RMK ENGINEERING COLLEGE

(An Autonomous Institution)

R.S.M. Nagar, Kavaraipettai-601 206, Gummidipoondi Taluk, Thiruvallur District.

PROJECT

SMART FASHION RECOMMENDER APPLICATION

DONE BY

TEAM ID: PNT2022TMID15761

MRITTHULA .B (111719104097)

PAMURU YAMINI (111719104109)

NEYA. A (111719104104)

NEENA SRI .S (111719104103)

TABLE OF CONTENT

1.INTRODUCTION

1.1 Project Overview :

Nowadays, fashion applications and e-commerce are growing more and more . And it also has some problems when finding the customer's wanted product in the web applications. Having a chatbot that understands the algorithm of a specific application can be of great aid. We are implementing such a chat bot in a web application, which is fed with the knowledge of the application's algorithm and helps the user completely from finding their needs to processing the payment and initiating delivery. It works as an advanced filter search that can bring the user what they want with the help of pictorial and named representation by getting simple user information and activities. The application also has two main UI interactions: one is the user panel and the other one is the admin panel. Users can interact with the chat bot to search for products, order them from the manufacturer or distributor through chatbot AI, and it can also make payment transactions, track the delivery, and so on. The admin interface enables the user to upload products details , user details, orders and find how many products have been bought; supervise the stock availability; and interact with the buyer regarding the product reviews.

1.2 Purpose :

Unlike other areas, fashion recommendations shouldn't be based solely on personal taste and past activity of the customer. There are many external factors (many of which are emotional) that make creating a fashion recommendation system all the more complex. Public perceptions must be taken into account, as well as fashion rules, dress rules and current trends.

2.LITERATURE SURVEY

2.1 Existing problem :

In traditional e-commerce websites, the users need to search for their required product using a search bar or go through the whole effects of their search. It will take a lot of users' time and it will create a lot of flawed user experiments. This approach will create bad marketing for the product. Later when the user comes again to purchase the product it will create a bad impression on the user. Even though the product is good the user will not buy the product.

This type of search will create miss matched products when the product has a different name. Let's say we search for oranges on amazon. Sometimes it will show orange color or sometimes it will show orange fruit. In recent times, fashion systems have been integrated with artificial intelligence and deep learning. These approaches provide a rich recommendation, but in most cases, it is prone to product mismatch. Even Though the recommender system recommends products based on the user's preference, this system lacks a chat bot that improves user experience by interacting with users. In most fashion systems, the user needs to navigate across multiple products to find the appropriate product. The users are made to filter products based on the long list of categories present in the system. If this system is integrated with an intelligent bot, it would be able to list out only required categories.

2.2 References:

Flask tutorial: https://www.youtube.com/watch?v=MwZwr5Tvyxo&list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH

ElasticSearch tutorial: https://medium.com/naukri-engineering/elasticsearch-tutorial-for-beginners-using-python-b9cb48edcedc

ChatterBot tutorial: https://chatterbot.readthedocs.io/en/latest/

Template: https://colorlib.com

Photos: https://unsplash.com/

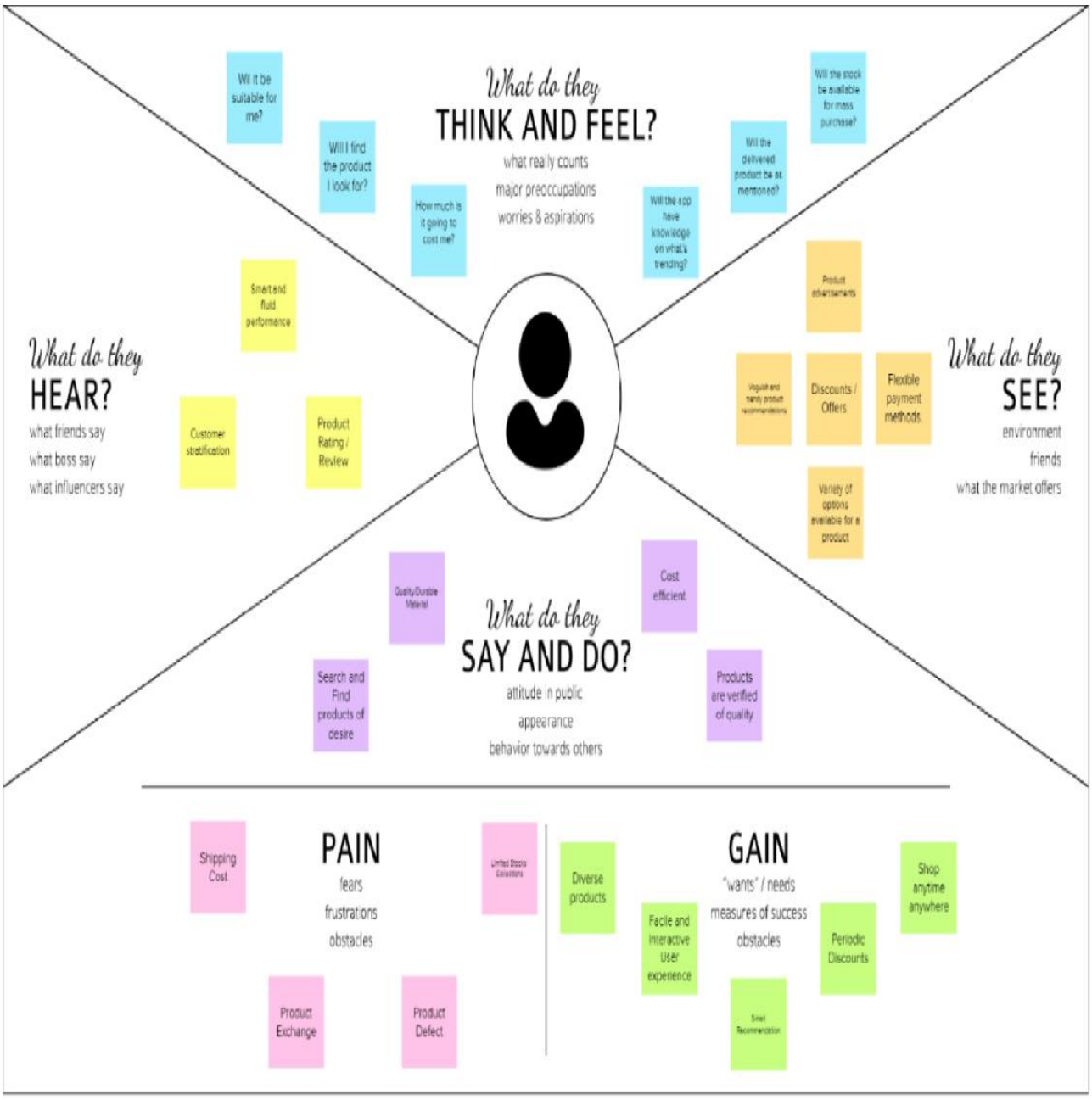
2.3 Problem Statement Definition:

In E-commerce websites, users need to search for products and navigate across screens to view the product, add them to the cart, and order products. The smart fashion recommender application leverages the use of a chat bot to interact with the users, gather information about their preferences, and recommend suitable products to the users. This application has two predefined roles assigned to the users. The roles are customer and admin. The application demands redirection of the user to the appropriate dashboard based on the assigned role. Admin should be able to track the number of different products and admin

