SMART FASHION RECOMMENDER APPLICATION
IBM – Documentation

Nalaiyathiran Project Based Learning
On
Professional Readiness For Innovation,
Employability And Entrepreneurship

TeamId:PNT2022TMID39959
PROJECT REPORT

JAYAKAMALESH K-511319205009
DINESHV-511319205005
PURUSHOTHAMAN C-511319205020
SARAVANAN L-511319205025

BACHELOR OF TECHNOLOGY
IN
INFORMATION TECHNOLOGY
KINGSTON ENGINEERING COLLEGE
VELLORE

faculty mentor : Industry
mentor:
Bhuvaneshwari G Krishna
Chaitanya
Assistant Professor,
Department Of It

1. INTRODUCTION

1.1 Project Overview

This web application to recommend a dress and its categories using the chatbot. It is make the shopping easier for the customer to access and to buy the dresses based on their requirements. The chatbot will recommend the dress category based on their kind of interest. Nowadays people are buying clothes in application so this will be more useful for the user to buy the dresses in our webapplication and it will have more categories of dresses with based on their needs. We propose an chatbot system which operates as a personal assistant to a fashion recommended system.

1.2 Purpose

Main purpose of the project is to make shopping easier and give a clear idea and suggestion to the user using the chatbot. Recommend dresses based on the user request using chatbot. Chatbot cotains of different categories for the user.

2. LITERATURE SURVEY

2.1 Existing Problem

In an existing platform the structure of an application is not clearly defined. In some of the applications there is an less availability of data and those application cannot recommend the more products based on the requirements of the user. There requires more time to access the data and order the dresses. There requires a huge amount of time to response for the users request. There is an less accuracy and it can suggest the product approximate to the user request. The process of an image processing is not fair and most of the images are not appropriate and it is aproximate to the user needs. It can consists of very less datasets. Some of the application can be made only for men or women in less varieties.

2.2 References

1. Seyed Omid Mohammadi, Ahmad Kalhor, 2021 November. Smart Fashion: A Review of AI Applications in Virtual Try-On & Fashion Synthesis.
2. Qingqing Tu; Le Dong, 2010 July. An Intelligent Personalized Fashion Recommendation System.
3. YASHAR DELDJOO, FATEMEH NAZARY, ARNAU RAMISA† JULIAN MCAULEY, GIOVANNI PELLEGRINI, ALEJANDRO BELLOGIN,TOMMASO DI NOIA, 2022 February. A Review of Modern Fashion Recommender Systems.
4. Maria Th. Kotouza, Sotirios–Filippos Tsarouchis, Alexandros-Charalampos Kyprianidis, Antonios C. Chrysopoulos & Pericles A. Mitkas, 2020 May. Towards Fashion Recommendation: An AI System for Clothing Data Retrieval and Analysis.

5. Kai Xin Thia, 2020 March. Building a Personalized Real-Time Fashion Collection Recommender.
6. SAMIT CHAKRABORTY, Md. Saiful Hoque, Naimur Rahman Jeem, Manik Chandra Biswas, 2021 August. Fashion Recommendation Systems.
7. Shaghayegh Shirkhani, 2021. Image-based fashion recommender systems Considering Deep learning role in computer vision development.
8. Seema Wazarkar, Shruti Patil, Pratik S. Gupta, Kriti Singh, Mukund Khandelwal, C.V. Sri Vaishnavi, Ketan Kotecha, 2022 July. Advanced Fashion Recommendation System for Different Body Types using Deep Learning Models.
9. Aneesh K, P V Rohith Kumar, Sai Uday Nagula, Archana Nagelli. 2022 June. Fashion Recommendation System.
10. M Sridevi, N ManikyaArun and M Sheshikala, Sudarshan E, 2020. Personalized fashion recommender system with image based neural networks.

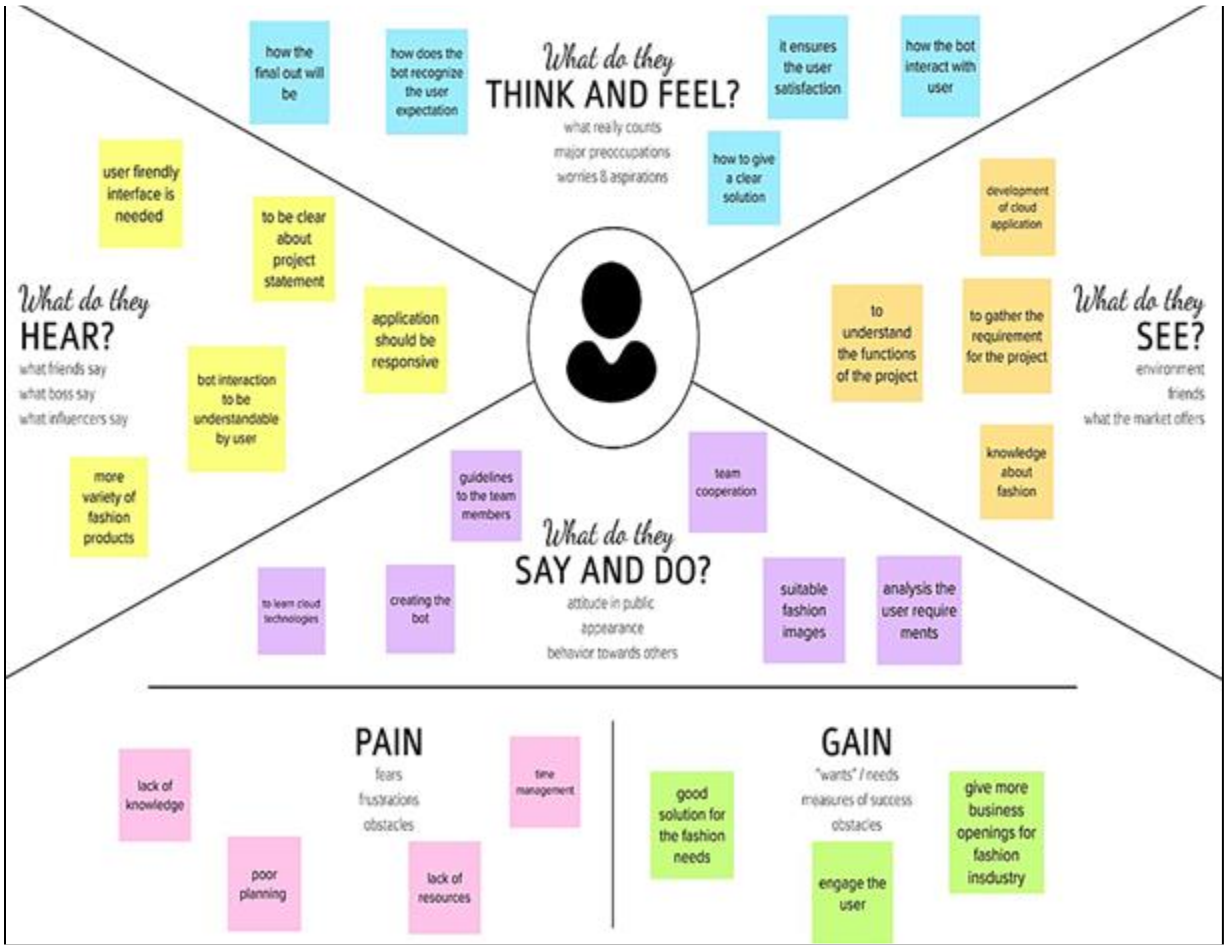
2.3 Problem Statement Definition

In recent years, the huge amount of information and users of the internet service, it is hard to know quickly and accurately what the user wants. This phenomenon leads to an extremely low utilization of information, also known as the information overload problem. Traditionally, keywords are used to retrieve images, but such methods require a lot of annotations on the image data, which will lead to serious problems such as inconsistent, inaccurate, and incomplete descriptions, and a huge amount of work to overcome this problem our application suggest the recommendation based on the user requirements.

3.IDEATION AND PROPOSED SYSTEM

3.1 Empathy Map Canvas

Our project can be developed using the chatbot.It consists of all the requirements for the user and the enough images for their search.Initially the user can login to the web application and can use the chatbot to make the request based on their requirements and it will show the variety of fashion products.Finally the user can select with their interest using the recommendation of the chatbot.



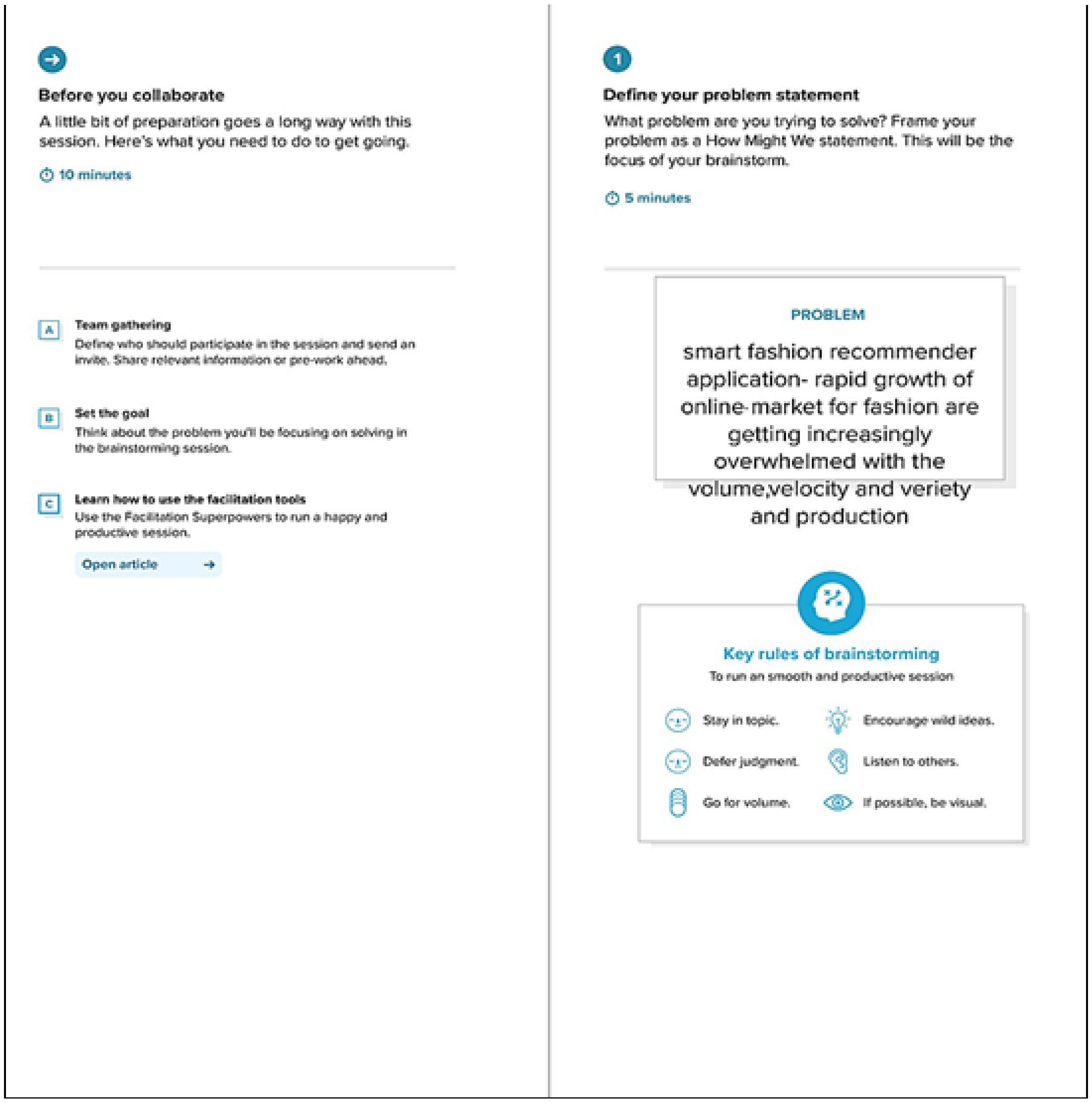
What do they THINK AND FEEL?
what really counts
major preoccupations
worries & aspirations
how the final out will be
how does the bot recognize the user expectation
it ensures the user satisfaction
how the bot interact with user
how to give a clear solution
What do they HEAR?
what friends say
what boss say
what influencers say
user firendly interface is needed
to be clear about project statement
application should be responsive
bot interaction to be understandable by user
more variety of fashion products
What do they SEE?
environment
friends
what the market offers
development of cloud application
to understand the functions of the project
to gather the requirement for the project
knowledge about fashion
What do they SAY AND DO?
attitude in public
appearance
behavior towards others
guidelines to the team members
team cooperation
to learn cloud technologies
creating the bot
suitable fashion images
analysis the user require ments
PAIN
fears
frustrations
obstacles
lack of knowledge
poor planning
lack of resources
time management
GAIN
"wants" / needs
measures of success
obstacles
good solution for the fashion needs
engage the user
give more business openings for fashion insdustry

Fig 3.1.1 Empathy Map

3.2 Ideation & Brainstroming

Our project is a user-friendly interface and we trained the Chatbot in our web application. There is an enough options are provided for the users.This application can be accessed in different platform. All the data are stored in the database and data base can be responsive quickly based on the users request. Chatbot can be build with many options it can recommend the product based on user needs.In this application the ordered details can be easily tracked and it can be stored in the database.

Step 1:Team gathering,Collabration and Select the Problem Statement



Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A Team gathering
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B Set the goal
Think about the problem you'll be focusing on solving in the brainstorming session.

C Learn how to use the facilitation tools
Use the Facilitation Superpowers to run a happy and productive session.

