1 INTRODUCTION

1.1 PROJECT OVER VIEW

The era of recommendation systems originally started in the 1990s based on the widespread research progress in Collective Intelligence. During this period, recommendations were generally provided to consumers based on their rating structure . The first consumer-focused recommendation system was developed and commercialized by Goldberg, Nichols, Oki and Terry in 1992. Tapestry, an electronic messaging system was developed to allow users only to rate messages as either a good or bad product and service . However, now there are plenty of methods to obtain information about the consumer's liking for a product through the Internet. These data can be retrieved in the forms of voting, tagging, reviewing and the number of likes or dislikes the user provides. It may also include reviews written in blogs, videos uploaded on YouTube or messages about a product. Regardless of communication and presentation, medium preferences are expressed in the form of numerical values.

1.2 PURPOSE

Drive Traffic

Through personalized email messages and targeted blasts, a recommendation engine can encourage elevated amounts of traffic to your site, thus increasing the opportunity to scoop up more data to further enrich a customer profile.

Deliver Relevant Content

By analyzing the customer's current site usage and previous browsing history, a recommendation engine can deliver relevant product recommendations as he or she shops based on said profile. The data is collected in real time so the software can react as shopping habits change on the fly.

Engage Shoppers

Shoppers become more engaged when personalized product recommendations are made to them across the customer journey. Through individualized product recs, customers are able to delve more deeply into your product line without having to dive into (and very likely get lost in) an ecommerce rabbit hole.

Kibo Research shows that 52% of retailers are leveraging AI-driven personalization to deliver personalized product recommendations to their customers.

Convert Shoppers to Customers

Converting shoppers into customers takes a special touch. Personalized interactions from a recommendation engine show your customer that he or she is valued as an individual, in turn, engendering long-term loyalty.

Increase Average Order Value

Average order values typically go up when an engine is leveraged to display personalized options as shoppers are more willing to spend generously on items they thoroughly covet.

Increase Number of Items per Order

In addition to the average order value rising, the number of items per order also typically rises when an engine is employed. When the customer is shown options that meet his or her interest, they are far more likely to add items to to their active purchase cart.

Control Merchandising and Inventory Rules

A recommendation engine can add your marketing and inventory control directives to a customer's profile to feature products that are on clearance or overstocked so as to avoid unnecessary shopping friction and tone deafness.

Reduce Workload and Overhead

The volume of data required to create a personal shopping experience for each customer is usually far too large to be managed manually. Using an engine automates this process, easing the workload for your IT staff.

A Recommendation Engine Provides Reports

Detailed reports are an integral part of a personalization system. Accurate and up-to-the-minute

reporting will allow you to make informed decisions about the direction of a campaign or the structure of a product page.

Offer Advice and Direction

An experienced recommendation provider like Kibo can offer advice on how to use the data collected from your recommendation engine. Acting as a partner and a consultant, the provider should have the industry know-how needed to help guide you and your ecommerce site to a prosperous future.

2 LITRETRURE SURVEY

2.1 Existing problem

I am	Describe customer with 3-4 characteristics-who are they?	Customer who wants to create a personalized collections. Customer wanting to buy good quality product in less time. Customer who wants to wear all kind of collections from traditional to western.
I am trying to	List their outcome or "job" the core about – what are they trying to achieve?	Can choose the product from the comfort of their homes.
but	Describe the problems or barriers that get in the way here	Working professionals could not spend much time on in-store shopping , hence this application might come in handy.
because	Enter the "root cause" of why the problem or barrier exists -what needs to be solved?	Nowadays people are so busy and lazy to go for shopping. At the same time they couldn't afford much time for it.
Which makes me feel	Describe the emotion from the customer's point of view-how does it impact them emotionally?	Customer feels so happy and satisfied for getting their personalized collection in less time.

2.2 REFERENCES

1. Barnard, M. *Fashion as Communication*, 2nd ed.; Routledge: London, UK, 2008. [**Google Scholar**]
2. Chakraborty, S.; Hoque, S.M.A.; Kabir, S.M.F. Predicting fashion trend using runway images: Application of logistic regression in trend forecasting. *Int. J. Fash. Des. Technol. Educ.* **2020**, *13*, 376–386. [**Google Scholar**] [**CrossRef**]
3. Karmaker Santu, S.K.; Sondhi, P.; Zhai, C. On application of learning to rank for e-commerce search. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, 7–11 August 2017; pp. 475–484. [**Google Scholar**] [**CrossRef**][**Green Version**]
4. Garude, D.; Khopkar, A.; Dhake, M.; Laghane, S.; Maktum, T. Skin-tone and occasion oriented outfit recommendation system. *SSRN Electron. J.* **2019**. [**Google Scholar**] [**CrossRef**]
5. Kang, W.-C.; Fang, C.; Wang, Z.; McAuley, J. Visually-aware fashion recommendation and design with generative image models. In Proceedings of the 2017 IEEE International Conference on Data Mining (ICDM), New Orleans, LA, USA, 18–21 November 2017; pp. 207–216. [**Google Scholar**] [**CrossRef**][**Green Version**]

2.3 PROBLEM STATEMENT DEFINITION

While the stratospheric growth shows that businesses all over the world are exploring what recommendation engines can do for them, effectively using this technology comes with its fair share of challenges.

1. Significant investments required

Recommendation engines are a big investment, not only financially, but in terms of time, too: it takes a long time and deep expertise to build an effective recommendation engine in-house.

2. Too many choices

Alternatively, you could employ an off-the-shelf solution from a third-party company, but with so many options available on the market, how do you know

which is the right one for your business? Evaluating different solutions can be enormously time consuming, as you need to evaluate their case studies, the technology, how the solution will be integrated into your current company setup, and so on.

3. The complex onboarding process

Bringing a recommendation engine into your business can be a complex affair. Sometimes, it might not be worth the effort, especially if it does not fit into your business vertical.

4. Lack of data analytics capability

Like all AI-based technologies, recommendation engines rely on data – if you do not have high-quality data, or cannot crunch and analyze it properly, you will not be able to make the most of the recommendation engine.

5. The 'cold start' problem

Relying on user data has its downsides, one of which is the issue of 'cold start'. This is when a new user enters the system or new items are added to the catalogue, and therefore, it will be difficult for the algorithm to predict the taste or preferences of the new user, or the rating of the new items, leading to less accurate recommendations.

6. Inability to capture changes in user behavior

Consumers do not stand still – they are constantly behaving and evolving both as people and customers. Staying on top of these changes is a constant battle.