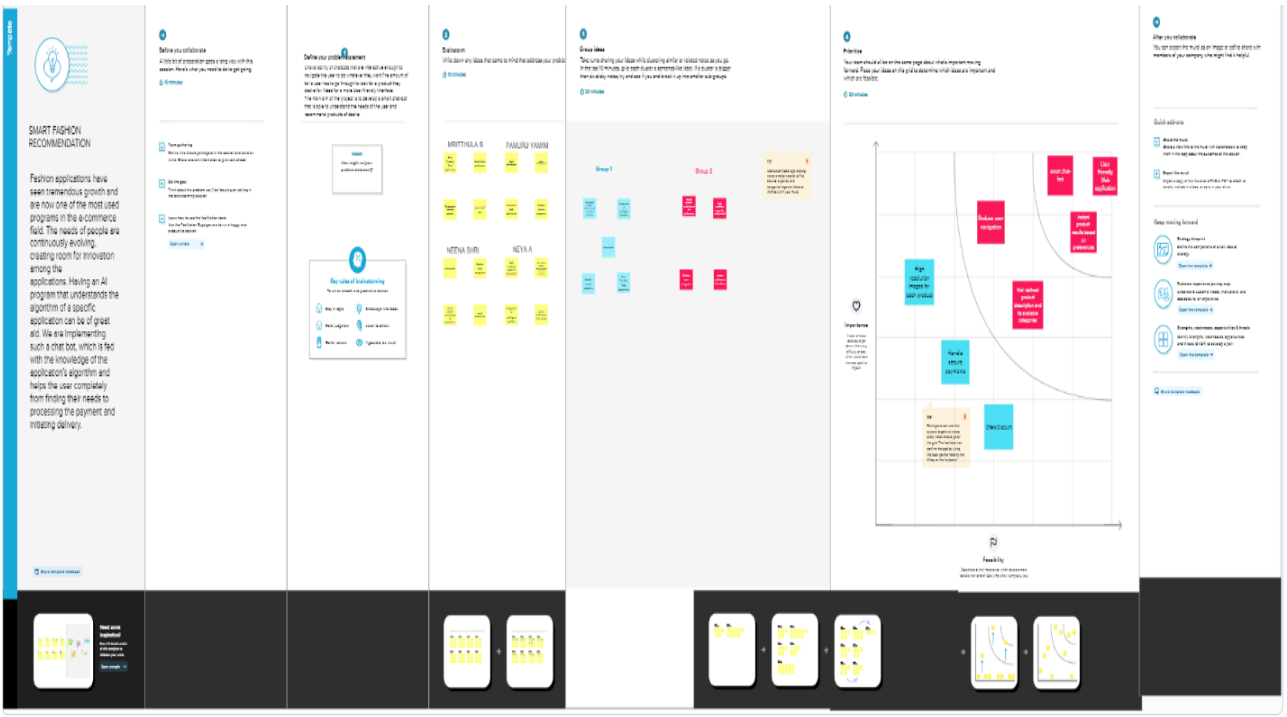
should be assigned the responsibility to create products with appropriate categories. The user should be able to mention their preferences using interacting with chat bots. The user must receive a notification on order confirmation/failure. The chat bot must gather feedback from the user at the end of order confirmation. The main objective of this application is to provide better interactivity with the user and to reduce navigating pages to find appropriate products.

3.IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming



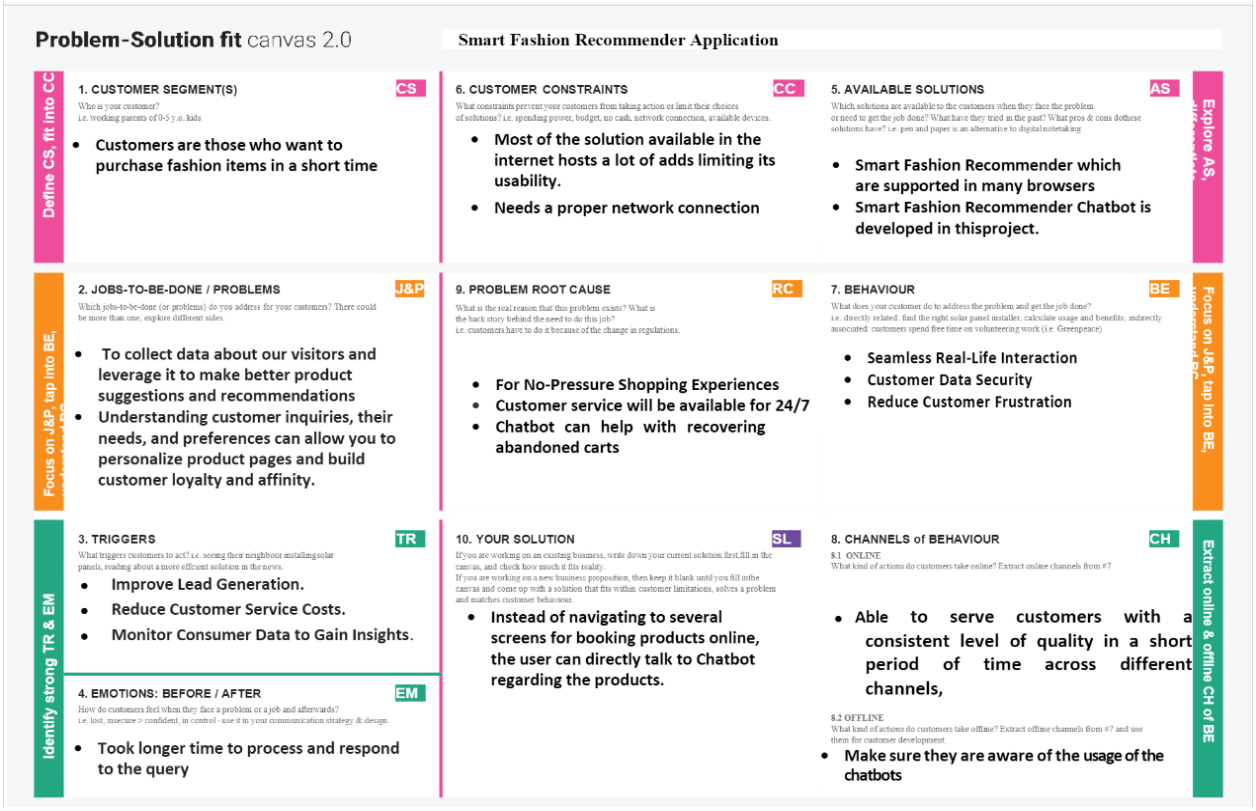
3.3 Proposed Solution

S.No.		Parameter	Description
1.		Problem Statement (Problem to be solved)	In normal Online Shopping: • Poor customer service due to lack of service teams. • There are many human error can occur. • Negative feedback from customer • There is no sufficient way to guide the customers properly. • Lack of sales. • Poor relationship between customer

2.		Idea / Solution description	By using Smart fashion recommender application: • Improve the effectiveness of customer service teams. • Reduce the potential for human error. • Collect candid and meaningful customer feedback. • Guide customers along the path to purchase. • Turns leads into sales (aka, boost conversion). Build stronger customer relationships
3.		Novelty / Uniqueness	• Improve your call/chat containment rate with the latest BERT-based natural language understanding (NLU) models that are capable of recognizing intent and context accurately and efficiently in more complex use cases.
4.		Social Impact / Customer Satisfaction	• Complaint handling might be one of the most critical factors to gain customer experience, from minor to big complaints. It is crucial to handle complaints as fast as possible to keep the customer satisfied.
5.		Business Model (Revenue Model)	• This application can be developed with minimum cost and at the same time it can provide high performance
6.		Scalability of the Solution	• This can be developed to a scalable product by using sensors and transmitting the data through Wireless Sensor Network and Analysing the data

			in cloud and operation is performed using chat bots.

