Smart Fashion Recommender Application

IBM-Project-8243-1658912566

TEAMID:PNT2022TMID04912

Submitted by

Aravind R

Arun T S

Arvindkumar S

Gokul Sri Ram S

In partial fulfilment for the award of the degreeof

BACHELOROFTECHNOLOGY in

Computer Science and Engineering

PSNA College of Engineering and Technology

Dindigul-624622

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, and so on. 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.

The rapid progress of computer vision, cloud computing and artificial intelligence combined with the current growing urge for online shopping systems opened an excellent opportunity for the fashion industry. As a result, many studies worldwide are dedicated to modern fashion related applications such as virtual try-on and fashion synthesis.

Traditionally, keywords are used to retrieve images, but such methods require a lot of annotations on the image data, which will lead to serious problems such as inconsistent, inaccurate, and incomplete descriptions, and a huge amount of work.

However, the accelerated evolution speed of the field makes it hard to track these many research branches in a structured framework. Such hierarchical application-based multi-label classification of studies increases the visibility of current research, promotes the field, provides research directions, and facilitates access to related studies.

INDEX

Smart Fashion Recommender Application

1. INTRODUCTION

1.1 ProjectOverview

Instead of searching for products in the search bar and navigating to individual products to find required preferences, this project leverages the use of chatbots to gather all required preferences and recommend products to the user. The solution is implemented in such a way as to improve the interactivity between customers and applications. The chatbot sends messages periodically to notify offers and preferences. For security concerns, this application uses a token to authenticate and authorize users securely. The token has encoded user id and role. Based on the encoded information, access to the resources is restricted to specific users. User is divided base don their roles. The roles comprises of admin and user. Admin is provided access with adding new products, update a product track user details. Users of the application get periodic recommendation via chatbot base don their preference and search. The main purpose of this application is to make process of searching, filtering and ordering of products simple and quickly that enhances the overall user experience. The chatbot is trained by providing different categories of product information and its related details.

1.2 Purpose

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 chatbot, 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 chatbot, search for products, order them from the manufacturer or distributor, make payment transactions, track the delivery, and so on. 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. 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.

2. LITERATURESURVEY

2.1 Existingsolutions

On 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 chatbot 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 of creating products with appropriate categories. The user should be able to mention their preferences using interacting with chatbots. The user must receive a notification on order confirmation/failure. The chatbot 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.

2.2 References

S.No	Name of the Journal	Author/Publisher	Year of Publication	Theme	Inference
1.	**A Review of Modern Fashion Recommender Systems**	Yashar Deldjoo, et al.	2022	The textile and apparel industries havegrown tremendously over the last years. Customers no longer have to visit many stores, stand in long queues, or try on garments in dressing rooms as millions of products are now available in online catalogs.	In this survey, we have analyzed and classified the recommender systems that function in a specific vertical market. This domain presents a unique collection of challenges and sub- problems pertinent to the development of successful recommender systems.
2.	**Fashion Recommendation System**	Aneesh K, et al.	2022	Fashion Recommendation System isused in order to classify the user’s clothes and recommend the most suitable outfit for a given occasion using a recommendation algorithm	The proposed system shows that it can process the user’s clothes from the images, identify the type and color of the outfit and finally recommend the most suitable outfit for the given occasion based on the user’s existing clothes.

3.	**Product Recommender Chatbot**	Neera Sanjay Agashe	2021	This research will recommend the perfumes according to customers moods, likings, etc. Customer just hasto write description of perfume which he/she wants to buy.	This system tries to recognise customers behaviour and then recommend the products according to their interest. Each shopping website has their own way of recommending products and follow different recommender system.
4.	**Image-based fashion recommender systems**	Shaghayegh Shirkhani	2021	This idea aims to provide deeper insight into the fashion recommendersystem domain by focusing on image-based fashion recommender systems considering computer vision advancements.	We can conclude that 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.

5.	**A Survey on Conversational Recommender Systems**	Dietmar Jannach, Ahtsham Manzoor	2021	A complete knowledge on Conversational Recommender Systems(CRS)	They support a task-oriented, multi- turn dialogue with their users. Duringsuch a dialogue, the system can elicitthe detailed and current preferences of the user, provide explanations for the item suggestions, or process feedback by users on the made suggestions.

