inventory	USN-7	As a user, I can add the items to inventory.	High	Pradeep Sriram
Sprint 2		USN-8	As a user, I can see the items in the inventory.	Low	Prithiv
Sprint 3	Indication	USN-9	As a user, I can be able to receive indication	High	Pradeep Sriram
Sprint 3	Location	USN-10	As a user, I can be able to see items from a particular store location	Medium	Mohamed Shiham
Sprint 3		USN-11	As a user, I can add a new location of my store	Medium	Nithyananthan
Sprint 4	Purchase	USN -12	As a customer, I can be able to purchase good from the particular location of the store	High	Prithiv

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Team Members
Sprint 4	Deployment	USN-13	As a user, I can access the software in the web	High	Pradeep Sriram

6.2 Milestone And Activities

Milestones and Activities:

Milestones	Activities
Registration	<ul style="list-style-type: none">• Retailer Registration
Login	<ul style="list-style-type: none">• Retailer Login
Item Actions	<ul style="list-style-type: none">• Add item type• Display item types
Inventory Actions	<ul style="list-style-type: none">• Add stock to inventory• Display stock from inventory
Notification	<ul style="list-style-type: none">• Sent notification when threshold of stock reached
Location	<ul style="list-style-type: none">• Add store location• Update operations to handle location
Customer	<ul style="list-style-type: none">• Able to see stocks• Able to make purchase
Deployment	<ul style="list-style-type: none">• Project deployment in IBM Cloud

7. CODING&SOLUTIONING

(Explain the features added in the project along with code)

7.1 Feature 1

Complete insights into key products and service drivers. With the help of tables and symbols, marketers can effectively track and analyse factors that have an effect on important bottom lines like profitability. Store Managers can also effectively optimise product mix across channels, lines and brands with the product scorecards available. Some of the different KPIs that managers can avail of from product performance metrics are product sales by region, change in sales and margin per product, ROI per product, top competitor by product category and much more..

7.2 Feature 2

The entire organisation can access the same store data simultaneously and thus everyone has an understanding of what the customer wants. Managers can better monitor progress, respond immediately to customer needs, adjust parameters for continuous improvement, and exercise greater control over the organisation.

One can record and analyze inventory results and merchandise processes daily to know whether business decisions are based on timely, accurate information.

7.3 Code

```
1  from flask import Flask, render_template, url_for, request, redirect
2  from flask_sqlalchemy import SQLAlchemy
3  from collections import defaultdict
4  from datetime import datetime
5
6  app = Flask(__name__)
7  app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///inventory.db'
8  db = SQLAlchemy(app)
9
10
11 class Product(db.Model):
12
13     __tablename__ = 'products'
14     product_id = db.Column(db.String(200), primary_key=True)
15     date_created = db.Column(db.DateTime, default=datetime.utcnow)
16
17     def __repr__(self):
18         return '<Product %r>' % self.product_id
19
20 class Location(db.Model):
21
22     __tablename__ = 'locations'
23     location_id = db.Column(db.String(200), primary_key=True)
24     date_created = db.Column(db.DateTime, default=datetime.utcnow)
25
26     def __repr__(self):
27         return '<Location %r>' % self.location_id
28
29 class ProductMovement(db.Model):
30
31     __tablename__ = 'productmovements'
32     movement_id = db.Column(db.Integer, primary_key=True)
33     product_id = db.Column(db.Integer, db.ForeignKey('products.product_id'))
34     qty = db.Column(db.Integer)
35     from_location = db.Column(db.Integer, db.ForeignKey('locations.location_id'))
36     to_location = db.Column(db.Integer, db.ForeignKey('locations.location_id'))
37     movement_time = db.Column(db.DateTime, default=datetime.utcnow)
38
39     product = db.relationship('Product', foreign_keys=product_id)
40     fromLoc = db.relationship('Location', foreign_keys=from_location)
41     toLoc = db.relationship('Location', foreign_keys=to_location)
```

```

42     def __repr__(self):
43         return '<ProductMovement %r>' % self.movement_id
44
45 @app.route('/', methods=["POST", "GET"])
46 def index():
47
48     if (request.method == "POST") and ('product_name' in request.form):
49         product_name = request.form["product_name"]
50         new_product = Product(product_id=product_name)
51
52         try:
53             db.session.add(new_product)
54             db.session.commit()
55             return redirect("/")
56
57         except:
58             return "There Was an issue while add a new Product"
59
60     if (request.method == "POST") and ('location_name' in request.form):
61         location_name = request.form["location_name"]
62         new_location = Location(location_id=location_name)
63
64         try:
65             db.session.add(new_location)
66             db.session.commit()
67             return redirect("/")
68
69         except:
70             return "There Was an issue while add a new Location"
71     else:
72         products = Product.query.order_by(Product.date_created).all()
73         locations = Location.query.order_by(Location.date_created).all()
74         return render_template("index.html", products = products, locations = locations)
75
76 @app.route('/locations/', methods=["POST", "GET"])
77 def viewLocation():
78     if (request.method == "POST") and ('location_name' in request.form):
79         location_name = request.form["location_name"]
80         new_location = Location(location_id=location_name)