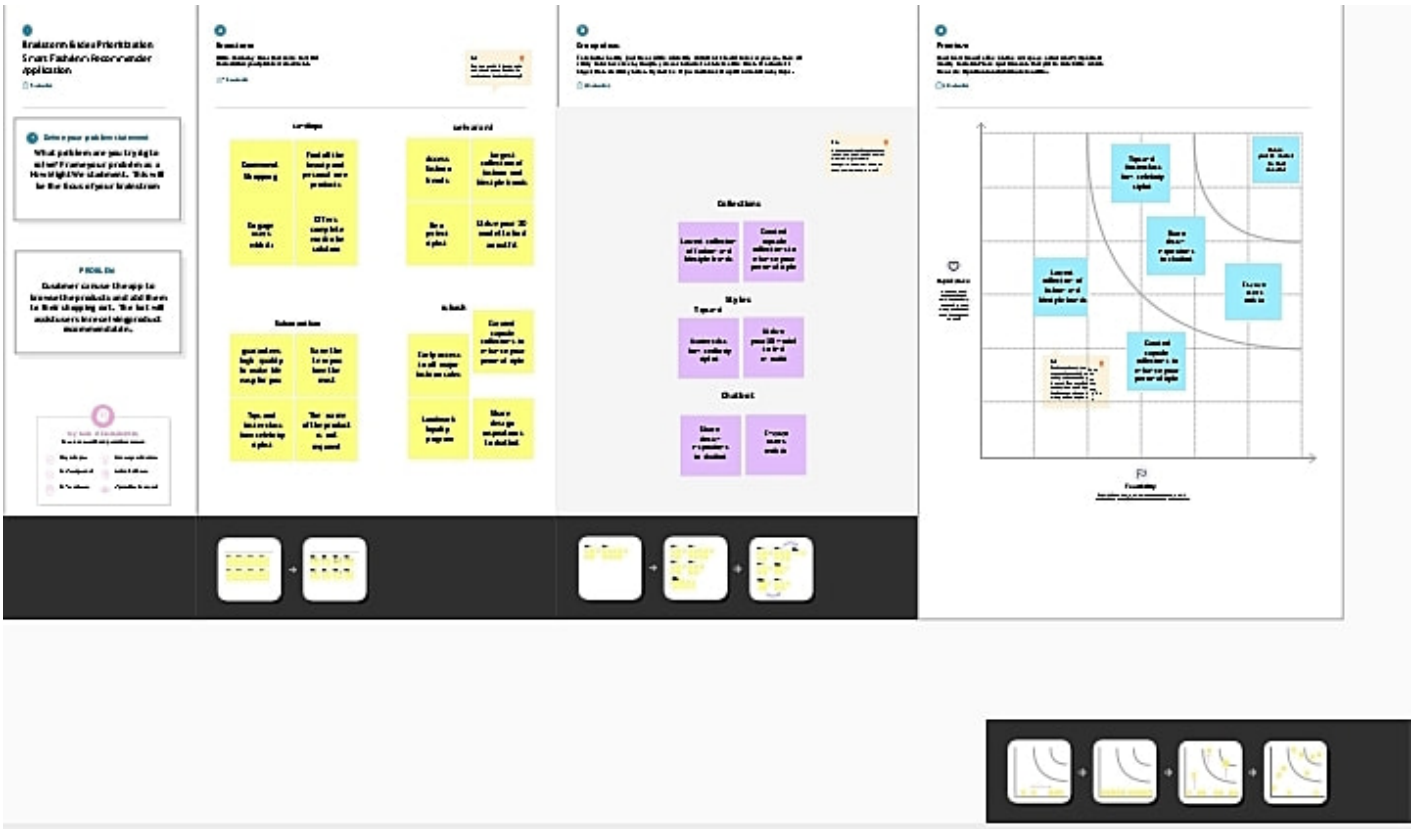
7. Privacy concerns

The more the algorithm knows about the customer, the more accurate its recommendations will be. However, many customers are hesitant to hand over personal information, especially given several high-profile cases of customer data leaks in recent years. However, without this customer data, the recommendation engine cannot function effectively.

3. IDEATION AND PROPOSED SOLUTION

3.1 EMPATHY MAP

3.2 IDEATION AND BRAIN STROMING



3.3 Proposed Solution

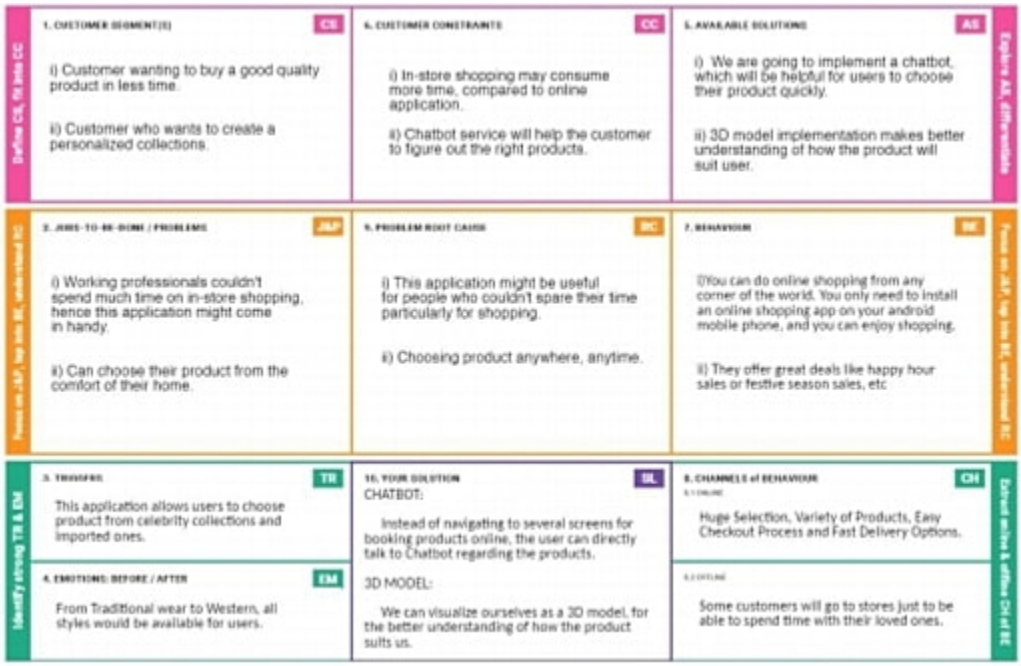
S.No.	Parameter	Description
•	Problem Statement (Problem to be solved)	Customer can use the app to browse the products and add them to their shopping cart. The bot will assist users in receiving product recommendation.
•	Idea / Solution description	We have come up with a new innovative solution through which you can directly do your online shopping based on your choice without any search. It can be done by using the chatbot.
•	Novelty / Uniqueness	Share design inspirations to chatbot. Utilize user's 3D model to find an outfit.
•	Social Impact / Customer Satisfaction	Instead of navigating to several screens for booking products online, the user can directly talk to Chatbot regarding the products. We can visualize ourselves as a 3D model, for the better understanding of how the product suits us.
•	Business Model (Revenue Model)	While getting a big order from a major retailer might sound like a good thing for a fledgling brand, it means the brand has a short time to somehow produce that inventory and hire the necessary employees without any money upfront.
•	Scalability of the Solution	Technological developments such as color changes and the integration of conductive sensors etc. Could revolutionize the way designers think about fashion.

