



**NAALAIYA THIRAN PROJECT - 2022
19ECI01-PROFESSIONAL READINESS FOR
INNOVATION, EMPLOYABILITY AND
ENTREPRENEURSHIP**



IT - ITeS SSC
NASSCOM



INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

A PROJECT REPORT

Submitted by

BHARATHRAM V	1904006
LAKSHMANAN P	1904023
MATHANRAJ T	1904024
RAMCHANDAR K G	1904040

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**

COIMBATORE INSTITUTE OF TECHNOLOGY, COIMBATORE – 641 014

(Government Aided Autonomous Institution affiliated to Anna University)

**ANNA UNIVERSITY: CHENNAI 600025
NOVEMBER 2022**

COIMBATORE INSTITUTE OF TECHNOLOGY

(Government aided Autonomous Institution Affiliated to Anna University)

COIMBATORE – 641014

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this report “**INVENTORY MANAGEMENT SYSTEM**” is the bonafide work of **BHARATHRAM V (1904006), LAKSHMANAN P (1904023), MATHANRAJ T (1904024) AND RAMCHANDAR KG (1904040)** who carried out **19ECI01 Professional Readiness for Innovation, Employability and Entrepreneurship** project offered by IBM and Anna University, Chennai.

SIGNATURE

Dr.M.POONGOTHAI

FACULTY MENTOR

Professor

Department of Electronics

and Communication

Engineering

SIGNATURE

Dr.I.S.AKILA

FACULTY EVALUATOR

Associate Professor

Department of Electronics

and Communication

Engineering

SIGNATURE

Dr. A. RAJESWARI, M.E. Ph.D.,

PRINCIPAL AND HEAD

Professor

Department of Electronics

and

Communication Engineering

PROJECT CALENDER

Phase	Phase Description	Week	Dates	Activity Details
1	Preparation Phase (Pre-requisites, Registrations, Environment Set-up, etc.)	2	22 - 27 Aug 2022	Creation GitHub account & collaborate with Project repository in project workspace
2	Ideation Phase (Literature Survey, Empathize, Defining Problem Statement, Ideation)	2	29 Aug – 3rd Sept 2022	Literature survey (Aim, objective, problem statement and need for the project)
		3	5 - 10th Sept 2022	Preparing Empathy Map Canvas to capture the user Pains & Gains
		4	12 - 17 Sept 2022	Listing of the ideas using brainstorming session
3	Project Design Phase -I (Proposed Solution, Problem- Solution Fit, Solution Architecture)	5	19 - 24 Sept 2022	Preparing the proposed solution document
		6	26 Sept - 01 Oct 2022	Preparing problem - solution fit document & Solution Architecture
4	Project Design Phase -II (Requirement Analysis, Customer Journey, Data Flow Diagrams, Technology Architecture)	7	3 - 8 Oct 2022	Preparing the customer journey maps
		8	10 - 15 Oct 2022	Preparing the Functional Requirement Document & Data- Flow Diagrams and Technology Architecture
5	Project Planning Phase (Milestones & Tasks, Sprint Schedules)	9	17 - 22 Oct 2022	Preparing Milestone & Activity List, Sprint Delivery Plan
6	Project Development Phase (Coding & Solutioning, acceptance Testing, Performance Testing)	10	24 - 29 Oct 2022	Preparing Project Development - Delivery of Sprint-1
		11	31 Oct - 5 Nov 2022	Preparing Project Development - Delivery of Sprint-2
		12	7 - 12 Nov 2022	Preparing Project Development - Delivery of Sprint-3
		13	14 - 19 Nov 2022	Preparing Project Development - Delivery of Sprint-4

TABLE OF CONTENTS

CHAPTER NO	CONTENTS	PAGE NO
	LIST OF FIGURES	
	LIST OF TABLES	
1	INTRODUCTION 1.1 PROJECT OVERVIEW 1.2 PURPOSE	1
2	LITERATURE SURVEY 2.1 EXISTING SOLUTION 2.2 PROBLEM STATEMENT DEFINITION	2
3	IDEATION & PROPOSED SOLUTION 3.1 EMPATHY MAP CANVAS 3.2 IDEATION AND BRAINSTORMING 3.3 PROPOSED SOLUTION 3.4 PROBLEM SOLUTION FIT	3
4	REQUIREMENT ANALYSIS 4.1 FUNCTIONAL REQUIREMENTS 4.2 NON FUNCTIONAL REQUIREMENTS	8
5	PROJECT DESIGN 5.1 DATA FLOW DIAGRAMS 5.2 SOLUTION AND TECHNICAL ARCHITECTURE	10
6	PROJECT PLANNING & SCHEDULING 6.1 SPRINT PLANNING AND ESTIMATION 6.2 SPRINT DELIVERY SCHEDULE	13
7	CODING & SOLUTIONING 7.1 IBM CLOUD 7.2 FLASK FRAMEWORK 7.3 IBM DB2 MODULE 7.4 DOCKER CLI 7.5 IBM CLOUD CLI 7.6 SENDGRID API 7.7 KUBERNETES	16
8	TESTING AND RESULTS	19
9	PERFORMANCE RESULTS 9.1 PERFORMANCE METRICS	22
10	ADVANTAGES & DISADVANTAGES 10.1 ADVANTAGES 10.2 DISADVANTAGES	23
11	CONCLUSION	24
12	FUTURE SCOPE	25

SOURCE CODE

GITHUB AND DEMO LINK

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
3.1	EMPATHY MAP	3
3.4	PROBLEM SOLUTION FIT	6
5.1	DATA FLOW DIAGRAM FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS	11
5.2	SOLUTION ARCHITECTURE FOR INVENTORY MANAGEMENT FOR RETAILERS	12
7.1	IBM CLOUD PLATFORM	16
8.1	SIGN UP PAGE FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS	19
8.2	LOGIN PAGE FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS	19
8.3	DASHBOARD PAGE FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS	20
8.4	ORDERS PAGE FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS	20
8.5	PROFILE PAGE FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS	21
9.1	PERFORMANCE METRICES	22

LIST OF TABLES

TABLE NO	TITLE	PAGE NO
3.2	IDEATION AND BRAINSTORMING	4
3.3	PROPOSED SOLUTION	5
4.1	FUNCTIONAL REQUIREMENTS FOR THE CLOUD BASED INVENTORY MANAGEMENT SYSTEM	8
4.2	NON-FUNCTIONAL REQUIREMENTS OF CLOUD-BASED INVENTORY MANAGEMENT SYSTEM	9
6.1	SPRINT PLANNING AND ESTIMATION FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS	13
6.2	SPRINT PLANNING DONE FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS	15

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The problem faced by the retailers is that they do not have any system to record and keep their inventory data. It is difficult for the owner to record the inventory data quickly and safely because they only keep it in the logbook and not properly organized.

Inventory management facilitates the smooth functioning of your business and enhances sales, promotes cost-effectiveness, and improves customer experience. Listed below are some of the reasons why businesses need inventory management:

- Managing Finances
- Tracking Inventory
- Avoiding late deliveries
- Managing time and effort
- Predicting future sales
- Enhancing customer loyalty

1.2 PURPOSE

Retail inventory management is the process of ensuring retailers meet customer demand without running out of stock or carrying excess supply. The objective of the project is to create an application that help retailers to track and manage stocks which results in lower costs and a better understanding of sales pattern. By creating an application, retailers can log in to it and can update inventory details, also users will be able to add new stock by submitting essential details related to the stock. Retailers can also 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.

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING SOLUTION

In recent years, the correct management of inventories has become a fundamental pillar for achieving success in enterprises. Unfortunately, studies suggesting the investment and adoption of advanced inventory management and control systems are not easy to find. In this context, this article aims to analyze and present an extensive literature concerning inventory management, containing multiple definitions and fundamental concepts for the retail sector. A systematic literature review was carried out to determine the main trends and indicators of inventory management in Small and Medium-sized Enterprises (SMEs). This focus specifically on the retail sector. The primary outcomes of this study are the leading inventory management systems and models, the Key Performance Indicators (KPIs) for their correct management, and the benefits and challenges for choosing or adopting an efficient inventory control and management system. Findings indicate that SMEs do not invest resources in sophisticated systems; instead, a simple Enterprise Resource Planning (ERP) system or even programs such as Excel or manual inventories are mainly used.

2.2 PROBLEM STATEMENT DEFINITION

The problem faced by the retailers is that they do not have any system to record and keep their inventory data. It is difficult for the owner to record the inventory data quickly and safely because they only keep it in the logbook and not properly organized.

