



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	4
a. Project Overview	4
b. Purpose	4
2. LITERATURE SURVEY	5
a. Existing problem	5
b. References	6
c. Problem Statement Definition	6
3. IDEATION & PROPOSED SOLUTION	7
a. Empathy Map Canvas	7
b. Ideation & Brainstorming	8
c. Proposed Solution	9
d. Problem Solution fit	11
4. REQUIREMENT ANALYSIS	12
a. Functional requirement	12
b. Non-Functional requirements	13
5. PROJECT DESIGN	14
a. Data Flow Diagrams	14
b. Solution & Technical Architecture	14
c. User Stories	15
6. PROJECT PLANNING & SCHEDULING	16
a. Sprint Planning & Estimation	16
b. Sprint Delivery Schedule	17
c. Reports from JIRA	17
7. CODING & SOLUTIONING (Explain the features added in the project along with code).....	18
a. Feature 1	18
b. Feature 2	19
c. Database Schema (if Applicable)	
8. TESTING	20
a. Test Cases	20
b. User Acceptance Testing	21

9. RESULTS	22
a. Performance Metrics	22
10. ADVANTAGES & DISADVANTAGES	23
11. CONCLUSION	25
12. FUTURE SCOPE	25
13. APPENDIX	26
Source Code	26
GitHub & Project Demo Link	38

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

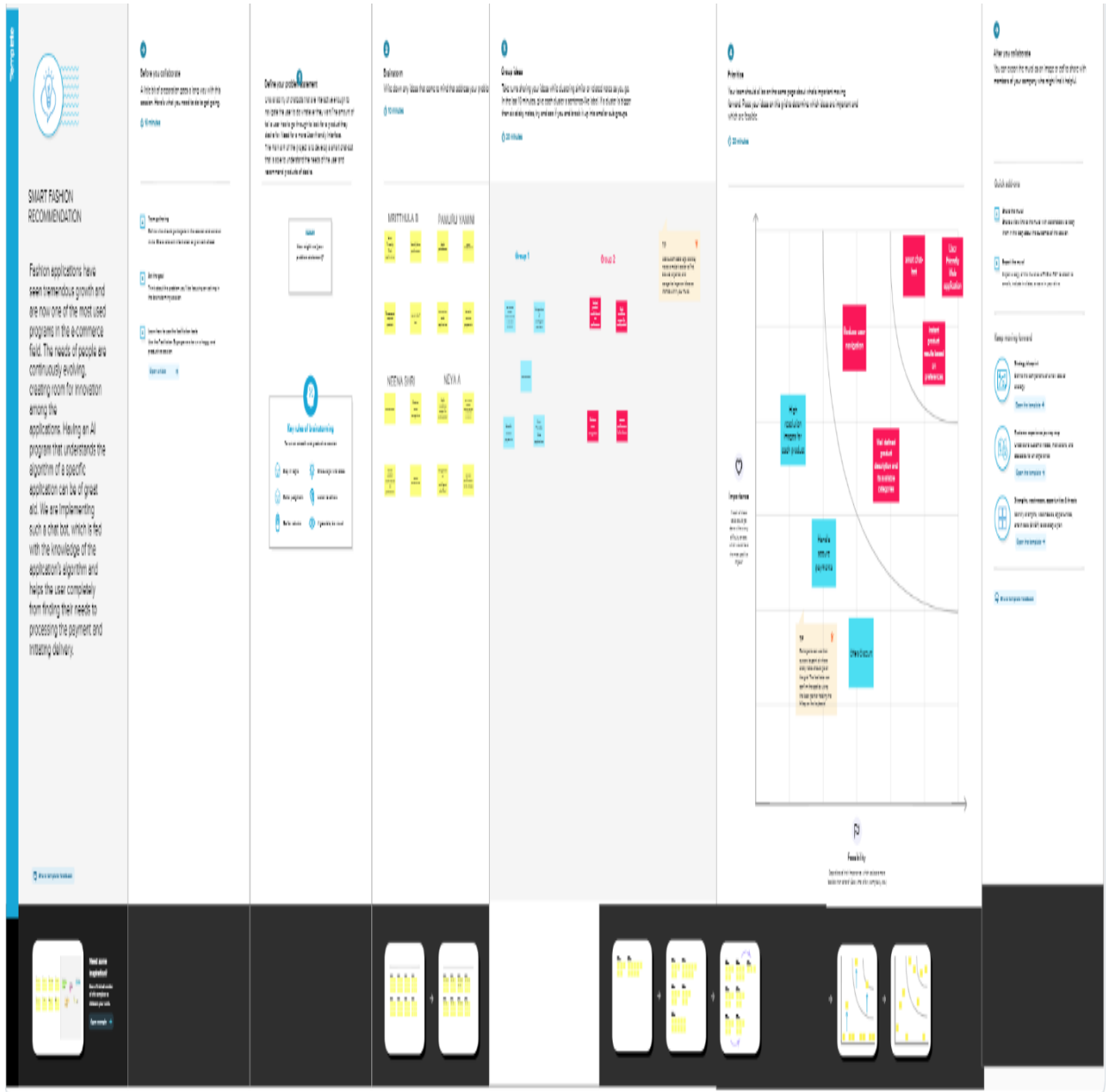
redirection of the user to the appropriate dashboard based on the assigned role. Admin should be able to track the number of different products and admin should be assigned the responsibility to create products with appropriate categories. The user should be able to mention their preferences using interacting 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