2.3 ProblemStatementDefinition

Problem Statement 1:

The User Needs a way to Find Trending Fashion Clothes so that Here find the All Collections

Problem Statement 2:

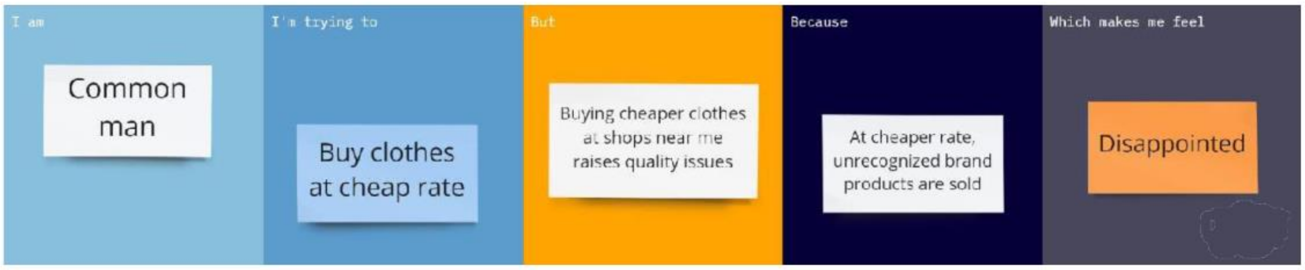
The User Needs a way to Find Offers and Discounts so that Here User easy to find Daily Offers

Problem Statement 3:

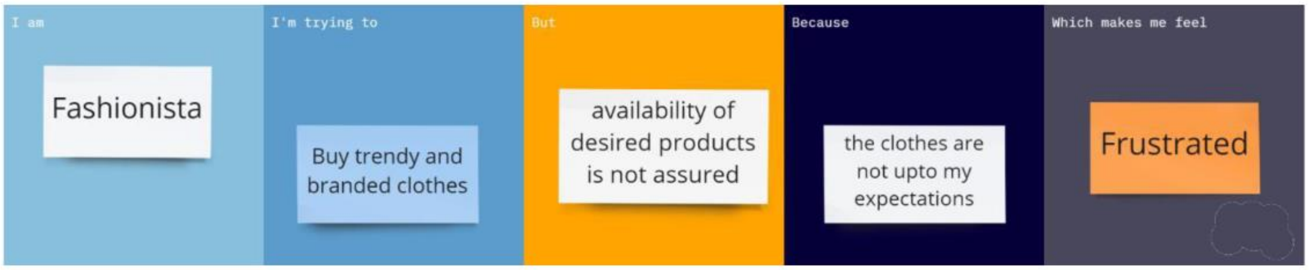
The User Needs a way to Assistant for finding Clothes so that Here User got the Chat Bot assistant

Problem Statement 4:

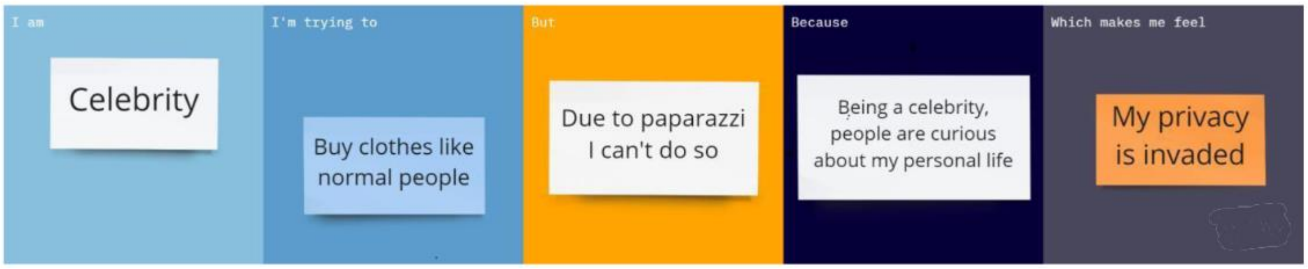
The Sellers Needs a way to struggling to sells products offline so that Here Sellers will Sell Products via our application.



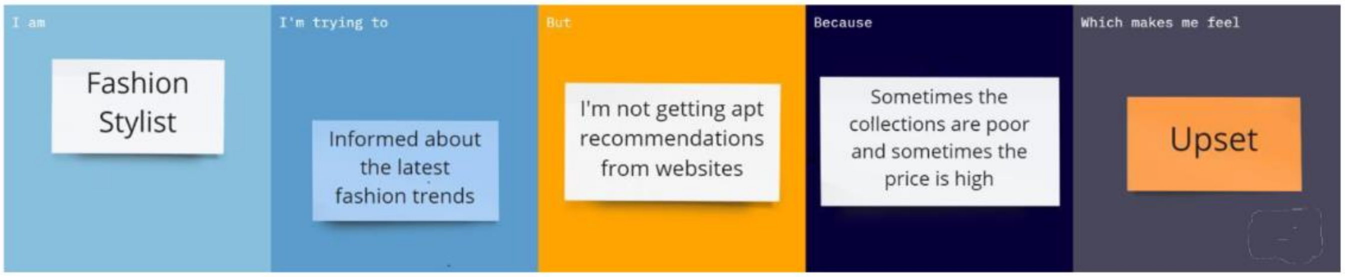
I am	I'm trying to	But	Because	Which makes me feel
Common man	Buy clothes at cheap rate	Buying cheaper clothes at shops near me raises quality issues	At cheaper rate, unrecognized brand products are sold	Disappointed