3.4 Problem Solution fit



4. REQUIREMENT ANALYSIS

2.3 Functional requirement

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	Registration	Registration can be done using mobile number or Gmail and needed some user Information
FR-2	Login	User only log in by user id and password, which is given during registration
FR-3	Delivery confirmation	Confirmation via email and phone number

FR-4	Assistance	Bot is integrated with the application to make the usability simple

2.4 Non-Functional requirements

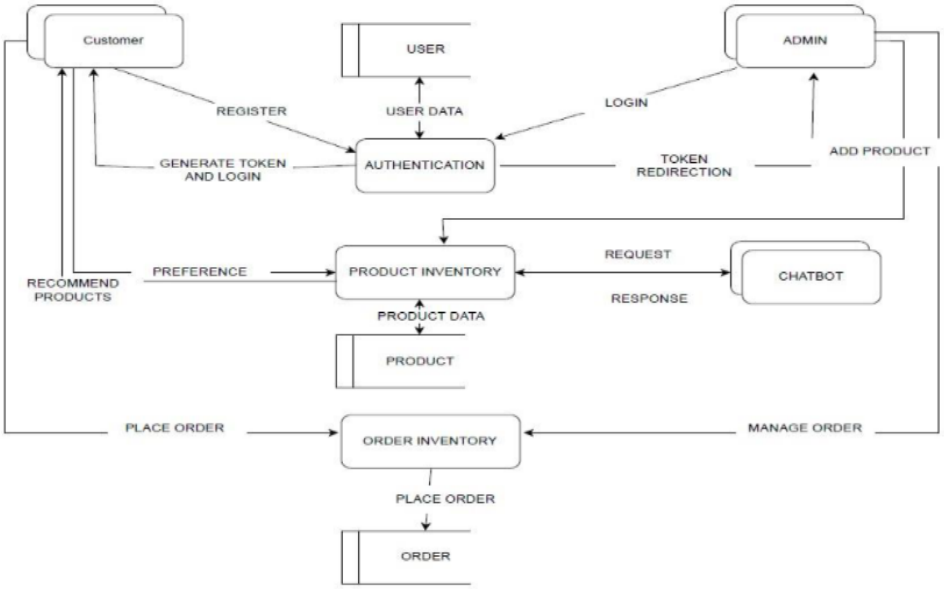
Following are the non-functional requirements of the proposed solution.

NFR-4	Performance	The system shall be able to handle multiple requests at any given point in time and generate an appropriate response.

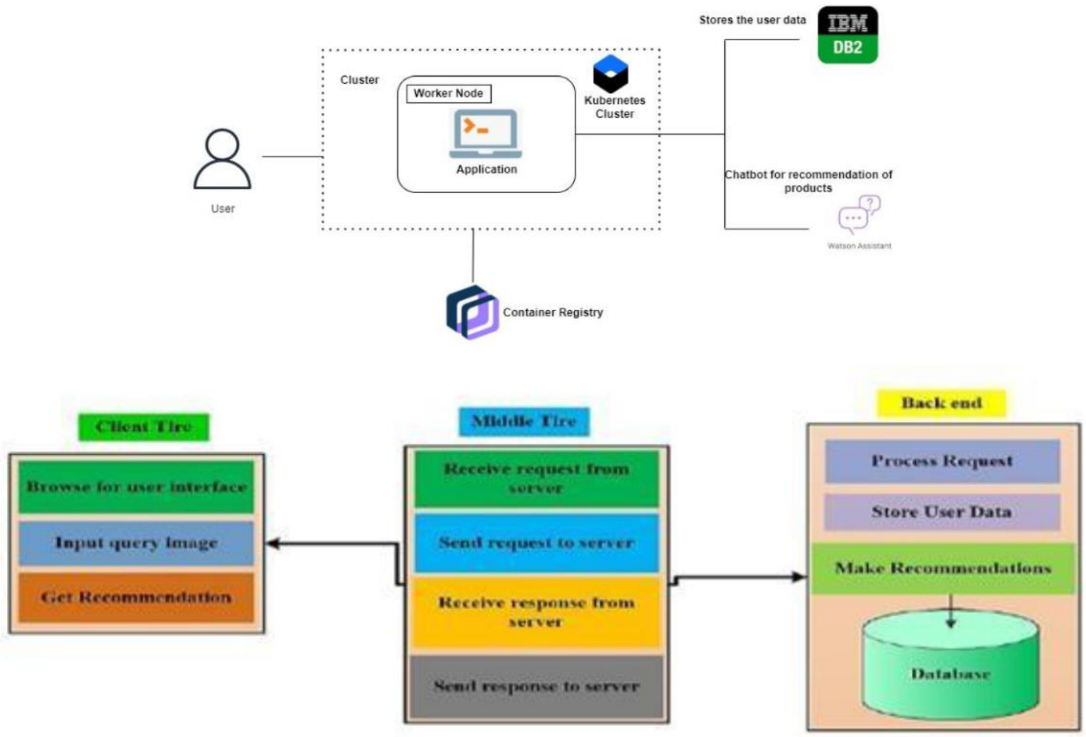
5.PROJECT DESIGN

5.1 Data Flow Diagrams:

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	A user-friendly interface with chat bot to make usability efficient
NFR-2	**Security**	Secured connection HTTPS should be established for transmitting requests and responses.
NFR-3	**Reliability**	The system should handle excepted as well as unexpected errors and exceptions to avoid termination of the program



5.2 Solution & Technical Architecture:



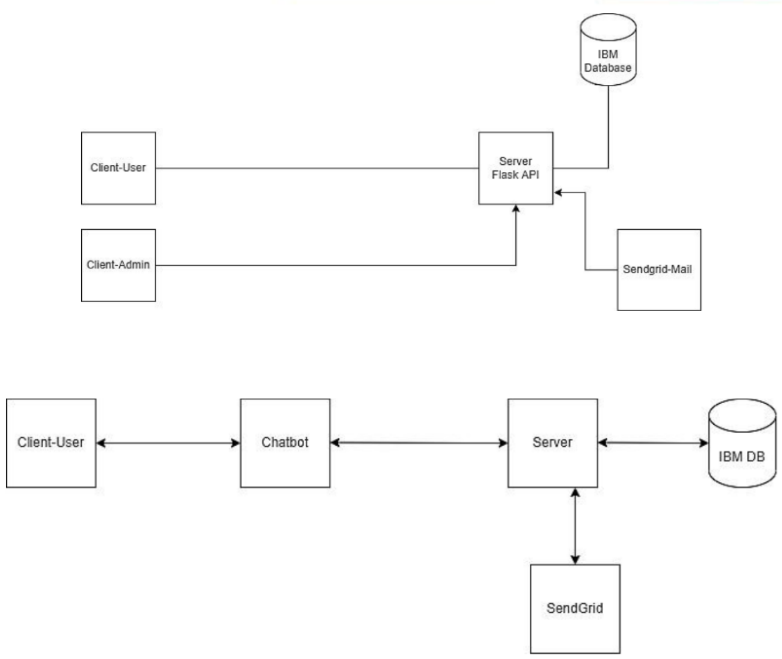
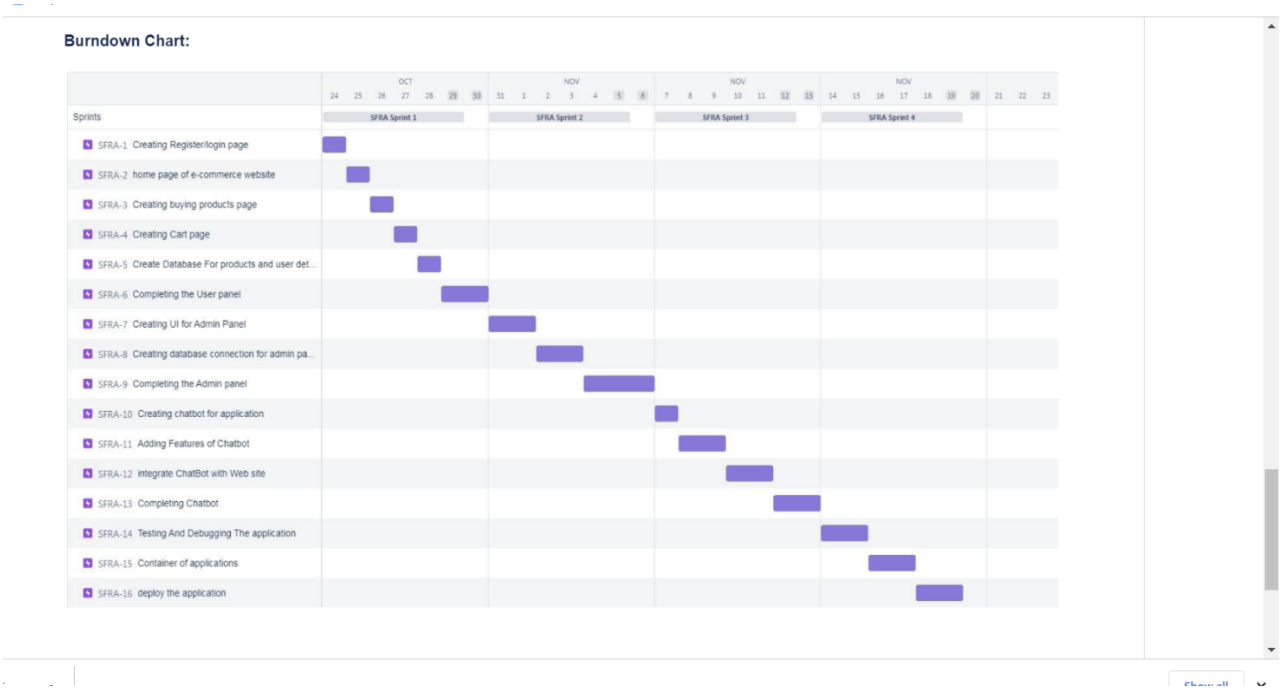