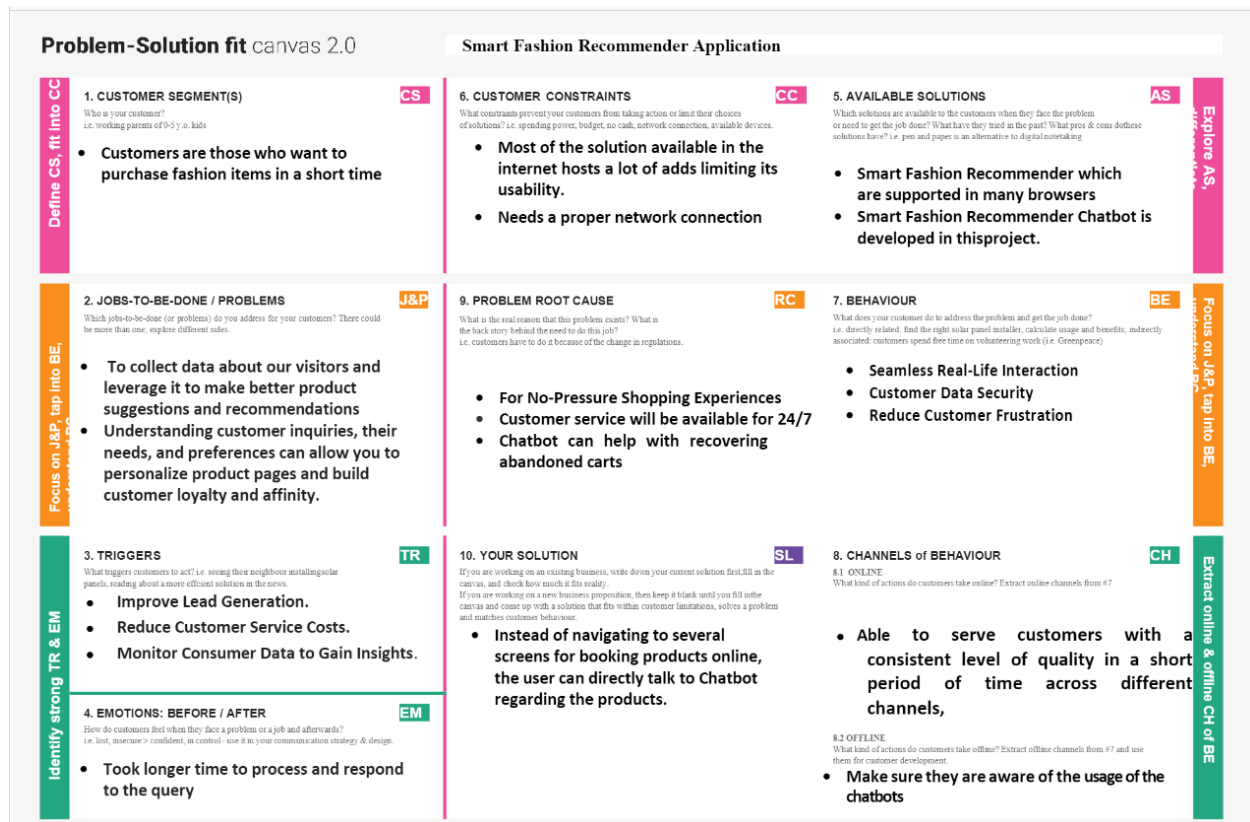
3.3 Proposed Solution

S.No.	Parameter	Description
-------	-----------	-------------

1.		Problem Statement (Problem to be solved)	<p>In normal Online Shopping:</p> <ol style="list-style-type: none"> 1. Poor customer service due to lack of service teams. 2. There are many human error can occur. 3. Negative feedback from customer 4. There is no sufficient way to guide the customers properly. 5. Lack of sales. 6. Poor relationship between customer
2.		Idea / Solution description	<p>By using Smart fashion recommender application:</p> <ol style="list-style-type: none"> 1. Improve the effectiveness of customer service teams. 2. Reduce the potential for human error. 