Open article

1 Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

PROBLEM

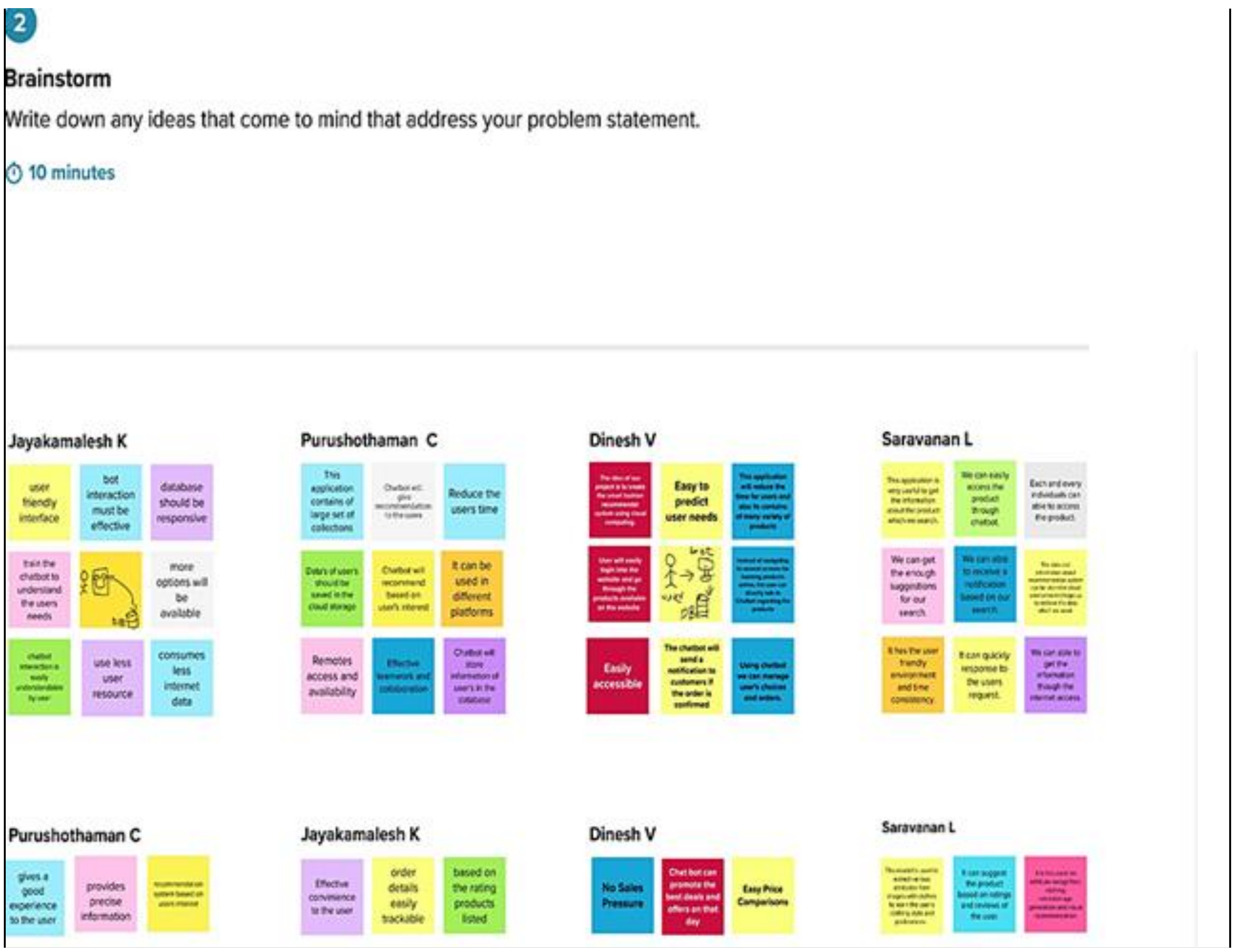
smart fashion recommender application- rapid growth of online-market for fashion are getting increasingly overwhelmed with the volume,velocity and veriety and production

Key rules of brainstorming
To run an smooth and productive session

Stay in topic.	Encourage wild ideas.
Defer judgment.	Listen to others.
Go for volume.	If possible, be visual.

Fig 3.2.1 Team gathering and Collabration

Step 2:Brainstrom



2

Brainstorm

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

10 minutes

Jayakamalesh K

- user friendly interface
- bot interaction must be effective
- database should be responsive
- train the chatbot to understand the users needs
- more options will be available
- chatbot interaction is easily understandable by user
- use less user resource
- consumes less internet data

Purushothaman C

- This application contains of large set of collections
- Chatbot will give recommendation to the users
- Reduce the users time
- Data's of users should be saved in the cloud storage
- Chatbot will recommend based on user's interest
- It can be used in different platforms
- Remotes access and availability
- Effective teamwork and collaboration
- Chatbot will store information of user's in the database

Dinesh V

- [illegible]
- Easy to predict user needs
- [illegible]
- [illegible]
- [illegible]
- Easily accessible
- The chatbot will send a notification to customers if the order is confirmed
- Using chatbot we can manage user's choices and orders.

Saravanan L

- This application is very useful to get the information about the product which we search.
- We can easily access the product through chatbot.
- Each and every individuals can able to access the product.
- We can get the enough suggestions for our search.
- We can able to receive a notification based on our search.
- [illegible]
- It has the user friendly environment and time consistency.
- It can quickly response to the users request.
- We can able to get the information though the internet access.

Purushothaman C

- gives a good experience to the user
- provides precise information
- recommendation system based on users interest

Jayakamalesh K

- Effective convinence to the user
- order details easily trackable
- based on the rating products listed

Dinesh V

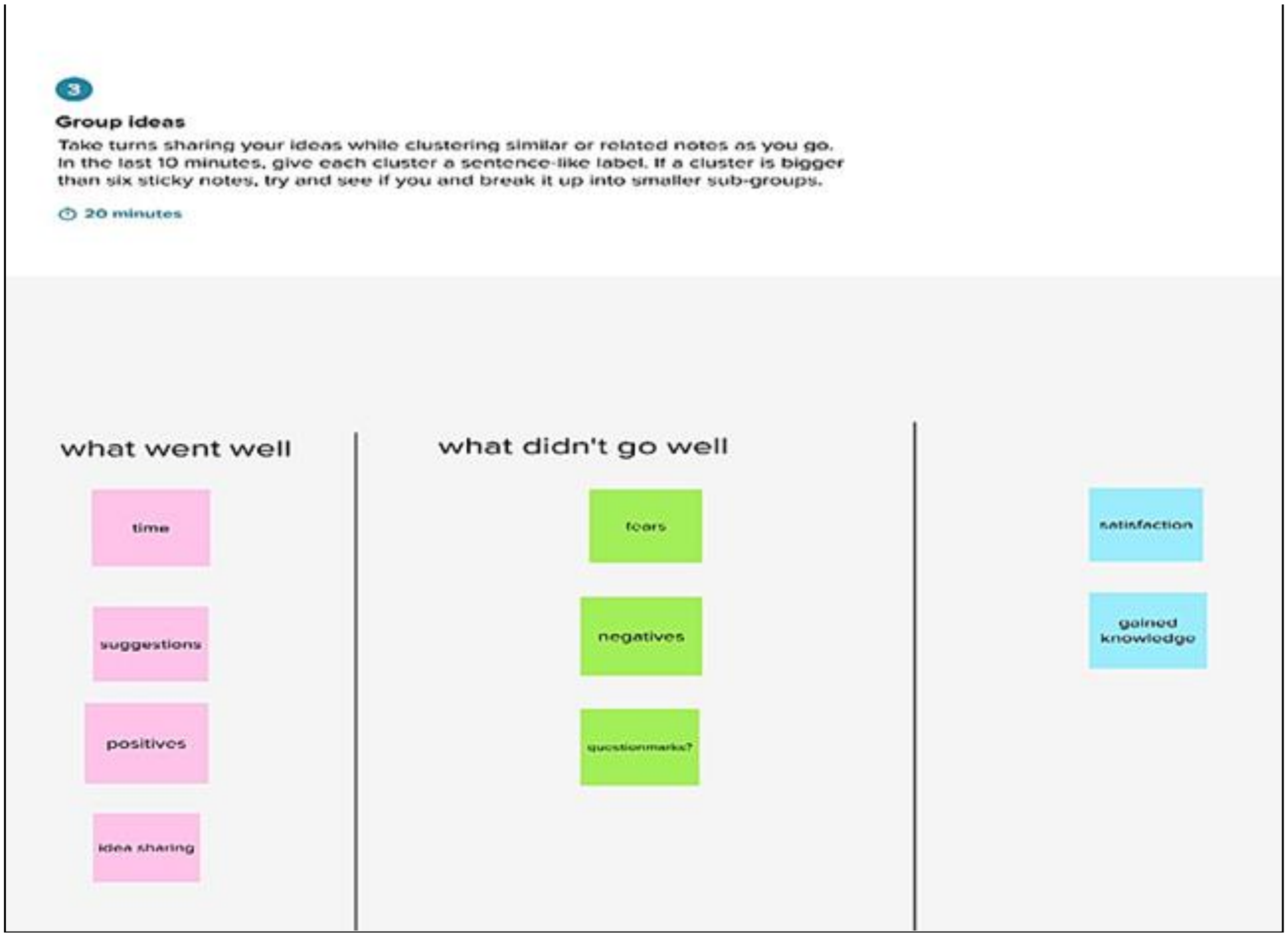
- No Sales Pressure
- Chat bot can promote the best deals and offers on that day
- Easy Price Comparisons

Saravanan L

- [illegible]
- It can suggest the product based on ratings and reviews of the user
- [illegible]

Fig 3.2.2 Brainstrom

Step 3: Idea listing and Grouping


3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

Fig 3.2.3 Group Ideas

Step 4: Idea Prioritization

In this idea prioritisation the basic ideas can be implemented to satisfy the requirements of the user. Initially the user can login to our application for searching their dresses based on their needs. The Chatbot can quickly respond to the user request. It has user-friendly interface so the user can easily access the dresses in our application. The data of the user can be stored in the cloud storage. So it will help to trace the data related to the user easily. Chatbot can reduce the time of the user. Then the database can be quickly respond to the user request and by using our application the quality and the price comparison of the difference dresses can be made. Our application can be easily used in any platform at anywhere and anytime. It can consists of the enough collection of dresses or data for different occasions to the users.

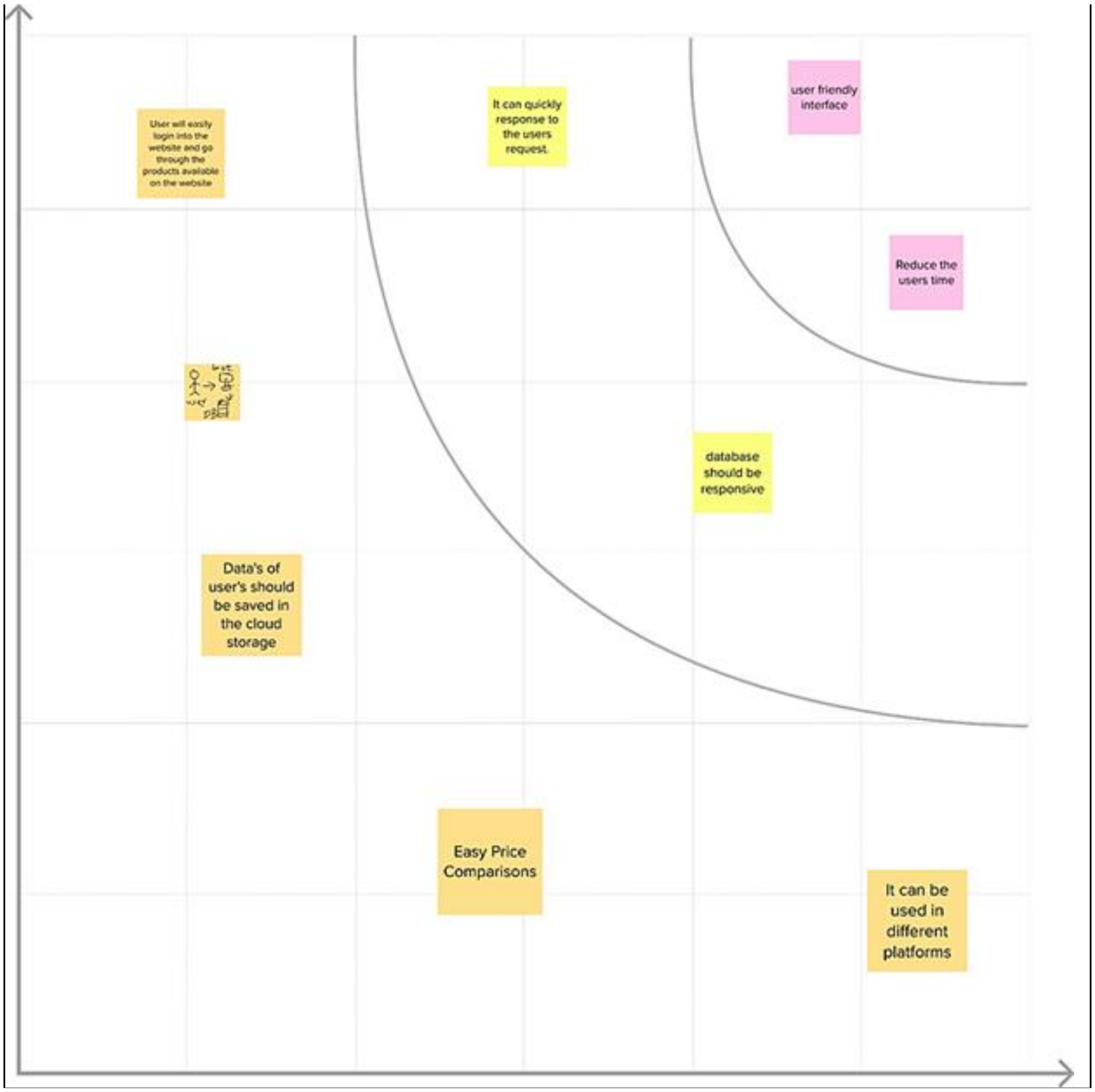


Fig 3.2.4 Idea Prioritization

3.3 Proposed Solution

Problem Statement:

With the rapid growth of online market for clothing and fashions consumers are getting increasingly overwhelmed with the volume, velocity and variety of production facing too many types of garments, consumers need to try them on repeatedly, which is somewhat time- and energy-consuming.

Solution Description:

Smart Fashion Recommender Application can tackle with choice overload by suggesting the most interesting apt products to the users.

Uniqueness:

Instead of searching manually a chatbot will help to find the right product effectively, with this feature user can save time and it is a easy process.

Customer Satisfaction:

This chatbot helps the users to find the right products easily, the innovations that all levels of business owners can take advantage of and the application used in all fashion markets.