Figure 1: Architecture and data flow of the smart fashion recommendation system

6 PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation & Sprint Delivery Schedule & Reports from JIRA



Project Planning Phase
Project Planning (Product Backlog, Sprint Planning, Stories, Story points)

Date	30 October 2022
Team ID	PNT2022TMID15761
Project Name	Project -Smart Fashion Recommender Application
Maximum Marks	8 Marks

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Use the below template to create product backlog and sprint schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	User Panel	USN-1	The user will login into the website and go through the products available on the website	20	High	Mritthula B Pamuru Yamini Neena sri Neya A
Sprint-2	Admin panel	USN-2	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.	20	High	Mritthula B Pamuru Yamini Neena sri Neya A
Sprint-3	Chat Bot	USN-3	The user can directly talk to Chatbot regarding the products. Get the recommendations based on information provided by the user.	20	High	Mritthula B Pamuru Yamini Neena sri Neya A
Sprint-4	final delivery	USN-4	Container of applications using docker kubernetes and deployment the application. Create the documentation and final submit the application	20	High	Mritthula B Pamuru Yamini Neena sri Neya A

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

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

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

Velocity:

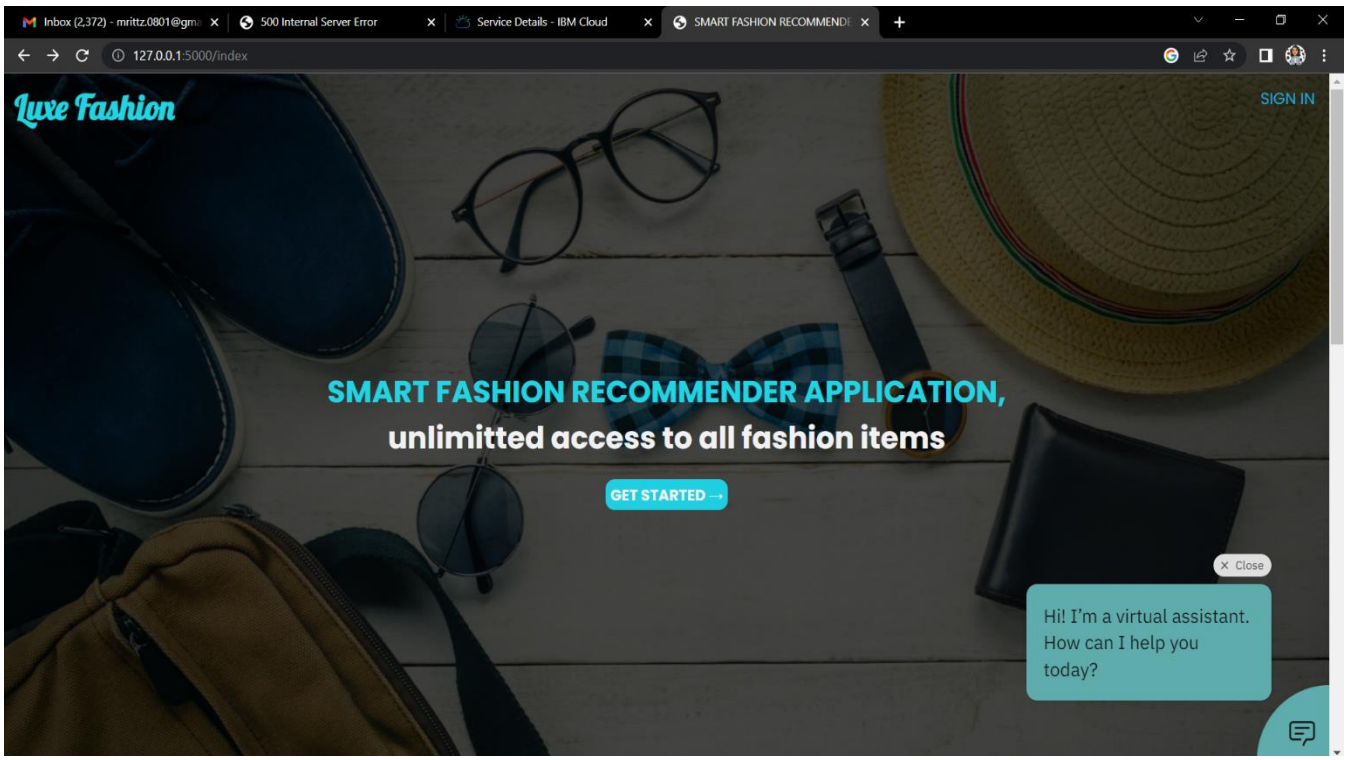
Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