I am	I'm trying to	But	Because	Which makes me feel
Fashionista	Buy trendy and branded clothes	availability of desired products is not assured	the clothes are not upto my expectations	Frustrated



I am	I'm trying to	But	Because	Which makes me feel
Celebrity	Buy clothes like normal people	Due to paparazzi I can't do so	Being a celebrity, people are curious about my personal life	My privacy is invaded

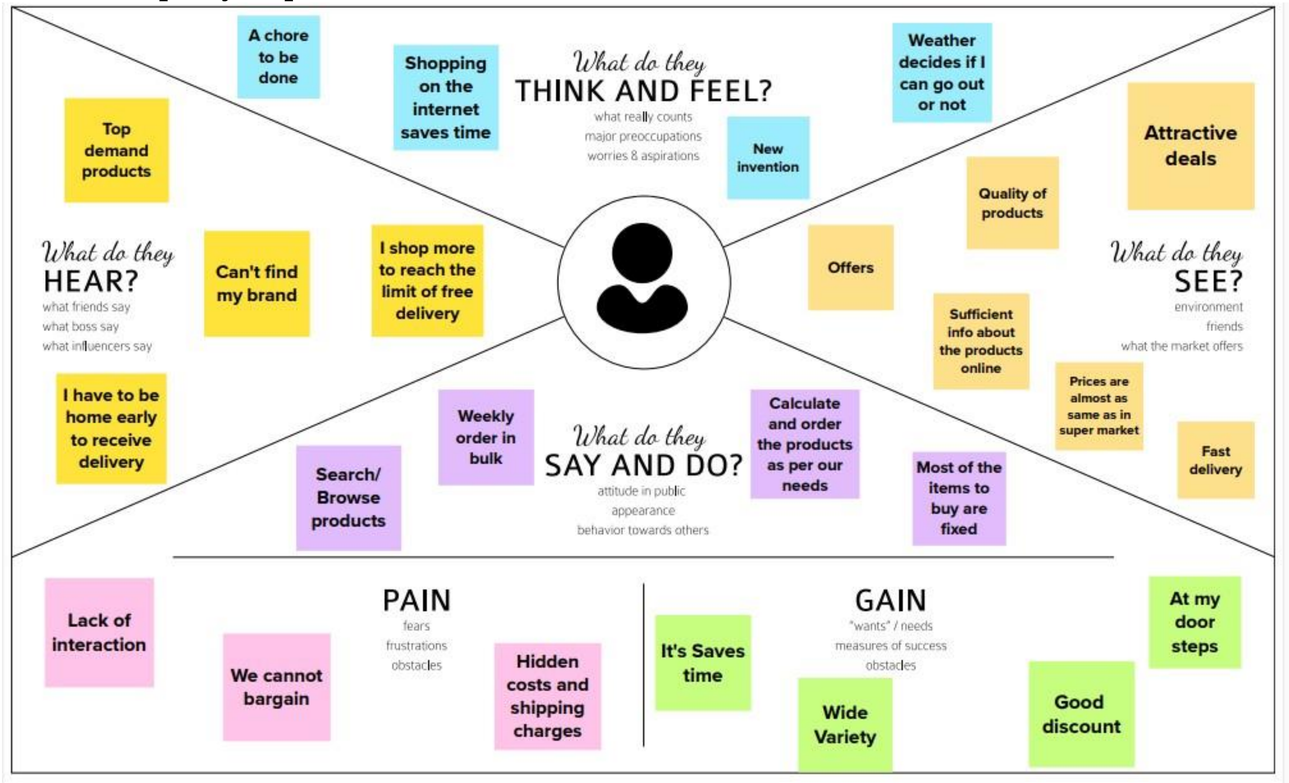


I am	I'm trying to	But	Because	Which makes me feel
Fashion Stylist	Informed about the latest fashion trends	I'm not getting apt recommendations from websites	Sometimes the collections are poor and sometimes the price is high	Upset

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	Common man	Buy clothes at cheap rate	Buying cheaper clothes at shops near me raises quality issues	At cheaper rate, unrecognized brand products are sold	Disappointed
PS-2	Fashionista	Buy trendy and branded clothes	Availability of desired products is not assured	The clothes are not up to my expectations	Frustrated
PS-3	Celebrity	Buy clothes like normal people	Due to paparazzi, I can't do so	Being a celebrity, people are curious about my personal life	My privacy is invaded
PS-4	Fashion Stylist	Informed about the latest fashion trends	I'm not getting apt recommendations from websites	Sometimes the collections are poor and sometimes the price is high	Upset