Business Model:

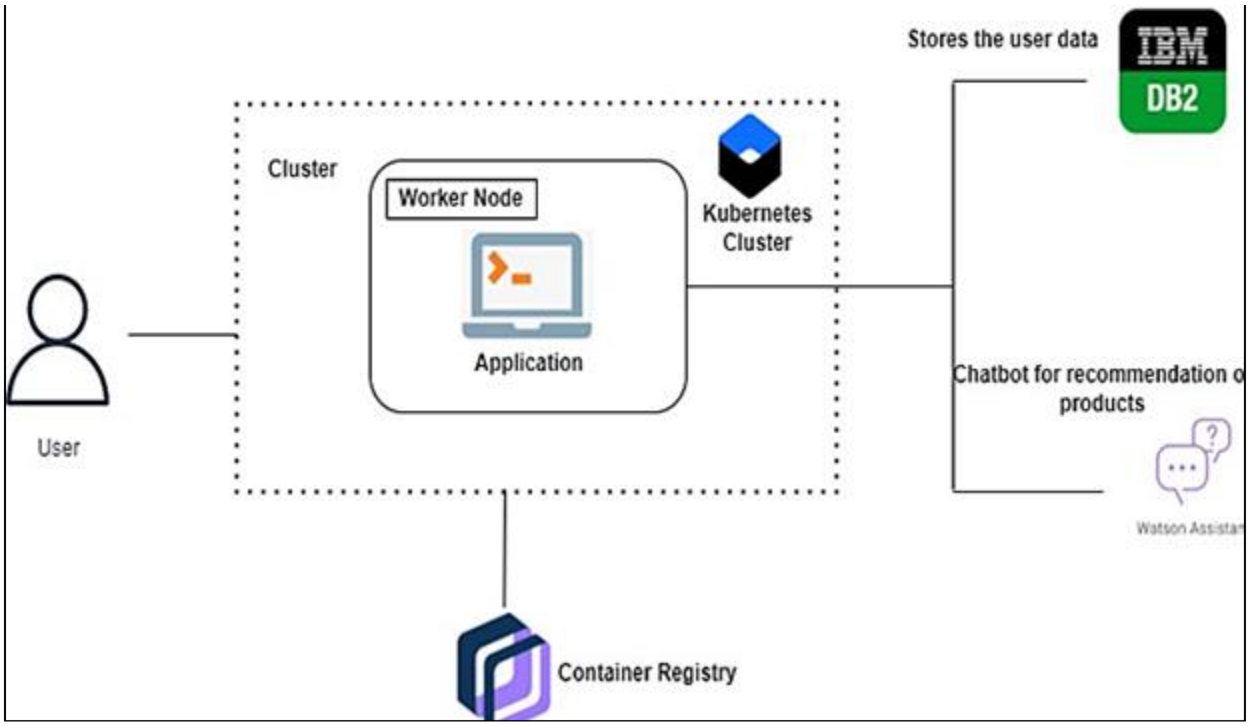


Fig 3.3.1 Business Model

Scalability of the solution:

- Responsiveness of the application
- Bot never runs into errors
- Optimized stock database

4. REQUIREMENT ANALYSIS

4.1 Functional Requirements

Registration:

Registration can be done using mobile number or g mail (primary key) and needed some user information.

Authentication:

User only log in by user id and password so the data will be secured without user knowledge know can access.

Reliable Servers:

Private database and Reliable and efficient sever was used.

Assistance:

Bot is integrated with the application to make the usability simple.

Certifications:

SSL certificate to enables encryption between the users and server.

4.2 Non-Functional Requirements

Usability:

A user-friendly interface with chat bot to make usability simple and efficient.

Security:

Log in credentials ensures the user data, Secure Service Socket (SSL) enables the encrypted transaction between the user and server.

Reliablity:

It has a fault-tolerance infrastructure it can be able to provide a reliable service.

Performance:

Modules are containerized to deploy the application faster and more securely so the application should be responsive.

Availability:

It is a cloud based web application so user can access with out any platform limitations ,just using a browsers with a internet connection is enough for use the application.

Scalability:

It has a quick request and response time, high throughput, enough network resources and so on.

5. PROJECT DESIGN

5.1 Data Flow Diagram

DFD (level 0):

Initially the administrator can login to the application. Then all the credentials in which the administrator is given that can be stored in the database. Then the chatbot can recommend the dresses based on the user needs. The administrator can monitoring the website security and website management of an application. The user of our application is made to login and they can get the recommendations based on their requirements. Then the new visitor of our application is made to register for a first time and the next time onwards they allow to login and access their products based on their interest

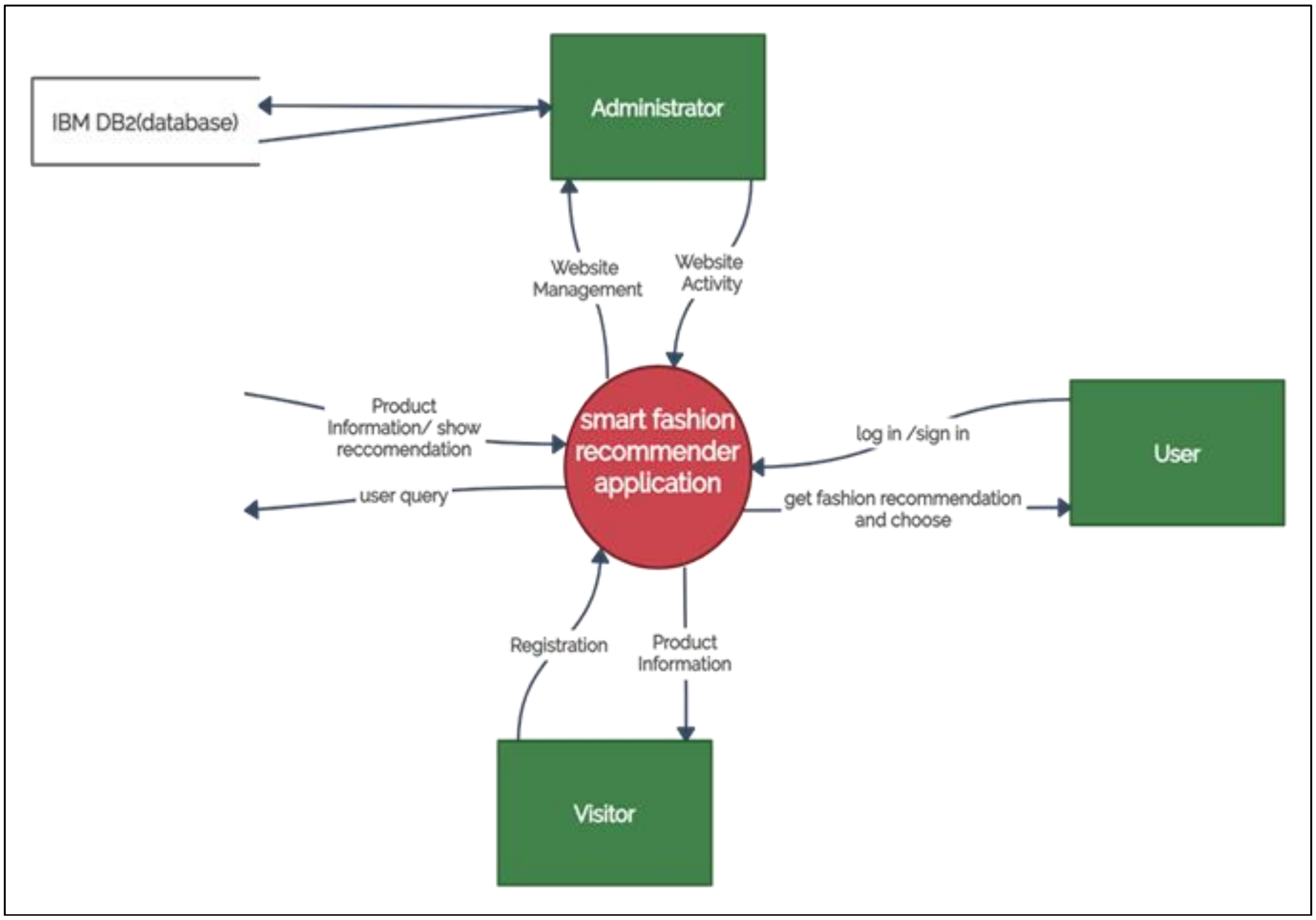


Fig 5.1.1 Data Flow Diagram DFD (level 0)

DFD (level 1):

In this level the user can login to the application. Users can make the query to the chatbot and it will respond to the query of the user. Then the main role of the chatbot is to interact with the user and suggest the product based on the user request. All the information about the user can be stored in the database. The administrator of the application have an access to add or remove the data and the images about the dresses from the database in the application.

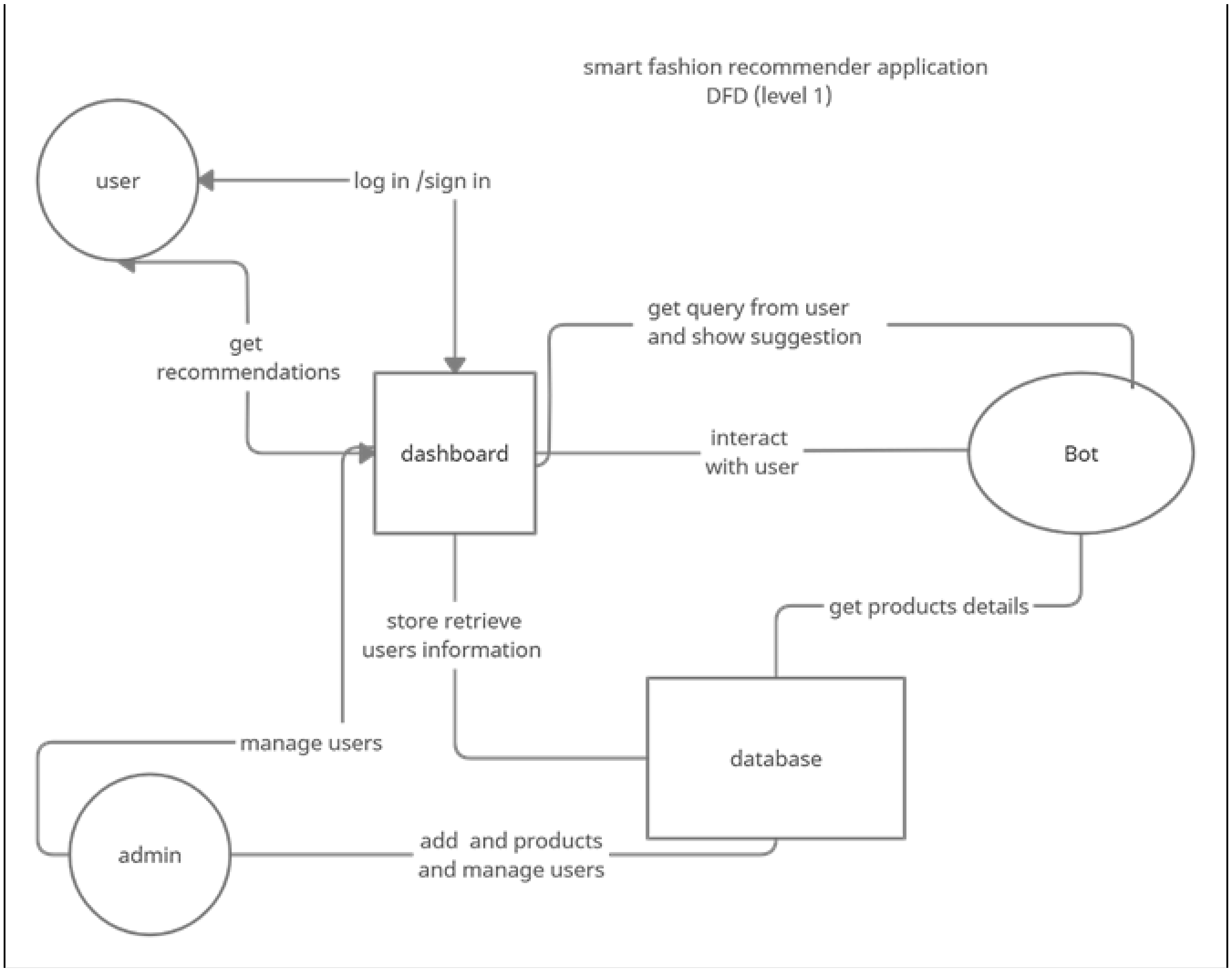

smart fashion recommender application
DFD (level 1)
user
log in /sign in
get recommendations
dashboard
get query from user and show suggestion
interact with user
Bot
get products details
store retrieve users information
manage users
database
admin
add and products and manage users

Fig 5.1.2 Data Flow Diagram (level 1)

DFD (level 2):

In this level the user can login to an application and they make the request based on their interest to the chatbot. The chatbot can quickly respond to their request. Then all the data of the user and the product can be stored in the database and it can be monitored by the administrator of the application. Then there can be a enough and accurate product is recommended to the user based on their request.