```

```

82         try:
83             db.session.add(new_location)
84             db.session.commit()
85             return redirect("/locations/")
86
87         except:
88             locations = Location.query.order_by(Location.date_created).all()
89             return "There Was an Issue while add a new Location"
90     else:
91         locations = Location.query.order_by(Location.date_created).all()
92         return render_template("locations.html", locations=locations)
93
94 @app.route('/products/', methods=["POST", "GET"])
95 def viewProduct():
96     if (request.method == "POST") and ('product_name' in request.form):
97         product_name = request.form["product_name"]
98         new_product = Product(product_id=product_name)
99
100         try:
101             db.session.add(new_product)
102             db.session.commit()
103             return redirect("/products/")
104
105         except:
106             products = Product.query.order_by(Product.date_created).all()
107             return "There Was an Issue while add a new Product"
108     else:
109         products = Product.query.order_by(Product.date_created).all()
110         return render_template("products.html", products=products)
111
112 @app.route("/update-product/<name>", methods=["POST", "GET"])
113 def updateProduct(name):
114     product = Product.query.get_or_404(name)
115     old_product_id = product.product_id
116
117     if request.method == "POST":
118         product.product_id = request.form['product_name']
119
120     try:
121         db.session.commit()

```

```

121         db.session.commit()
122         updateProductInMovements(old_product, request.form['product_name'])
123         return redirect("/products/")
124
125     except:
126         return "There was an issue while updating the Product"
127     else:
128         return render_template("update-product.html", product=product)
129
130 @app.route("/delete-product/<name>")
131 def deleteProduct(name):
132     product_to_delete = Product.query.get_or_404(name)
133
134     try:
135         db.session.delete(product_to_delete)
136         db.session.commit()
137         return redirect("/products/")
138     except:
139         return "There was an issue while deleting the Product"
140
141 @app.route("/update-location/<name>", methods=["POST", "GET"])
142 def updateLocation(name):
143     location = Location.query.get_or_404(name)
144     old_location = location.location_id
145
146     if request.method == "POST":
147         location.location_id = request.form['location_name']
148
149         try:
150             db.session.commit()
151             updateLocationInMovements(
152                 old_location, request.form['location_name'])
153             return redirect("/locations/")
154
155         except:
156             return "There was an issue while updating the Location"
157     else:
158         return render_template("update-location.html", location=location)
159
160 @app.route("/delete-location/<name>")

```

```

161 def deleteLocation(id):
162     location_to_delete = Location.query.get_or_404(id)
163
164     try:
165         db.session.delete(location_to_delete)
166         db.session.commit()
167         return redirect("/locations/")
168     except:
169         return "There was an issue while deleteing the Location"
170
171 @app.route("/movements/", methods=["POST", "GET"])
172 def viewMovements():
173     if request.method == "POST" :
174         product_id = request.form["productId"]
175         qty = request.form["qty"]
176         fromLocation = request.form["fromLocation"]
177         toLocation = request.form["toLocation"]
178         new_movement = ProductMovement(
179             product_id=product_id, qty=qty, from_location=fromLocation, to_location=toLocation)
180
181         try:
182             db.session.add(new_movement)
183             db.session.commit()
184             return redirect("/movements/")
185
186         except:
187             return "There Was an issue while add a new Movement"
188     else:
189         products = Product.query.order_by(Product.date_created).all()
190         locations = Location.query.order_by(Location.date_created).all()
191         movs = ProductMovement.query\
192             .join(Product, ProductMovement.product_id == Product.product_id)\
193             .add_columns(
194                 ProductMovement.movement_id,
195                 ProductMovement.qty,
196                 Product.product_id,
197                 ProductMovement.from_location,
198                 ProductMovement.to_location,
199                 ProductMovement.movement_time)\
200             .all()