3. Collect candid and meaningful customer feedback. 4. Guide customers along the path to purchase. 5. Turns leads into sales (aka, boost conversion). <p>Build stronger customer relationships</p>
3.		Novelty / Uniqueness	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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)	<ul style="list-style-type: none"> • This application can be developed with minimum cost and at the same time it can provide high performance
6.		Scalability of the Solution	<ul style="list-style-type: none"> • 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

4.1 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
------	------------	---

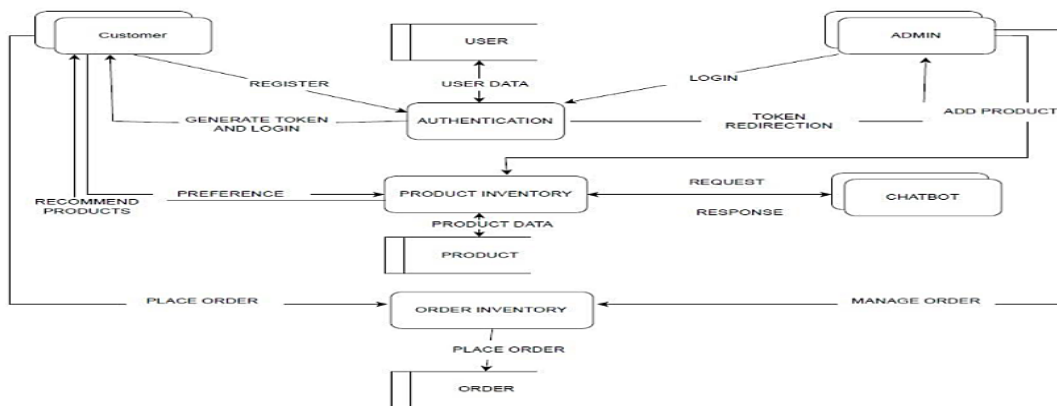
Non-Functional requirements:

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

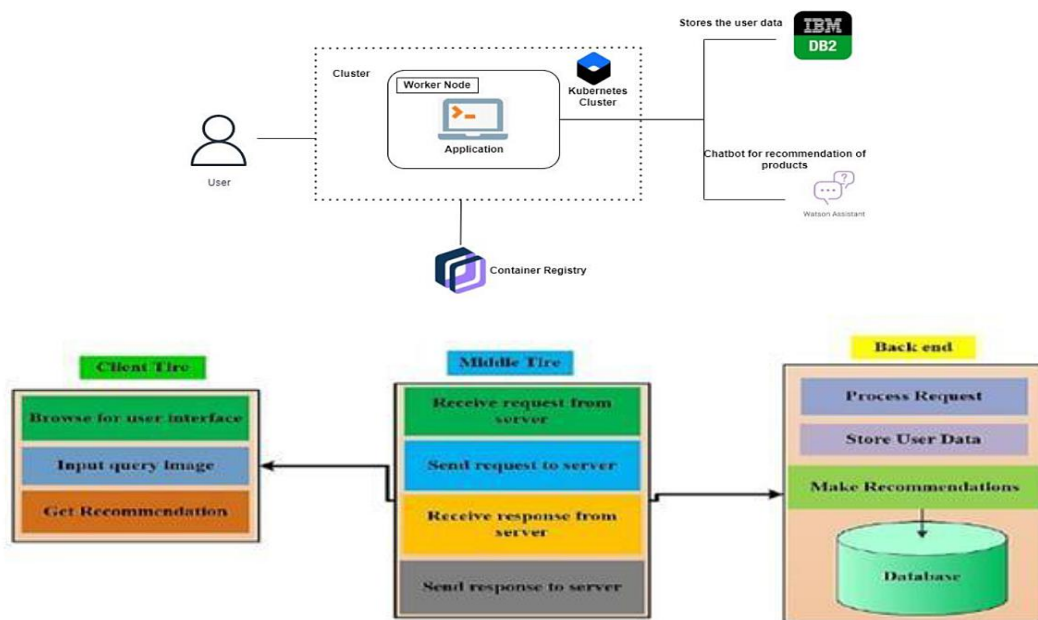
FR No.	Non-Functional Requirement	Description
NFR-1	Usability	A user-friendly interface with chat bot to make usability efficient
NFR-2	Security	Secured connection HTTPS should be established for transmitting requests and responses
NFR-3	Reliability	The system should handle excepted as well as unexpected errors and exceptions to avoid termination of the program
NFR-4	Performance	The system shall be able to handle multiple requests at any given point in time and generate an appropriate response.
NFR-5	Availability	It is a cloud based web application so user can access without any platform limitations ,just using a browsers with a internet connection is enough for use the application

5.PROJECT DESIGN

5.1 Data Flow Diagrams:



5.2 Solution & Technical Architecture:



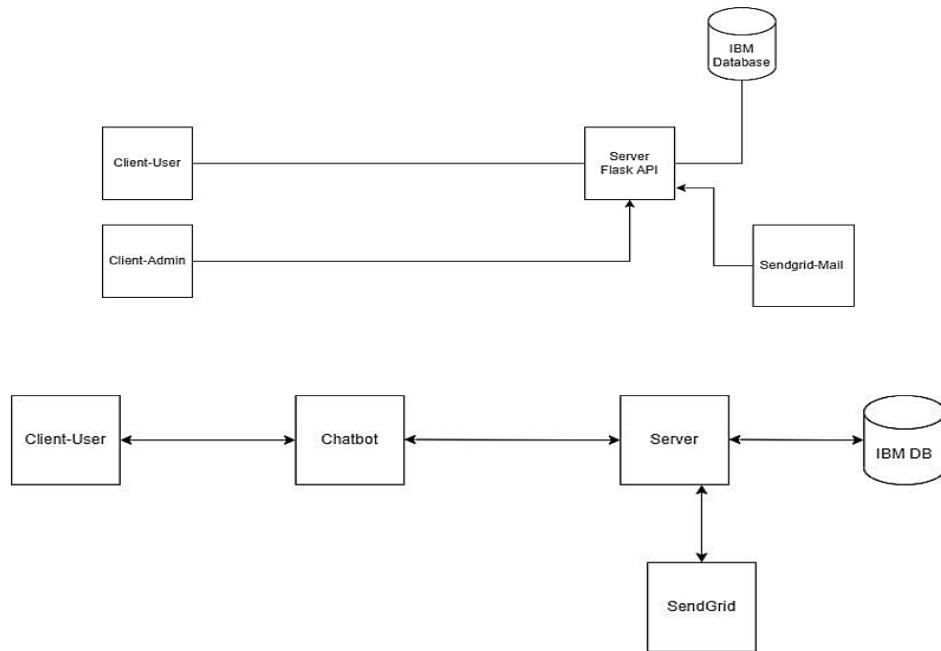


Figure 1: Architecture and data flow of the smart fashion recommendation system

5.3 User Stories

User Stories

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

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password	I can access and make purchases.	High	Sprint-1