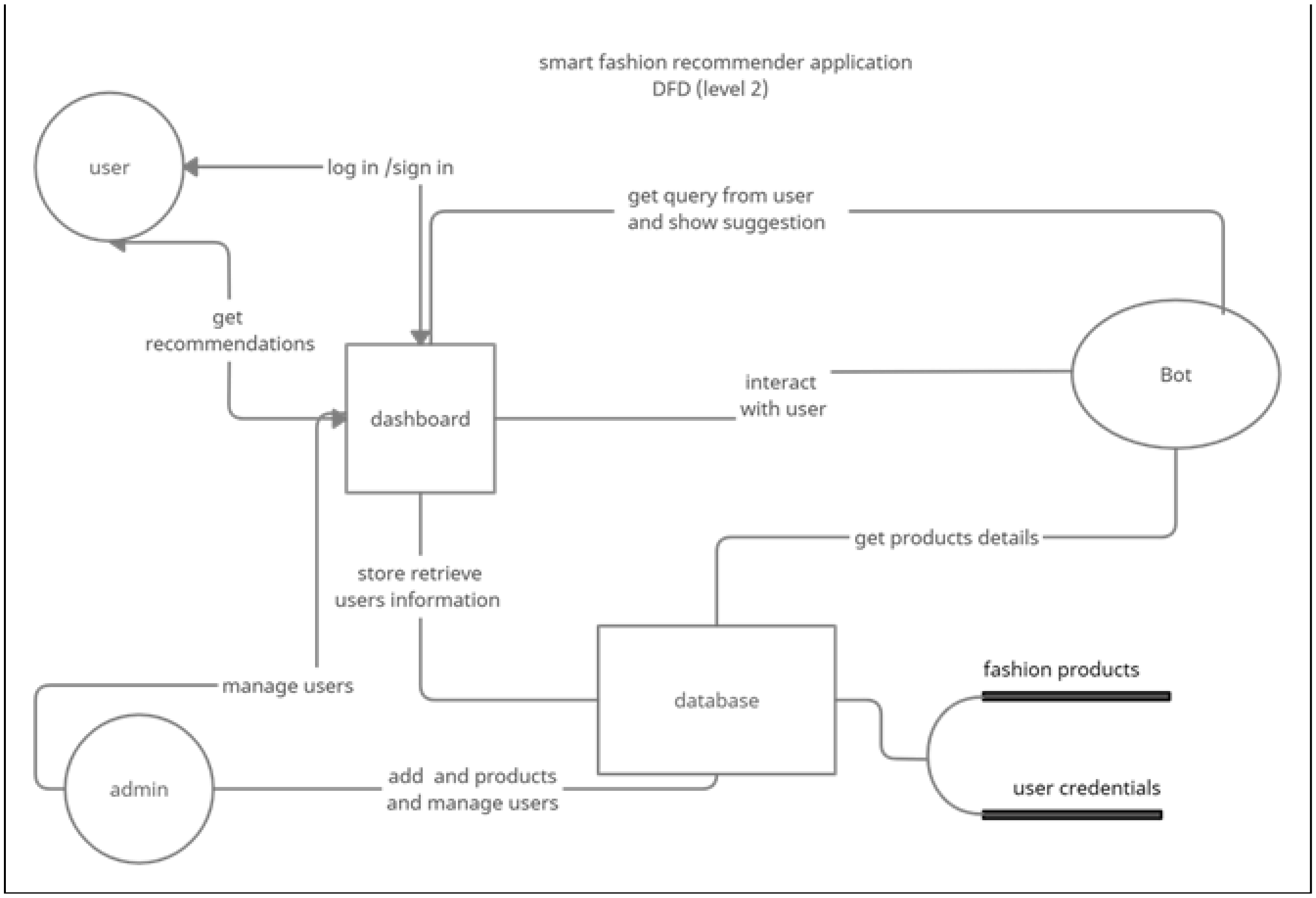


Fig 5.1.3 Data Flow Diagram (level 2)

5.2 Solution & Technical Architecture

In this the user can login or register to the web application with the details.In the chatbot the user can select the dresses with their interest and the chatbot will show the categories of dresses in the database and the user can order the dress with their details.The details will be stored in the database.In this the admin can login and add or remove the image of dresses from the database and maintain the stocks.

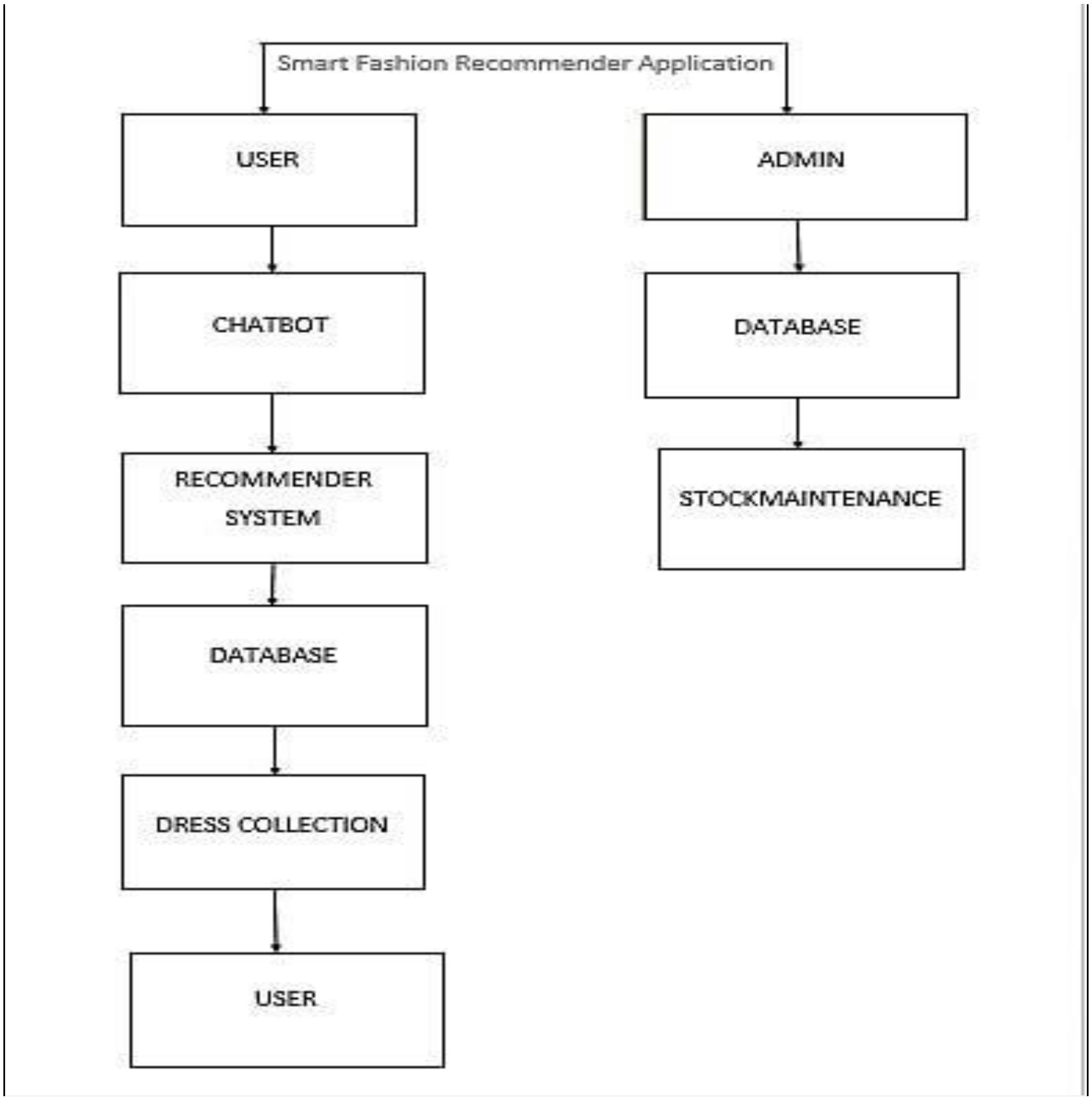


Fig 5.2.1 Solution Architecture

5.3 User Stories

User Type	User Story/Task	Acceptance Criteria
Customer-1 (Mobile User)	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard.
Customer-2 (Mobile User)	As a user, I will receive confirmation email once I have registered for the application.	I can receive confirmation email & click confirm.
Customer-3 (Mobile User)	As a user, I can register for the application through Gmail.	I can use my gmail credentials instead of manually entering my details.
Customer-4 (Mobile User)	As a user, I can log into the application by entering email & password.	I can retrieve my information.
Customer-5 (Mobile User)	As a user, I can continue my old progress.	I can continue where I left.
Customer-6 (Mobile User)	As a user, I expect a bot to assist me.	I can interact with the bot.
Customer-1 (Desktop User)	I can register for the application using browser by entering the email, password,	I can access my profile.

	name and personal information and conforming my password.	
Customer-2 (Desktop User)	As a user, I will receive a confirmation and greeting mail from the web application.	I can receive the confirmation and greeting mail.
Customer-3 (Desktop User)	As a user, I can register for the application through Gmail.	I can use my gmail credentials instead of manually entering my details.
Customer-4 (Desktop User)	As a user, I can log into the application by entering email & password.	I can retrieve my profile.
Customer-5 (Desktop User)	As a user, I can view my personalized dashboard.	I can continue where I left.
Customer-6 (Desktop User)	As a user, I expect a bot to assist me.	I can interact with the bot.
Adiministration	As a admin, login on admin page.	I can access admin menu.
	As a admin, access the admin menu using admin credentials.	I can log in using admin credentials.
	As a admin, manage the application through admin panel.	I can access admin panel.
	As a admin, add and remove available products from data base.	I can access database.

Table 5.3.1 User Stories

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Sprint	**Functional Requirements**	**User Story /Task**
Sprint-1	User Panel	The user will login into the website and go through the products available on the website.
Sprint-2	Admin Panel	The role of the admin is to check out the database about the stock and have a track of all the things that the users are purchasing.

Sprint-3	Chat Bot	The user can directly talk to Chatbot regarding the products. Get the recommendations based on information provided by the user.
Sprint-4	Final Delivery	Container of applications using docker kubernetes and deployment the application. Create the documentation and final submit the application.

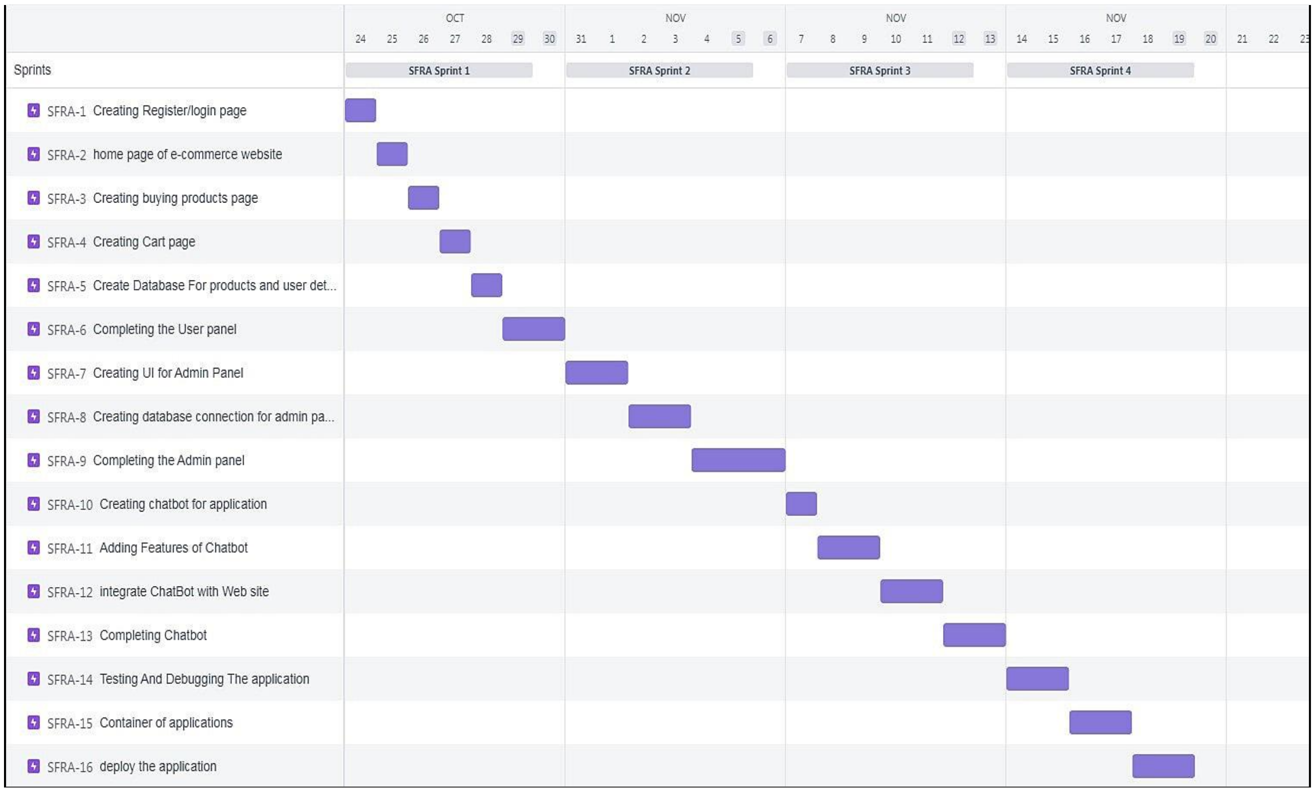
Table 6.1.1 Sprint Planning & Estimation

6.2 Sprint Delivery Schedule

Sprint	Duration	Sprint Start Date	Sprint End Date (Planned)	Sprint Release Date (Actual)
Sprint-1	6 Days	24 Oct 2022	29 Oct 2022	29 Oct 2022
Sprint-2	8 Days	31 Oct 2022	05 Nov 2022	07 Nov 2022
Sprint-3	6 Days	07 Nov 2022	12 Nov 2022	12 Nov 2022
Sprint-4	6 Days	14 Nov 2022	19 Nov 2022	19 Nov 2022

Table 6.2.1 Sprint Delivery Schedule

6.3 Reports From JIRA



Sprints

SFRA-1 Creating Register/login page
SFRA-2 home page of e-commerce website
SFRA-3 Creating buying products page
SFRA-4 Creating Cart page
SFRA-5 Create Database For products and user det...
SFRA-6 Completing the User panel
SFRA-7 Creating UI for Admin Panel
SFRA-8 Creating database connection for admin pa...
SFRA-9 Completing the Admin panel
SFRA-10 Creating chatbot for application
SFRA-11 Adding Features of Chatbot
SFRA-12 integrate ChatBot with Web site
SFRA-13 Completing Chatbot
SFRA-14 Testing And Debugging The application
SFRA-15 Container of applications
SFRA-16 deploy the application

Fig 6.3.1 JIRA Report

7.CODING AND SOLUTION

7.1 Login and Register page

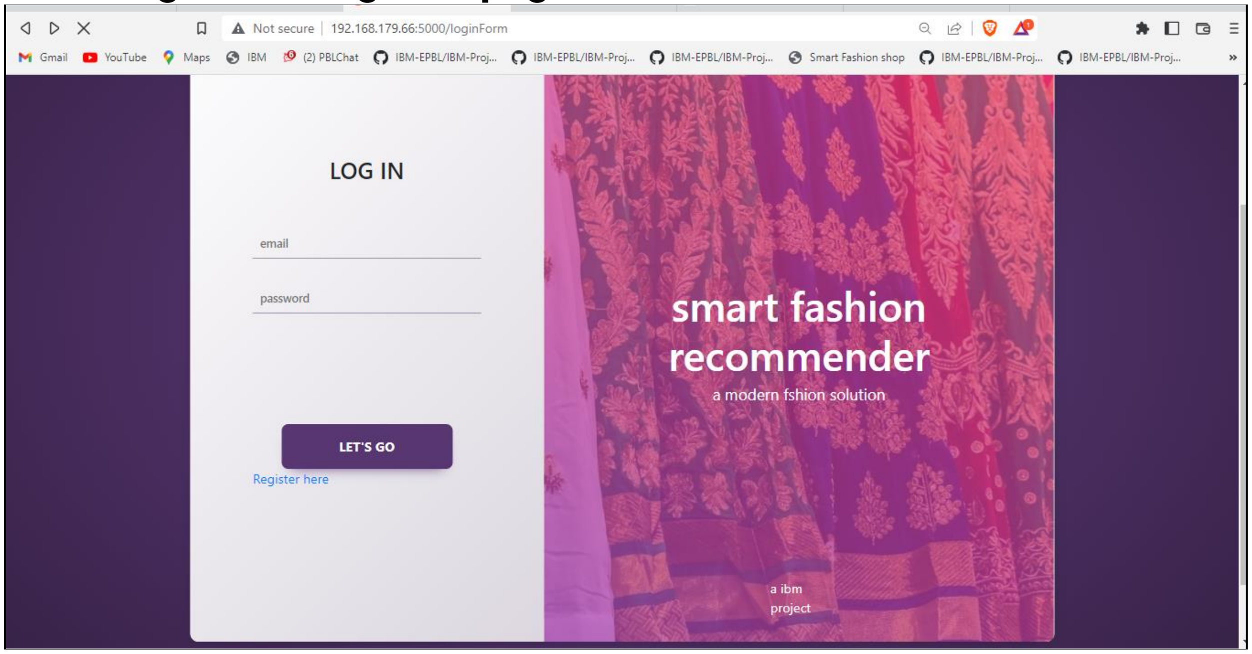


Fig 7.1.1 Log in Page

Existing user can directly login using the login credentials, the credentials will checked in database if the entered credentials is match in db user will redirected to home page other wise redirected to registerpage.