3.4 PROBLEM SOLUTION FIT

The Problem-Solution Fit simply means that you have found a problem with your customer and that the solution you have realized for it actually solves the customer's problem. It helps entrepreneurs, marketers and corporate innovators identify behavioral patterns and recognize what would work and why

Purpose

• Users find it difficult to choose their product, here the bot will assist user in receiving product recommendation.
• To reduce search time, from the user interaction with bot, the similar product will be displayed based on user's requirements.
• The implemented 3D model will help user to decide how the product will look on them.
• This would be a one stop solution for all kinds of users.



4 REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through mobile number Registration through LinkedIN
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Advanced Search Capabilities	sorting and filtering options
FR-4	Checking item availability	item availability in specific locations
FR-5	Shopping cart	My cart button Add-to-cart button Remove-from-cart button
FR-6	Super-fast checkout	Online transfer, credit card payment, paying with mobile wallets
FR-7	Checking the shipping status	Option to easily check the shipping status of items ordered in the store

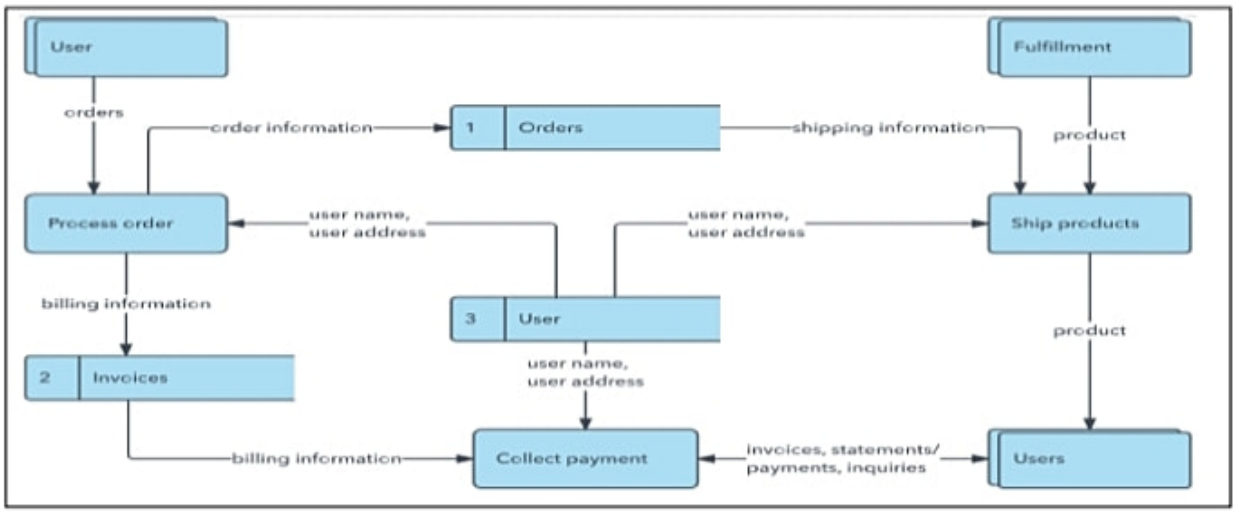
4.2 NON-FUNCTIONAL REQUIREMENTS

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Specific user in a specific context can use a product/design to achieve a defined goal effectively, efficiently and satisfactorily.
NFR-2	**Security**	This Application will collect a lot of users' private information to complete a purchase (banking, shipping/home address, email, etc.) Data protection is the priority.
NFR-3	**Reliability**	Ability of the software to perform critical tasks like collecting and securing customer data, providing payment gateway to function correctly in a given environment, for a particular amount of time
NFR-4	**Performance**	Online shopping behavior is no different from offline — people love places and platforms that help them to find the best deals and products in a single place with minimal effort
NFR-5	**Availability**	Online consumers do not adhere to closing times. Information should be available wherever and whenever required within a time limit specified.
NFR-6	**Scalability**	Having a plan to handle demand peaks. Avoid downtime, preserve the customer experience, and ensure deliveries go out on time at all costs

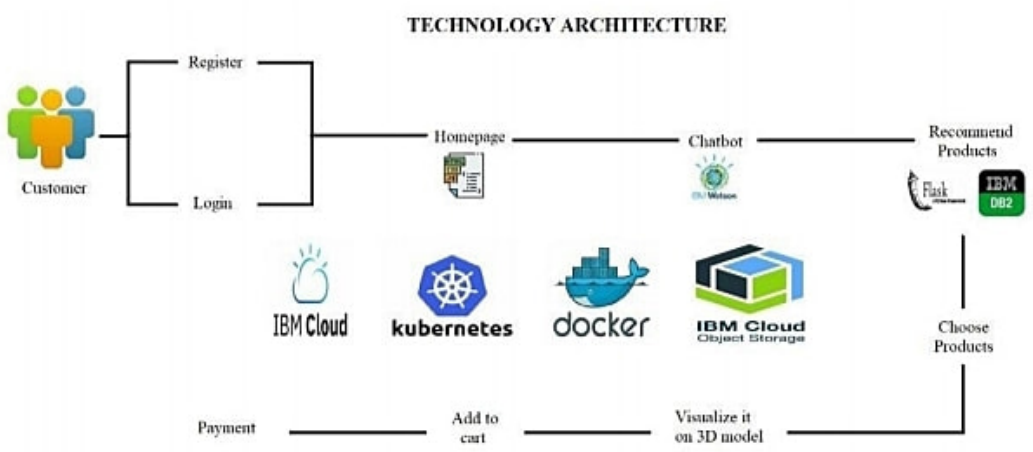
5. PROJECT DESING

5.1 DATA FLOW DIAGRAMS

A data flow daigram (DFD) is a traditional visual representaion of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows it shows how data enters and leaves the system ,what changes the information,and where data is stored



5.2 SOLUTION AND TECHNICAL ARCHITECTURE



1: ComponentsTable & Technologies:

S.No	Component	Description	Technology
1.	User Interface	How user interacts with application e.g. Web UI, Mobile App, Chatbot etc.	HTML, CSS, JavaScript / Angular Js / React Js etc.
2.	Application Logic-1	Logic for a process in the application	Java / Python
3.	Application Logic-2	Logic for a process in the application	IBM Watson STT service
4.	Application Logic-3	Logic for a process in the application	IBM Watson Assistant
5.	Database	Data Type, Configurations etc.	MySQL, NoSQL, etc.
6.	Cloud Database	Database Service on Cloud	IBM DB2, IBM Cloudant etc.
7.	File Storage	File storage requirements	IBM Block Storage or Other Storage Service or Local Filesystem
8.	External API-1	Purpose of External API used in the application	IBM Weather API, etc.
9.	External API-2	Purpose of External API used in the application	Aadhar API, etc.
10.	Machine Learning Model	Purpose of Machine Learning Model	Object Recognition Model, etc.
11.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud Local Server Configuration: Cloud Server Configuration :	Local, Cloud Foundry, Kubernetes, etc.