	Dashboard					
Customer (Web user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
		USN-5	As a user, I can log into the application by entering email & password	I can access and make purchases.	High	Sprint-1
Administrator	Login	USN-1	I enter my mail and password on organisation's approval	I can approve products and purchases	High	Sprint-1 Administrator

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

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

6.2 Sprint Delivery Schedule

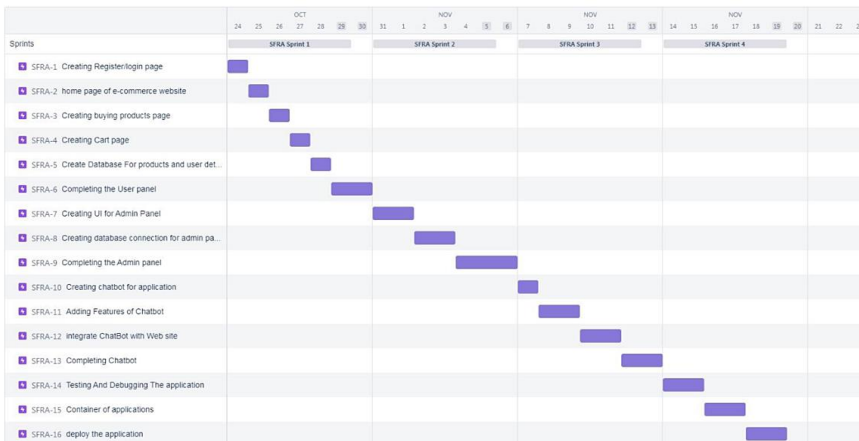
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022		29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022		05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12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