CHAPTER 3

IDEATION AND PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS

An empathy map is a collaborative visualization used to express clearly what one knows about a particular type of user. It externalizes knowledge about users in order to create a shared understanding of user needs, and aid in decision making.

Empathy maps are split into 4 quadrants (Says, Thinks, Does, and Feels), with the user in the middle. Empathy maps provide a glance into who a user is as a whole. The *Says* quadrant contains what the user says or what he needs. The *Thinks* quadrant captures what the user is thinking throughout the experience. The *Does* quadrant encloses the actions the user takes. The *Feels* quadrant is the user's emotional state.

The empathy map for Inventory management system for retailers is shown in Fig 3.1

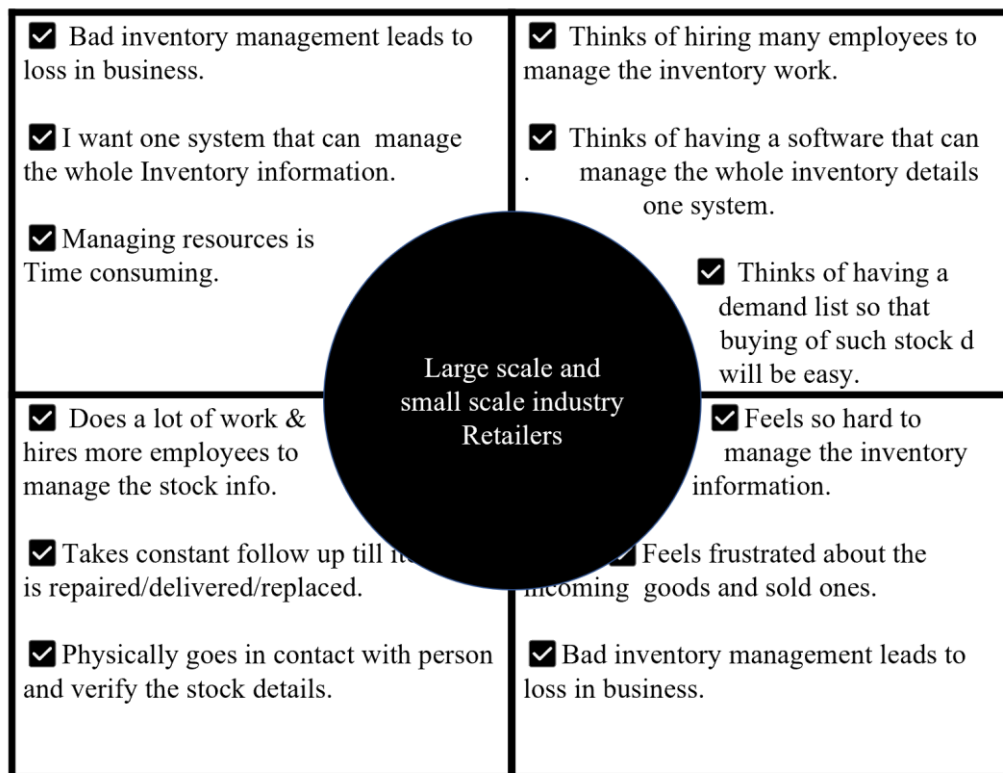


Fig 3.1 Empathy map

3.2 IDEATION AND BRAINSTORMING

Ideation is often closely related to the practice of brainstorming, a specific technique that is utilized to generate new ideas. Brainstorming is usually conducted by getting a group of people together to come up with either general new ideas or ideas for solving a specific problem or dealing with a specific situation. A principal difference between ideation and brainstorming is that ideation is commonly more thought of as being an individual pursuit, while brainstorming is almost always a group activity. Both brainstorming and ideation are processes invented to create new valuable ideas, perspectives, concepts and insights, and both are methods for envisioning new frameworks and systemic problem solving.

The Ideation chart for Inventory management system for retailers is shown in Table 3.2.

IDEA 1	IDEA 2	IDEA 3
While billing, scanning the bar codes detects the product and manages stock according to the quantity sold. Demand based advanced stock pre-order. Providing feedback and Rating of the wholesale agent.	Minimum stock alert at the level of 10 pieces of a product. Demand based stopping of pre ordering (Least sold goods). Providing Key Performance Indicator for analyzing stock. Out of stock E-mail alert.	Providing daily reports on the stock and inventory details for better Business analysis. Tools (such as RFID and barcodes) to maintain correct records across digital and physical databases. Data from different key performance metrics, which take into account several aspects of the inventory influencing the business

Table 3.2 Ideation and Brainstorming

3.3 PROPOSED SOLUTION

The proposed solution for Inventory management system for retailers is shown in table 3.3

S.No.	Parameter	Description
1.	Problem Statement (Problem to besolved)	The problem faced by the retailers is that they do not have any system to record and keep their inventory data. It is difficult for the owner to recordthe inventory data quickly and safely because they only keep it in the logbook and not properly organized.
2.	Idea / Solution description	We aim to design an Inventory Management systemwhich is used to manage the inventory details and aims to save for the future investments. User can track the stocks sold and yet to be sold and can visualize it. The Application will notify the user when a stock is about to complete. Our web application will monitor user's stock by tracking the received SMSs from the user's mobile.
3.	Novelty / Uniqueness	Retailers get notified when the stock is about to getover and intimates the user to buy more stock. Providing Key Performance Indicator for analyzing stock. Demand based advanced stock pre-order.
4.	Social Impact / Customer Satisfaction	Encourages user to track Stock availability and increase profit. It helps to make a better budget that he will have a financial control.
5.	Business Model (Revenue Model)	The low- c o s t requirement for designing this proposed model makes it more reliable and userfriendly.
6.	Scalability of the Solution	With efficient usage of IBM cloud, this proposed model will be able to handle a large number of user data. This makes a huge number of users to easily access and efficiently use it.

Table 3.3 Proposed Solution

3.4 PROBLEM SOLUTION FIT

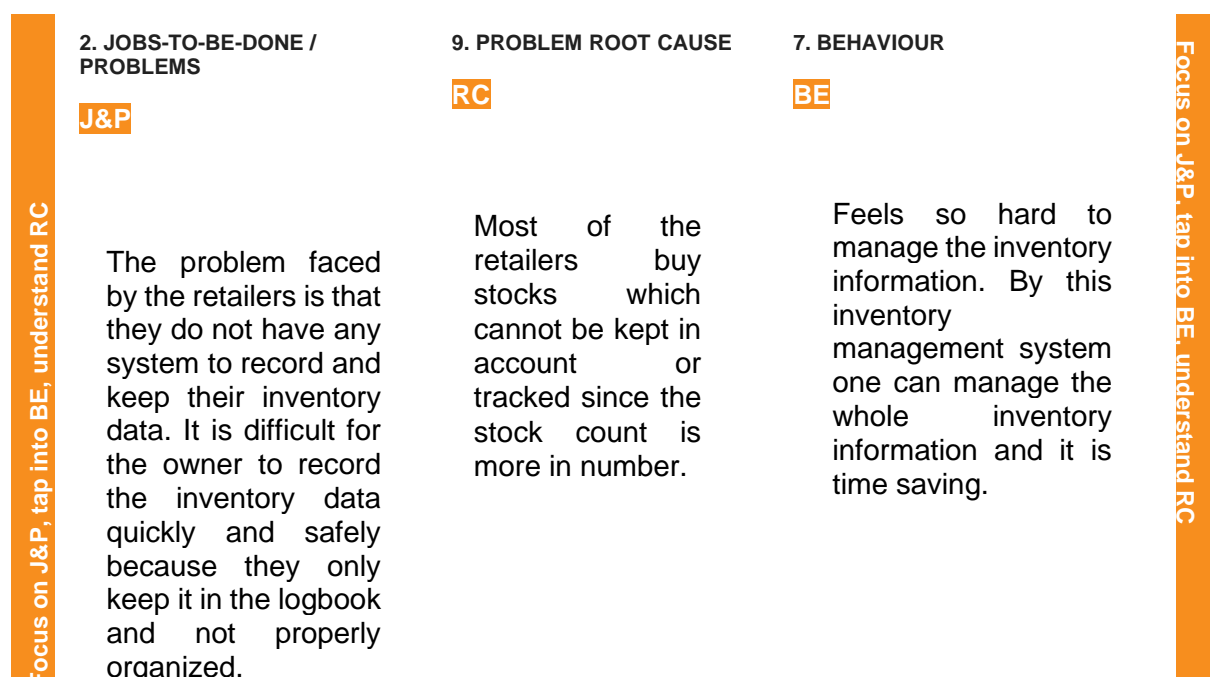
The Problem solution fit simply means that one have found a problem with the customer and that the solution one have realised for it actually solves the customers problem. The problem solution fit is an important step towards the Product-Market Fit. The structure of problem solution fit is given below.