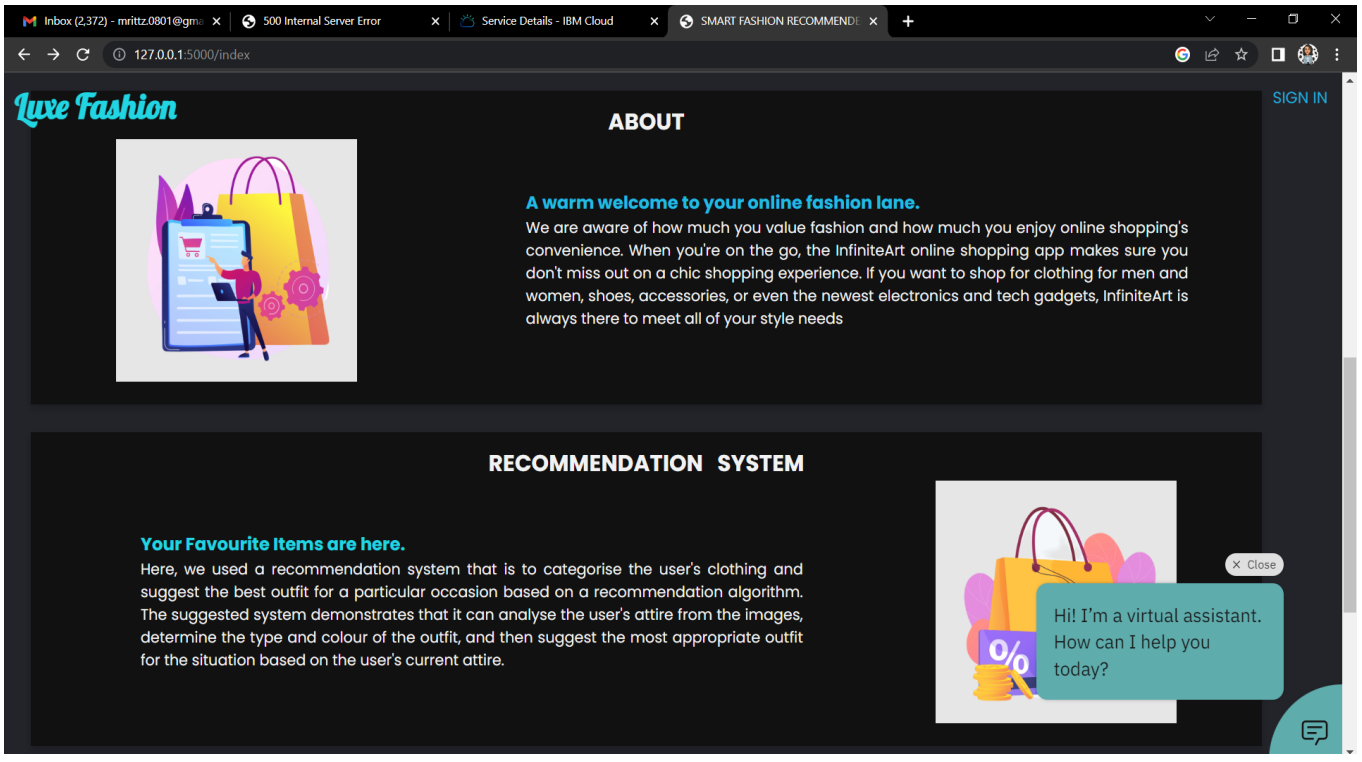
7 TESTING

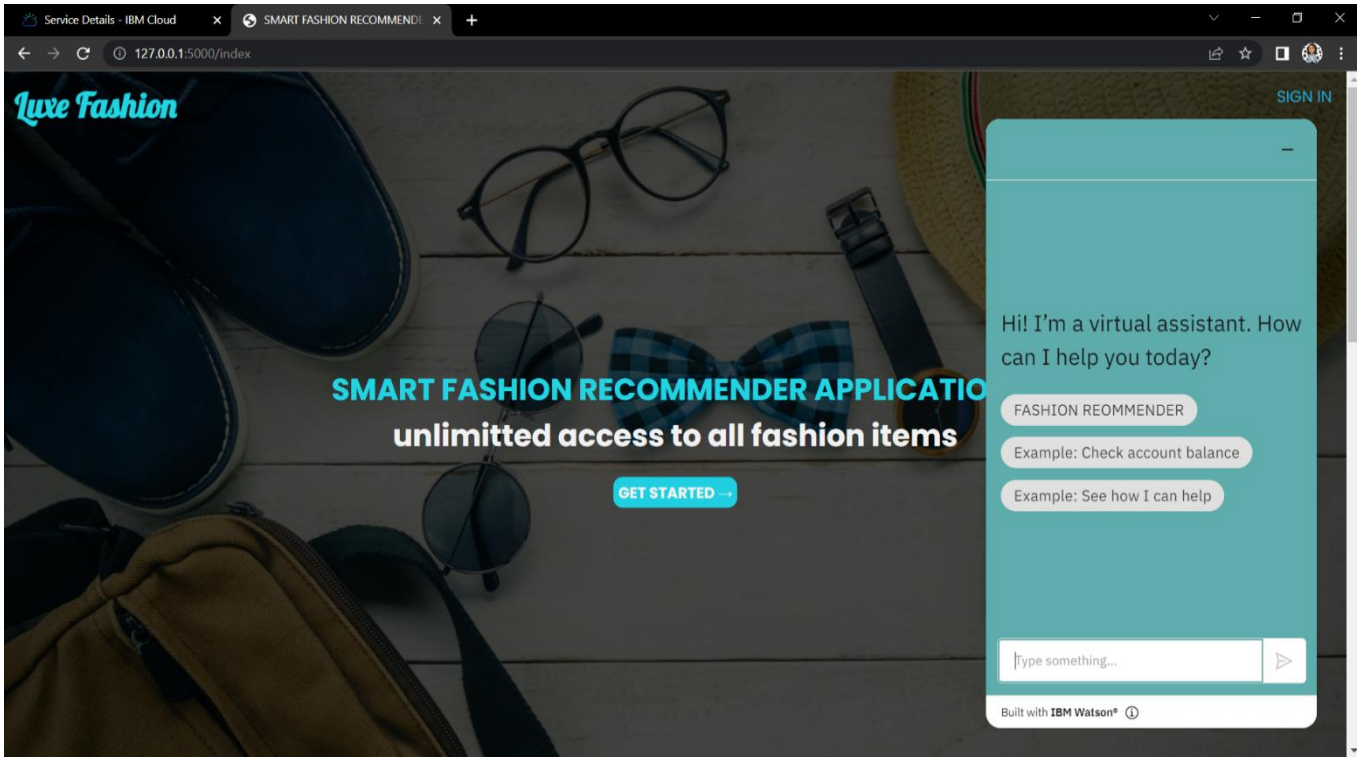
7.1 Test Cases

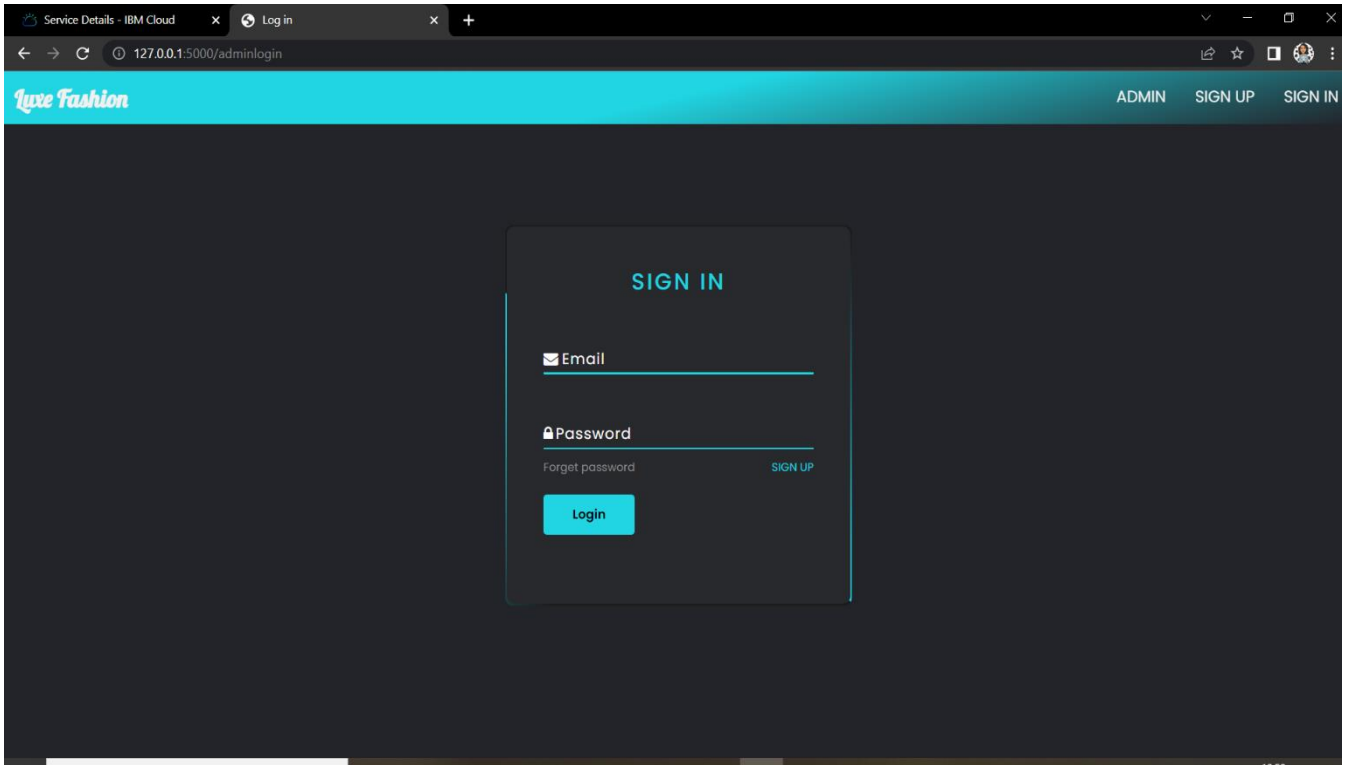
Test Case Id	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
TC-1	Install PET app in android phone	Transfer PET app	Open Application with it home page	Application executed with home page	Pass
TC-2	Enter valid data in username and password field	neya	Show home page for user neya	Displayed home page for user neya	Pass
TC-3	Enter a valid data in isername and leave password field empty	neya	Show error	Didn't show any error	Fail