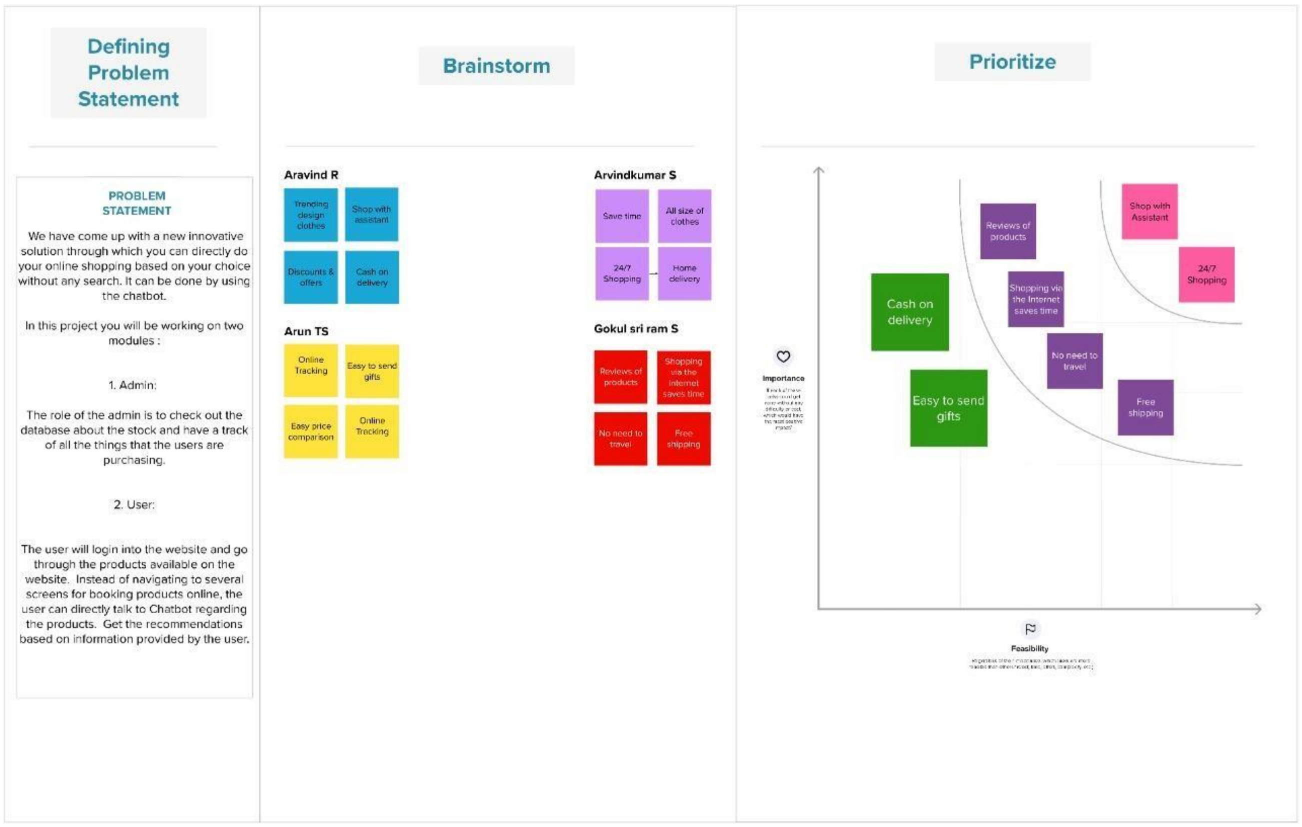
3. IDEATION&PROPOSEDSOLUTION

3.1 EmpathyMapCanvas



3.2 Ideation&Brainstorming

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions. Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room



Defining Problem Statement

PROBLEM STATEMENT

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.

In this project you will be working on two modules :

1. Admin:

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.

2. User:

The user will login into the website and go through the products available on the website. Instead of navigating to several screens for booking products online, the user can directly talk to Chatbot regarding the products. Get the recommendations based on information provided by the user.

3.3 Proposed Solution

Proposed solution Definition :

Proposed Solution refers to the technical response that the Implementation agency will offer in response to the Project's requirements and objectives.

How to write a problem statement

1. Describe how things should work.
2. Explain the problem and state why it matters.
3. Explain your problem's financial costs.
4. Back up your claims.
5. Propose a solution.
6. Explain the benefits of your proposed solution(s).
7. Conclude by summarizing the problem and solution.