Customer state fit: To make sure one understands the target group, their limitations and their currently available solutions, against which one is going to compete.

Problem-Behavior fit: To help one to identify the most urgent and frequent problems, understand the real reasons behind them and see which behavior supports it.

Communication-Channel fit: To help one to sharpen the communication with strong triggers, emotional messaging and reaching customers via the right channels.

Solution guess: Translate all the validated data one have gathered into a solution that fits the customer state and his/her limitations, solves a real problem and taps into the common behavior of the target group. The below fig 3.4 shows the problem solution fit



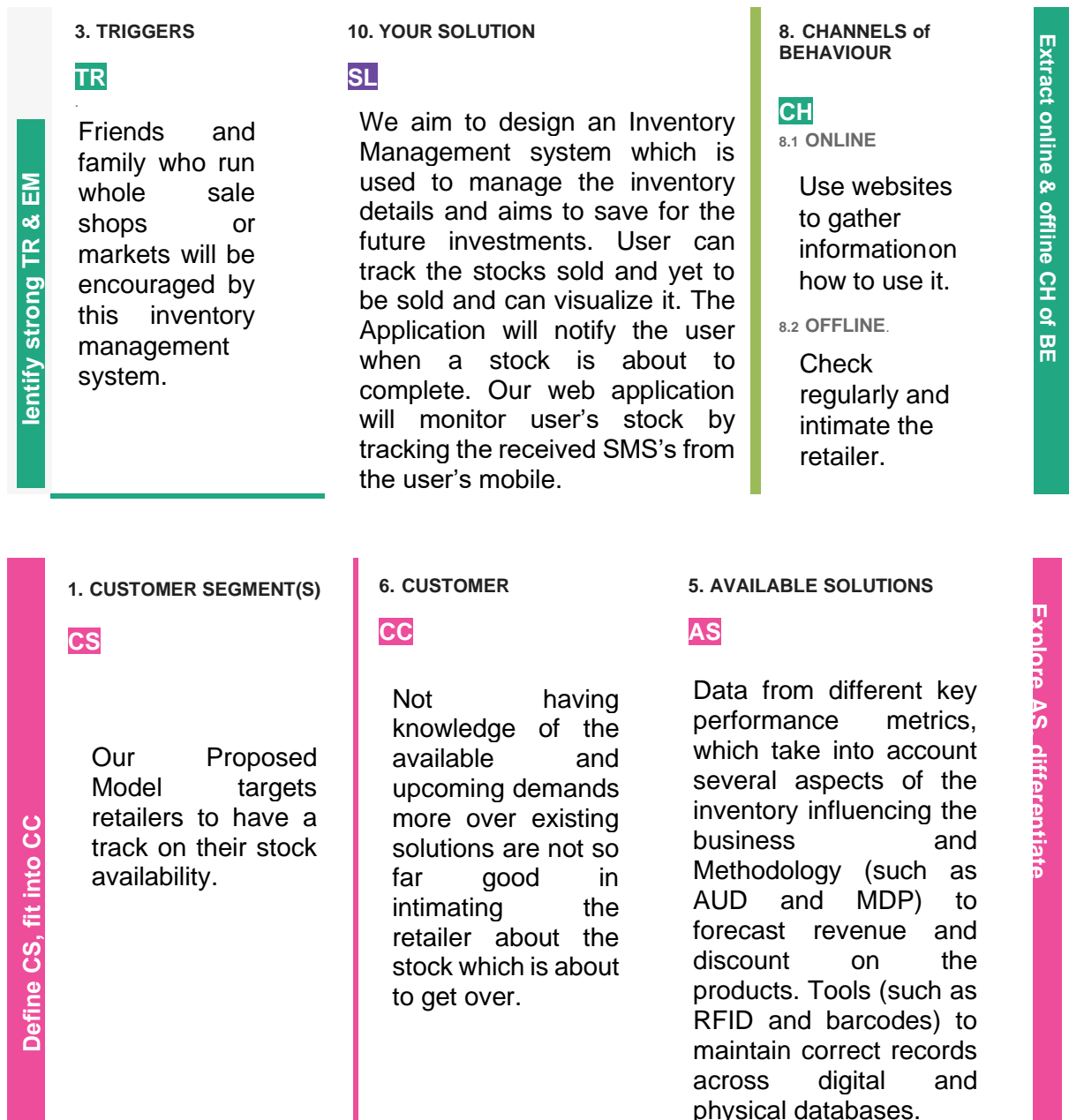


Fig 3.4 Problem Solution fit

CHAPTER 4

REQUIREMENT ANALYSIS

Requirements analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and Non-functional requirements.

4.1 FUNCTIONAL REQUIREMENTS

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements. The below table 4.1 shows the Functional Requirements for the cloud Based Inventory management System

FR No.	Functional Requirement (EPIC)	Sub Requirement (Story/ Sub Task)
FR-1	User Registration	Registration using Business e-mail
FR-2	User Confirmation	Confirmation Via Email Confirmation Via OTP
FR-3	Account Completion	Filling the details required to maintain an inventory
FR-4	Allocating hub manager	Providing restricted access to the hub managers to maintain entire hub's in a single logic using access policy
FR-5	Analysis of demand/ stock moment	Regular communication regarding stock moment to the user. Using sendgrid mail service and also using IBM Watson Ai chat Bot

Table 4.1 Functional Requirements for the cloud Based Inventory management System

4.2 NON-FUNCTIONAL REQUIREMENTS

These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. The below table 4.2 shows the Non-Functional Requirements for the cloud Based Inventory management System

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	It avoids manual entry so mistakes can be avoided and it also gives us regular notification about the stock moment because of that we can be able to maintain inventory in an efficient way.
NFR- 2	Security	Here we are providing two step authentications to provide a better security as well the owner of the account has full access to them accounts he/she can restrict other user this will provide better security because each and every action will be monitored.
NFR- 3	Reliability	The product is going to be developed in microservice architecture so the complexity of the software will be reduced and it will improve the performance. each and every module will be loosely coupled so failure of one module will not lead to failure of entire system.
NFR- 4	Performance	Hence, we are going to develop the product in microservice architecture each and every module is going to have their own environment so traffic in one module will not affect other.

Table 4.2 Non-Functional Requirements of cloud-based Inventory management System

CHAPTER 5

PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled. They can be used to analyze an existing system or model a new one. Like all the best diagrams and charts, a DFD can often visually “say” things that would be hard to explain in words, and they work for both technical and nontechnical audiences, from developer to CEO. That’s why DFDs remain so popular after all these years. While they work well for data flow software and systems, they are less applicable nowadays to visualizing interactive, real-time or database-oriented software or systems.

There are four main elements of a DFD — external entity, process, data store, and data flow.

- **External entity**

An external entity, which are also known as terminators, sources, sinks, or actors, are an outside system or process that sends or receives data to and from the diagrammed system. They’re either the sources or destinations of information, so they’re usually placed on the diagram’s edges. External entity symbols are similar across models except for Unified, which uses a stick-figure drawing instead of a rectangle, circle, or square.

- **Process**

Process is a procedure that manipulates the data and its flow by taking incoming data, changing it, and producing an output with it. A process can do this by performing computations and using logic to sort the data, or change its flow of direction. Processes usually start from the top left of the DFD and finish on the bottom right of the diagram.

- **Data store**

Data stores hold information for later use, like a file of documents that's waiting to be processed. Data inputs flow through a process and then through a data store while data outputs flow out of a data store and then through a process.