```



```

201
202     movements = ProductMovement.query.order_by(
203         ProductMovement.movement_time).all()
204     return render_template("movements.html", movements=movs, products=products, locations=locations)
205
206 @app.route("/update-movement/<int:id>", methods=["POST", "GET"])
207 def updateMovement(id):
208
209     movement = ProductMovement.query.get_or_404(id)
210     products = Product.query.order_by(Product.date_created).all()
211     locations = Location.query.order_by(Location.date_created).all()
212
213     if request.method == "POST":
214         movement.product_id = request.form["productId"]
215         movement.qty = request.form["qty"]
216         movement.from_location = request.form["fromLocation"]
217         movement.to_location = request.form["toLocation"]
218
219         try:
220             db.session.commit()
221             return redirect("/movements/")
222
223         except:
224             return "There was an issue while updating the Product Movement"
225     else:
226         return render_template("update-movement.html", movement=movement, locations=locations, products=products)
227
228 @app.route("/delete-movement/<int:id>")
229 def deleteMovement(id):
230     movement_to_delete = ProductMovement.query.get_or_404(id)
231
232     try:
233         db.session.delete(movement_to_delete)
234         db.session.commit()
235         return redirect("/movements/")
236     except:
237         return "There was an issue while deleteing the Prodcut Movement"
238
239 @app.route("/product-balance/", methods=["POST", "GET"])
240 def productBalanceReport():

```



```

241     movs = ProductMovement.query.\
242         join(Product, ProductMovement.product_id == Product.product_id).\
243         add_columns(
244             Product.product_id,
245             ProductMovement.qty,
246             ProductMovement.from_location,
247             ProductMovement.to_location,
248             ProductMovement.movement_time).\
249         order_by(ProductMovement.product_id).\
250         order_by(ProductMovement.movement_id).\
251         all()
252     balancedDict = defaultdict(lambda: defaultdict(dict))
253     tempProduct = ''
254     for mov in movs:
255         row = mov[0]
256         if(tempProduct == row.product_id):
257             if(row.to_location and not "qty" in balancedDict[row.product_id][row.to_location]):
258                 balancedDict[row.product_id][row.to_location]["qty"] = 0
259             elif (row.from_location and not "qty" in balancedDict[row.product_id][row.from_location]):
260                 balancedDict[row.product_id][row.from_location]["qty"] = 0
261             if (row.to_location and "qty" in balancedDict[row.product_id][row.to_location]):
262                 balancedDict[row.product_id][row.to_location]["qty"] += row.qty
263             if (row.from_location and "qty" in balancedDict[row.product_id][row.from_location]):
264                 balancedDict[row.product_id][row.from_location]["qty"] -= row.qty
265             pass
266         else :
267             tempProduct = row.product_id
268             if(row.to_location and not row.from_location):
269                 if(balancedDict):
270                     balancedDict[row.product_id][row.to_location]["qty"] = row.qty
271                 else:
272                     balancedDict[row.product_id][row.to_location]["qty"] = row.qty
273
274     return render_template("product-balance.html", movements=balancedDict)
275
276 @app.route("/movements/get-from-locations/", methods=["POST"])
277 def getLocations():
278     product = request.form["productId"]
279     location = request.form["location"]
280     locationDict = defaultdict(lambda: defaultdict(dict))