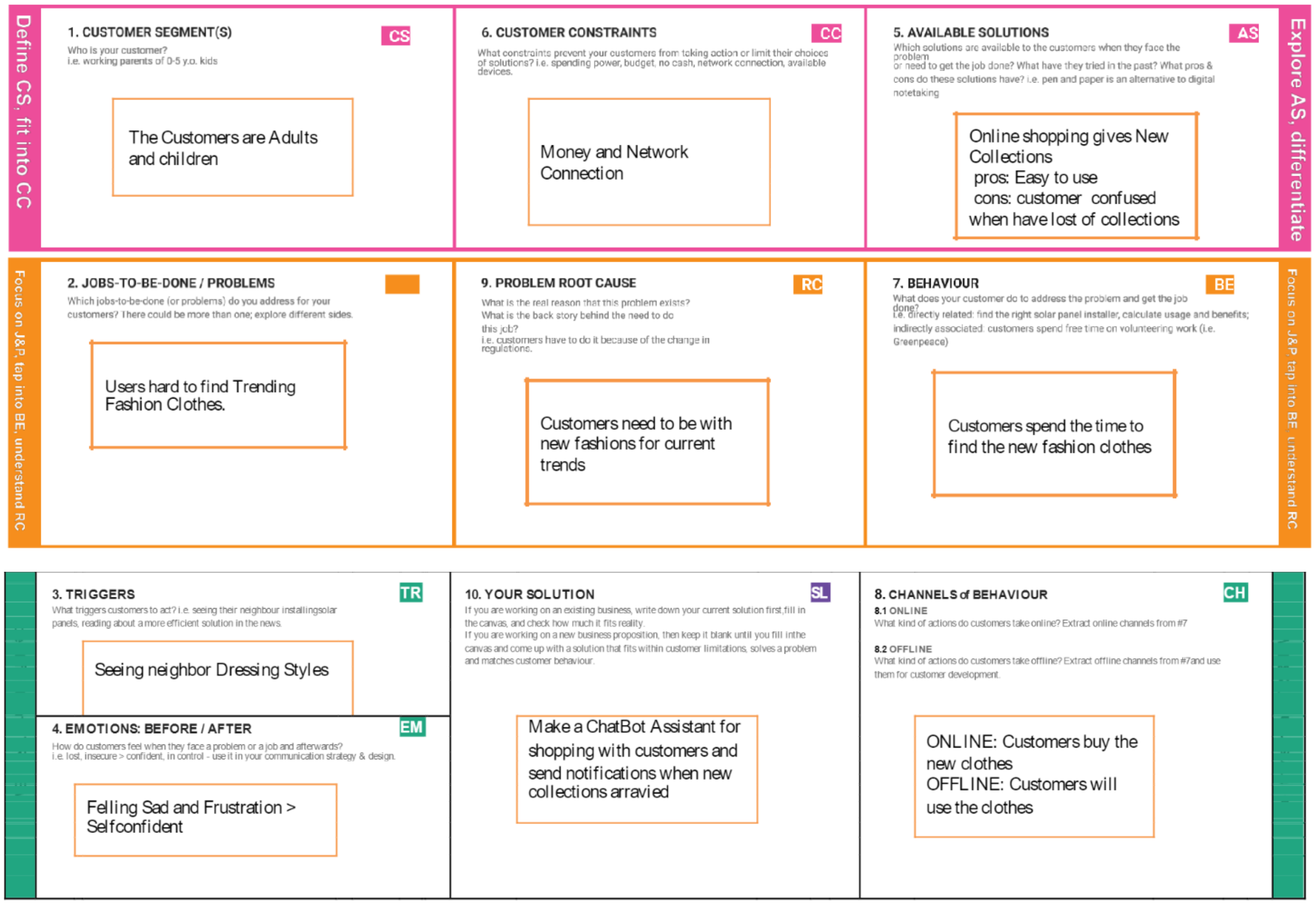
The main goal of presenting a business proposal is to provide a solution to a problem faced by a potential buyer. This section should be as comprehensive as possible, and able to address all the needs that we have pointed out in the first section.