- **Data flow**

Data flow is the path the system's information takes from external entities through processes and data stores. With arrows and succinct labels, the DFD can show the direction of the data flow. The below Fig 5.1 shows the Data Flow Diagram for Inventory Management system for retailers

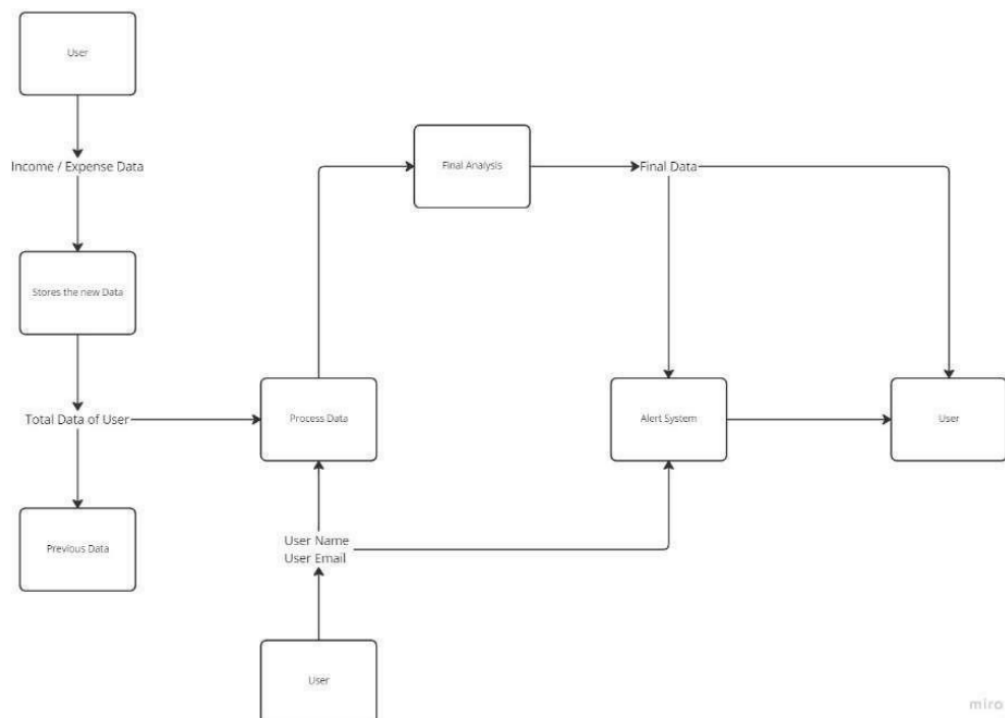


Fig 5.1: Data Flow Diagram for Inventory Management system for retailers

5.2 SOLUTION AND TECHNICAL ARCHITECHTURE

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

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behaviour, and other aspects of the software to project stakeholders.

- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.

The below figure 5.2 shows the Solution architecture of Inventory Management for retailers

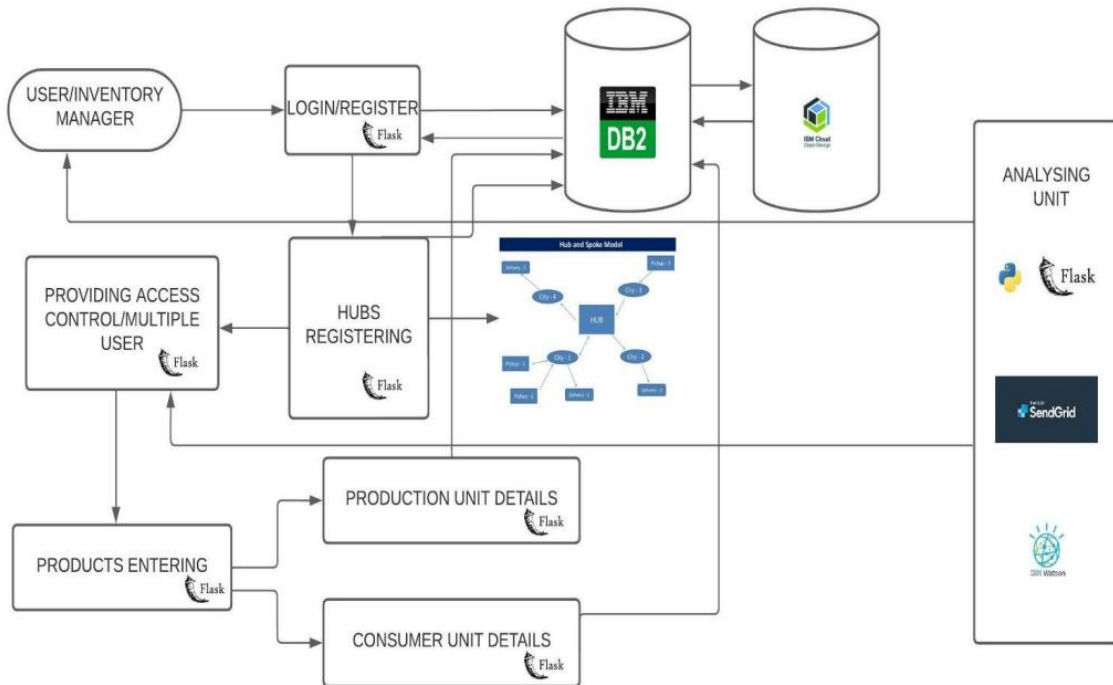


Fig 5.2: Solution architecture of Inventory Management for retailers

CHAPTER 6
PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION

Sprint planning is an event in scrum that kicks off the sprint. The purpose of sprint planning is to define what can be delivered in the sprint and how that work will be achieved. Sprint planning is done in collaboration with the whole scrum team.

The sprint is a set period of time where all the work is done. However, before leap into action it is necessary to set up the sprint. It needs to decide on how long the time box is going to be, the sprint goal, and where it is going to start. The sprint planning session kicks off the sprint by setting the agenda and focus. If done correctly, it also creates an environment where the team is motivated, challenged, and can be successful. The below table 6.1 shows the Sprint Planning and estimation for Inventory Management system for Retailers

SPRINT	FUNCTIONAL REQUIREMENT	USER STORY NUMBER	USER STORY / TASK	STORY POINTS	PRIORITY
Sprint-1	Registration	USN-1	As a user, I can register for the application by using my email & password and confirming my login credentials.	3	High
Sprint-1	Confirmation	USN-3	As a user, I can receive my confirmation email once I have registered for the application.	2	High
Sprint-1	Login	USN-4	As a user, I can log in to the authorized account by entering the registered email and password.	3	Medium

Sprint-2	Dashboard	USN-5	As a user, I can view the products that are available currently.	4	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.	3	Medium
Sprint-3	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.	6	Medium
Sprint-4	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.	4	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.	3	Medium

Table 6.1: Sprint Planning and estimation for Inventory Management system for Retailers

6.2 SPRINT DELIVERY SCHEDULE

The sprint delivery plan is scheduled accordingly as shown in the below table 6.2 which consists of the sprints with respective to their duration, sprint start and end date and the releasing data.

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	11	6 Days	24 Oct 2022	29 Oct 2022	11	29 Oct 2022
Sprint-2	7	6 Days	31 Oct 2022	05 Nov 2022	7	05 Nov 2022
Sprint-3	6	6 Days	07 Nov 2022	12 Nov 2022	6	12 Nov 2022
Sprint-4	7	6 Days	14 Nov 2022	19 Nov 2022	7	19 Nov 2022

Table 6.2: Sprint Planning done for Inventory Management system for Retailers

CHAPTER 7

CODING & SOLUTIONING

7.1 IBM Cloud

The IBM Cloud platform combines platform as a service (PaaS) with infrastructure as a service (IaaS) to provide an integrated experience. The platform scales and supports both small development teams and organizations, and large enterprise businesses. Globally deployed across data centers around the world, the solution you build on IBM Cloud spins up fast and performs reliably in a tested and supported environment you can trust!

IBM Cloud provides solutions that enable higher levels of compliance, security, and management, with proven architecture patterns and methods for rapid delivery for running mission-critical workloads.

