PROJECT REPORT
SMART FASHION RECOMMENDER

Abstract:

Fashion applications have seen tremendous growth and are now one of the most used programs in the e-commerce field. The needs of people are continuously evolving, creating room for innovation among the applications. One of the tedious processes and presumably the main activities is choosing what you want to wear. Having an AI program that understands the algorithm of a specific application can be of great aid. We are implementing such a chat bot, 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. The application also has two main user interfaces - the user and the admin. The users can interact with the chat bot, search for products, order them from the manufacturer or distributor, make payment transactions, track the delivery, among other examples. The admin interface enables the user to upload products, find how many products have been bought, supervise the stock availability and interact with the buyer regarding the product as reviews.

Introduction

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.

Description

We have developed 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 chat bot. In this project you will be working on two modules:

- Admin
- User

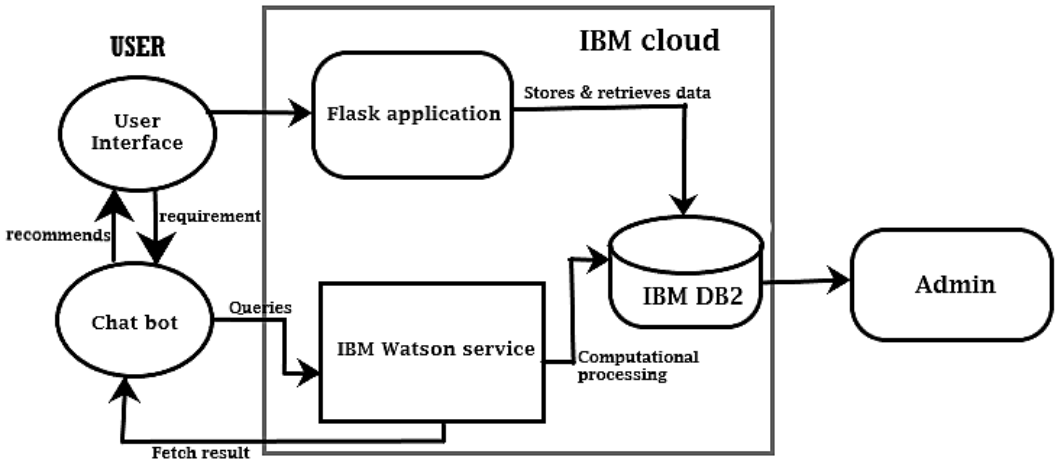
Hardware Requirements:

- 8GB RAM
- Intel Core i3
- Laptop/Desktop
- Windows/MAC/Linux OS.

Software Requirements:

- Python
- Flask
- Docker
- Kubernetes
- IBM DB

System Architecture:



Features:

- Using chat bot we can manage users' choices and orders.
- The chat bot can give recommendations to users based on their interests.
- It can promote the best deals and offers on that day.
- It will store the customer's details and orders in the database.
- The chat bot will send a notification to customers if the order is confirmed.
- Chat bots can also help in collecting customer feedback.

Functional Requirements:

Redirect users to their respective dashboards based on their roles such as admin and customer

- Allow admin to track sales of individual products
- Allow admin to manage orders made by a particular customer.
- Allow users to interact with the chat bot.
- Manage users' choices and charges using the chat bot.
- Promote the best deals and offers.
- Store customer details and orders.
- Send Notifications to customers if the order is confirmed.
- Collect user feedback.
- Recommend products based on user preference.
- Enable online payment features.
- Generate reports for order summary and order histories.

Non-Functional Requirement:

Performance Requirements:

- The system shall be able to handle multiple requests at any given point in time and generate an appropriate response.
- The response should not take longer than five seconds to appear on the client side.
- The client application should lazy load images of the product to minimize network calls over the network.
- The responses from the server should be cached on the client side.