```

```

298     location = request.form["location"]
299     locations = Location.query.\
300         filter(Location.location_id == location).\
301         all()
302     print(locations)
303     if locations:
304         return {"output": False}
305     else:
306         return {"output": True}
307
308 @app.route("/dub-products/", methods=["POST", "GET"])
309 def getPDuplicate():
310     product_name = request.form["product_name"]
311     products = Product.query.\
312         filter(Product.product_id == product_name).\
313         all()
314     print(products)
315     if products:
316         return {"output": False}
317     else:
318         return {"output": True}
319
320 def updateLocationInMovements(oldLocation, newLocation):
321     movement = ProductMovement.query.filter(ProductMovement.from_location == oldLocation).all()
322     movement2 = ProductMovement.query.filter(ProductMovement.to_location == oldLocation).all()
323     for mov in movement2:
324         mov.to_location = newLocation
325     for mov in movement:
326         mov.from_location = newLocation
327
328     db.session.commit()
329
330 def updateProductInMovements(oldProduct, newProduct):
331     movement = ProductMovement.query.filter(ProductMovement.product_id == oldProduct).all()
332     for mov in movement:
333         mov.product_id = newProduct
334
335     db.session.commit()
336
337 if (__name__ == "__main__"):
338     app.run(debug=True)

```

8. RESULTS

8.1 Performance Metrics

Inventory Performance is a measure of how effectively and efficiently inventory is used and replenished. The goal of inventory performance metrics is to compare actual on-hand dollars versus forecasted cost of goods sold. Many Lean practitioners claim that inventory performance is the single best indicator of the overall operational performance of a facility.

9. ADVANTAGES & DISADVANTAGES

- Paper-based retail inventory management can take a lot of time and effort. The retail inventory management software can cut short your in-store inventory process cycles through automation. Automation would give you time to focus on other productive business tasks.
- Inventory management is one of the crucial retail processes. Thus, any discrepancy in the inventory control would impact all other operations in your company. The retail inventory software can streamline the inventory processes, which would, in turn, improve the efficiency of your entire business
- Manual inventory control would increase your labor and process costs. The software would not only help you save time, but it would also help you reduce costs. As a result, the profitability of your business would improve. Also, you can invest the excess funds in activities that promote your business growth.
- One of the biggest problems with any computerized system is the potential for a system crash. A corrupt hard drive, power outages and other technical issues can result in the loss of needed data. At the least, businesses are interrupted when they are unable to access data they need. Business owners should back up data regularly to protect against data loss.
- Hackers look for any way to get company or consumer information. An inventory system connected to point-of-sale devices and accounting is a valuable resource to hack into in search of potential financial information or personal details of owners, vendors or clients. Updating firewalls and anti-virus software can mitigate this potential issue.
- When everything is automated, it is easy to forego time-consuming physical inventory audits. They may no longer seem necessary when the computers are doing their work. However, it is important to continue to do regular audits to identify loss such as spoilage or breakage. Audits also help business owners identify potential internal theft and manipulation of the computerized inventory system.

10. **CONCLUSION**

Inventory management is a very complex but essential part of the supply chain. An effective inventory management system helps to reduce stock-related costs such as warehousing, carrying, and ordering costs. As you have read above, there are different techniques that businesses can utilize to simplify and optimize stock management processes and control systems.

11. **FUTURE SCOPE**

In summary, successful companies will embrace the challenges of inventory management in the 21st century by leveraging the technology that is being offered through the Fourth Industrial Revolution. More important, companies will look at inventory as a strategic asset, that when properly deployed will deliver increased value and competitive advantage. Effective collaboration between supply chain partners will take on increased importance. The intensifying risks inherent with global sourcing in combination with a better appreciation of TCO will motivate companies to rethink their global inventory strategies.

12. **REFERENCES**

- Aggarwal, S.: A review of current inventory theory and its applications. International Journal of Production Research 12, 443–472 (1974)
- Anily, S., Federgruen, A.: One warehouse multiple retailer systems with vehicle routing costs. Management Science 36, 92–114 (1990)
- Beckmann, M.: An inventory model for arbitrary interval and quantity distributions of demand. Management Science 8, 35–57 (1961)
- Hamann, T., Proth, J.: Inventory control of repairable tools with incomplete information. International Journal of Production Economics 31, 543–550 (1993)

GitHub link

<https://github.com/IBM-EPBL/IBM-Project-29634-1660127869.git>