Table 2:Application Cherecteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source frameworks used	Technology of Opensource framework
2.	Security Implementations	List all the security / access controls implemented, use of firewalls etc.	e.g. SHA-256, Encryptions, IAM Controls, OWASP etc.
3.	Scalable Architecture	Justify the scalability of architecture (3 – tier, Micro-services)	Technology used
4.	Availability	Justify the availability of application (e.g. use of load balancers, distributed servers etc.)	Technology used
5.	Performance	Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc.	Technology used

5.3 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/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	
		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	
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	
		USN-4	As a user, I can register for the application through Gmail		Medium	
	Login	USN-5	As a user, I can log into the application by entering email & password		High	
Customer Care Executive		USN-7	As a customer care executive i can solve the login issues and other issues of the application.	I can provide support or solution at any time 24*7	Medium	
Administrator	Application	USN-8	As a administrator i can upgrad or update the application.	I can fix the bugs which arises for the customers and users of the application	Medium	

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story	Points	Team Members
Sprint-1	User	USN-1	As a user, I can register for the application entering email, password, and confirming password I will go through the products on the website	20 by my available	High	Sandhiya,selvarani
Sprint-2	Admin	USN-2	As an Admin, I can check out the database about the stock and have a track of all the that the users are purchasing.	20 things	High	Subash , subananthan

Sprint-3	Chat Bot	USN-3	The user can directly talk to Chat bot regarding the products. Get the recommendations on information provided by the user	20 based	High	Sandhiya,selvarani
Sprint-4	Final Delivery	USN-4	Container of applications using dockerkubernetes and deployment the application. Create the documentation and final submit the application	20	High	Subash,subananthan

Project Tracker, Velocity & Burn down Chart: (4 Marks)

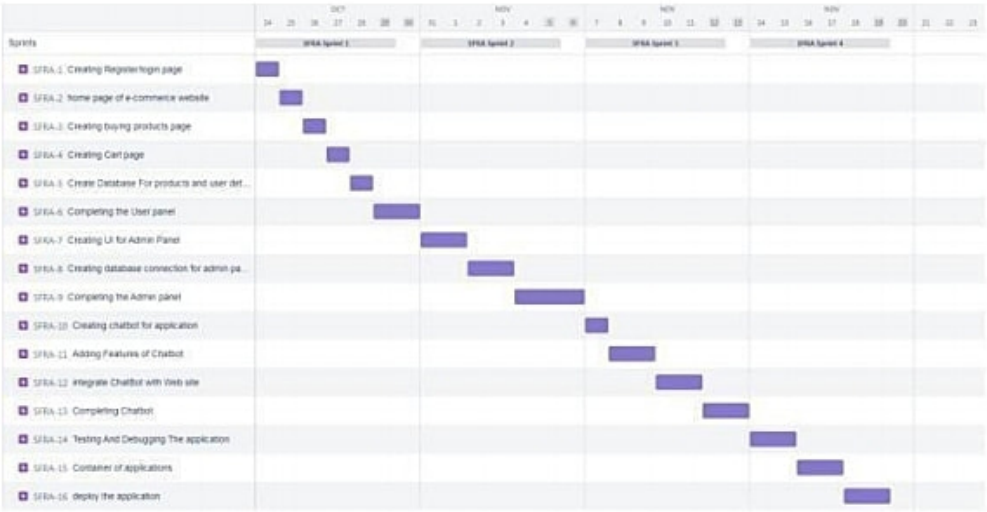
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed Planned	Sprint Release Date(Actual) (as on End Date)
Sprint-1	20	6	Days 24 Oct 2022	29 Oct 2022		1 Nov2022
Sprint-2	20	6	Days 31 Oct 2022	05 Nov 2022		05 Nov 2022
Sprint-3	20	6	Days 07 Nov 2022	12 Nov 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) periteration unit (story points per day)

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

Burn down Chart:



7. CODING & SOLUTION

7.1 Feature

```python
import ibm_db
from flask_session import Session
from flask import Flask, render_template, redirect, request, session, jsonify
from datetime import datetime

# # Instantiate Flask object named app
app = Flask(__name__)

# # Configure sessions
app.config["DB2_DATABASE"] = 'bludb'
app.config["DB2_HOSTNAME"] = '9938aec0-8105-433e-8bf9-0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud'
app.config["DB2_PORT"] = '32459'
app.config["DB2_PROTOCOL"] = 'TCPIP'
app.config["DB2_USER"] = 'mqs19694'
app.config["DB2_PASSWORD"] = 'SsDjCqU5ECrgxjRF'

Session(app)

# Creates a connection to the database
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=9938aec0-8105-433e-8bf9-0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32459/bludb:userid=<mqs19694>;password=<r1WN
db = conn.connection.cursor()

@app.route("/")
def index():
    shirts = db.execute("SELECT * FROM shirts ORDER BY onSalePrice")
    shirtsLen = len(shirts)
    # Initialize variables
    shoppingCart = []
    shopLen = len(shoppingCart)
    totItems, total, display = 0, 0, 0
    if 'user' in session:
        shoppingCart = db.execute("SELECT samplename, image, SUM(qty), SUM(subTotal), price, id FROM cart GROUP BY samplename")
        shopLen = len(shoppingCart)
```

```python
        for i in range(shopLen):
            total += shoppingCart[i]["SUM(subTotal)"]
            totItems += shoppingCart[i]["SUM(qty)"]
        shirts = db.execute("SELECT * FROM shirts ORDER BY onSalePrice ASC")
        shirtsLen = len(shirts)
        return render_template ("index.html", shoppingCart=shoppingCart, shirts=shirts, shopLen=shopLen, shirtsLen=shirtsLen, total=total, totItems=totItems, display=display,
    return render_template ( "index.html", shirts=shirts, shoppingCart=shoppingCart, shirtsLen=shirtsLen, shopLen=shopLen, total=total, totItems=totItems, display=display)


@app.route("/buy/")
def buy():
    # Initialize shopping cart variables
    shoppingCart = []
    shopLen = len(shoppingCart)
    totItems, total, display = 0, 0, 0
    qty = int(request.args.get('quantity'))
    if session:
        # Store id of the selected shirt
        id = int(request.args.get('id'))
        # Select info of selected shirt from database
        goods = db.execute("SELECT * FROM shirts WHERE id = :id", id=id)
        # Extract values from selected shirt record
        # Check if shirt is on sale to determine price
        if(goods[0]["onSale"] == 1):
            price = goods[0]["onSalePrice"]
        else:
            price = goods[0]["price"]
        samplename = goods[0]["samplename"]
        image = goods[0]["image"]
        subTotal = qty * price
        # Insert selected shirt into shopping cart
        db.execute("INSERT INTO cart (id, qty, samplename, image, price, subTotal) VALUES (:id, :qty, :samplename, :image, :price, :subTotal)", id=id, qty=qty, samplename=samp
```

```
        shoppingCart = db.execute("SELECT samplename, image, SUM(qty), SUM(subTotal), price, id FROM cart GROUP BY samplename")
        shopLen = len(shoppingCart)
        # Rebuild shopping cart
        for i in range(shopLen):
            total += shoppingCart[i]["SUM(subTotal)"]
            totItems += shoppingCart[i]["SUM(qty)"]
        # Select all shirts for home page view
        shirts = db.execute("SELECT * FROM shirts ORDER BY samplename ASC")
        shirtsLen = len(shirts)
        # Go back to home page
        return render_template ("index.html", shoppingCart=shoppingCart, shirts=shirts, shopLen=shopLen, shirtsLen=shirtsLen, total=total, totItems=totItems, display=display,


@app.route("/update/")
def update():
    # Initialize shopping cart variables
    shoppingCart = []
    shopLen = len(shoppingCart)
    totItems, total, display = 0, 0, 0
    qty = int(request.args.get('quantity'))
    if session:
        # Store id of the selected shirt
        id = int(request.args.get('id'))
        db.execute("DELETE FROM cart WHERE id = :id", id=id)
        # Select info of selected shirt from database
        goods = db.execute("SELECT * FROM shirts WHERE id = :id", id=id)
        # Extract values from selected shirt record
        # Check if shirt is on sale to determine price
        if(goods[0]["onSale"] == 1):
            price = goods[0]["onSalePrice"]
        else:
            price = goods[0]["price"]
```

```
        samplename = goods[0]["samplename"]
        image = goods[0]["image"]
        subTotal = qty * price
        # Insert selected shirt into shopping cart
        db.execute("INSERT INTO cart (id, qty, samplename, image, price, subTotal) VALUES (:id, :qty, :samplename, :image, :price, :subTotal)", id=id, qty=qty, samplename=samp
        shoppingCart = db.execute("SELECT samplename, image, SUM(qty), SUM(subTotal), price, id FROM cart GROUP BY samplename")
        shopLen = len(shoppingCart)
        # Rebuild shopping cart
        for i in range(shopLen):
            total += shoppingCart[i]["SUM(subTotal)"]
            totItems += shoppingCart[i]["SUM(qty)"]
        # Go back to cart page
        return render_template ("cart.html", shoppingCart=shoppingCart, shopLen=shopLen, total=total, totItems=totItems, display=display, session=session )


@app.route("/filter/")
def filter():
    if request.args.get('typeClothes'):
        query = request.args.get('typeClothes')
        shirts = db.execute("SELECT * FROM shirts WHERE typeClothes = :query ORDER BY samplename ASC", query=query )
    if request.args.get('sale'):
        query = request.args.get('sale')
        shirts = db.execute("SELECT * FROM shirts WHERE onSale = :query ORDER BY samplename ASC", query=query)
    if request.args.get('id'):
        query = int(request.args.get('id'))
        shirts = db.execute("SELECT * FROM shirts WHERE id = :query ORDER BY samplename ASC", query=query)
    if request.args.get('kind'):
        query = request.args.get('kind')
        shirts = db.execute("SELECT * FROM shirts WHERE kind = :query ORDER BY samplename ASC", query=query)
    if request.args.get('price'):
        query = request.args.get('price')
        shirts = db.execute("SELECT * FROM shirts ORDER BY onSalePrice ASC")
```

```
    shirtsLen = len(shirts)
    # Initialize shopping cart variables
    shoppingCart = []
    shopLen = len(shoppingCart)
    totItems, total, display = 0, 0, 0
    if 'user' in session:
        # Rebuild shopping cart
        shoppingCart = db.execute("SELECT samplename, image, SUM(qty), SUM(subTotal), price, id FROM cart GROUP BY samplename")
        shopLen = len(shoppingCart)
        for i in range(shopLen):
            total += shoppingCart[i]["SUM(subTotal)"]
            totItems += shoppingCart[i]["SUM(qty)"]
        # Render filtered view
        return render_template ("index.html", shoppingCart=shoppingCart, shirts=shirts, shopLen=shopLen, shirtsLen=shirtsLen, total=total, totItems=totItems, display=display,
    # Render filtered view
    return render_template ( "index.html", shirts=shirts, shoppingCart=shoppingCart, shirtsLen=shirtsLen, shopLen=shopLen, total=total, totItems=totItems, display=display)


@app.route("/checkout/")
def checkout():
    order = db.execute("SELECT * from cart")
    # Update purchase history of current customer
    for item in order:
        db.execute("INSERT INTO purchases (uid, id, samplename, image, quantity) VALUES(:uid, :id, :samplename, :image, :quantity)", uid=session["uid"], id=item["id"], samplen
    # Clear shopping cart
    db.execute("DELETE from cart")
    shoppingCart = []
    shopLen = len(shoppingCart)
    totItems, total, display = 0, 0, 0
    # Redirect to home page
    return redirect('/')


```

```


@app.route("/remove/", methods=["GET"])
def remove():
    # Get the id of shirt selected to be removed
    out = int(request.args.get("id"))
    # Remove shirt from shopping cart
    db.execute("DELETE from cart WHERE id=:id", id=out)
    # Initialize shopping cart variables
    totItems, total, display = 0, 0, 0
    # Rebuild shopping cart
    shoppingCart = db.execute("SELECT samplename, image, SUM(qty), SUM(subTotal), price, id FROM cart GROUP BY samplename")
    shopLen = len(shoppingCart)
    for i in range(shopLen):
        total += shoppingCart[i]["SUM(subTotal)"]
        totItems += shoppingCart[i]["SUM(qty)"]
    # Turn on "remove success" flag
    display = 1
    # Render shopping cart
    return render_template ("cart.html", shoppingCart=shoppingCart, shopLen=shopLen, total=total, totItems=totItems, display=display, session=session )


@app.route("/login/", methods=["GET"])
def login():
    return render_template("login.html")


@app.route("/new/", methods=["GET"])
def new():
    # Render log in page
    return render_template("new.html")