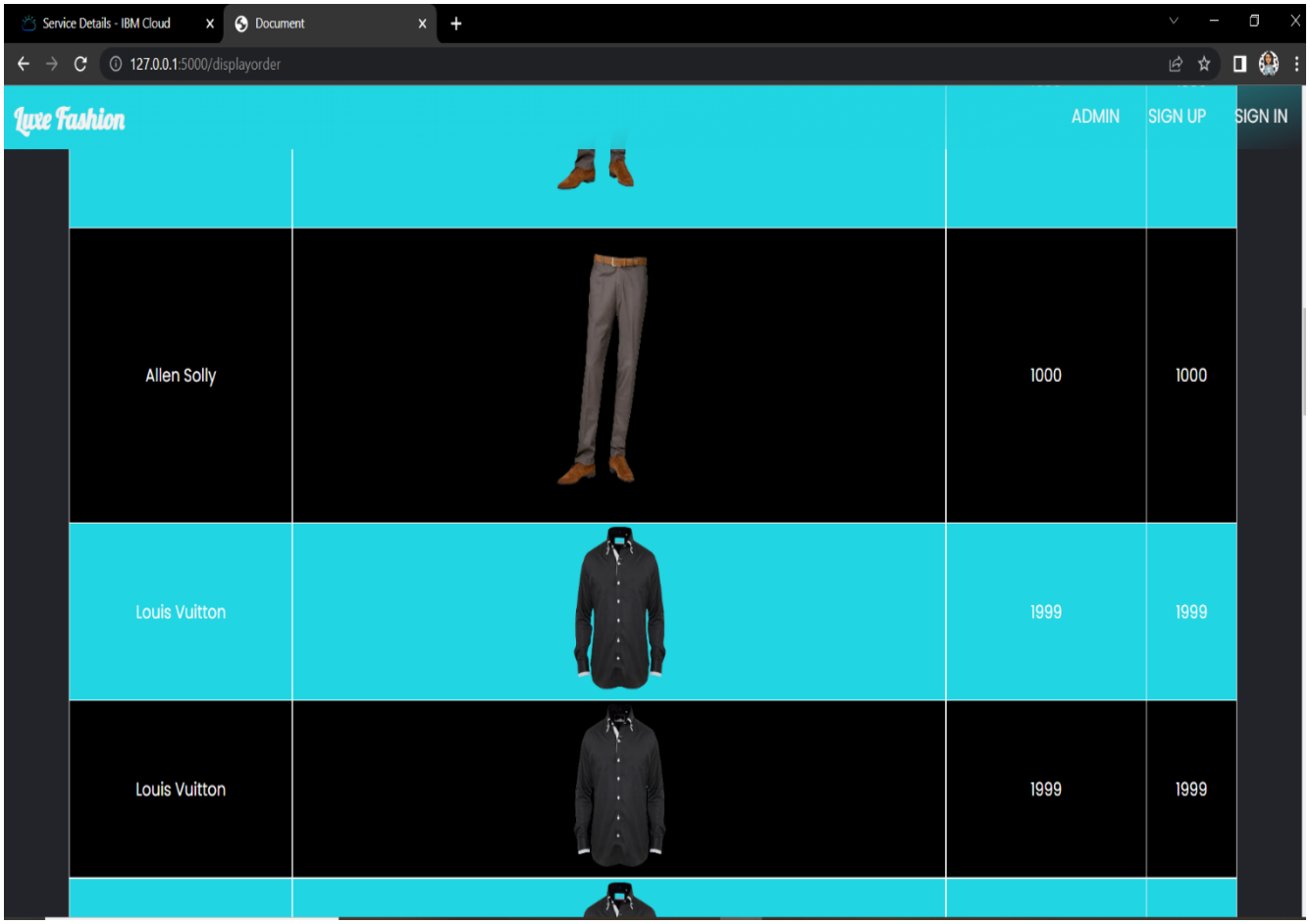
8 RESULTS


Inbox (2,372) - mrittz.0801@gm
500 Internal Server Error
Service Details - IBM Cloud
SMART FASHION RECOMMENDE
127.0.0.1:5000/index
Luxe Fashion
SIGN IN
SMART FASHION RECOMMENDER APPLICATION,
unlimitted access to all fashion items
GET STARTED
Close
Hi! I'm a virtual assistant. How can I help you today?


Inbox (2,372) - mrittz.0801@gm
500 Internal Server Error
Service Details - IBM Cloud
SMART FASHION RECOMMENDE
127.0.0.1:5000/index
Luxe Fashion
SIGN IN
ABOUT
A warm welcome to your online fashion lane.
We are aware of how much you value fashion and how much you enjoy online shopping's convenience. When you're on the go, the InfiniteArt online shopping app makes sure you don't miss out on a chic shopping experience. If you want to shop for clothing for men and women, shoes, accessories, or even the newest electronics and tech gadgets, InfiniteArt is always there to meet all of your style needs
RECOMMENDATION SYSTEM
Your Favourite Items are here.
Here, we used a recommendation system that is to categorise the user's clothing and suggest the best outfit for a particular occasion based on a recommendation algorithm. The suggested system demonstrates that it can analyse the user's attire from the images, determine the type and colour of the outfit, and then suggest the most appropriate outfit for the situation based on the user's current attire.
Close
Hi! I'm a virtual assistant. How can I help you today?


Service Details - IBM Cloud
SMART FASHION RECOMMENDE
127.0.0.1:5000/index
Luxe Fashion
SIGN IN
SMART FASHION RECOMMENDER APPLICATIO
unlimitted access to all fashion items
GET STARTED
Hi! I'm a virtual assistant. How can I help you today?
FASHION REOMMENDER
Example: Check account balance
Example: See how I can help
Type something...
Built with IBM Watson®


Service Details - IBM Cloud
Log in
127.0.0.1:5000/adminlogin
Luxe Fashion
ADMIN
SIGN UP
SIGN IN
SIGN IN
Email
Password
Forget password
SIGN UP
Login



9 ADVANTAGES & DISADVANTAGES

ADVANTAGES:

1. CUSTOMER SATISFACTION:

Many a time customers tend to look at their product recommendation from their last browsing. Mainly because they think they will find better opportunities for good products. When they leave the site and come back later; it would help if their browsing data from the previous session was available. This could further help and guide their e-Commerce activities, similar to experienced assistants at Brick and Mortar stores. This type of customer satisfaction leads to customer retention.

2.REVENUE:

With years of research, experiments and execution primarily driven by Amazon, not only is there less of a learning curve for online customers today. Many different algorithms have also been explored, executed, and proven to drive high conversion rate vs. non-personalized product recommendations.

DISADVANTAGES:

1. LACK OF DATA

Perhaps the biggest issue facing recommender systems is that they need a lot of data to effectively make recommendations. It's no coincidence that the companies most identified with having excellent recommendations are those with a lot of consumer user data: Google, Amazon, Netflix, Last.fm. As illustrated in the slide below from Strands' presentation at Recked, a good recommender system firstly needs item data (from a catalog or other form), then it must capture and analyze user data (behavioral events), and then the magic algorithm does its work. The more item and user data a recommender system has to work with, the stronger the chances of getting good recommendations. But it can be a chicken and egg problem – to get good recommendations, you need a lot of users, so you can get a lot of data for the recommendations.