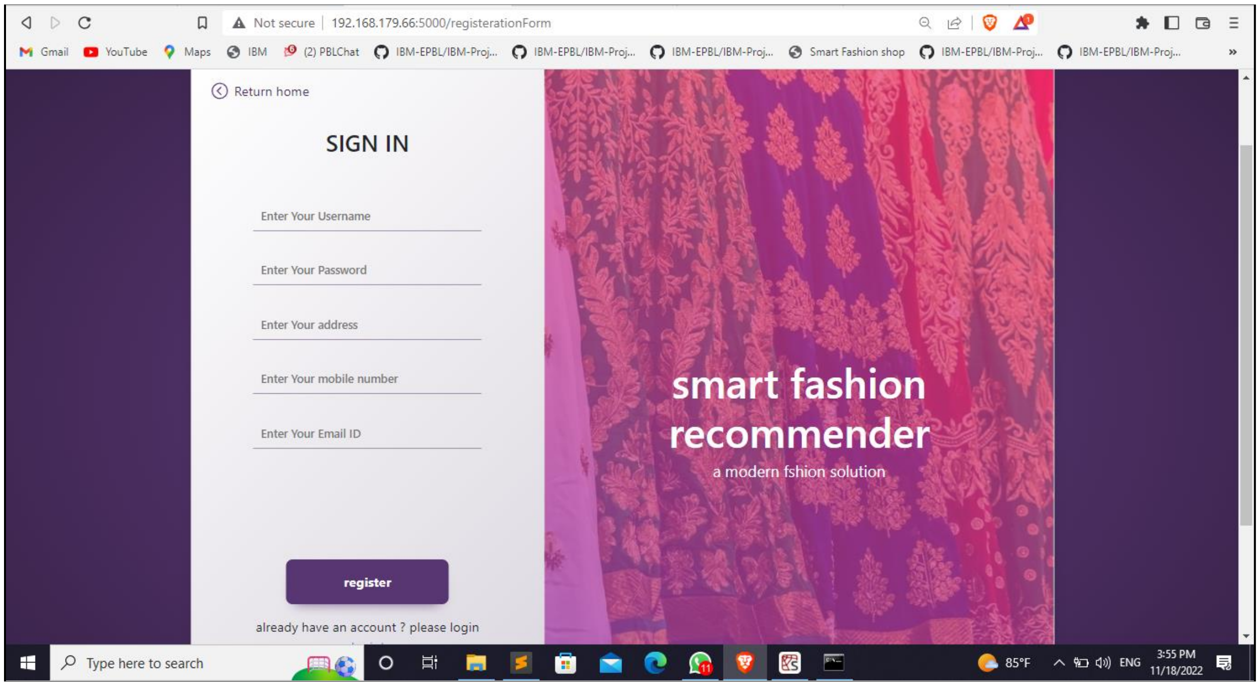


Fig 7.1.2 Register Page

New user can register in sign in page with entering the mentioned details. Once the user completed the details entering, they want to click the register button . Then only user credentials are stored at the database.

7.2.1 Home Page

Smart Fashion Recommender is the place to find fashionable and affordable dresses for both men and women. With a constantly updating inventory of casual, formal, traditonal, vacations and partywear etc. It contains different variety of fashion products.

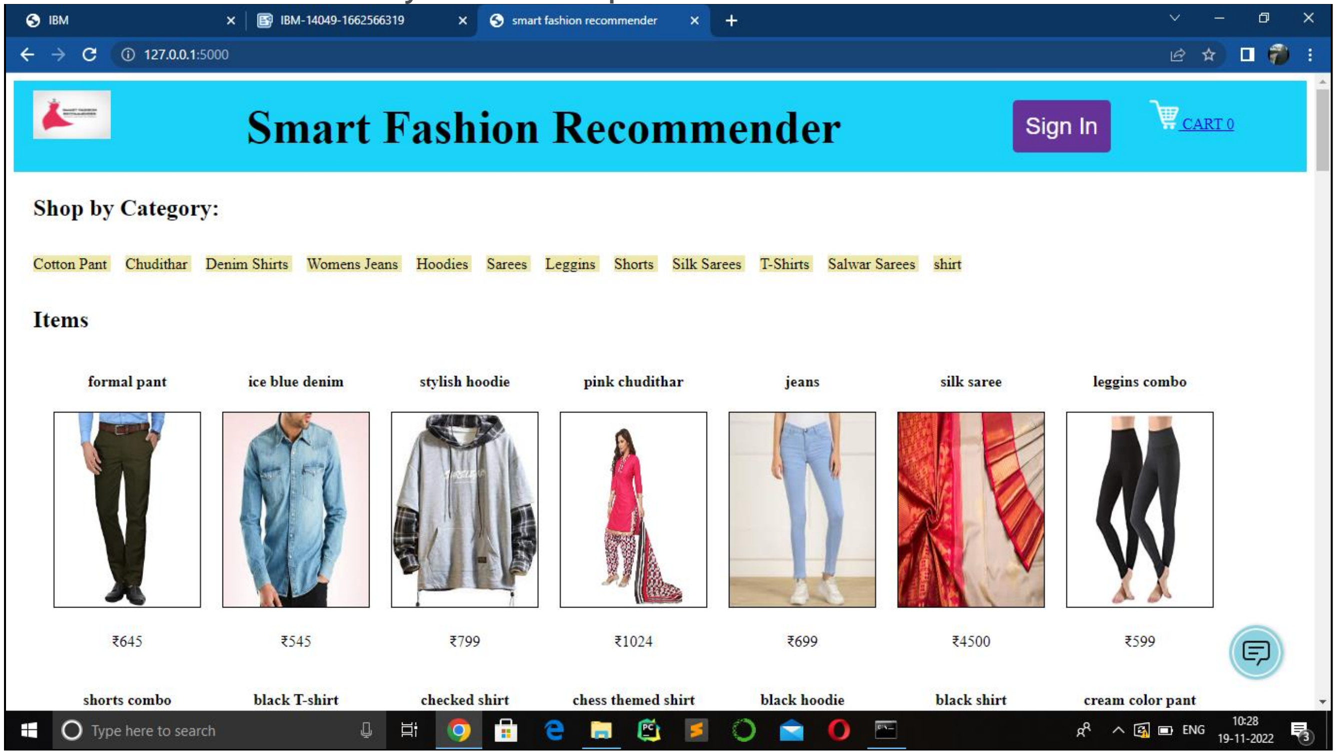
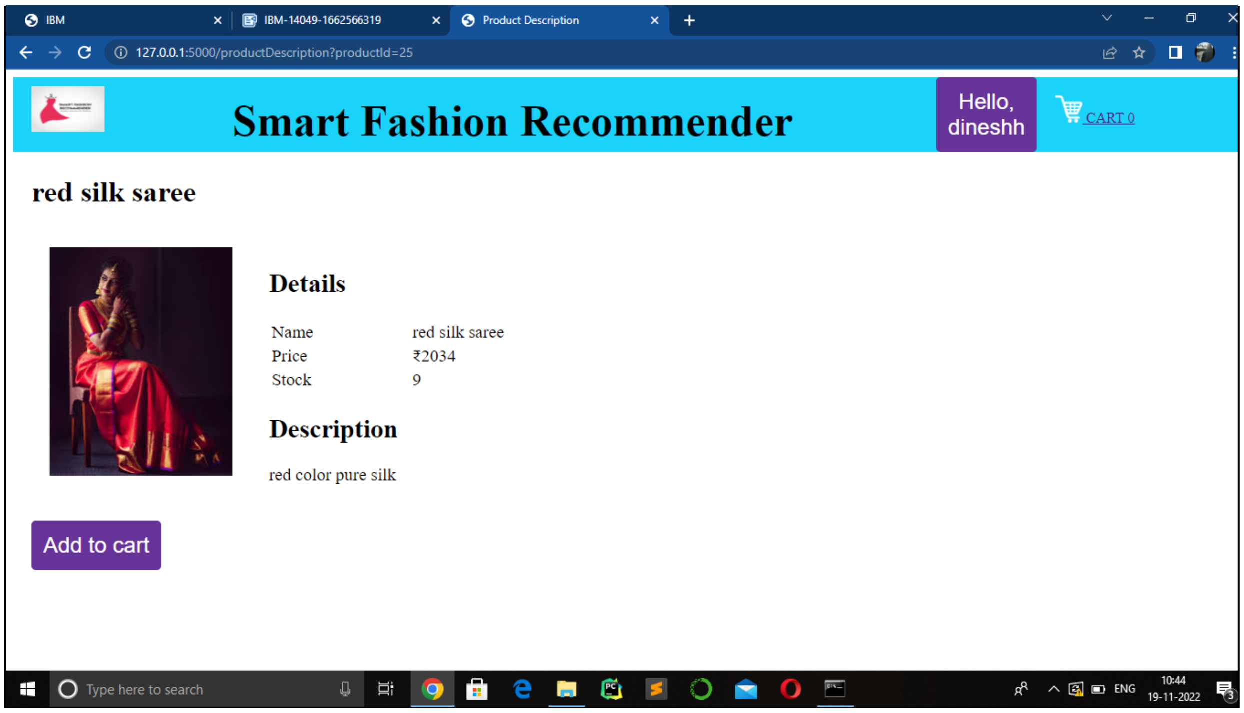


Fig 7.2.1.1 Home Page

7.2.2 Product description

Once the user click the product image in home its move on to the product description page, the sample is shown below. The description page conatins details about the product which is name, price, stock details and detail description about the product.



Smart Fashion Recommender

red silk saree

Details

Name	red silk saree
Price	₹2034
Stock	9

Description

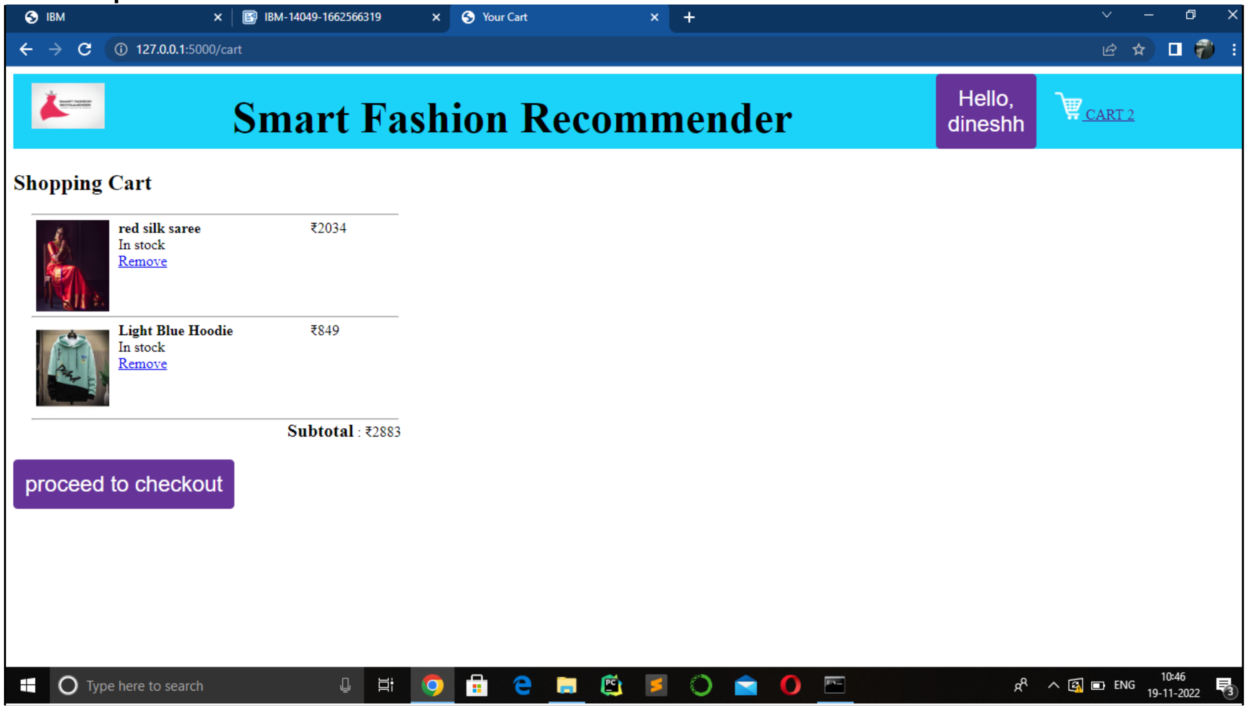
red color pure silk

Add to cart

Fig 7.2.2.1 Product Description Page

7.2.3 Cart

When the user want buy the product means they can easily add the product to the cart. The sample figure of product added to cart is shown below.This page contains the product that are added by the user.It contains total price of the product in cart.