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Customers feels difficult when Search many websites to find Fashion clothes and accessories.
2.	Idea / Solution description	Customers directly make online shopping basedon customer choice without any search.
3.	Novelty / Uniqueness	The customer will talk to Chat Bot regarding theProducts. Get the recommendations based on information provided by the

		user
4.	Social Impact / Customer Satisfaction	The user friendly interface, Assistants form chat bot finding dress makes customersatisfied.
5.	Business Model (Revenue Model)	The chat bot sells our Products to customer. Customers buy our products and generate revenue
6.	Scalability of the Solution	We can easily scalable our Applications by increases the items and products

3.4 Problem Solution Fit

The Lean Startup, LUM (Lazy User Model), and User Experience design tenets serve as the foundation for the Problem-Solution Fit canvas. It aids in the identification of behavioural patterns by business innovators, marketers, and entrepreneurs. • It serves as a template for identifying solutions with the best prospects of being adopted, cutting down on testing time, and getting a clearer picture of the situation as it stands. My objective was to develop a tool that transforms a problem into a solution while considering client behavior and the surrounding circumstances. 17 • With the help of this template, we will be able to thoroughly examine problem solving and take important information into account earlier. • It increases our chances of finding a product-market fit and a problem-solution fit.



Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Who is your customer?
i.e. working parents of 0-5 y.o. kids

The Customers are Adults and children

6. CUSTOMER CONSTRAINTS CC

What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices.

Money and Network Connection

5. AVAILABLE SOLUTIONS AS

Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking

Online shopping gives New Collections
pros: Easy to use
cons: customer confused when have lost of collections

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS

Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.

Users hard to find Trending Fashion Clothes.

9. PROBLEM ROOT CAUSE RC

What is the real reason that this problem exists?
What is the back story behind the need to do this job?
i.e. customers have to do it because of the change in regulations.

Customers need to be with new fashions for current trends

7. BEHAVIOUR BE

What does your customer do to address the problem and get the job done?
i.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)

Customers spend the time to find the new fashion clothes

Focus on J&P, tap into BE, understand RC

3. TRIGGERS TR

What triggers customers to act? i.e. seeing their neighbour installingsolar panels, reading about a more efficient solution in the news.

Seeing neighbor Dressing Styles

4. EMOTIONS: BEFORE / AFTER EM

How do customers feel when they face a problem or a job and afterwards?
i.e. lost, insecure > confident, in control - use it in your communication strategy & design.

Felling Sad and Frustration > Selfconfident

10. YOUR SOLUTION SL

If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality.
If you are working on a new business proposition, then keep it blank until you fill inthe canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour.

Make a ChatBot Assistant for shopping with customers and send notifications when new collections arravied

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE
What kind of actions do customers take online? Extract online channels from #7

8.2 OFFLINE
What kind of actions do customers take offline? Extract offline channels from #7and use them for customer development.

ONLINE: Customers buy the new clothes
OFFLINE: Customers will use the clothes

4. REQUIREMENTANALYSIS

4.1 Functionalrequirement

The functional requirements of the application are:
● Redirect users to their respective dashboards
● Allow admin to track sales of individual products
● Allow admin to manage orders made by a particular customer.
● Allow users to interact with the chatbot.
● Manage users' choices and charges using the chatbot.
● 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.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form
FR-2	User Interaction	Interact through the Chat Bot
FR-3	Buying Products	Through the chat Bot Recommendation
FR-4	Track Products	Ask the Chat Bot to Track my Orders
FR-5	Return Products	Through the chat Bot
FR_6	New Collections	Recommended from chat Bot

4.2 Non-Functionalrequirements

Following are the Non-Functional requirements of the proposed solution. Performance Requirements
● The response should not take longer than 5 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.