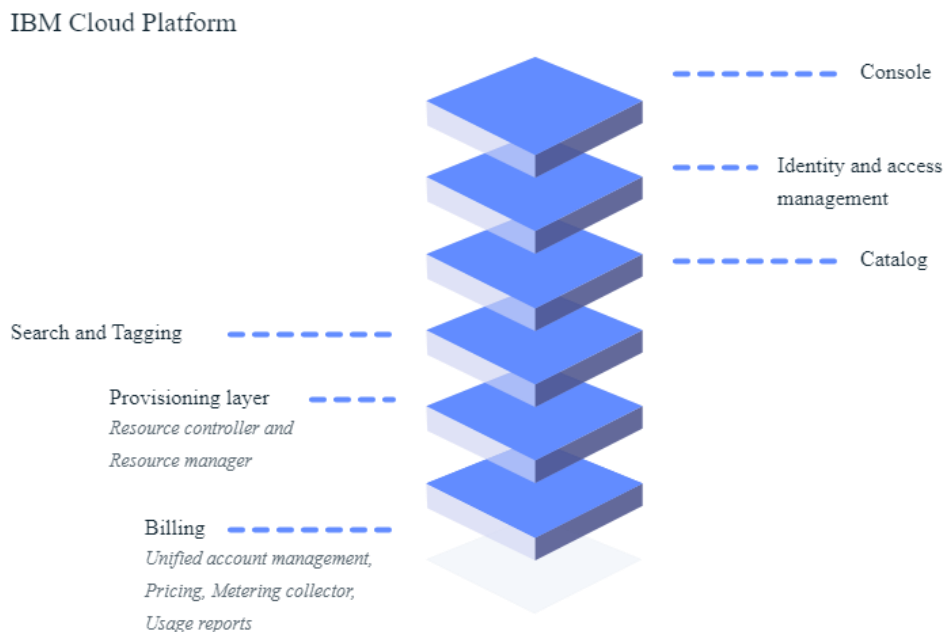


Fig 7.1: IBM cloud platform

7.2 Flask framework

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist

for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

7.3 IBM DB2 Module

Module features allow you to

- Extend schema support by allowing you to group together, in a named set, a collection of related data type definitions, database object definitions and other logic elements including:
 - SQL procedures
 - A module initialization procedure for implicit execution upon module initialization
 - User-defined data type definitions including: distinct type, array type, associative array type, row type, and cursor type
- Define a namespace such that objects defined within the module can refer to other objects defined in the module without providing an explicit qualifier.
- Add object definitions that are private to the module. These objects can only be referenced by other objects within the module.
- Add object definitions that are published. Published objects can be referenced from within the module or from outside of the module.
- Define published prototypes of routines without routine-bodies in modules and later implement the routine-bodies using the routine prototype.
- Initialize the module by executing the module initialization procedure for the module. This procedure can include SQL statements, SQL PL statements, and can be used to set default values for global variables or to open cursors.
- Reference objects defined in the module from within the module and from outside of the module by using the module name as a qualifier (2-part name support) or a combination of the module name and schema name as qualifiers (3-part name support).
- Drop objects defined within the module.
- Drop the module.
- Manage who can reference objects in a module by allowing you to grant and revoke the EXECUTE privilege for the module.

7.4 Docker CLI

The Docker client enables users to interact with Docker. The Docker client can reside on the same host as the daemon or connect to a daemon on a remote host. A docker client can communicate with more than one daemon. The Docker client provides a command line interface (CLI) that allows you to issue build, run, and stop application commands to a Docker daemon. The main purpose of the Docker Client is to provide a means to direct the pull of images from a registry and to have it run on a Docker host. Common commands issued by a client are:

- docker build
- docker pull
- docker run

7.5 IBM cloud CLI

IBM Cloud CLI provides full management of your IBM Cloud account via command line. Some installation steps described along this guide may need the IBM Cloud Command Line Interface (CLI) available to be performed.

7.6 SendGrid API

SendGrid's web API allows users to pull information about their email program without having to actually log on to SendGrid.com. Users can pull lists, statistics, and even email reports. In addition to this, users can send email via the web API without using traditional SMTP.

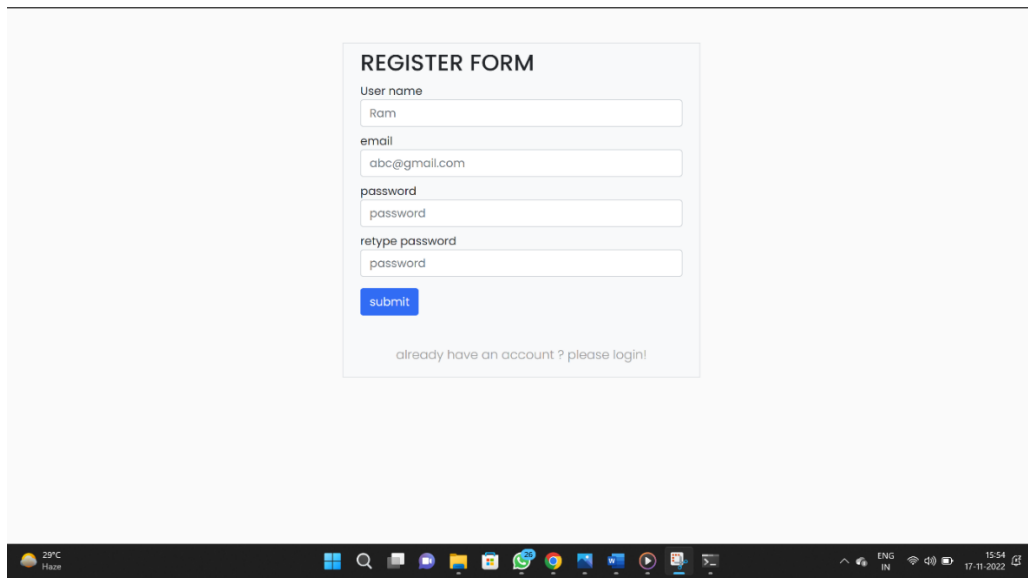
7.7 Kubernetes

Kubernetes is an open-source Container Management tool which automates container deployment, container scaling, and descaling and container load balancing (also called as container orchestration tool). It is written in Golang and has a huge community because it was first developed by Google and later donated to CNCF (Cloud Native Computing Foundation). Kubernetes can group 'n' number of containers into one logical unit for managing and deploying them easily. It works brilliantly with all cloud vendors i.e. public, hybrid and on-premises. Kubernetes is an open-source platform that manages Docker containers in the form of a cluster. Along with the automated deployment and scaling of containers, it provides healing by automatically restarting failed containers and rescheduling them when their hosts die. This capability improves the application's availability.

CHAPTER 8

TESTING AND RESULTS

This Chapter presents the results the results of Inventory Management System for Retailers. The below Fig 8.1 shows the Sign-Up page for Inventory Management System for Retailers



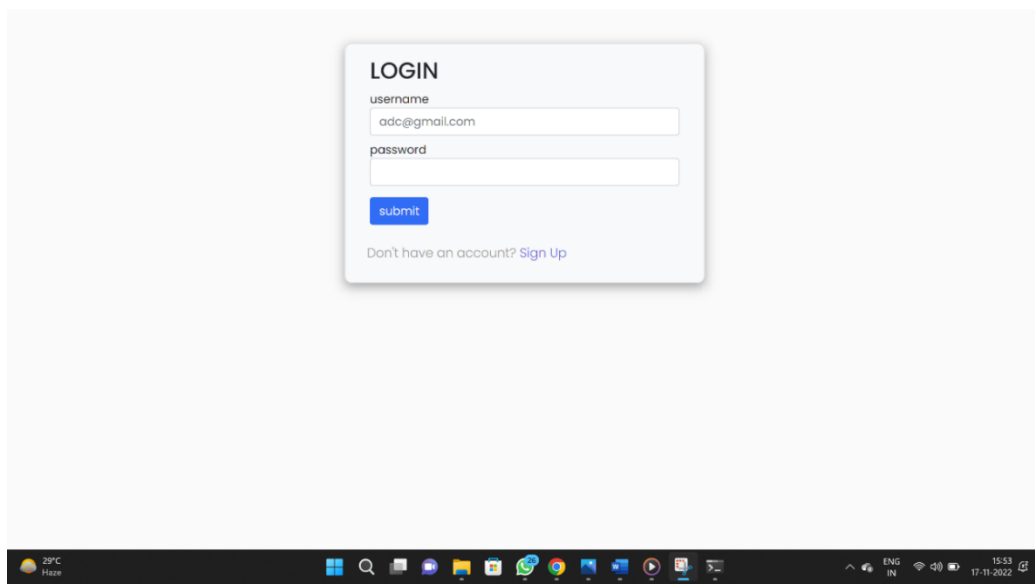
The screenshot displays a web browser window with a 'REGISTER FORM' centered on the page. The form contains the following fields and elements:

- REGISTER FORM** (Section Header)
- User name**: Input field containing 'Ram'.
- email**: Input field containing 'abc@gmail.com'.
- password**: Input field containing 'password'.
- retype password**: Input field containing 'password'.
- submit**: A blue button.
- Below the form, the text 'already have an account ? please login!' is displayed.

The browser's taskbar at the bottom shows the system clock as 15:54 on 17-11-2022, with a language setting of 'ENG IN'.

Fig 8.1: Sign Up page for Inventory Management System for Retailers

The below Fig 8.2 shows the Login Page for Inventory Management System for Retailers



The screenshot displays a web browser window with a 'LOGIN' form centered on the page. The form contains the following fields and elements:

- LOGIN** (Section Header)
- username**: Input field containing 'adc@gmail.com'.
- password**: Input field (empty).
- submit**: A blue button.
- Below the form, the text 'Don't have an account? [Sign Up](#)' is displayed.