Smart Fashion Recommender

Shopping Cart

red silk saree In stock Remove	₹2034
Light Blue Hoodie In stock Remove	₹849

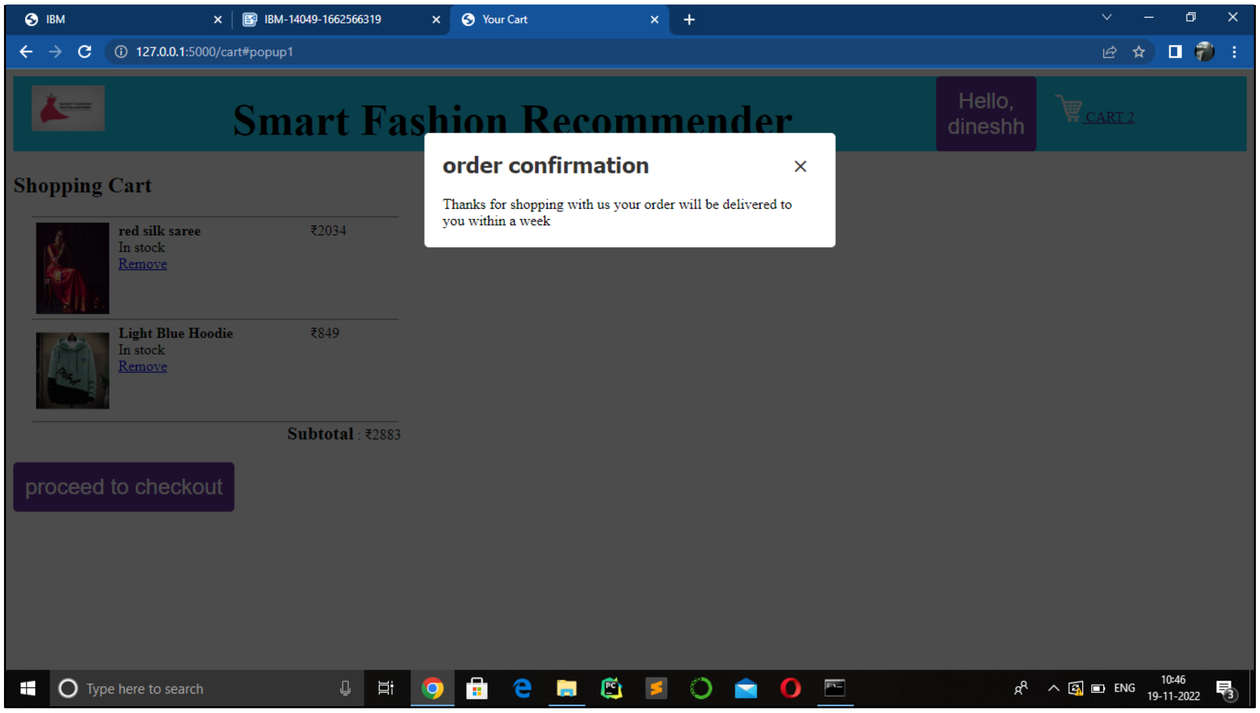
Subtotal : ₹2883

proceed to checkout

Fig 7.2.3.1 Cart Page

When the user confirmed to buy the product they want to click the proceed to check out button.

Then only the order will be confirmed. Then the order confirmation pop up is shown to the user.



Shopping Cart

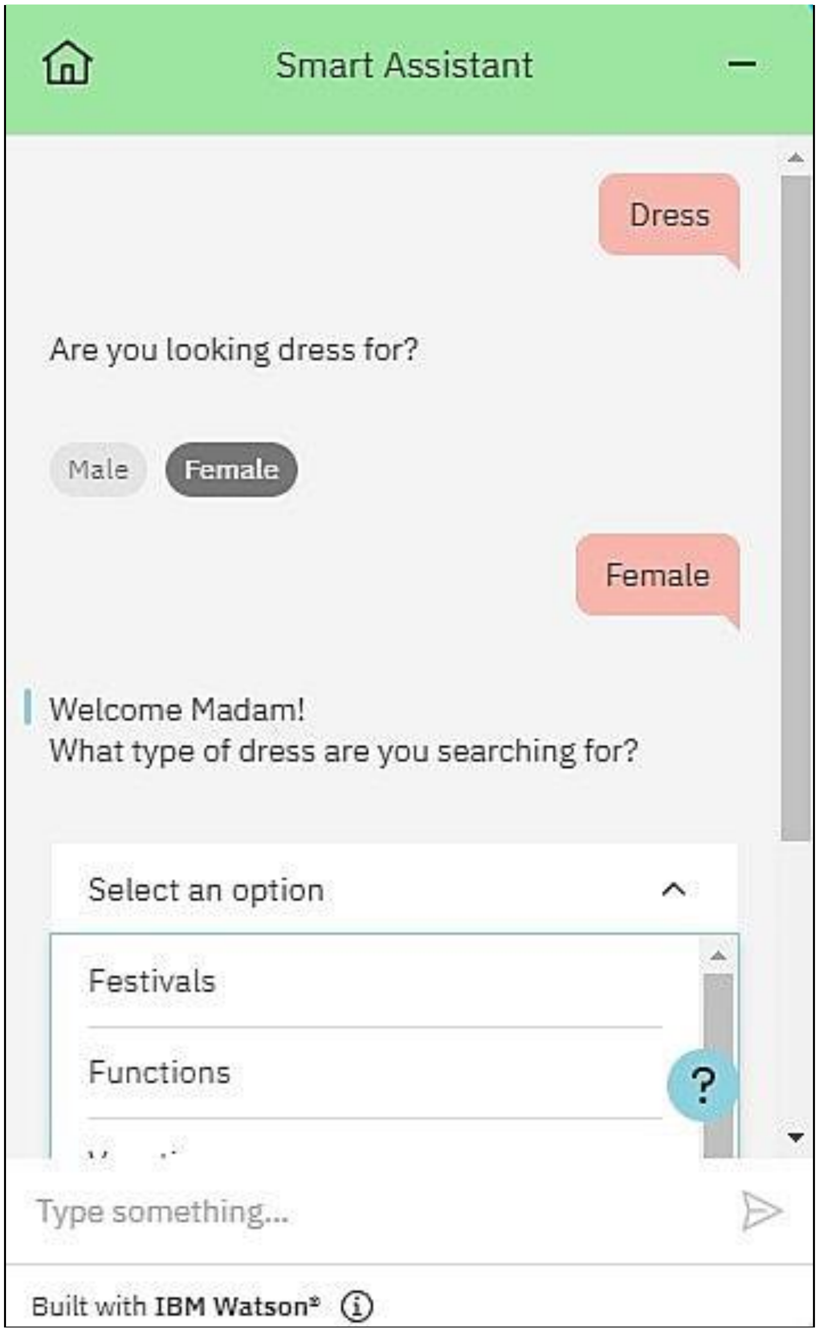
red silk saree In stock Remove	₹2034
Light Blue Hoodie In stock Remove	₹849
	Subtotal : ₹2883

order confirmation

Thanks for shopping with us your order will be delivered to you within a week

Fig 7.2.3.2 Confirmation Window

7.3 Chatbot


Smart Assistant

Dress

Are you looking dress for?

Female

Welcome Madam!
What type of dress are you searching for?

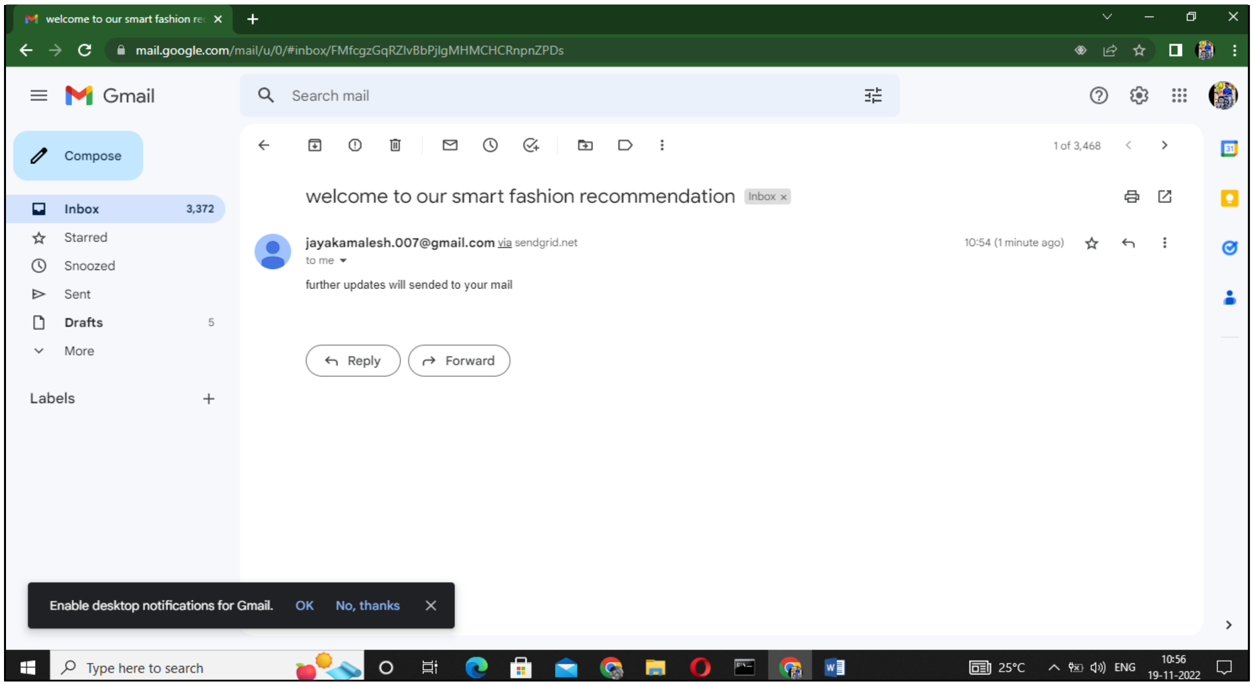
Fig 7.3.1 Chatbot Window

It is the smart assistant for the smart fashion recommemder. Where the user can able to get the product suggestions from our assistant.It contains of products like traditional, formal, vacation,
party wear etc.When the user click any of the given option, then our assistant show products to the user.

7.4 Sendgrid

Sendgrid is a type of mail system which is used to send the mail to the user who are registered.

The Sendgrid send the mail to the user regarding confirmation of registration, product offer details, and order details of the product to the users.



welcome to our smart fashion recommendation

jayakamalesh.007@gmail.com via sendgrid.net
to me
10:54 (1 minute ago)

further updates will sended to your mail

Enable desktop notifications for Gmail. OK No, thanks

Fig 7.4.1 Sendrid Mail Service Sample Mail

7.5 SQL Schema

Table Name	**Schema**
Categories	CREATE TABLE categories (categoryId INTEGER PRIMARY KEY, name char(20))
Cart	CREATE TABLE cart (userId INTEGER, productId INTEGER, FOREIGN KEY(userId) REFERENCES users(userId), FOREIGN KEY(productId) REFERENCES products(productId))
Products	CREATE TABLE products (productId INTEGER PRIMARY KEY, name CHAR, price REAL, description CHAR, image

	CHAR, stock INTEGER, categoryId INTEGER, FOREIGN KEY(categoryId) REFERENCES categories(categoryId))
Users	CREATE TABLE users (userId INTEGER PRIMARY KEY, password CHAR, email CHAR, firstName CHAR, lastName CHAR, address1 CHAR, address2 CHAR, zipcode CHAR, city CHAR, state CHAR, country CHAR, phone CHAR)

8. TESTING

8.1 Test Cases

This report shows the number of test cases that have passed, failed, and untested.

Section	**Total Cases**	**Not Tested**	**Fail**	**Pass**
Login	5	0	0	5
Register	7	0	0	7
Home Page	2	0	0	2
Order Page	3	0	0	3
Order Products	9	0	0	9
Final Report Output	4	0	0	4

Version Control	2	0	0	2

Table 8.2.2 Test Case Table

8.2 User Acceptance Testing

1. Purpose of Document

The purpose of this document is to briefly explain the test coverage and open issues of the Smart Fashion Recommender Application project at the time of the release to User Acceptance Testing (UAT).

2. Defect Analysis

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved.

Resolution	**Severity 1**	**Severity 2**	**Severity 3**	**Severity 4**	**Subtotal**
By Design	5	5	2	3	21
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	11	2	4	20	37
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	24	14	13	26	77

Table 8.2.1 Defect Analysis Table

9. RESULTS

9.1 Performance Metrices

Project team shall fill the following information in model performance testing.

1. NFT - Risk Assessment

S. No	Project Name	Scope/feature	Functional Changes	Hardware Changes	Software Changes	Load/Volume Changes	Risk Score	Justification
1	Smart Fashion Recommender Application	New	Low	No Changes	Moderate	>5 to 10%	ORANGE	As we have seenthe changes

Fig 9.1.1 Risk Assessment

2. NFT - Detailed Test Plan

S.No	Project Overview	NFT Testapproach	Assumptions/Dependencies/Risks	Approvals/Sign Off
1	Smart Fashion Recommender Application	Manual testing	laptop or mobilewith internet connection V.Dinesh	

Fig 9.1.2 Detailed Test Plan

3. End of the Test Report

S.No	Project Overview	NFT Test approach	Test Outcome	Go/No-Go Decision	Recommendations	Identified Defects(Detected/Closed/Open)	Approvals/Sign Off

1	Smart FashionRecommender Application	Manual testing	Work as we expexted		Use Laptops/Desktop Mode	No Defects	V.Dinesh

Fig 9.1.3 End of the Test Report

10. ADVANTAGES AND DISADVANTAGES

Advantages

- It reduces the user's time
- It helps the user to Shopping with Smart Assistant
- It helps the user by Shopping at home
- It helps to User manage their order list

Disadvantages

- User's have fear about Online Shopping
- User's have fear about Online Payment
- User have receive some wrong items
- User have receive bad quality items

11. CONCLUSION

Recommendation systems have the potential to explore new opportunities for retailers by enabling them to provide customized recommendations to consumers based on information retrieved from the Internet. They help consumers to instantly find the products and services that closely match with their choices. Moreover, different stat-of-the-art algorithms have been developed to recommend products based on users' interactions with their social groups. Therefore, research on embedding social media images within fashion recommendation systems has gained huge popularity in recent times. This paper presented a review of the fashion recommendation systems, algorithmic models and filtering techniques based on the academic articles related to this topic. The technical aspects, strengths and weaknesses of the filtering techniques have been discussed elaborately, which will help future researchers gain an in-depth understanding of fashion recommender systems. However, the proposed prototypes should be tested in commercial applications to understand their feasibility and accuracy in the retail market, because inaccurate recommendations can produce a negative impact on a customer. Moreover, future research should concentrate on including time series analysis and accurate categorization of product images based on the variation in color, trend and clothing style in order to develop an effective recommendation system. The proposed model will follow brand specific personalization campaigns and hence it will ensure highly curated and tailored offerings for users. Hence, this research will be highly beneficial for researchers interested in using augmented and virtual reality features to develop recommendation systems.