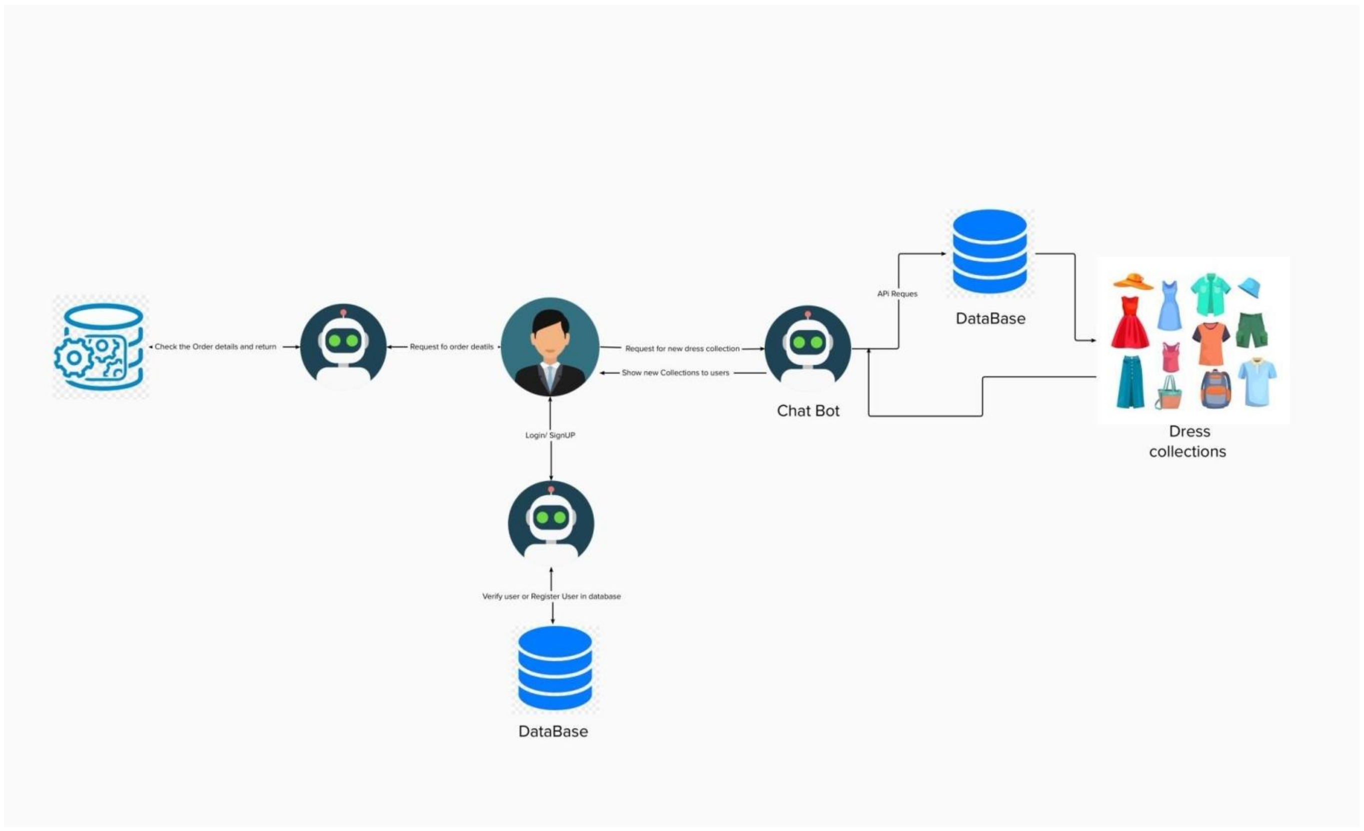
● Clients should implement mechanisms to prevent XSS attacks.

● 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 avoid termination of the program. Appropriate error messages should be generated and displayed to the client.

FR No.	**Non-Functional Requirement**	**Description**
NFR-1	**Usability**	Using Android or IOS or windows applications.
NFR-2	**Security**	The user data is stored securely in IBM cloud.
NFR-3	**Reliability**	The Quality of the services are trusted.
NFR-4	**Performance**	Its Provide smooth user experience.
NFR-5	**Availability**	The services are available for 24/7.
NFR-6	**Scalability**	Its easy to scalable size of users and products.

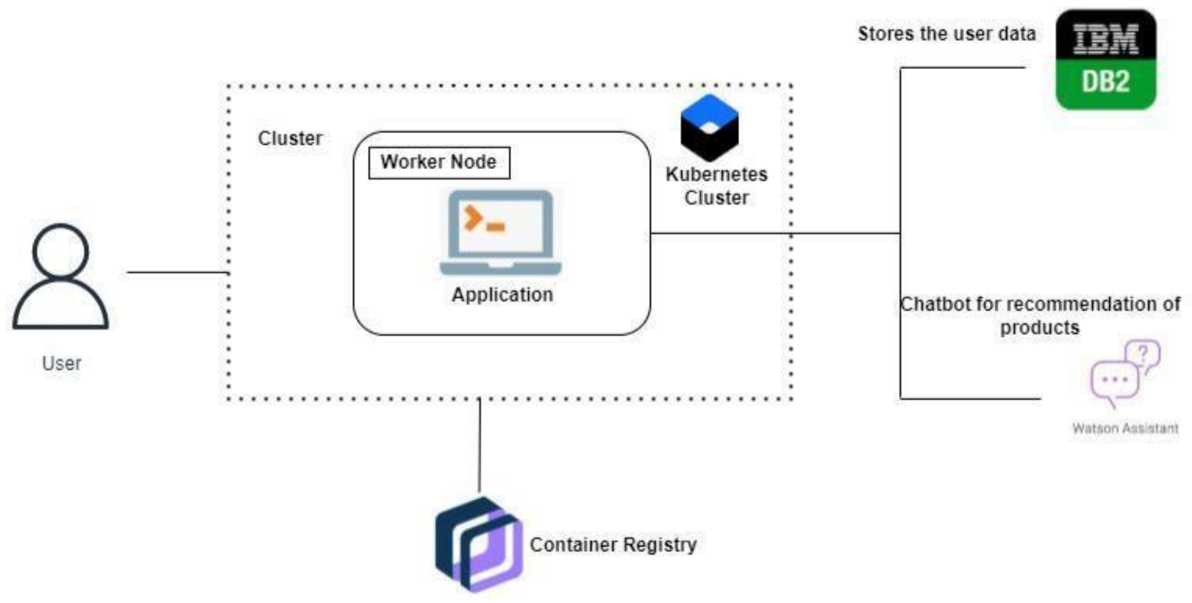
5. PROJECTDESIGN

5.1 DataFlowDiagrams


Check the Order details and return
Request fo order deatils
Request for new dress collection
Show new Collections to users
APi Reques
DataBase
Chat Bot
Dress collections
Login/ SignUP
Verify user or Register User in database
DataBase