The browser's taskbar at the bottom shows the system clock as 15:53 on 17-11-2022, with a language setting of 'ENG IN'.

Fig 8.2: Login Page for Inventory Management System for Retailers

The below Fig 8.3 shows the Dashboard page for Inventory Management system for Retailers

Inventory

Dashboard

Orders

Suppliers

Profile

Logout

Dashboard

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,

ID	NAME	QUANTITY	PRICE_PER_QUANTITY	TOTAL_PRICE
1	Book	100	10.0	1000.0
2	Laptop	100	10.0	1000.0
3	Table	100	20.0	2000.0
10	dkds	100	10.0	1000.0
5	Pencil	300	5.0	1500.0

Update Stock

Enter Item

Choose a field

Enter Value

Update

Add New Stock

Enter the item

Enter quantity

Enter price

Add Stock

Remove stocks

Enter the item

Remove

25°C Haze

ENG IN 15:58 17-11-2022

Fig 8.3: Dashboard page for Inventory Management system for Retailers

The below Fig 8.4 shows the Orders Page for Inventory Management system for Retailers

Inventory

Dashboard

Orders

Suppliers

Profile

Logout

Orders

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,

ID	STOCKS_ID	QUANTITY	DATE	DELIVERY_DATE	PRICE
1	10	10	2022-11-13	2022-11-20	1000.0
8	10	555	2022-11-13	2022-11-20	275.0
3	3	10	2022-11-13	2022-11-20	1000.0

Create Order

Enter Stock ID

Enter Quantity

Create

Update Order

Enter Order ID

Choose a field

Enter Value

Update

Cancel Order

Enter Order ID

Cancel

25°C Haze

ENG IN 15:58 17-11-2022

Fig 8.4: Orders Page for Inventory Management system for Retailers

The below Fig 8.5 shows the Profile Page for Inventory Management system for Retailers

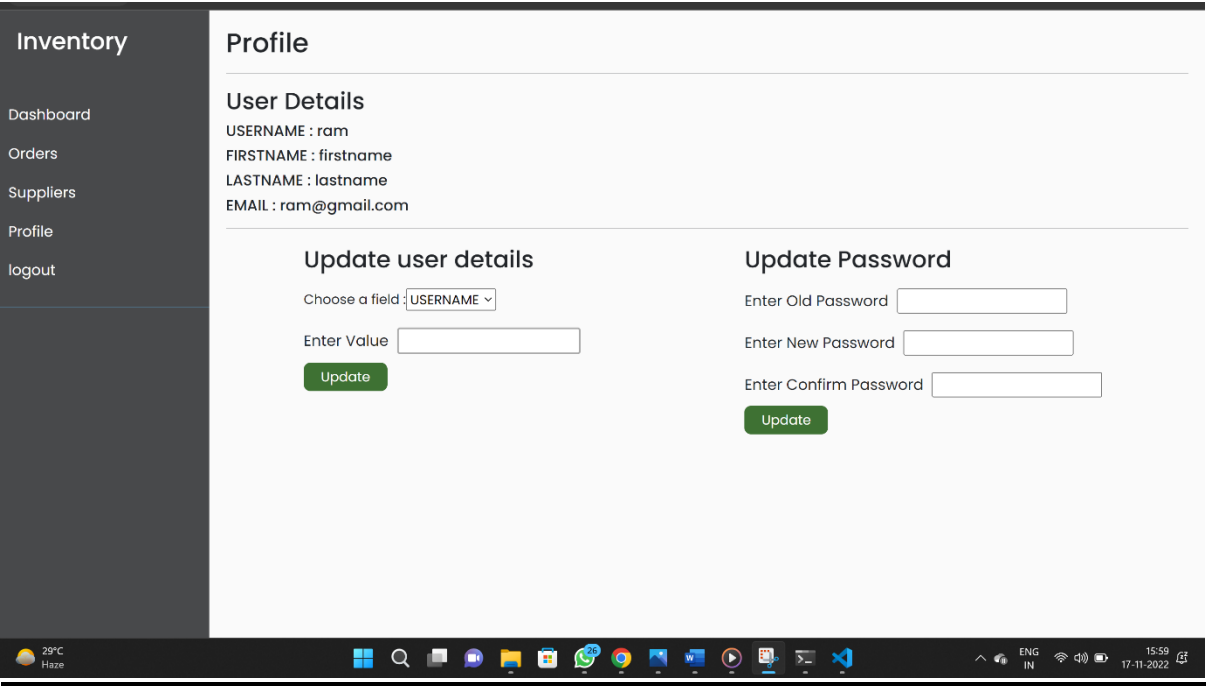


Fig 8.5: Profile Page for Inventory Management system for Retailers

CHAPTER 9

PERFORMANCE RESULTS

9.1 PERFORMANCE METRICS

Fig 9.1 shows the performance metrics of the flask application using Google Developer Tools.

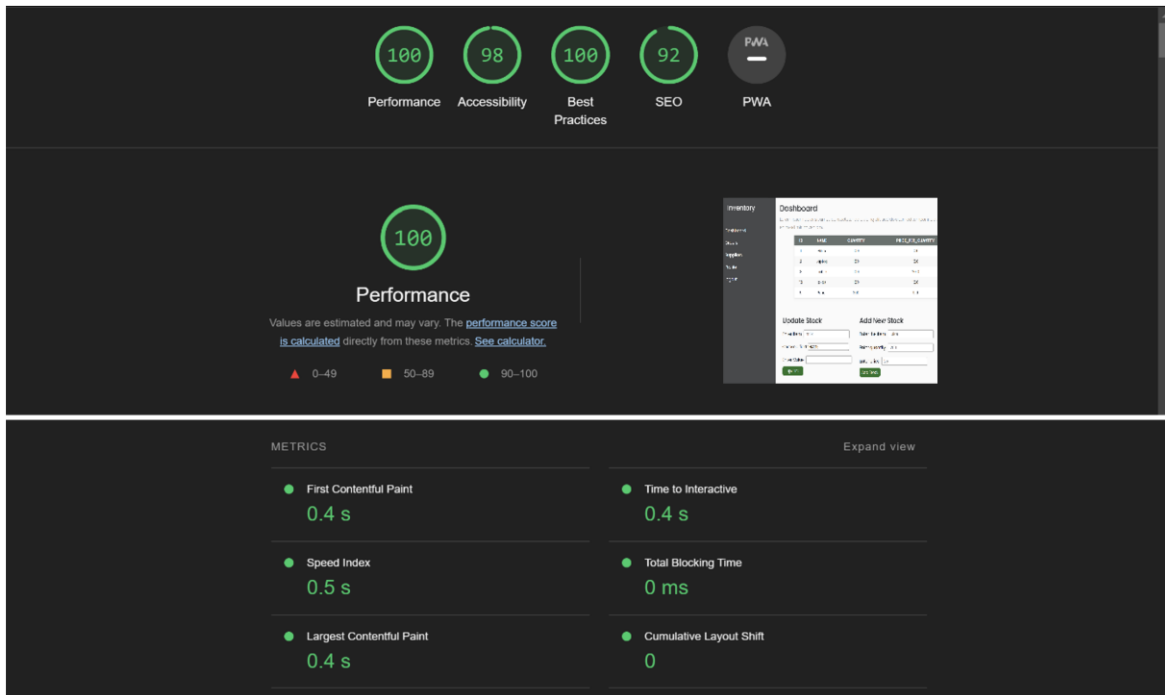


Fig 9.1: Performance metrics

CHAPTER 10

ADVANTAGES AND DISADVANTAGES

10.1 ADVANTAGES

The inventory management system for retailers is a web-based application. A dashboard is given to retailers where they can able to update, manage and create a product and assign the number of quantities for the products. Whenever a product goes out of stock the inventory management system for retailers alerts the retailers through email stating product goes out of stocks. Thus, inventory management system helps retailers to increase the profit and reduce the risk of holding large quantity of a particular stock. The inventory management system helps the retailers to efficiently utilizes the inventory area i.e., where the store all the products

10.2 DISADVANTAGES

The inventory management system for retailers helps them in many ways, but it needs a manual way of updating the quantity of the product. The retailers need to create and update the quantity of stock in the web-based application. The inventory management system necessitates manual updating of stock quantities, which adds to their workload.

CHAPTER 11

CONCLUSION

A web-based application is created to manage inventory stocks. Retailers can able to update, create, and manage products in this web application.

The inventory management system gives a mail alert, if a product goes out of stock, stating that a product has gone out of stock.