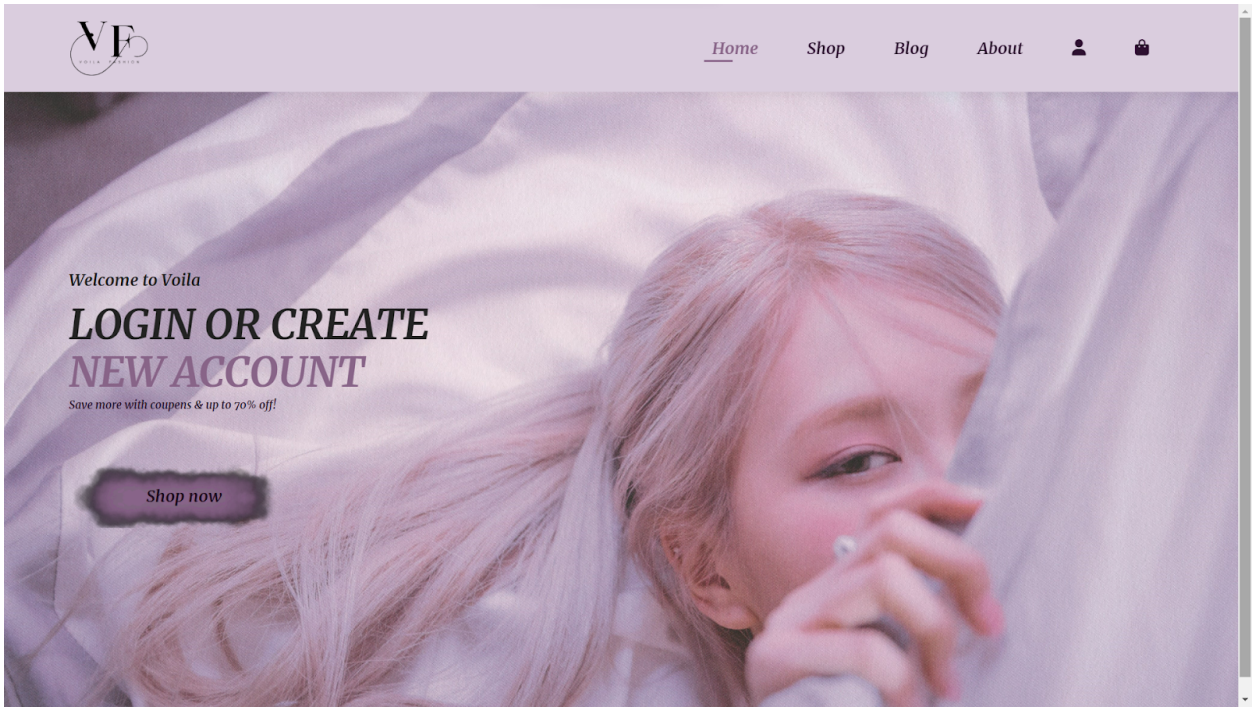
Security Requirements :

- Credentials and secrets should be stored securely and should not be leaked.
- Secured connection HTTPS should be established for transmitting requests and responses between client and server.
- The system has different roles assigned to a user and every user has access constraints.
- User access token should be valid for a shorter period and needs to be refreshed periodically.
- Clients should implement mechanisms to prevent XSS attacks.
- The server should restrict access to the resources for the particular client domain.

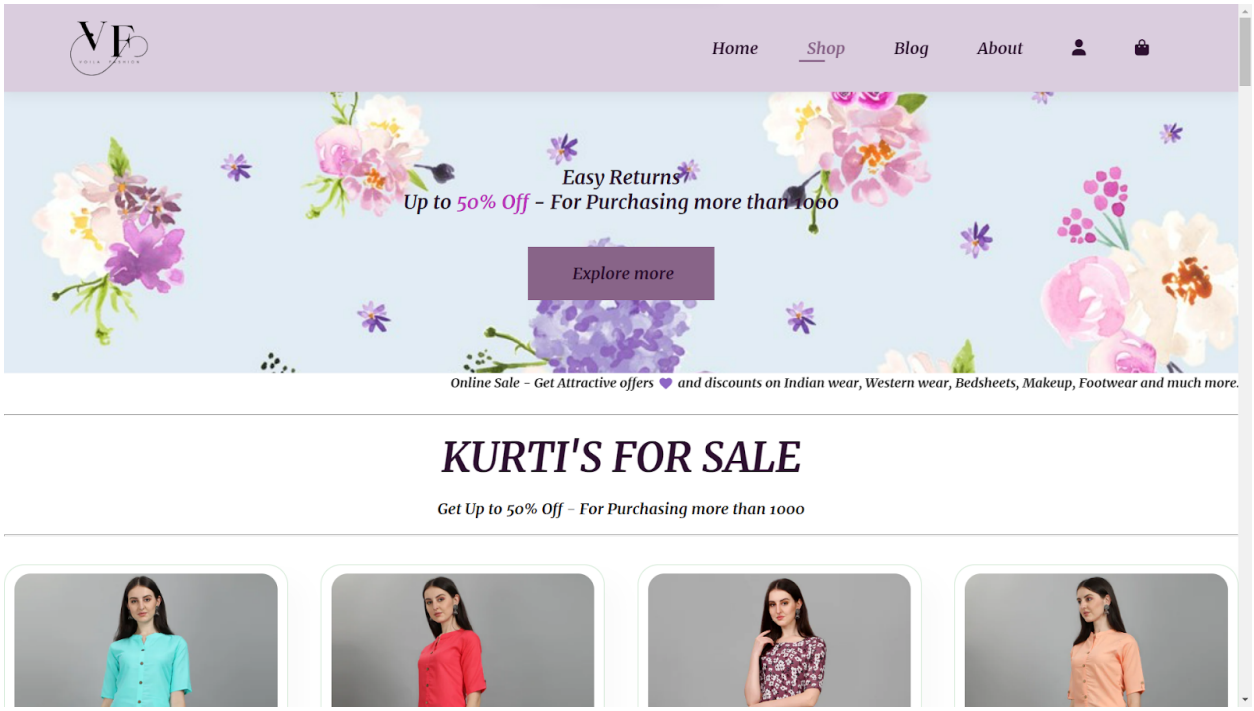
Error Handling:

- The system should handle expected as well as unexpected errors and exceptions to S
- Appropriate error messages should be generated and displayed to the client.

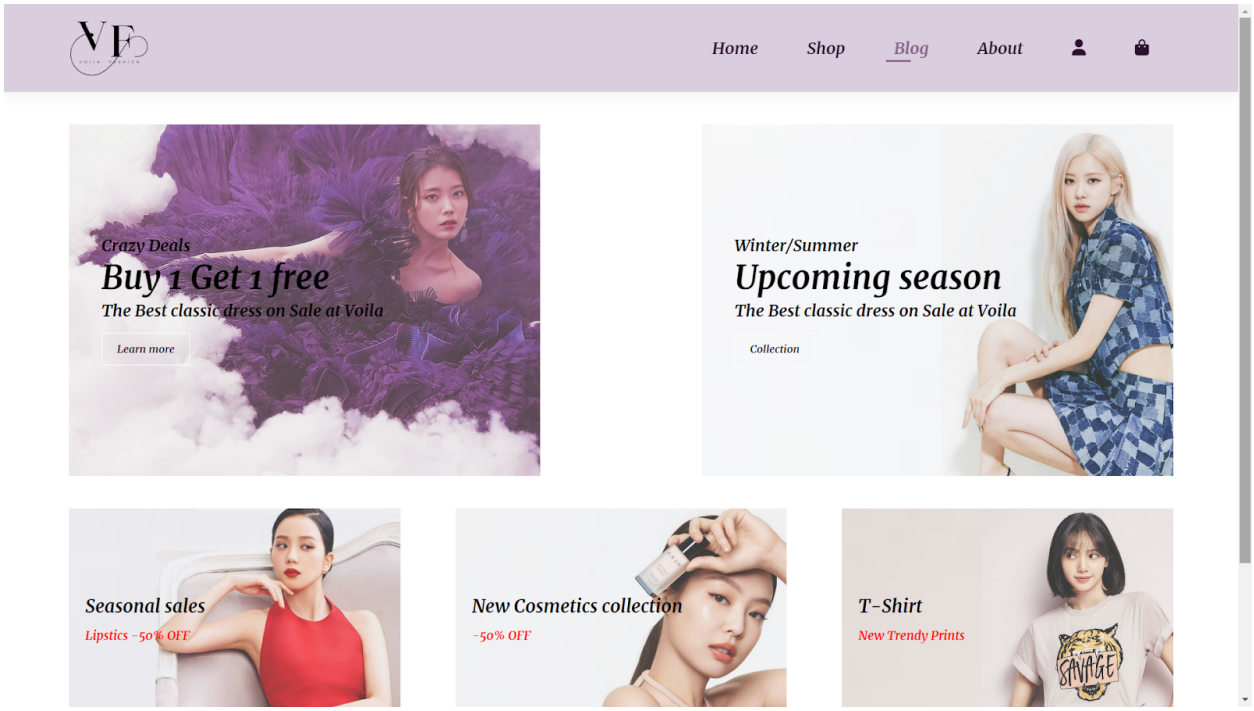
index.html:



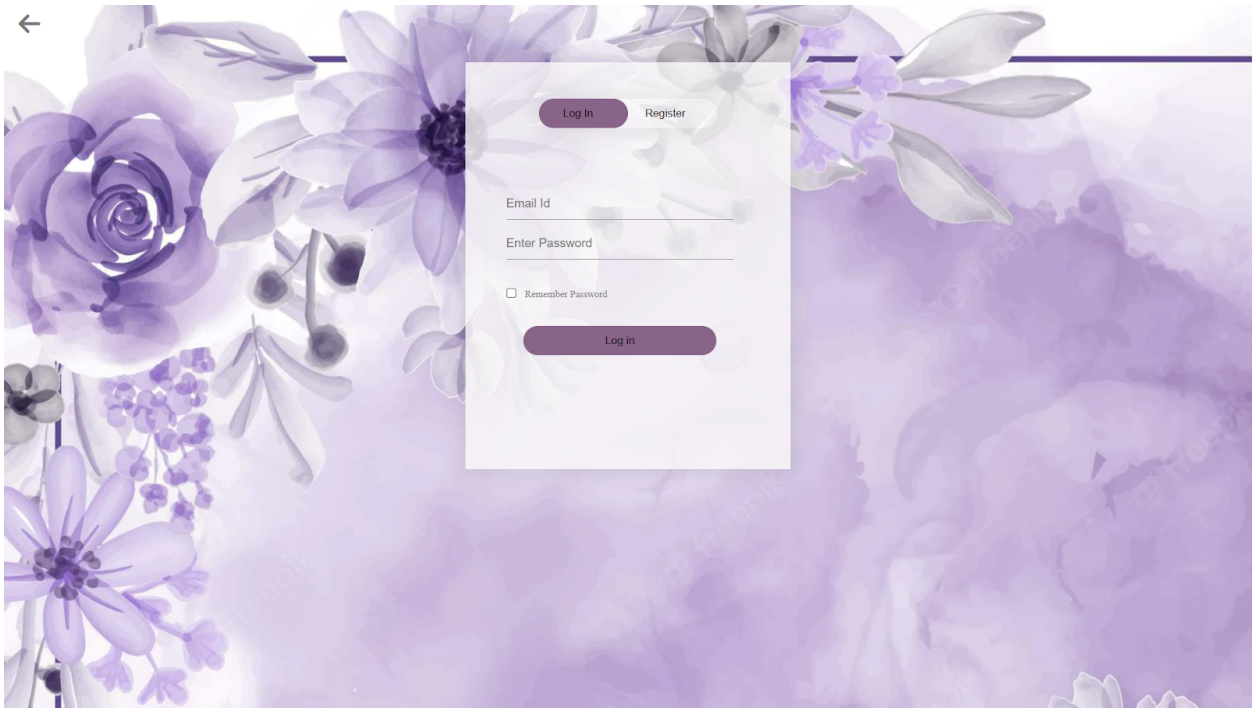
Shop.html:

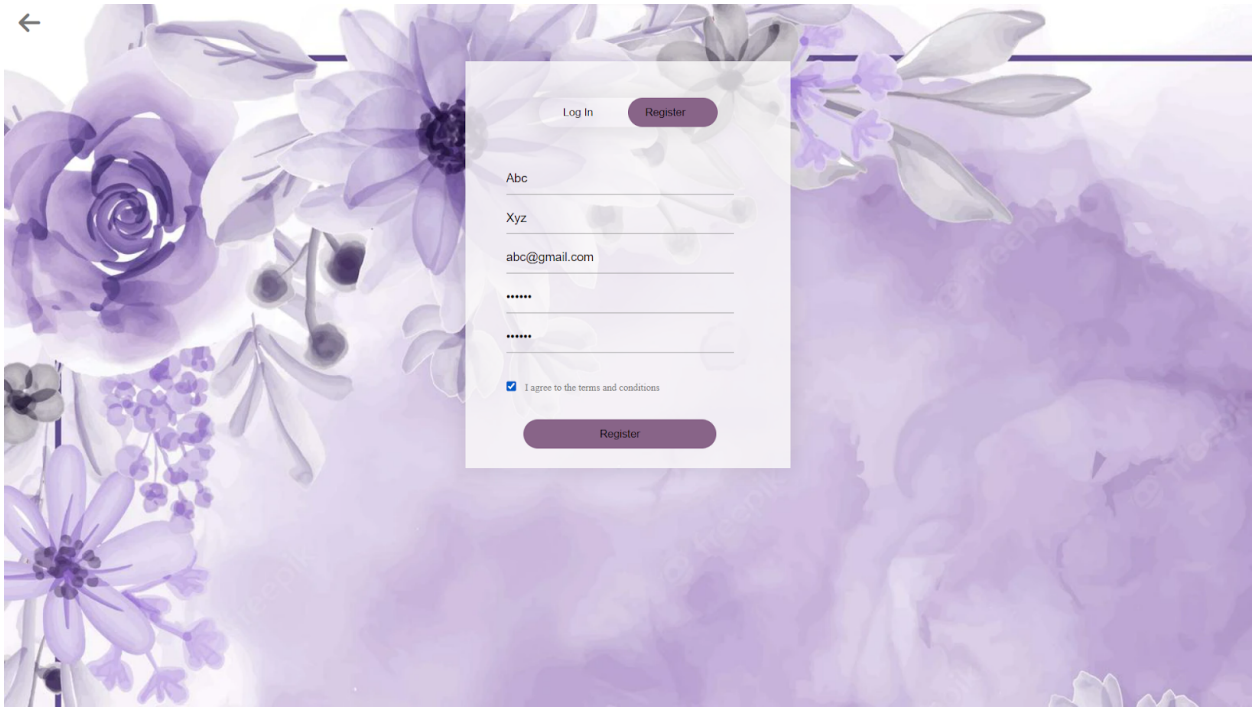


Blog.html:



Login.html:


Log In
Register
Email Id
Enter Password
Remember Password
Log in


Log In
Register
Abc
Xyz
abc@gmail.com
I agree to the terms and conditions
Register

Database.py:

```
import ibm_db
dictionary={}
def printTableData(conn):
   sql = "SELECT * FROM userdetails" out = ibm_db.exec_immediate(conn, sql)document
=ibm_db.fetch_assoc(out)
while:
   document != False:
      dictionary.update({document['EmailID']:document['password']})
      document = ibm_db.fetch_assoc(out)
def insertTableData(Firstname,Lastname,EmailID,password):
   sql="INSERT INTO userdetails(Firstname,Lastname,EmailID,password) VALUES
('{}','{}','{}','{}')".format(Firstname,Lastname,EmailID,password) out =
ibm_db.exec_immediate(conn,sql)
print('Number of affected rows : ',ibm_db.num_rows(out),"\n")
def updateTableData(Firstname,Lastname,EmailID,password):
   sql = "UPDATE userdetails SET ( Firstname,Lastname,email,password)=('{}','{}','{}')
WHERE rollno={}".format(Firstname,Lastname,EmailID,password) out =
ibm_db.exec_immediate(conn,sql)
print('Number of affected rows : ', ibm_db.num_rows(out), "\n")
try:

conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=2d46b6b4-cbf6-40eb-bbce-6251
e6ba0300.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=32328;SECURITY
=SSL;SSLServerCertificate=DigiCertGlo
balRootCA.crt;PROTOCOL=TCPIP;UID=bjn03696;PWD=ef96tLJX2VjzaCPX;", "", "")
print("Db connected")
except:
   print("Error")
```

App.py:

```
from flask import
Flask,render_template,request,url_for,session app=Flask(__name__)
@app.route("/")
@app.route("/login",methods=['POST','GET'])
def login():
   if:
```

```
        request.method=="POST"
        printTableData(conn)
        username=request.form['EmailID']
        password=request.form['password']
        try:
            if dictionary[username] == password and username in dictionary:
                return "Logged in successfully"
            except:
                return "Invalid username or password"
            return
        render_template('login.html')
        @app.route("/register",methods=['POST','GET'])
        def register():
            if request.method=="POST":
                request.form['Firstname'] Firstname =
                request.form['Lastname'] Lastname =
                request.form['EmailID'] email =
                request.form['password'] password=
                insertTableData(Firstname,Lastname,EmailID, password)
                return
            render_template('login.html')
    return
render_template('register.html')
if:
    __name__=="__main__":
app.run(debug=True)
late,request,url_for,session
app=Flask(__name__)
@app.route("/&quo t;)
@app.route("/login",methods=['POST','GET'])
def login():
    if:
        request.method=="POST":
            printTableData(conn)
            username=request.form['username']
            password=request.form['password']
            try:
                if dictionary[EmailID] == password and EmailID in dictionary:
                    return "Logged in successfully"
```

```
    except:
        return "Invalid username or password"
    return
render_template('log')
```

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.