12. FUTURE SCOPE

There has been significant progress recently in fashion recommendation system research, which will benefit both consumers and retailers soon. The use of product and user images, textual content, demographic history, and cultural information is crucial in developing recommendation frameworks.

Product attributes and clothing style matching are common features of collaborative and content-based filtering techniques. Researchers can develop more sophisticated hyper personalized filtering techniques considering the correlation between consumers' clothing stylesand personalities. The methods based on employing a scoring system for quantifying each product attribute will be helpful in increasing the precision of the model. The use of virtual sales advisers in an online shopping portal would provide consumers with a real time offline shopping experience. Retailers can collect the data on users' purchase history and product reviews from the recommendation system and subsequently use them in style prediction for the upcoming seasons. The integration of different domain information strengthens the deep learning paradigm by enabling the detection of design component variation, which improves the performance of the recommendation system in the long run. Deep learning approaches should be more frequently used to quickly explore fashion items from different online databases to provide prompt recommendations to users or consumers.

13. APPENDIX

Source Code

app.py

```
from flask import *
import re
import ibm_db
import ibm_db_dbi
import ibm_boto3
from ibm_botocore.client import Config
from ibm_botocore.exceptions import ClientError
import ibm_s3transfer.manager
import sendgrid
import os
from sendgrid.helpers.mail import *

app = Flask(__name__)

app.secret_key = 'a'
conn                                                                =
ibm_db_dbi.Connection(ibm_db.connect("DATABASE=bludb;HOSTNAME=9
938aec0-8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","",""))
con        =ibm_db.connect("DATABASE=bludb;HOSTNAME=9938aec0-8105-
433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","","")

def getLoginDetails():
   conn                                                             =
ibm_db_dbi.Connection(ibm_db.connect("DATABASE=bludb;HOSTNAME=9
938aec0-8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","",""))
   cur = conn.cursor()
   if 'email' not in session:
          loggedIn = False
          username = ''
          noOfItems = 0
   else:
          loggedIn = True
          cur.execute("SELECT userId, username FROM users WHERE email =
?", (session['email'], ))
          userId, username = cur.fetchone()
          cur.execute("SELECT count(productId) FROM kart WHERE userId = ?",
(userId, ))
          noOfItems = cur.fetchone()[0]
   conn.close()
   return (loggedIn, username, noOfItems)
```

```
@app.route("/", methods=['GET', 'POST'])
def root():
   conn                                                                    =
ibm_db_dbi.Connection(ibm_db.connect("DATABASE=bludb;HOSTNAME=9
938aec0-8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","",""))
   loggedIn, username, noOfItems = getLoginDetails()
   cur = conn.cursor()
   cur.execute('SELECT  productId,  name,  price,  description,  image,  stock
FROM products')
   itemData = cur.fetchall()
   cur.execute('SELECT categoryId, name FROM categories')
   categoryData = cur.fetchall()
   itemData = parse(itemData)
   return
render_template('home.html',itemData=itemData,categoryData=categoryDa
ta,loggedIn=loggedIn,username=username, noOfItems=noOfItems)

@app.route("/add")
def admin():
   conn                                                                    =
ibm_db_dbi.Connection(ibm_db.connect("DATABASE=bludb;HOSTNAME=9
938aec0-8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","",""))
   cur = conn.cursor()
   cur.execute("SELECT categoryId, name FROM categories")
   categories = cur.fetchall()
   conn.close()
   return render_template('add.html', categories=categories)

@app.route("/addItem", methods=["GET", "POST"])
def addItem():
   if request.method == "POST":
       name = request.form['name']
       price = float(request.form['price'])
       description = request.form['description']
```

```
        stock = int(request.form['stock'])
        categoryId = int(request.form['category'])
        new_bucket_name="menspants"
        #Uploading image procedure
        pics = request.files['image']
        new_item_name= name +".png"
        new_file_path=pics
        image="https://menspants.s3.jp-tok.cloud-object-
storage.appdomain.cloud/"+name+".png"
        upload_large_file(new_bucket_name, new_item_name,new_file_path)
        try:
          conn     =ibm_db.connect("DATABASE=bludb;HOSTNAME=9938aec0-
8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","","")

          insert_sql = "INSERT INTO  products (name, price, description, image,
stock, categoryId) VALUES (?, ?, ?, ?, ?, ?)"
          prep_stmt = ibm_db.prepare(conn, insert_sql)
          ibm_db.bind_param(prep_stmt, 1, name)
          ibm_db.bind_param(prep_stmt, 2, price)
          ibm_db.bind_param(prep_stmt, 3, description)
          ibm_db.bind_param(prep_stmt, 4, image)
          ibm_db.bind_param(prep_stmt, 5, stock)
          ibm_db.bind_param(prep_stmt, 6, categoryId)
          ibm_db.execute(prep_stmt)
          msg="added successfully"
        except:
          msg="error occured"
          conn.rollback()
        print(msg)
        return redirect(url_for('root'))

@app.route("/remove")
def remove():
   conn                                                                        =
ibm_db_dbi.Connection(ibm_db.connect("DATABASE=bludb;HOSTNAME=9
938aec0-8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","",""))
   cur = conn.cursor()
   cur.execute('SELECT productId, name, price, description, image, stock
FROM products')
   data = cur.fetchall()
   conn.close()
   return render_template('remove.html', data=data)

@app.route("/removeItem")
def removeItem():
   productId = request.args.get('productId')
   conn                                                              =
ibm_db_dbi.Connection(ibm_db.connect("DATABASE=bludb;HOSTNAME=9
938aec0-8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","",""))
   try:
         cur = conn.cursor()
         cur.execute('DELETE FROM products WHERE productID = ?',
(productId, ))
         conn.commit()
         msg = "Deleted successsfully"
   except:
         conn.rollback()
         msg = "Error occured"
   conn.close()
   print(msg)
   return redirect(url_for('root'))

@app.route("/displayCategory")
def displayCategory():
      loggedIn, firstName, noOfItems = getLoginDetails()
      categoryId = request.args.get("categoryId")
      conn                                                           =
ibm_db_dbi.Connection(ibm_db.connect("DATABASE=bludb;HOSTNAME=9
938aec0-8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","",""))
```

```
        cur = conn.cursor()
        cur.execute("SELECT products.productId, products.name, products.price, products.image, categories.name FROM products, categories WHERE products.categoryId = categories.categoryId AND categories.categoryId = ?", (categoryId, ))
        data = cur.fetchall()
        conn.close()
        categoryName = data[0][4]
        data = parse(data)
        return render_template('displayCategory.html', data=data, loggedIn=loggedIn, firstName=firstName, noOfItems=noOfItems, categoryName=categoryName)

@app.route("/account/profile")
def profileHome():
    if 'email' not in session:
        return redirect(url_for('root'))
    loggedIn, firstName, noOfItems = getLoginDetails()
    return render_template("profileHome.html", loggedIn=loggedIn, firstName=firstName, noOfItems=noOfItems)

@app.route("/loginForm")
def loginForm():
    if 'email' in session:
        return redirect(url_for('root'))
    else:
        return render_template('login.html', error='')

@app.route("/login", methods = ['POST', 'GET'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        if is_valid(email, password):
            session['email'] = email
            return redirect(url_for('root'))
        else:
            error = 'Invalid UserId / Password'
            return render_template('login.html', error=error)
def is_valid(email, password):
   cur = conn.cursor()
   cur.execute('SELECT email, password FROM users')
   data = cur.fetchall()
   for row in data:
     if row[0] == email and row[1] == password:
          return True
   return False

@app.route("/productDescription")
def productDescription():
    conn                                                                    =
ibm_db_dbi.Connection(ibm_db.connect("DATABASE=bludb;HOSTNAME=9
938aec0-8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","",""))
    loggedIn, username, noOfItems = getLoginDetails()
    productId = request.args.get('productId')
    cur = conn.cursor()
    cur.execute('SELECT productId, name, price, description, image, stock
FROM products WHERE productId = ?', (productId, ))
    productData = cur.fetchone()
    conn.close()
    return    render_template("productDescription.html",    data=productData,
loggedIn = loggedIn, username = username, noOfItems = noOfItems)

@app.route("/addToCart")
def addToCart():
    if 'email' not in session:
        return redirect(url_for('loginForm'))
    else:
        productId = int(request.args.get('productId'))
        conn                                                                =
ibm_db_dbi.Connection(ibm_db.connect("DATABASE=bludb;HOSTNAME=9
938aec0-8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","",""))
        cur = conn.cursor()
        cur.execute("SELECT    userId    FROM    users    WHERE    email    =    ?",
(session['email'], ))
        userId = cur.fetchone()[0]
        try:
            cur.execute("INSERT INTO kart (userId, productId) VALUES (?, ?)",
(userId, productId))
            conn.commit()
            msg = "Added successfully"
        except:
            conn.rollback()
            msg = "Error occured"
        conn.close()
        return redirect(url_for('root'))

@app.route("/cart")
def cart():
    if 'email' not in session:
        return redirect(url_for('loginForm'))
    loggedIn, username, noOfItems = getLoginDetails()
    email = session['email']
    conn                                                                  =
ibm_db_dbi.Connection(ibm_db.connect("DATABASE=bludb;HOSTNAME=9
938aec0-8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","",""))
    cur = conn.cursor()
    cur.execute("SELECT userId FROM users WHERE email = ?", (email, ))
    userId = cur.fetchone()[0]
    cur.execute("SELECT products.productId, products.name, products.price,
products.image FROM products, kart WHERE products.productId =
kart.productId AND kart.userId = ?", (userId, ))
    products = cur.fetchall()
    totalPrice = 0
    for row in products:
        totalPrice += row[2]
    return     render_template("cart.html",     products     =     products,
totalPrice=totalPrice,         loggedIn=loggedIn,         username=username,
noOfItems=noOfItems)

@app.route("/removeFromCart")
def removeFromCart():
    if 'email' not in session:
        return redirect(url_for('loginForm'))
```

```
   email = session['email']
   productId = int(request.args.get('productId'))
   conn                                                                                    =
ibm_db_dbi.Connection(ibm_db.connect("DATABASE=bludb;HOSTNAME=9
938aec0-8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","",""))
   cur = conn.cursor()
   cur.execute("SELECT userId FROM users WHERE email = ?", (email, ))
   userId = cur.fetchone()[0]
   try:
         cur.execute("DELETE FROM kart WHERE userId = ? AND productId =
?", (userId, productId))
         conn.commit()
         msg = "removed successfully"
   except:
         conn.rollback()
         msg = "error occured"
   conn.close()
   return redirect(url_for('root'))


@app.route("/registerationForm")
def registrationForm():
   return render_template("register.html")

@app.route("/register", methods = ['GET', 'POST'])
def register():
    error = ''
    con    =ibm_db.connect("DATABASE=bludb;HOSTNAME=9938aec0-8105-
433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=
zlp37297;PWD=so1e9MAlFSDzhxfs","","")

    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        address = request.form['address']
        mobileNo = request.form['mobileNo']
        email = request.form['email']
        sql = "SELECT * FROM users WHERE username =?"
        stmt = ibm_db.prepare(con, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            msg = 'Account already exists !'
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = 'Invalid email address !'
        elif not re.match(r'[A-Za-z0-9]+', username):
            msg = 'name must contain only characters and numbers !'
        else:
         insert_sql = "INSERT INTO users
(username,password,address,mobileNO,email) VALUES (?, ?, ?, ?, ?)"
         prep_stmt = ibm_db.prepare(con, insert_sql)
         ibm_db.bind_param(prep_stmt, 1, username)
         ibm_db.bind_param(prep_stmt, 2, password)
         ibm_db.bind_param(prep_stmt, 3, address)
         ibm_db.bind_param(prep_stmt, 4, mobileNo)
         ibm_db.bind_param(prep_stmt, 5, email)
         ibm_db.execute(prep_stmt)
         error = 'You have successfully registered !'
         sg =
sendgrid.SendGridAPIClient('SG.SghpqcwHSC6F8CBqwCplpQ.pv-
ilF3fYOs976zRDrM57NwalJogmjWJQpgaMyEFOYg')
         from_email = Email("jayakamalesh.007@gmail.com")
         to_email = To(email)
         subject = "welcome to our smart fashion recommendation"
         content = Content("text/plain", "further updates will sended to your
mail")
         mail = Mail(from_email, to_email, subject, content)
         response = sg.client.mail.send.post(request_body=mail.get())
         print(response.status_code)
         print(response.body)
         print(response.headers)
    elif request.method == 'POST':
        error = 'Please fill out the form !'
    return render_template('login.html', error = error)
```

```
@app.route("/logout")
def logout():
    session.pop('email', None)
    return redirect(url_for('root'))


def log_done():
    print("DONE!\n")

def log_client_error(e):
    print("CLIENT ERROR: {0}\n".format(e))

def log_error(msg):
    print("UNKNOWN ERROR: {0}\n".format(msg))

def upload_large_file(bucket_name, item_name, file_path):
    print("Starting large file upload for {0} to bucket: {1}".format(item_name,
bucket_name))

    # set the chunk size to 5 MB
    part_size = 1024 * 1024 * 5

    # set threadhold to 5 MB
    file_threshold = 1024 * 1024 * 5

    # set the transfer threshold and chunk size in config settings
    transfer_config = ibm_boto3.s3.transfer.TransferConfig(
        multipart_threshold=file_threshold,
        multipart_chunksize=part_size
    )

    # create transfer manager
    transfer_mgr        =        ibm_boto3.s3.transfer.TransferManager(cos_cli,
config=transfer_config)

    try:
        # initiate file upload
        future = transfer_mgr.upload(file_path, bucket_name, item_name)

        # wait for upload to complete
        future.result()
        print ("Large file upload complete!")
    except Exception as e:
        print("Unable to complete large file upload: {0}".format(e))
    finally:
        transfer_mgr.shutdown()
COS_ENDPOINT = "https://s3.jp-tok.cloud-object-storage.appdomain.cloud"
# example: https://s3.us-south.cloud-object-storage.appdomain.cloud
COS_API_KEY_ID                                                          =
"l3fM5rXs5ycS8W4SAclnSNJPsk6MMEVuExuEoeVJPJFW"    #    example:
xxxd12V2QHXbjaM99G9tWyYDgF_0gYdlQ8aWALIQxXx4
COS_AUTH_ENDPOINT = "https://iam.cloud.ibm.com/identity/token"
COS_SERVICE_CRN            =            "crn:v1:bluemix:public:cloud-object-
storage:global:a/b9916d3d37d24674be1984ac2159a53c:da69c0d9-9adb-
46b9-b72c-582469fca7ab::" # example: crn:v1:bluemix:public:cloud-object-
storage:global:a/xx999cd94a0dda86fd8eff3191349999:9999b05b-x999-
4917-xxxx-9d5b326a1111::
COS_STORAGE_CLASS = "us-south-smart"

    # Create client connection
cos_cli = ibm_boto3.client("s3",
ibm_api_key_id=COS_API_KEY_ID,
ibm_service_instance_id=COS_SERVICE_CRN,
config=Config(signature_version="oauth"),
endpoint_url=COS_ENDPOINT
)


def parse(data):
    ans = []
    i = 0
    while i < len(data):
        curr = []
        for j in range(7):
            if i >= len(data):
                break
            curr.append(data[i])
            i += 1
        ans.append(curr)
    return ans
if __name__ == '__main__':
  app.run(host='0.0.0.0')
```