```

```python
@app.route("/logged/", methods=["POST"] )
def logged():
    # Get log in info from log in form
    user = request.form["username"].lower()
    pwd = request.form["password"]
    #pwd = str(sha1(request.form["password"].encode('utf-8')).hexdigest())
    # Make sure form input is not blank and re-render log in page if blank
    if user == "" or pwd == "":
        return render_template ( "login.html" )
    # Find out if info in form matches a record in user database
    query = "SELECT * FROM users WHERE username = :user AND password = :pwd"
    rows = db.execute ( query, user=user, pwd=pwd )

    # If username and password match a record in database, set session variables
    if len(rows) == 1:
        session['user'] = user
        session['time'] = datetime.now( )
        session['uid'] = rows[0]["id"]
    # Redirect to Home Page
    if 'user' in session:
        return redirect ( "/" )
    # If username is not in the database return the log in page
    return render_template ( "login.html", msg="Wrong username or password." )


@app.route("/history/")
def history():
    # Initialize shopping cart variables
    shoppingCart = []
    shopLen = len(shoppingCart)
    totItems, total, display = 0, 0, 0
```

```python
    # Retrieve all shirts ever bought by current user
    myShirts = db.execute("SELECT * FROM purchases WHERE uid=:uid", uid=session["uid"])
    myShirtsLen = len(myShirts)
    # Render table with shopping history of current user
    return render_template("history.html", shoppingCart=shoppingCart, shopLen=shopLen, total=total, totItems=totItems, display=display, session=session, myShirts=myShirts, myS


@app.route("/logout/")
def logout():
    # clear shopping cart
    db.execute("DELETE from cart")
    # Forget any user_id
    session.clear()
    # Redirect user to login form
    return redirect("/")


@app.route("/register/", methods=["POST"] )
def registration():
    # Get info from form
    username = request.form["username"]
    password = request.form["password"]
    confirm = request.form["confirm"]
    fname = request.form["fname"]
    lname = request.form["lname"]
    email = request.form["email"]
    # See if username already in the database
    rows = db.execute( "SELECT * FROM users WHERE username = :username ", username = username )
    # If username already exists, alert user
    if len( rows ) > 0:
        return render_template ( "new.html", msg="Username already exists!" )
    # If new user, upload his/her info into the users database
```

```python
    new = db.execute ( "INSERT INTO users (username, password, fname, lname, email) VALUES (:username, :password, :fname, :lname, :email)",
                       username=username, password=password, fname=fname, lname=lname, email=email )
    # Render login template
    return render_template ( "login.html" )


@app.route("/cart/")
def cart():
    if 'user' in session:
        # Clear shopping cart variables
        totItems, total, display = 0, 0, 0
        # Grab info currently in database
        shoppingCart = db.execute("SELECT samplename, image, SUM(qty), SUM(subTotal), price, id FROM cart GROUP BY samplename")
        # Get variable values
        shopLen = len(shoppingCart)
        for i in range(shopLen):
            total += shoppingCart[i]["SUM(subTotal)"]
            totItems += shoppingCart[i]["SUM(qty)"]
    # Render shopping cart
    return render_template("cart.html", shoppingCart=shoppingCart, shopLen=shopLen, total=total, totItems=totItems, display=display, session=session)
```

8.TESTING

8.1 Test cases

Shopping Cart

x

{% if shopLen ! = 0 %} {% for i in range(shopLen) %} {% endfor %} {% else %} {% endif %}

#	Item	Name	Quantity	Unit Price	Sub-Total
{{ i + 1 }}	{{ shoppingCart[i] [	{{ shoppingCart[i] ["samplename"] }}	{{ shoppingCart[i] ['SUM(qty)'] }}	{{ '${:.. 2f}'. format(shoppingCart[i] ["price"]) }}	{{ '${:.. 2f}'. format(shoppingCart[i] ['SUM(subTotal)']) }}

Your cart is empty :\

Get some shirts now!

Continue Shopping

Total: {{ '${:.. 2f}'. format(total) }}

Make Changes | Continue Shopping | Quick Checkout

Roopesh Clothing Store {% if session %}

- Logout
- You Bought
  {% else %}
  - Register
  - Login
    {% endif %}
  - Filter By
    All Shirts Trousers Shoes Casual Clothing Formal Clothing On Sale Price $0-$000

No. of Items: {{ totItems }} Total: ${{ '{:.. 2f}'. format(total) }}

{% if display == 1 %}
Your item was successfully removed from shopping cart!
{% endif %} {% block body %}{% endblock %}

© Roopesh Clothing Store

{% extends "base. html" %} {% block title %} Roopesh Clothing Store - Home {% endblock %} {% block body %}

Shopping Cart

{% if shopLen ! = 0 %} {% for i in range(shopLen) %} {% endfor %} {% else %} {% endif %}

#	Item	samplename	Quantity	Unit Price	Sub-Total	
{{ i + 1 }}	{{	{{ shoppingCart[i] ["samplename"] }}	Update	{{ '${:. . 2f}'. format(shoppingCart[i] ["price"]) }}	{{ '${:. . 2f}'. format(shoppingCart[i] ['SUM(subTotal)']) }}	Remove

Your cart is empty :\

Get some shirts now!

Continue Shopping

Total: {{ '${:. . 2f}'. format(total) }}

Continue Shopping Proceed to Checkout

{% endblock %}

{% extends "base. html" %} {% block title %} Roopesh Clothing Store - Home {% endblock %} {% block body %}

Your Shopping History

Items you've bought in the past.

{% for i in range(myShirtsLen) %} {% endfor %}

#	Item	Name	Quantity	Date	
{{ i + 1 }}	{{	{{ myShirts[i]["samplename"] }}	{{ myShirts[i]["quantity"] }}	{{ myShirts[i]["date"] }}	Buy Again

{% endblock %}

{% extends "base. html" %} {% block title %} Roopesh Clothing Store - Home {% endblock %} {% block body %}

Your Shopping History

Items you've bought in the past.

{% for i in range(myShirtsLen) %} {% endfor %}

#	Item	Name	Quantity	Date	
{{ i + 1 }}	{{	{{ myShirts[i]["samplename"] }}	{{ myShirts[i]["quantity"] }}	{{ myShirts[i]["date"] }}	Buy Again

{% endblock %}

```
{% extends "base. html" %}

{% block title %}
Roopesh Clothing Store - Home
{% endblock %}

{% block body %}
<! -- Main Store Body -->
{% if session['user'] %}
<div class="alert alert-warning alert-dismissible fade show" role="alert">
<button type="button" class="close" data-dismiss="alert" aria-label="Close">
<span aria-hidden="true">&times; </span>
</button>
<strong>Welcome, {{ session['user'] }}</strong> Hope you have a pleasant experience shopping with us.
</div>

{% endif %}
<div class="row" id="shirtCard">
{% for i in range(shirtsLen) %}
<div class="col-sm">
<div class="card text-center">
<div class="card-body">
```