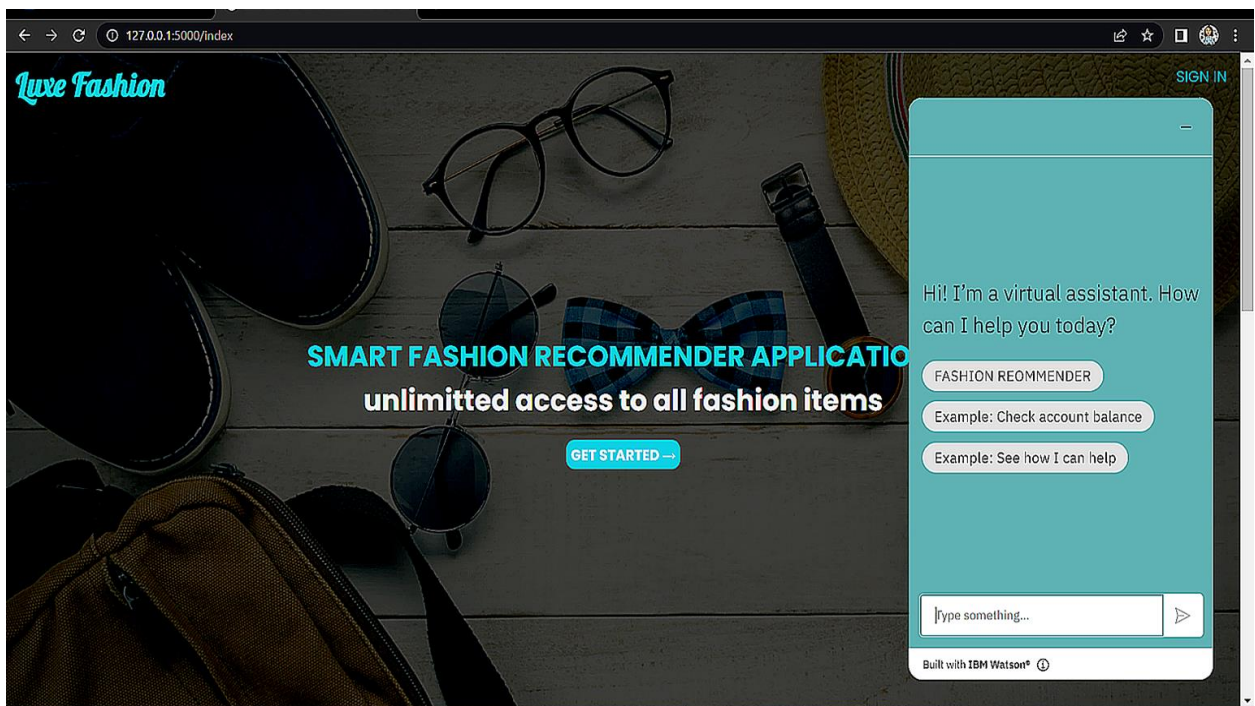
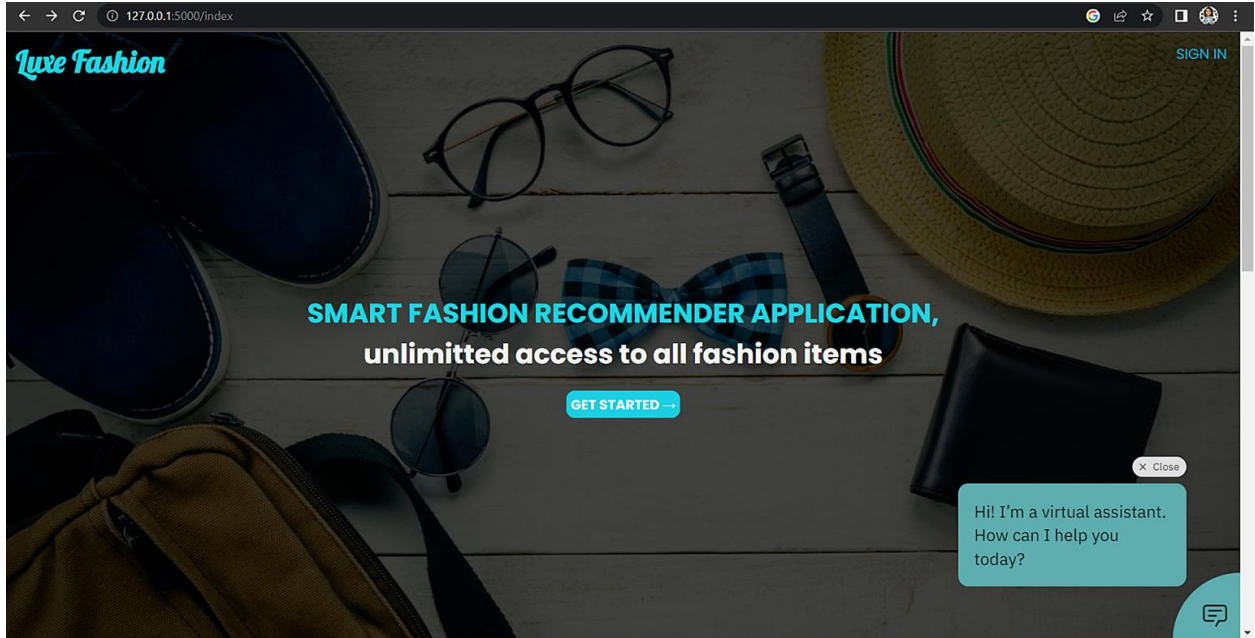
6.3 Reports from JIRA

Burndown Chart:

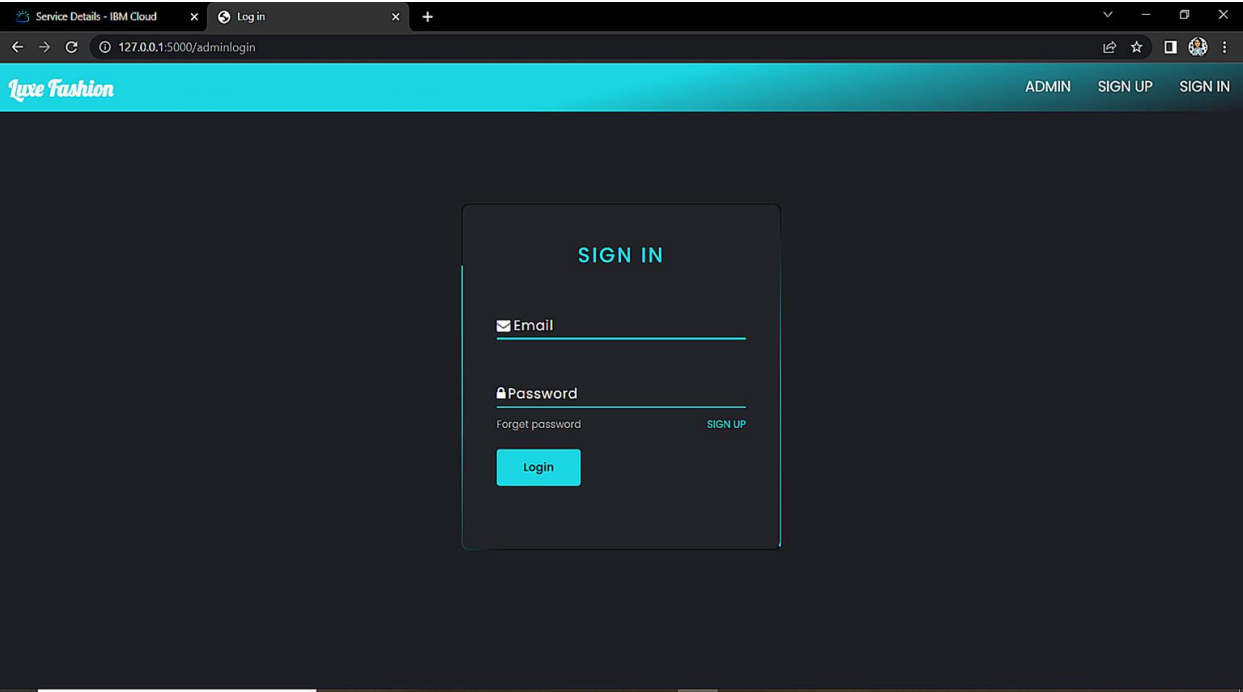


7. CODING & SOLUTIONING (Explain the features added in the project along with code)

7.1 Feature 1








7.2 Feature 2



127.0.0.1:5000/displayorder

Luxé Fashion ADMIN SIGN UP SIGN IN

				
Allen Solly		1000	1000	
Louis Vuitton		1999	1999	
Louis Vuitton		1999	1999	
				

8. TESTING

8.1 Test Cases

Test Case Id	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
TC-1	Install SFRA app in android phone	Transfer SFRA 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 username and leave password field empty	neya	Show error	Didn't show any error	Fail

8.2 User Acceptance Testing

Acceptance Testing UAT Execution & Report Submission

Date	03 November 2022
Team ID	PNT2022TMID15761
Project Name	Project – Smart Fashion Recommender Application
Maximum Marks	4 Marks

1. Purpose of Document

The purpose of this document is to briefly explain the test coverage and open issues of the [ProductName] 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	6	4	2	3	15
Duplicate	1	0	3	0	4
External	1	0	0	1	2
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	15	10	13	26	60

3. Test Case Analysis

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

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	7	0	0	7
Client Application	51	0	0	51
Security	2	0	0	2

Outsource Shipping	3	0	0	3
Exception Reporting	9	0	0	9
Final Report Output	4	0	0	4
Version Control	2	0	0	2

9.RESULTS

9.1 Performance Metrics

performance template for Internet of Things & Cloud Application Development - Excel (Product Activation Failed)

File

Home

Insert

Page Layout

Formulas

Data

Review

View

Tell me what you want to do...

Cut

Copy

Paste

Format Painter

Clipboard

Calibri

28

A

A

B

I

U

Font

Wrap Text

General

Alignment

Number

Normal

Bad

Good

Neutral

Calculation

Check Cell

Styles

Insert

Delete

Format

Cells

AutoSum

Fill

Clear

Editing

Sort & Find & Filter

Select

Editing

D2

X

✓

fx

NFT - Risk Assessment

Formula Bar

			NFT - Risk Assessment							
S.No	Project Name	Scope/feature	Functional Changes	Hardware Changes	Software Changes	Impact of Downtime	Load/Volume Changes	Risk Score	Justification	
1	Smart Fashion Recommender Application	New	Low	No Changes	No Changes	Moderate	>5 to 10%	ORANGE	As we have seen the changes	
			NFT - Detailed Test Plan							
	S.No	Project Overview	NFT Test approach	Assumptions/Dependencies/R		Approvals/SignOff				
	1	Smart Fashion Recommender Application	Manual Testing	Mobile or laptop with internet		Mritthula				
			End Of Test Report							
S.No	Project Overview	NFT Test approach	NFR - Met	Test Outcome	GO/NO-GO decision	Recommendations	Identified Defects (Detected/Closed/Open)		Approvals/SignOff	
1	Smart Fashion Recommender Application	Manual		Worked fine		Use laptop	No defect		Mritthula	

10.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.