home.html

```
<!DOCTYPE HTML>
<html>
<head>
<title>smart fashion recommender</title>
<link rel="stylesheet" href={{ url_for('static', filename='css/home.css') }} />
<link rel="stylesheet" href={{ url_for('static', filename='css/topStyle.css') }} />
<script>
  window.watsonAssistantChatOptions = {
    integrationID: "c70ef7b0-391d-4712-b4c1-40bdafd3bab9", // The ID of this integration.
    region: "au-syd", // The region your integration is hosted in.
    serviceInstanceID: "c0fe8cfd-53ab-4906-b9da-116a1247d13f", // The ID of your service instance.
    onLoad: function(instance) { instance.render(); }
  };
  setTimeout(function(){
    const t=document.createElement('script');
    t.src="https://web-chat.global.assistant.watson.appdomain.cloud/versions/" + (window.watsonAssistantChatOptions.clientVersion || 'latest') + "/WatsonAssistantChatEntry.js";
    document.head.appendChild(t);
  });
</script>
</head>
<body>
<div id="title">
<a href="/">
<img id="logo" src="https://menspants.s3.jp-tok.cloud-object-storage.appdomain.cloud/Photo_1668739605187.png" />
</a>
<form>
<h2 id="searchBox">Smart Fashion Recommender</h2>
</form>

{% if not loggedIn %}
```

```
<div id="signInButton">
<button          type="button"          onclick="location.href='/loginForm'">Sign
In</a></button>
</div>
{% else %}
<div class="dropdown">
<button class="dropbtn">Hello, <br>{{username}}</button>
<div class="dropdown-content">
<a href="/logout">Sign Out</a>
</div>
</div>
{% endif %}
<div id="kart">
<a class="link" href="/cart">
<img         src={{url_for('static',         filename='images/shoppingCart.png')}}
id="cartIcon" />
CART {{noOfItems}}
</a>
</div>
</div>
<div class="display">
<div class="displayCategory">
<h2>Shop by Category: </h2>

{% for row in categoryData %}
<a href="/displayCategory?categoryId={{row[0]}}">{{row[1]}}</a>
{% endfor %}
</div>
<div>
<h2>Items</h2>
{% for data in itemData %}
<table>
<tr id="productName">
{% for row in data %}
<td>
{{row[1]}}
</td>
{% endfor %}
</tr>
<tr id="productImage">
{% for row in data %}
```

```
<td>
<a href="/productDescription?productId={{row[0]}}">
<img src='{{row[4]}}' id="itemImage" />
</a>
</td>
{% endfor %}
</tr>
<tr id="productPrice">
{% for row in data %}
<td>
₹{{row[2]}}
</td>
{% endfor %}
</tr>
</table>
{% endfor %}

</div>
</div>
</body>
</html>
```

register.html

```
<html>
<title>sign in</title>
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css" integrity="sha384-
JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGM
N5t9UJ0Z" crossorigin="anonymous">
<link rel="stylesheet" href="/static/css/style1.css">
<body>
<section class="login">
<div class="login_box">
<div class="left">
<div class="top_link"><a href="#"><img
src="https://drive.google.com/u/0/uc?id=16U__U5dJdaTfNGobB_OpwAJ7
3vM50rPV&export=download" alt="">Return home</a></div>
<div class="contact">
```

```
<form action="/register" method="post" >
<div class="msg"><center>{{ error }}</center></div>
<h3>SIGN IN</h3>
<input type="text" name="username" placeholder="Enter Your Username"
id="username"required>
<input type="text" name="password" placeholder="Enter Your Password"
id="password"required>
<input type="text" name="address" placeholder="Enter Your address"
id="address"required>
<input type="text" name="mobileNo" placeholder="Enter Your mobile
number" id="mobileNO"required>
<input type="text" name="email" placeholder="Enter Your Email ID"
id="email" required>
<button class="submit" id="button">register</button>
<div class="note mt-3 text-center">
 <!--Register form -->
<p> already have an account ? please login <a href="/login">login! </a> </p>
 </div>
</form>
</div>
</div>
<div class="right">
<div class="right-text">
<h2>smart fashion recommender</h2>
<h5>a modern fshion solution</h5>
</div>
<div class="right-inductor"><img src="" alt=""></div>
</div>
</div>
</section>
</body>
</html>
```

login.html

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css" integrity="sha384-
JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGM
N5t9UJ0Z" crossorigin="anonymous">
<link rel="stylesheet" href="/static/css/style1.css">
```

```
<body>
<section class="login">
<div class="login_box">
<div class="left">
<div                    class="top_link"><a                    href="#"><img
src="https://drive.google.com/u/0/uc?id=16U__U5dJdaTfNGobB_OpwAJ7
3vM50rPV&export=download" alt="">Return home</a></div>
<div class="contact">
<form action="/login" method="post">
<div class="msg">{{ error }}</div>
<h3>LOG IN</h3>
<input type="text"  placeholder="email" name="email" required>
<input type="text" placeholder="password" name="password" required>
<button class="submit" id="button">LET'S GO</button>
<a href="/registerationForm">Register here</a>
</form>
</div>
</div>
<div class="right">
<div class="right-text">
<h2>smart fashion recommender</h2>
<h5>a modern fshion solution</h5>
</div>
<div class="right-inductor"><p>a ibm project</p></div>
</div>
</div>
</section>
</body>
</html>
```

cart.html

```
<!DOCTYPE HTML>
<html>
<head>
<title>Your Cart</title>
<link rel="stylesheet" href="/static/css/cart.css">
<link rel="stylesheet" href="/static/css/topStyle.css">
</head>
<body>
```

```
<div id="title">
<a href="/">
<img        id="logo"        src="https://menspants.s3.jp-tok.cloud-object-
storage.appdomain.cloud/Photo_1668739605187.png"/>
</a>
<form>
<h2 id="searchBox">Smart Fashion Recommender</h2>
</form>

{% if not loggedIn %}
<div id="signInButton">
<button        type="button"        onclick="location.href='/loginForm'">Sign
In</a></button>
 </div>
{% else %}
<div class="dropdown">
<button class="dropbtn">Hello,<br>{{username}}</button>
<div class="dropdown-content">
<a href="/account/orders">Your orders</a>
<a href="/account/profile">Your profile</a>
<hr>
<a href="/logout">Sign Out</a>
</div>
</div>
{% endif %}
<div id="kart">
<a class="link" href="/cart">
<img        src={{url_for('static',        filename='images/shoppingCart.png')}}
id="cartIcon" />
CART {{noOfItems}}
</a>
</div>
</div>
<div id="cartItems">
<h2>Shopping Cart</h2>
<div id="tableItems">
{% for row in products %}
<div>
<hr id="seperator">
<div id="itemImage">
<img src='{{row[3]}}' id="image"/>
```

```
</div>
<div id="itemName">
<span id="itemNameTag">{{row[1]}}</span><br>In stock<br>
<a href="/removeFromCart?productId={{row[0]}}">Remove</a>
</div>
<div id="itemPrice">
₹{{row[2]}}
</div>
</div>
{% endfor %}
<hr id="seperator">
<div id="total">
<span id="subtotal">Subtotal</span> : ₹{{totalPrice}}
</div>
</div>
</div>
<button type="button" onclick="location.href='#popup1'">proceed to
checkout</button>
<div id="popup1" class="overlay">
<div class="popup">
<h2>order confirmation</h2>
<a class="close" href="#">&times;</a>
<div class="content">
Thanks for shopping with us your order will be delivered to you within a week
</div>
</div>
</div>
</body>
</html>
```

product description.html

```
<!DOCTYPE HTML>
<html>
<head>
<title>Product Description</title>
<link rel="stylesheet" href={{url_for('static',
filename='css/productDescription.css')}} />
<link rel="stylesheet" href={{ url_for('static', filename='css/topStyle.css')}} />
</head>
<body>
```

```
<div id="title">
<a href="/">
<img          id="logo"          src="https://menspants.s3.jp-tok.cloud-object-
storage.appdomain.cloud/Photo_1668739605187.png" />
</a>
<form>
<h2 id="searchBox">Smart Fashion Recommender</h2>
</form>

{% if not loggedIn %}
<div id="signInButton">
<button          type="button"          onclick="location.href='/loginForm'">Sign
In</a></button>
</div>
{% else %}
<div class="dropdown">
<button class="dropbtn">Hello,<br>{{username}}</button>
<div class="dropdown-content">
<a href="/account/orders">Your orders</a>
<a href="/account/profile">Your profile</a>
<hr>
<a href="/logout">Sign Out</a>
</div>
</div>
{% endif %}
<div id="kart">
<a class="link" href="/cart">
<img        src={{url_for('static',        filename='images/shoppingCart.png')}}
id="cartIcon" />
CART {{noOfItems}}
</a>
</div>
</div>
<div id="display">
<div id="productName">
<h1>{{data[1]}}</h1>
</div>
<div>
<img src='{{data[4]}}' id="productImage"/>
</div>
```

```
<div id="productDescription">
<h2>Details</h2>
<table id="descriptionTable">
<tr>
<td>Name</td>
<td>{{data[1]}}</td>
</tr>
<tr>
<td>Price</td>
<td>₹{{data[2]}}</td>
</tr>
<tr>
<td>Stock</td>
<td>{{data[5]}}</td>
</tr>
</table>
<h2>Description</h2>
<p>{{data[3]}}</p>
</div>
<div id="addToCart">
<button                                                              type="button"
onclick="location.href='/addToCart?productId={{request.args.get('productI
d')}}'">Add to cart</a></button>
</div>
</div>
</body>
</html>
```

display category.html

```
<!DOCTYPE HTML>
<html>
<head>
<title>Category: {{categoryName}}</title>
<link rel="stylesheet" href={{ url_for('static', filename='css/home.css') }} />
<link rel="stylesheet" href={{ url_for('static', filename='css/topStyle.css') }} />
</head>
<body>
<div id="title">
<a href="/">
<img id="logo" src= {{ url_for('static', filename='images/logo.png') }
```

```
<form>
<input id="searchBox" type="text" name="searchQuery">
<input id="searchButton" type="submit" value="Search">
</form>

{% if not loggedIn %}
<div id="signInButton">
<a class="link" href="/loginForm">Sign In</a>
</div>
{% else %}
<div class="dropdown">
<button class="dropbtn">Hello,<br>{{firstName}}</button>
<div class="dropdown-content">
<a href="/account/orders">Your orders</a>
<a href="/account/profile">Your profile</a>
<hr>
<a href="/logout">Sign Out</a>
</div>
</div>
{% endif %}
<div id="kart">
<a class="link" href="/cart">
<img      src={{url_for('static',      filename='images/shoppingCart.png')}}
id="cartIcon" />
CART {{noOfItems}}
</a>
</div>
</div>

<div>
<h2>Showing all products of Category {{categoryName}}:</h2>
{% for itemData in data %}
<table>
<tr id="productName">
{% for row in itemData %}
<td>
{{row[1]}}
</td>
{% endfor %}
</tr>
<tr id="productImage">
```

```
{% for row in itemData %}
<td>
<a href="/productDescription?productId={{row[0]}}">
<img src={{ url_for('static', filename='uploads/' + row[3]) }} id="itemImage" />
</a>
</td>
{% endfor %}
</tr>
<tr id="productPrice">
{% for row in itemData %}
<td>
${{row[2]}}
</td>
{% endfor %}
</tr>
</table>
{% endfor %}
</div>
</body>
</html>
```

add.html

```
<!DOCTYPE HTML>
<html>
<head>
<title>Admin</title>
</head>
<body>
<h2>Add items</h2>
<form action="/addItem" method="POST" enctype="multipart/form-data">
Name: <input type="text" name="name"><br>
Price: <input type="text" name="price"><br>
Description: <textarea name="description" rows=3
cols="40"></textarea><br>
Image: <input type="file" name="image"><br>
Stock: <input type="text" name="stock"><br>
Category: <select name="category">
{% for row in categories %}
<option value="{{row[0]}}">{{row[1]}}</option>
```

```
{% endfor %}
</select><br>
<input type="submit">
</form>
</body>
</html>
```

13.2 GitHub & Project Demo Link

- ☆ Our GitHub Repository Link

https://github.com/IBM-EPBL/IBM-Project-14049-1659539603

- ☆ Project Demostration Video Link

https://www.youtube.com/watch?v=WorphEkk3hg