Roopesh Clothing Store

Log In to Buy

{{ msg }}

Username | Password | Login

Roopesh Clothing Store

Register

{{msg}}

Username

Password

Confirm Password

First Name

Last Name

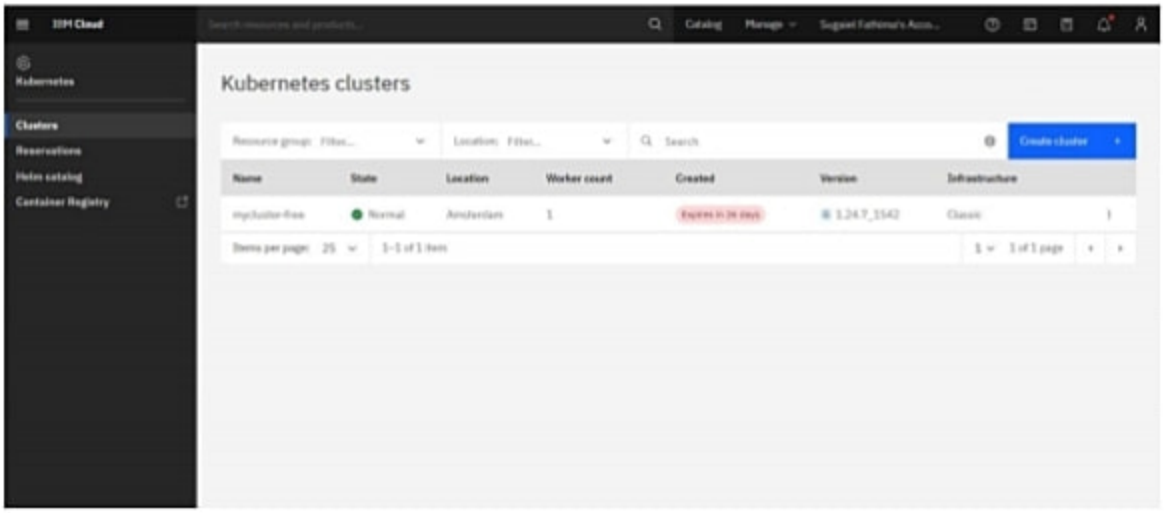
Email

Clear Register

9. RESULT

9.1 Performance Mentries

Kubernetes Cluster



10. ADVANTAGE & DISADVANTAGE

ADVANTAGE:

*Model doesn't need data of other users since recommendations are specific to a single user

*it makes its easier to scale to a large number of users

DISADVANTAGE:

*Feature representation of items is hand engineered to some extent this tech require

11.CONCLUSION:

In this Literature review, we have illustrated a big picture on different research approaches towards fashion recommender systems. We introduced the trajectory of studies in fashion recommender systems from the

very beginning. The main categories have been defined. We clarified what makes developing fashion recommender systems a necessity for the fashion domain, in this contemporary society, as a competitive advantage leveraging the power of data within employing machine learning methods and AI solutions for different purposes, including marketing, decision making, cross-selling, etc. Representing what makes the fashion domain distinguished from other recommender system domains, we conceptualized the sources of complexity in the fashion domain by illustrating how interconnected these concepts are, as a framework that any fashion recommender system can be defined and understood through it. Focusing on image-based fashion recommender systems, we identified four main tasks in fashion recommender systems, bringing their characteristics to the fore, including cloth-item retrievals, Complementary item recommendation, Outfit recommendation, and Capsule wardrobes.

12:FUTURE SCOPE:

Only a selected few are creative and bold enough to choose a ***career in fashion designing in India***. if you are among those few, exciting and colorful (literally) opportunities await you. But, before you jump right into how the life of a fashion designer looks, let's first evaluate what is the scope, what are the career options, and whether a ***diploma in fashion designing*** is what you should consider now.

13.APPENDIX:

Source code

```
import ibm_db
from flask_session import Session
from flask import Flask, render_template, redirect, request, session, jsonify
from datetime import datetime

# # Instantiate Flask object named app
app = Flask(__name__)

# # Configure sessions
app.config["DB2_DATABASE"] = 'bludb'
app.config["DB2_HOSTNAME"] = '9938aec0-8105-433e-8bf9-0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud'
app.config["DB2_PORT"] = '32459'
app.config["DB2_PROTOCOL"] = 'TCPIP'
app.config["DB2_USER"] = 'mqs19694'
app.config["DB2_PASSWORD"] = 'SsDjCqU5ECrgxjRF'

Session(app)

# Creates a connection to the database
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=9938aec0-8105-433e-8bf9-0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32459/bludb:userid=<mqs19694>;password=<r1WN
db = conn.connection.cursor()

@app.route("/")
def index():
    shirts = db.execute("SELECT * FROM shirts ORDER BY onSalePrice")
    shirtsLen = len(shirts)
    # Initialize variables
    shoppingCart = []
    shopLen = len(shoppingCart)
    totItems, total, display = 0, 0, 0
    if 'user' in session:
        shoppingCart = db.execute("SELECT samplename, image, SUM(qty), SUM(subTotal), price, id FROM cart GROUP BY samplename")
        shopLen = len(shoppingCart)
```

```
        for i in range(shopLen):
            total += shoppingCart[i]["SUM(subTotal)"]
            totItems += shoppingCart[i]["SUM(qty)"]
        shirts = db.execute("SELECT * FROM shirts ORDER BY onSalePrice ASC")
        shirtsLen = len(shirts)
        return render_template ("index.html", shoppingCart=shoppingCart, shirts=shirts, shopLen=shopLen, shirtsLen=shirtsLen, total=total, totItems=totItems, display=display,
    return render_template ( "index.html", shirts=shirts, shoppingCart=shoppingCart, shirtsLen=shirtsLen, shopLen=shopLen, total=total, totItems=totItems, display=display)


@app.route("/buy/")
def buy():
    # Initialize shopping cart variables
    shoppingCart = []
    shopLen = len(shoppingCart)
    totItems, total, display = 0, 0, 0
    qty = int(request.args.get('quantity'))
    if session:
        # Store id of the selected shirt
        id = int(request.args.get('id'))
        # Select info of selected shirt from database
        goods = db.execute("SELECT * FROM shirts WHERE id = :id", id=id)
        # Extract values from selected shirt record
        # Check if shirt is on sale to determine price
        if(goods[0]["onSale"] == 1):
            price = goods[0]["onSalePrice"]
        else:
            price = goods[0]["price"]
        samplename = goods[0]["samplename"]
        image = goods[0]["image"]
        subTotal = qty * price
        # Insert selected shirt into shopping cart
        db.execute("INSERT INTO cart (id, qty, samplename, image, price, subTotal) VALUES (:id, :qty, :samplename, :image, :price, :subTotal)", id=id, qty=qty, samplename=samp
```

```
        shoppingCart = db.execute("SELECT samplename, image, SUM(qty), SUM(subTotal), price, id FROM cart GROUP BY samplename")
        shopLen = len(shoppingCart)
        # Rebuild shopping cart
        for i in range(shopLen):
            total += shoppingCart[i]["SUM(subTotal)"]
            totItems += shoppingCart[i]["SUM(qty)"]
        # Select all shirts for home page view
        shirts = db.execute("SELECT * FROM shirts ORDER BY samplename ASC")
        shirtsLen = len(shirts)
        # Go back to home page
        return render_template ("index.html", shoppingCart=shoppingCart, shirts=shirts, shopLen=shopLen, shirtsLen=shirtsLen, total=total, totItems=totItems, display=display,


@app.route("/update/")
def update():
    # Initialize shopping cart variables
    shoppingCart = []
    shopLen = len(shoppingCart)
    totItems, total, display = 0, 0, 0
    qty = int(request.args.get('quantity'))
    if session:
        # Store id of the selected shirt
        id = int(request.args.get('id'))
        db.execute("DELETE FROM cart WHERE id = :id", id=id)
        # Select info of selected shirt from database
        goods = db.execute("SELECT * FROM shirts WHERE id = :id", id=id)
        # Extract values from selected shirt record
        # Check if shirt is on sale to determine price
        if(goods[0]["onSale"] == 1):
            price = goods[0]["onSalePrice"]
        else:
            price = goods[0]["price"]
```

```
        samplename = goods[0]["samplename"]
        image = goods[0]["image"]
        subTotal = qty * price
        # Insert selected shirt into shopping cart
        db.execute("INSERT INTO cart (id, qty, samplename, image, price, subTotal) VALUES (:id, :qty, :samplename, :image, :price, :subTotal)", id=id, qty=qty, samplename=samp
        shoppingCart = db.execute("SELECT samplename, image, SUM(qty), SUM(subTotal), price, id FROM cart GROUP BY samplename")
        shopLen = len(shoppingCart)
        # Rebuild shopping cart
        for i in range(shopLen):
            total += shoppingCart[i]["SUM(subTotal)"]
            totItems += shoppingCart[i]["SUM(qty)"]
        # Go back to cart page
        return render_template ("cart.html", shoppingCart=shoppingCart, shopLen=shopLen, total=total, totItems=totItems, display=display, session=session )


@app.route("/filter/")
def filter():
    if request.args.get('typeClothes'):
        query = request.args.get('typeClothes')
        shirts = db.execute("SELECT * FROM shirts WHERE typeClothes = :query ORDER BY samplename ASC", query=query )
    if request.args.get('sale'):
        query = request.args.get('sale')
        shirts = db.execute("SELECT * FROM shirts WHERE onSale = :query ORDER BY samplename ASC", query=query)
    if request.args.get('id'):
        query = int(request.args.get('id'))
        shirts = db.execute("SELECT * FROM shirts WHERE id = :query ORDER BY samplename ASC", query=query)
    if request.args.get('kind'):
        query = request.args.get('kind')
        shirts = db.execute("SELECT * FROM shirts WHERE kind = :query ORDER BY samplename ASC", query=query)
    if request.args.get('price'):
        query = request.args.get('price')
        shirts = db.execute("SELECT * FROM shirts ORDER BY onSalePrice ASC")
```

```
    shirtsLen = len(shirts)
    # Initialize shopping cart variables
    shoppingCart = []
    shopLen = len(shoppingCart)
    totItems, total, display = 0, 0, 0
    if 'user' in session:
        # Rebuild shopping cart
        shoppingCart = db.execute("SELECT samplename, image, SUM(qty), SUM(subTotal), price, id FROM cart GROUP BY samplename")
        shopLen = len(shoppingCart)
        for i in range(shopLen):
            total += shoppingCart[i]["SUM(subTotal)"]
            totItems += shoppingCart[i]["SUM(qty)"]
        # Render filtered view
        return render_template ("index.html", shoppingCart=shoppingCart, shirts=shirts, shopLen=shopLen, shirtsLen=shirtsLen, total=total, totItems=totItems, display=display,
    # Render filtered view
    return render_template ( "index.html", shirts=shirts, shoppingCart=shoppingCart, shirtsLen=shirtsLen, shopLen=shopLen, total=total, totItems=totItems, display=display)


@app.route("/checkout/")
def checkout():
    order = db.execute("SELECT * from cart")
    # Update purchase history of current customer
    for item in order:
        db.execute("INSERT INTO purchases (uid, id, samplename, image, quantity) VALUES(:uid, :id, :samplename, :image, :quantity)", uid=session["uid"], id=item["id"], samplen
    # Clear shopping cart
    db.execute("DELETE from cart")
    shoppingCart = []
    shopLen = len(shoppingCart)
    totItems, total, display = 0, 0, 0
    # Redirect to home page
    return redirect('/')