10 CONCLUSION

The smart fashion recommender system uses a chat bot as a primary mechanism to interact with users, collect user interest and recommend products periodically. A chat bot is designed to improve user experience by interacting with users. Users need not navigate between multiple pages to find an appropriate product. The system is designed to minimize the efforts taken by customers to search for the required product. The future enhancements of the chat bot include adding products to the cart, displaying cart items, order history, and payment through the chat bot.

11 APPENDIX

Source Code

```
import secrets
from turtle import title
from unicodedata import category
from flask import Flask, render_template, request, redirect, url_for, session
import ibm_db
import bcrypt
import base64
import os

conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=2f3279a5-73d1-4859-88f0-
a6c3e6b4b907.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud;PORT=30756;SECURITY=SSL;
SSLServerCertificateDigiCertGlobalRootCA.crt;PROTOCOL=TCPIP;UID=nhl80748;PWD=3yD0G9e6VuQHsOBX;",
"", "")
#url_for('static', filename='style.css')



app = Flask(_name_)
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'

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

def home():
    if 'email' not in session:
      return redirect(url_for('index'))
    shirt_list=[]
    pant_list=[]
    watch_list=[]
    ring_list=[]

    #selecting_shirt
    sql = "SELECT * FROM SHIRT"
    stmt = ibm_db.exec_immediate(conn, sql)
    shirt = ibm_db.fetch_both(stmt)
    while shirt != False :
        shirt_list.append(shirt)
        shirt = ibm_db.fetch_both(stmt)
    print(shirt_list)
```

```
  #selecting_pant

    sql1="SELECT * FROM PANT"
    stmt1 = ibm_db.exec_immediate(conn, sql1)
    pant=ibm_db.fetch_both(stmt1)
    while pant != False :
        pant_list.append(pant)
        pant = ibm_db.fetch_both(stmt1)
    print(pant_list)

  #selecting_watch
    sql2="SELECT * FROM WATCH"
    stmt2 = ibm_db.exec_immediate(conn, sql2)
    watch=ibm_db.fetch_both(stmt2)
    while watch != False :
        watch_list.append(watch)
        watch = ibm_db.fetch_both(stmt2)
    print(watch_list)

    #selecting_rings
    sql3="SELECT * FROM RINGS"
    stmt3 = ibm_db.exec_immediate(conn, sql3)
    ring=ibm_db.fetch_both(stmt3)
    while ring != False :
        ring_list.append(ring)
        ring = ibm_db.fetch_both(stmt3)
    print(ring_list)
    #returning to HTML
    return render_template('home.html',dictionary= shirt_list,pants=pant_list,watchs=watch_list,rings=ring_list)

@app.route("/index")
def index():
  return render_template('index.html')

@app.route("/register",methods=['GET','POST'])
def register():
  if request.method == 'POST':
    username = request.form['username']
    email = request.form['email']
    phoneno = request.form['phoneno']
    password = request.form['password']

    if not username or not email or not phoneno or not password:
```

```
      return render_template('register.html',error='Please fill all fields')
    hash=bcrypt.hashpw(password.encode('utf-8'),bcrypt.gensalt())
    query = "SELECT * FROM user_detail WHERE email=? OR phoneno=?"
    stmt = ibm_db.prepare(conn, query)
    # ibm_db.bind_param(stmt,1,email)
    # ibm_db.bind_param(stmt,2,phoneno)
    # ibm_db.execute(stmt)
    isUser = ibm_db.fetch_assoc(stmt)
    # if not isUser:
    insert_sql = "INSERT INTO user_detail(username, email, phoneno, password) VALUES (?,?,?,?)"
    prep_stmt = ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(prep_stmt, 1, username)
    ibm_db.bind_param(prep_stmt, 2, email)
    ibm_db.bind_param(prep_stmt, 3, phoneno)
    ibm_db.bind_param(prep_stmt, 4, hash)
    ibm_db.execute(prep_stmt)
    return render_template('register.html',success="You can login")
    # else:
    #   return render_template('register.html',error='Invalid Credentials')

  return render_template('register.html',name='Home')

@app.route("/login",methods=['GET','POST'])
def login():
    if request.method == 'POST':
      email = request.form['email']
      password = request.form['password']

      if not email or not password:
        return render_template('login.html',error='Please fill all fields')
      query = "SELECT * FROM user_detail WHERE email=?"
      stmt = ibm_db.prepare(conn, query)
      ibm_db.bind_param(stmt,1,email)
      ibm_db.execute(stmt)
      isUser = ibm_db.fetch_assoc(stmt)
      print(isUser,password)

      if not isUser:
        return render_template('login.html',error='Invalid Credentials')
      print("flag1")
      isPasswordMatch = bcrypt.checkpw(password.encode('utf-8'),isUser['PASSWORD'].encode('utf-8'))
      print("flag2")
      if not isPasswordMatch:
```

```
        return render_template('login.html',error='Invalid Credentials')
      print("flag3")
      session['email'] = isUser['EMAIL']
      print("flag4")
      return redirect(url_for('home') )
    print("flag5")
    return render_template('login.html',name='Home')

@app.route("/admin",methods=['GET','POST'])
def adregister():
  if request.method == 'POST':
    username = request.form['username']
    email = request.form['email']
    phoneno = request.form['phoneno']
    password = request.form['password']

    if not username or not email or not phoneno or not password:
      return render_template('adminregister.html',error='Please fill all fields')
    hash=bcrypt.hashpw(password.encode('utf-8'),bcrypt.gensalt())
    query = "SELECT * FROM admin_detail WHERE email=? OR phoneno=?"
    stmt = ibm_db.prepare(conn, query)
    ibm_db.bind_param(stmt,1,email)
    ibm_db.bind_param(stmt,2,phoneno)
    ibm_db.execute(stmt)
    isUser = ibm_db.fetch_assoc(stmt)
    if not isUser:
      insert_sql = "INSERT INTO admin_detail(username, email, phoneno, password) VALUES (?,?,?,?)"
      prep_stmt = ibm_db.prepare(conn, insert_sql)
      ibm_db.bind_param(prep_stmt, 1, username)
      ibm_db.bind_param(prep_stmt, 2, email)
      ibm_db.bind_param(prep_stmt, 3, phoneno)
      ibm_db.bind_param(prep_stmt, 4, hash)
      ibm_db.execute(prep_stmt)
      return render_template('adminregister.html',success="You can login")
    else:
      return render_template('adminregister.html',error='Invalid Credentials')

  return render_template('adminregister.html',name='Home')

@app.route("/adminlogin",methods=['GET','POST'])
def adlogin():
    if request.method == 'POST':
      email = request.form['email']
```

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

        if not email or not password:
          return render_template('adminlogin.html',error='Please fill all fields')
        query = "SELECT * FROM admin_detail WHERE email=?"
        stmt = ibm_db.prepare(conn, query)
        ibm_db.bind_param(stmt,1,email)
        ibm_db.execute(stmt)
        isUser = ibm_db.fetch_assoc(stmt)
        print(isUser,password)

        if not isUser:
          return render_template('adminlogin.html',error='Invalid Credentials')

        isPasswordMatch = bcrypt.checkpw(password.encode('utf-8'),isUser['PASSWORD'].encode('utf-8'))

        if not isPasswordMatch:
          return render_template('adminlogin.html',error='Invalid Credentials')

        session['email'] = isUser['EMAIL']
        return redirect(url_for('addproduct'))

    return render_template('adminlogin.html',name='Home')

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

def addproduct():
  if request.method == 'POST':
    types=request.form['cc']
    name = request.form['name']
    image = request.form['image']
    rate = request.form['rate']
    categorie = request.form['categorie']
    if types =='shirt':
      insert_sql = "INSERT INTO SHIRT(name, image, categorie,rate) VALUES (?,?,?,?)"
      prep_stmt = ibm_db.prepare(conn, insert_sql)
      ibm_db.bind_param(prep_stmt, 1, name)
      ibm_db.bind_param(prep_stmt, 2, image)
      ibm_db.bind_param(prep_stmt, 3, categorie)
      ibm_db.bind_param(prep_stmt, 4, rate)
      ibm_db.execute(prep_stmt)
    if types =='pant':
      insert_sql = "INSERT INTO SHIRT(name, image, categorie,rate) VALUES (?,?,?,?)"
```