This allows retailers to increase their profit, reduce the risk of having too much stock, and make better use of their inventory space.

CHAPTER 12

FUTURE SCOPE

The future scope of the web-based inventory management application for retailers includes building a charting system into the application that helps them know the sales performance of a product for different time periods like a day, a week, or a year.

Automation of updating the quantity of stock for each product using technologies like barcodes, QR codes, etc.

Analyze each product's sales performance in relation to key performance indicators to determine when to offer discounts and offers on products with good and bad stock performance.

SOURCE CODE

App.py

```
from flask import Flask, render_template, url_for, request, redirect, session,
make_response
```

```
import sqlite3 as sql
```

```
from functools import wraps
```

```
import re
```

```
import ibm_db
```

```
import os
```

```
from sendgrid import SendGridAPIClient
```

```
from sendgrid.helpers.mail import Mail
```

```
from datetime import datetime, timedelta
```

```
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=815fa4db-dc03-4c70-869a-
a9cc13f33084.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=30367;SECURI
TY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=gkx49901;PWD=kvWC
sySl7vApfsy2", "", "")
```

```
app = Flask(__name__)
```

```
app.secret_key = 'jackiechan'
```

```
def rewrite(url):
```

```
    view_func, view_args = app.create_url_adapter(request).match(url)
```

```
    return app.view_functions[view_func](**view_args)
```

```
def login_required(f):
```

```
    @wraps(f)
```



```

def decorated_function(*args, **kwargs):
    if "id" not in session:
        return redirect(url_for('login'))
    return f(*args, **kwargs)
return decorated_function

@app.route('/')
def root():
    return render_template('login.html')

@app.route('/user/<id>')
@login_required
def user_info(id):
    with sql.connect('inventorymanagement.db') as con:
        con.row_factory = sql.Row
        cur = con.cursor()
        cur.execute(f'SELECT * FROM users WHERE email="{id}"')
        user = cur.fetchall()
    return render_template("user_info.html", user=user[0])

@app.route('/login', methods=['GET', 'POST'])
def login():
    global userid
    msg = "

```

```

if request.method == 'POST':

    un = request.form['username']

    pd = request.form['password_1']

    print(un, pd)

    sql = "SELECT * FROM users WHERE email=? AND password=?"

    stmt = ibm_db.prepare(conn, sql)

    ibm_db.bind_param(stmt, 1, un)

    ibm_db.bind_param(stmt, 2, pd)

    ibm_db.execute(stmt)

    account = ibm_db.fetch_assoc(stmt)

    print(account)

    if account:

        session['loggedin'] = True

        session['id'] = account['EMAIL']

        userid = account['EMAIL']

        session['username'] = account['USERNAME']

        msg = 'Logged in successfully !'

        return rewrite('/dashboard')

    else:

        msg = 'Incorrect username / password !'

return render_template('login.html', msg=msg)

```

```

@app.route('/signup', methods=['POST', 'GET'])

```

```

def signup():

```

```

    mg = "

```

```

    if request.method == "POST":

```

```

username = request.form['username']

email = request.form['email']

pw = request.form['password']

sql = 'SELECT * FROM users WHERE email =?'

stmt = ibm_db.prepare(conn, sql)

ibm_db.bind_param(stmt, 1, email)

ibm_db.execute(stmt)

acnt = ibm_db.fetch_assoc(stmt)

print(acnt)


if acnt:

    mg = 'Account already exists!!'


elif not re.match(r'^[^\@]+\@[^\@]+\.[^\@]+', email):

    mg = 'Please enter the avalid email address'

elif not re.match(r'[A-Za-z0-9]+', username):

    ms = 'name must contain only character and number'

else:

    insert_sql = 'INSERT INTO users
(USERNAME,FIRSTNAME,LASTNAME,EMAIL,PASSWORD) VALUES (?, ?, ?, ?, ?)'

    pstmt = ibm_db.prepare(conn, insert_sql)

    ibm_db.bind_param(pstmt, 1, username)

    ibm_db.bind_param(pstmt, 2, "firstname")

    ibm_db.bind_param(pstmt, 3, "lastname")

    # ibm_db.bind_param(pstmt,4,"123456789")

    ibm_db.bind_param(pstmt, 4, email)

    ibm_db.bind_param(pstmt, 5, pw)

    print(pstmt)

```

```

    ibm_db.execute(pstmt)

    mg = 'You have successfully registered click login!'

    message = Mail(

        from_email=os.environ.get('MAIL_DEFAULT_SENDER'),

        to_emails=email,

        subject='New SignUp',

        html_content='<p>Hello, Your Registration was successfull. <br><br> Thank
you for choosing us.</p>')

    sg = SendGridAPIClient(

        api_key=os.environ.get('SENDGRID_API_KEY'))

    response = sg.send(message)

    print(response.status_code, response.body)

    return render_template("login.html", meg=mg)

elif request.method == 'POST':

    msg = "fill out the form first!"

    return render_template("signup.html", meg=mg)

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

    sql = "SELECT * FROM stocks"

    stmt = ibm_db.exec_immediate(conn, sql)

    dictionary = ibm_db.fetch_assoc(stmt)

    stocks = []

```

```

headings = [*dictionary]

while dictionary != False:

    stocks.append(dictionary)

    # print(f"The ID is : ", dictionary["NAME"])

    # print(f"The name is : ", dictionary["QUANTITY"])

    dictionary = ibm_db.fetch_assoc(stmt)


return render_template("dashboard.html", headings=headings, data=stocks)


@app.route('/addstocks', methods=['POST'])
@login_required
def addStocks():

    if request.method == "POST":

        print(request.form['item'])

        try:

            item = request.form['item']

            quantity = request.form['quantity']

            price = request.form['price']

            total = int(price) * int(quantity)

            insert_sql = 'INSERT INTO stocks
(NAME,QUANTITY,PRICE_PER_QUANTITY,TOTAL_PRICE) VALUES (?,?=?,?)'

            pstmt = ibm_db.prepare(conn, insert_sql)

            ibm_db.bind_param(pstmt, 1, item)

            ibm_db.bind_param(pstmt, 2, quantity)

            ibm_db.bind_param(pstmt, 3, price)

            ibm_db.bind_param(pstmt, 4, total)

            ibm_db.execute(pstmt)

```

```
except Exception as e:
```

```
    msg = e
```

```
finally:
```

```
    # print(msg)
```

```
    return redirect(url_for('dashBoard'))
```

```
@app.route('/updatestocks', methods=['POST'])
```

```
@login_required
```

```
def UpdateStocks():
```

```
    if request.method == "POST":
```

```
        try:
```

```
            item = request.form['item']
```

```
            print("hello")
```

```
            field = request.form['input-field']
```

```
            value = request.form['input-value']
```

```
            print(item, field, value)
```

```
            insert_sql = 'UPDATE stocks SET ' + field + "= ?" + " WHERE NAME=?"
```

```
            print(insert_sql)
```

```
            pstmt = ibm_db.prepare(conn, insert_sql)
```

```
            ibm_db.bind_param(pstmt, 1, value)
```

```
            ibm_db.bind_param(pstmt, 2, item)
```

```
            ibm_db.execute(pstmt)
```

```
            if field == 'PRICE_PER_QUANTITY' or field == 'QUANTITY':
```

```
                insert_sql = 'SELECT * FROM stocks WHERE NAME= ?'
```

```

        pstmt = ibm_db.prepare(conn, insert_sql)

        ibm_db.bind_param(pstmt, 1, item)

        ibm_db.execute(pstmt)

        dictionary = ibm_db.fetch_assoc(pstmt)

        print(dictionary)

        total = dictionary['QUANTITY'] * dictionary['PRICE_PER_QUANTITY']

        insert_sql = 'UPDATE stocks SET TOTAL_PRICE=? WHERE NAME=?'

        pstmt = ibm_db.prepare(conn, insert_sql)

        ibm_db.bind_param(pstmt, 1, total)

        ibm_db.bind_param(pstmt, 2, item)

        ibm_db.execute(pstmt)

    except Exception as e:

        msg = e

    finally:

        # print(msg)

        return redirect(url_for('dashBoard'))

```

```

@app.route('/deletestocks', methods=['POST'])

```

```

@login_required

```

```

def deleteStocks():

```

```

    if request.method == "POST":

```

```

        print(request.form['item'])

```

```

    try:

```

```

        item = request.form['item']

```

```

        insert_sql = 'DELETE FROM stocks WHERE NAME=?'