```

```python

@app.route("/remove/", methods=["GET"])
def remove():
    # Get the id of shirt selected to be removed
    out = int(request.args.get("id"))
    # Remove shirt from shopping cart
    db.execute("DELETE from cart WHERE id=:id", id=out)
    # Initialize shopping cart variables
    totItems, total, display = 0, 0, 0
    # Rebuild shopping cart
    shoppingCart = db.execute("SELECT samplename, image, SUM(qty), SUM(subTotal), price, id FROM cart GROUP BY samplename")
    shopLen = len(shoppingCart)
    for i in range(shopLen):
        total += shoppingCart[i]["SUM(subTotal)"]
        totItems += shoppingCart[i]["SUM(qty)"]
    # Turn on "remove success" flag
    display = 1
    # Render shopping cart
    return render_template ("cart.html", shoppingCart=shoppingCart, shopLen=shopLen, total=total, totItems=totItems, display=display, session=session )


@app.route("/login/", methods=["GET"])
def login():
    return render_template("login.html")


@app.route("/new/", methods=["GET"])
def new():
    # Render log in page
    return render_template("new.html")


```

```python
    # Retrieve all shirts ever bought by current user
    myShirts = db.execute("SELECT * FROM purchases WHERE uid=:uid", uid=session["uid"])
    myShirtsLen = len(myShirts)
    # Render table with shopping history of current user
    return render_template("history.html", shoppingCart=shoppingCart, shopLen=shopLen, total=total, totItems=totItems, display=display, session=session, myShirts=myShirts, myS


@app.route("/logout/")
def logout():
    # clear shopping cart
    db.execute("DELETE from cart")
    # Forget any user_id
    session.clear()
    # Redirect user to login form
    return redirect("/")


@app.route("/register/", methods=["POST"] )
def registration():
    # Get info from form
    username = request.form["username"]
    password = request.form["password"]
    confirm = request.form["confirm"]
    fname = request.form["fname"]
    lname = request.form["lname"]
    email = request.form["email"]
    # See if username already in the database
    rows = db.execute( "SELECT * FROM users WHERE username = :username ", username = username )
    # If username already exists, alert user
    if len( rows ) > 0:
        return render_template ( "new.html", msg="Username already exists!" )
    # If new user, upload his/her info into the users database
```

```python
    new = db.execute ( "INSERT INTO users (username, password, fname, lname, email) VALUES (:username, :password, :fname, :lname, :email)",
                    username=username, password=password, fname=fname, lname=lname, email=email )
    # Render login template
    return render_template ( "login.html" )


@app.route("/cart/")
def cart():
    if 'user' in session:
        # Clear shopping cart variables
        totItems, total, display = 0, 0, 0
        # Grab info currently in database
        shoppingCart = db.execute("SELECT samplename, image, SUM(qty), SUM(subTotal), price, id FROM cart GROUP BY samplename")
        # Get variable values
        shopLen = len(shoppingCart)
        for i in range(shopLen):
            total += shoppingCart[i]["SUM(subTotal)"]
            totItems += shoppingCart[i]["SUM(qty)"]
    # Render shopping cart
    return render_template("cart.html", shoppingCart=shoppingCart, shopLen=shopLen, total=total, totItems=totItems, display=display, session=session)
```

GitHub link: https://github.com/IBM-EPBL/IBM-Project-28841-1660117280

Project Demo Link: https://drive.google.com/file/d/1ZbacicgkOfKiWPOZIENPLhUokszGX1NI/view?usp=share_link