```
        prep_stmt = ibm_db.prepare(conn, insert_sql)
        ibm_db.bind_param(prep_stmt, 1, name)
        ibm_db.bind_param(prep_stmt, 2, image)
        ibm_db.bind_param(prep_stmt, 3, categorie)
        ibm_db.bind_param(prep_stmt, 4, rate)
        ibm_db.execute(prep_stmt)
      if types =='watch':
        insert_sql = "INSERT INTO WATCH(name, image, rate) VALUES (?,?,?)"
        prep_stmt = ibm_db.prepare(conn, insert_sql)
        ibm_db.bind_param(prep_stmt, 1, name)
        ibm_db.bind_param(prep_stmt, 2, image)
        ibm_db.bind_param(prep_stmt, 3, rate)
        ibm_db.execute(prep_stmt)
      if types =='ring':
        insert_sql = "INSERT INTO RINGS(name, image, categorie,rate) VALUES (?,?,?,?)"
        prep_stmt = ibm_db.prepare(conn, insert_sql)
        ibm_db.bind_param(prep_stmt, 1, name)
        ibm_db.bind_param(prep_stmt, 2, image)
        ibm_db.bind_param(prep_stmt, 3, categorie)
        ibm_db.bind_param(prep_stmt, 4, rate)
        ibm_db.execute(prep_stmt)

    return render_template('addproduct.html',success="You can login")

@app.route("/data")
def display():
    shirt_list=[]
    pant_list=[]
    watch_list=[]
    ring_list=[]

    #selecting_shirt
    sql = "SELECT * FROM SHIRT"
    stmt = ibm_db.exec_immediate(conn, sql)
    shirt = ibm_db.fetch_both(stmt)
    while shirt != False :
        shirt_list.append(shirt)
        shirt = ibm_db.fetch_both(stmt)
    print(shirt_list)

  #selecting_pant

    sql1="SELECT * FROM PANT"
```

```
  stmt1 = ibm_db.exec_immediate(conn, sql1)
  pant=ibm_db.fetch_both(stmt1)
  while pant != False :
     pant_list.append(pant)
     pant = ibm_db.fetch_both(stmt1)
  print(pant_list)

#selecting_watch
  sql2="SELECT * FROM WATCH"
  stmt2 = ibm_db.exec_immediate(conn, sql2)
  watch=ibm_db.fetch_both(stmt2)
  while watch != False :
     watch_list.append(watch)
     watch = ibm_db.fetch_both(stmt2)
  print(watch_list)

  #selecting_rings
  sql3="SELECT * FROM RINGS"
  stmt3 = ibm_db.exec_immediate(conn, sql3)
  ring=ibm_db.fetch_both(stmt3)
  while ring != False :
     ring_list.append(ring)
     ring = ibm_db.fetch_both(stmt3)
  print(ring_list)
  #returning to HTML
  return render_template('home.html',dictionary= shirt_list,pants=pant_list,watchs=watch_list,rings=ring_list)

@app.route("/orderplaced",methods=['GET','POST'])
def dis():
  if request.method == 'POST':
    pname=request.form['name']
    img=request.form['image']
    rate=request.form['rate']
    categorie=request.form['categorie']
  return render_template('order.html',pname=pname,img=img,rate=rate,categorie=categorie)

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

def orderdisplay():
  if request.method == 'POST':
    name = request.form['order_name']
    image = request.form['order_image']
    rate = request.form['order_rate']
```

```
    categorie = request.form['order_categorie']
    insert_sql = "INSERT INTO ORDERS(oname, oimage,orate, ocategorie) VALUES (?,?,?,?)"
    prep_stmt = ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(prep_stmt, 1, name)
    ibm_db.bind_param(prep_stmt, 2, image)
    ibm_db.bind_param(prep_stmt, 3, rate)
    ibm_db.bind_param(prep_stmt, 4, categorie)
    ibm_db.execute(prep_stmt)
  return render_template('success.html',success="You can login")

@app.route("/displayorder")
def displayorder():
  details_list=[]
  #selecting_shirt
  sql = "SELECT * FROM ORDERS"
  stmt = ibm_db.exec_immediate(conn, sql)
  detail = ibm_db.fetch_both(stmt)
  while detail != False :
      details_list.append(detail)
      detail = ibm_db.fetch_both(stmt)
  print(details_list)
  return render_template('displayorder.html',details=details_list)

@app.route('/logout')
def logout():
    session.pop('email', None)
    return redirect(url_for('login'))
if _name_ == '_main_':
    port=int(os.environ.get('PORT',5000))
    app.run(port=port,host='0.0.0.0')

#url_for('static', filename='style.css')




app = Flask(_name_)
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'

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

def home():
    if 'email' not in session:
      return redirect(url_for('index'))
```

```
  shirt_list=[]
  pant_list=[]
  watch_list=[]
  ring_list=[]

  #selecting_shirt
  sql = "SELECT * FROM SHIRT"
  stmt = ibm_db.exec_immediate(conn, sql)
  shirt = ibm_db.fetch_both(stmt)
  while shirt != False :
      shirt_list.append(shirt)
      shirt = ibm_db.fetch_both(stmt)
  print(shirt_list)

#selecting_pant

  sql1="SELECT * FROM PANT"
  stmt1 = ibm_db.exec_immediate(conn, sql1)
  pant=ibm_db.fetch_both(stmt1)
  while pant != False :
      pant_list.append(pant)
      pant = ibm_db.fetch_both(stmt1)
  print(pant_list)

#selecting_watch
  sql2="SELECT * FROM WATCH"
  stmt2 = ibm_db.exec_immediate(conn, sql2)
  watch=ibm_db.fetch_both(stmt2)
  while watch != False :
      watch_list.append(watch)
      watch = ibm_db.fetch_both(stmt2)
  print(watch_list)

  #selecting_rings
  sql3="SELECT * FROM RINGS"
  stmt3 = ibm_db.exec_immediate(conn, sql3)
  ring=ibm_db.fetch_both(stmt3)
  while ring != False :
      ring_list.append(ring)
      ring = ibm_db.fetch_both(stmt3)
  print(ring_list)
  #returning to HTML
  return render_template('home.html',dictionary= shirt_list,pants=pant_list,watchs=watch_list,rings=ring_list)
```

```
@app.route("/index")
def index():
  return render_template('index.html')

@app.route("/register",methods=['GET','POST'])
def register():
  if request.method == 'POST':
    username = request.form['username']
    email = request.form['email']
    phoneno = request.form['phoneno']
    password = request.form['password']

    # if not username or not email or not phoneno or not password:
    #   return render_template('register.html',error='Please fill all fields')
    hash=bcrypt.hashpw(password.encode('utf-8'),bcrypt.gensalt())
    # query = "SELECT * FROM user_detail WHERE email=? OR phoneno=?"
    # stmt = ibm_db.prepare(conn, query)
    # ibm_db.bind_param(stmt,1,email)
    # ibm_db.bind_param(stmt,2,phoneno)
    # ibm_db.execute(stmt)
    # isUser = ibm_db.fetch_assoc(stmt)
    # if not isUser:
    insert_sql = "INSERT INTO user_detail(username, email, phoneno, password) VALUES (?,?,?,?)"
    prep_stmt = ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(prep_stmt, 1, username)
    ibm_db.bind_param(prep_stmt, 2, email)
    ibm_db.bind_param(prep_stmt, 3, phoneno)
    ibm_db.bind_param(prep_stmt, 4, hash)
    ibm_db.execute(prep_stmt)
    return render_template('register.html',success="You can login")
```

GitHub & Project Demo Link

https://drive.google.com/file/d/1MiSHW_uGmNSi9YFT67oUwypvJr7PVqjS/view?usp=share_link