```

```

        pstmt = ibm_db.prepare(conn, insert_sql)

```

```

        ibm_db.bind_param(pstmt, 1, item)

        ibm_db.execute(pstmt)

    except Exception as e:

        msg = e

    finally:

        # print(msg)

        return redirect(url_for('dashBoard'))


@app.route('/update-user', methods=['POST', 'GET'])
@login_required
def updateUser():

    if request.method == "POST":

        try:

            email = session['id']

            field = request.form['input-field']

            value = request.form['input-value']

            insert_sql = 'UPDATE users SET ' + field + '= ? WHERE EMAIL=?'

            pstmt = ibm_db.prepare(conn, insert_sql)

            ibm_db.bind_param(pstmt, 1, value)

            ibm_db.bind_param(pstmt, 2, email)

            ibm_db.execute(pstmt)

        except Exception as e:

            msg = e

    finally:

        # print(msg)

```



```

return redirect(url_for('profile'))

@app.route('/update-password', methods=['POST', 'GET'])
@login_required
def updatePassword():
    if request.method == "POST":
        try:
            email = session['id']
            password = request.form['prev-password']
            curPassword = request.form['cur-password']
            confirmPassword = request.form['confirm-password']
            insert_sql = 'SELECT * FROM users WHERE EMAIL=? AND PASSWORD=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, email)
            ibm_db.bind_param(pstmt, 2, password)
            ibm_db.execute(pstmt)
            dictionary = ibm_db.fetch_assoc(pstmt)
            print(dictionary)
            if curPassword == confirmPassword:
                insert_sql = 'UPDATE users SET PASSWORD=? WHERE EMAIL=?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, confirmPassword)
                ibm_db.bind_param(pstmt, 2, email)
                ibm_db.execute(pstmt)
        except Exception as e:
            msg = e
    finally:

```

```

        # print(msg)

        return render_template('result.html')


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

    query = "SELECT * FROM orders"

    stmt = ibm_db.exec_immediate(conn, query)

    dictionary = ibm_db.fetch_assoc(stmt)

    orders = []

    headings = [*dictionary]

    while dictionary != False:

        orders.append(dictionary)

        dictionary = ibm_db.fetch_assoc(stmt)

    return render_template("orders.html", headings=headings, data=orders)


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

    if request.method == "POST":

        try:

            stock_id = request.form['stock_id']

            query = 'SELECT PRICE_PER_QUANTITY FROM stocks WHERE ID= ?'

            stmt = ibm_db.prepare(conn, query)

            ibm_db.bind_param(stmt, 1, stock_id)

            ibm_db.execute(stmt)

```

```

dictionary = ibm_db.fetch_assoc(stmt)

if dictionary:

    quantity = request.form['quantity']

    date = str(datetime.now().year) + "-" + str(
        datetime.now().month) + "-" + str(datetime.now().day)

    delivery = datetime.now() + timedelta(days=7)

    delivery_date = str(delivery.year) + "-" + str(
        delivery.month) + "-" + str(delivery.day)

    price = float(quantity) * \
        float(dictionary['PRICE_PER_QUANTITY'])

    query = 'INSERT INTO orders
(STOCKS_ID,QUANTITY,DATE,DELIVERY_DATE,PRICE) VALUES (?, ?, ?, ?, ?)'

    pstmt = ibm_db.prepare(conn, query)

    ibm_db.bind_param(pstmt, 1, stock_id)

    ibm_db.bind_param(pstmt, 2, quantity)

    ibm_db.bind_param(pstmt, 3, date)

    ibm_db.bind_param(pstmt, 4, delivery_date)

    ibm_db.bind_param(pstmt, 5, price)

    ibm_db.execute(pstmt)

except Exception as e:

    print(e)

finally:

    return redirect(url_for('orders'))

@app.route('/updateOrder', methods=['POST'])
@login_required

```

```

def updateOrder():
    if request.method == "POST":
        try:
            item = request.form['item']
            field = request.form['input-field']
            value = request.form['input-value']
            query = 'UPDATE orders SET ' + field + "= ?" + " WHERE ID=?"
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

    finally:
        return redirect(url_for('orders'))

```

```

@app.route('/cancelOrder', methods=['POST'])
@login_required
def cancelOrder():
    if request.method == "POST":
        try:
            order_id = request.form['order_id']
            query = 'DELETE FROM orders WHERE ID=?'
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, order_id)
            ibm_db.execute(pstmt)

```

```

except Exception as e:

    print(e)


finally:

    return redirect(url_for('orders'))


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

    sql = "SELECT * FROM suppliers"

    stmt = ibm_db.exec_immediate(conn, sql)

    dictionary = ibm_db.fetch_assoc(stmt)

    suppliers = []

    orders_assigned = []

    headings = [*dictionary]

    while dictionary != False:

        suppliers.append(dictionary)

        orders_assigned.append(dictionary['ORDER_ID'])

        dictionary = ibm_db.fetch_assoc(stmt)


# get order ids from orders table and identify unassigned order ids

    sql = "SELECT ID FROM orders"

    stmt = ibm_db.exec_immediate(conn, sql)

    dictionary = ibm_db.fetch_assoc(stmt)

    order_ids = []

    while dictionary != False:

        order_ids.append(dictionary['ID'])

```

```

dictionary = ibm_db.fetch_assoc(stmt)

unassigned_order_ids = set(order_ids) - set(orders_assigned)

return
render_template("suppliers.html", headings=headings, data=suppliers, order_ids=unassigned_order_ids)

@app.route('/updatesupplier', methods=['POST'])
@login_required
def UpdateSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            field = request.form['input-field']
            value = request.form['input-value']
            print(item, field, value)
            insert_sql = 'UPDATE suppliers SET ' + field + "= ?" + " WHERE NAME=?"
            print(insert_sql)
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

    finally:
        return redirect(url_for('suppliers'))

```

```

@app.route('/addsupplier', methods=['POST'])
@login_required
def addSupplier():
    if request.method == "POST":
        try:
            name = request.form['name']
            order_id = request.form.get('order-id-select')
            print(order_id)
            print("Hello world")
            location = request.form['location']

            insert_sql = 'INSERT INTO suppliers (NAME,ORDER_ID,LOCATION)
VALUES (?, ?,?)'

            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, name)
            ibm_db.bind_param(pstmt, 2, order_id)
            ibm_db.bind_param(pstmt, 3, location)
            ibm_db.execute(pstmt)

        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))

@app.route('/deletesupplier', methods=['POST'])
@login_required
def deleteSupplier():
    if request.method == "POST":

```

```

try:

    item = request.form['name']

    insert_sql = 'DELETE FROM suppliers WHERE NAME=?'

    pstmt = ibm_db.prepare(conn, insert_sql)

    ibm_db.bind_param(pstmt, 1, item)

    ibm_db.execute(pstmt)

except Exception as e:

    msg = e


finally:

    return redirect(url_for('suppliers'))

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

    if request.method == "GET":

        try:

            email = session['id']

            insert_sql = 'SELECT * FROM users WHERE EMAIL=?'

            pstmt = ibm_db.prepare(conn, insert_sql)

            ibm_db.bind_param(pstmt, 1, email)

            ibm_db.execute(pstmt)

            dictionary = ibm_db.fetch_assoc(pstmt)

            print(dictionary)

        except Exception as e:

            msg = e

    finally:

        # print(msg)

        return render_template("profile.html", data=dictionary)

```



```
@app.route('/logout', methods=['GET'])  
  
@login_required  
def logout():  
    print(request)  
  
    resp = make_response(render_template("login.html"))  
  
    session.clear()  
  
    return resp  
  
if __name__ == '__main__':  
    app.run(debug=True)
```

GitHub: <https://github.com/IBM-EPBL/IBM-Project-8994-1658941244>

Demo link:

https://drive.google.com/drive/u/1/folders/19qeDVVXsatUFL9tAe_VAHlxqma27h7tt

REFERENCES

1. Y. Fan, 2010, "Development of inventory management system," 2nd IEEE International Conference on Information Management and Engineering, 2010, pp. 207-210, Doi: 10.1109/ICIME.2010.5478077.
2. A. Milella, A. Petitti, R. Marani, G. Cicirelli and T. D’orazio, "Towards Intelligent Retail: Automated on-Shelf Availability Estimation Using a Depth Camera," in IEEE Access, vol. 8, pp. 19353-19363, 2020, doi: 10.1109/ACCESS.2020.2968175.
3. Inventory management for retail companies, “A literature review and current trends”, March 2021, DOI:10.1109/ICI2ST51859.2021.00018, Conference: 2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)