11. 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.

12. FUTURE SCOPE

we suggest some future research directions for fashion recommender systems. Considering the rapid growth of multimedia data, where visual information will be the critical component. More indepth research in applications of multi-model fusion and multi-task learning in fashion recommender systems are required to model recommender system to be capable of profiling users comprehensively. Besides, while the majority of researches in fashion recommender systems is mainly based on similaritybased retrieval techniques, there is a need for more studies in the development of new functions such as designing clothes, which are highly demanded in future fashion recommender systems. Furthermore, most of the current fashion datasets do not contain outfit compatibility annotations, or they are limited in terms of size and the type of annotations they provide. Consequently, most researchers built their dataset, which is a labor-costing process, and most of them are not accessible publicly for further research. So, the other future direction for subsequent studies may be focusing on developing automatic annotation methods, constructing large-scale rich annotated data sets for particular task definitions in fashion recommender systems. From an ethical perspective in fashion recommender systems also there is a need for performing the comprehensive study since it has not been

studied in almost any of the researches, which have been reviewed through this thesis.

13. 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=3yD0G9e6V
uQHsOBX;", "", "")

#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,watches=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(prepare_stmt, 1, username)

```

```

        ibm_db.bind_param(prepare_stmt, 2, email)

```

```

        ibm_db.bind_param(prepare_stmt, 3, phoneno)

```

```

        ibm_db.bind_param(prepare_stmt, 4, hash)

```

```

        ibm_db.execute(prepare_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(prepare_stmt, 1, username)
    ibm_db.bind_param(prepare_stmt, 2, email)
    ibm_db.bind_param(prepare_stmt, 3, phoneno)
    ibm_db.bind_param(prepare_stmt, 4, hash)
    ibm_db.execute(prepare_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(prepare_stmt, 1, name)
            ibm_db.bind_param(prepare_stmt, 2, image)
            ibm_db.bind_param(prepare_stmt, 3, categorie)
            ibm_db.bind_param(prepare_stmt, 4, rate)
            ibm_db.execute(prepare_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(prepare_stmt, 1, name)
ibm_db.bind_param(prepare_stmt, 2, image)
ibm_db.bind_param(prepare_stmt, 3, categorie)
ibm_db.bind_param(prepare_stmt, 4, rate)
ibm_db.execute(prepare_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(prepare_stmt, 1, name)
    ibm_db.bind_param(prepare_stmt, 2, image)
    ibm_db.bind_param(prepare_stmt, 3, rate)
    ibm_db.execute(prepare_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(prepare_stmt, 1, name)
    ibm_db.bind_param(prepare_stmt, 2, image)
    ibm_db.bind_param(prepare_stmt, 3, categorie)
    ibm_db.bind_param(prepare_stmt, 4, rate)
    ibm_db.execute(prepare_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,watches=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(prepare_stmt, 1, name)
        ibm_db.bind_param(prepare_stmt, 2, image)
        ibm_db.bind_param(prepare_stmt, 3, rate)
        ibm_db.bind_param(prepare_stmt, 4, categorie)
        ibm_db.execute(prepare_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,watches=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(prepare_stmt, 1, username)
ibm_db.bind_param(prepare_stmt, 2, email)
ibm_db.bind_param(prepare_stmt, 3, phoneno)
ibm_db.bind_param(prepare_stmt, 4, hash)
ibm_db.execute(prepare_stmt)
return render_template('register.html',success="You can login")

```

GitHub Link

<https://github.com/IBM-EPBL/IBM-Project-24384-1659942296>

Project Demo Link

https://drive.google.com/file/d/1dO1IZSdr-Hko6cK-_a6I9j44leeE0tU0/view?usp=sharing