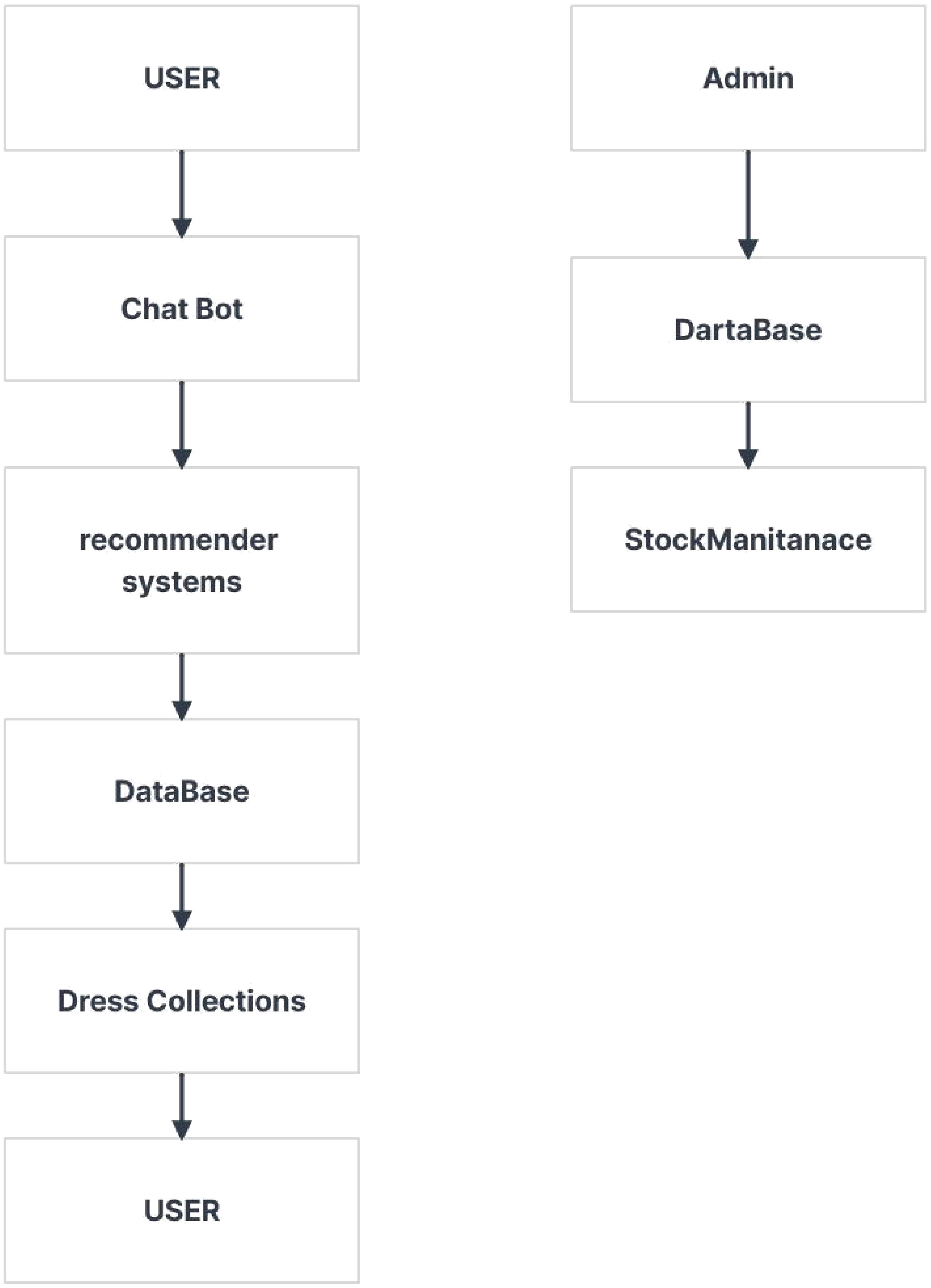
5.2 Solution&TechnicalArchitecture

5.2.1 SolutionArchitecture

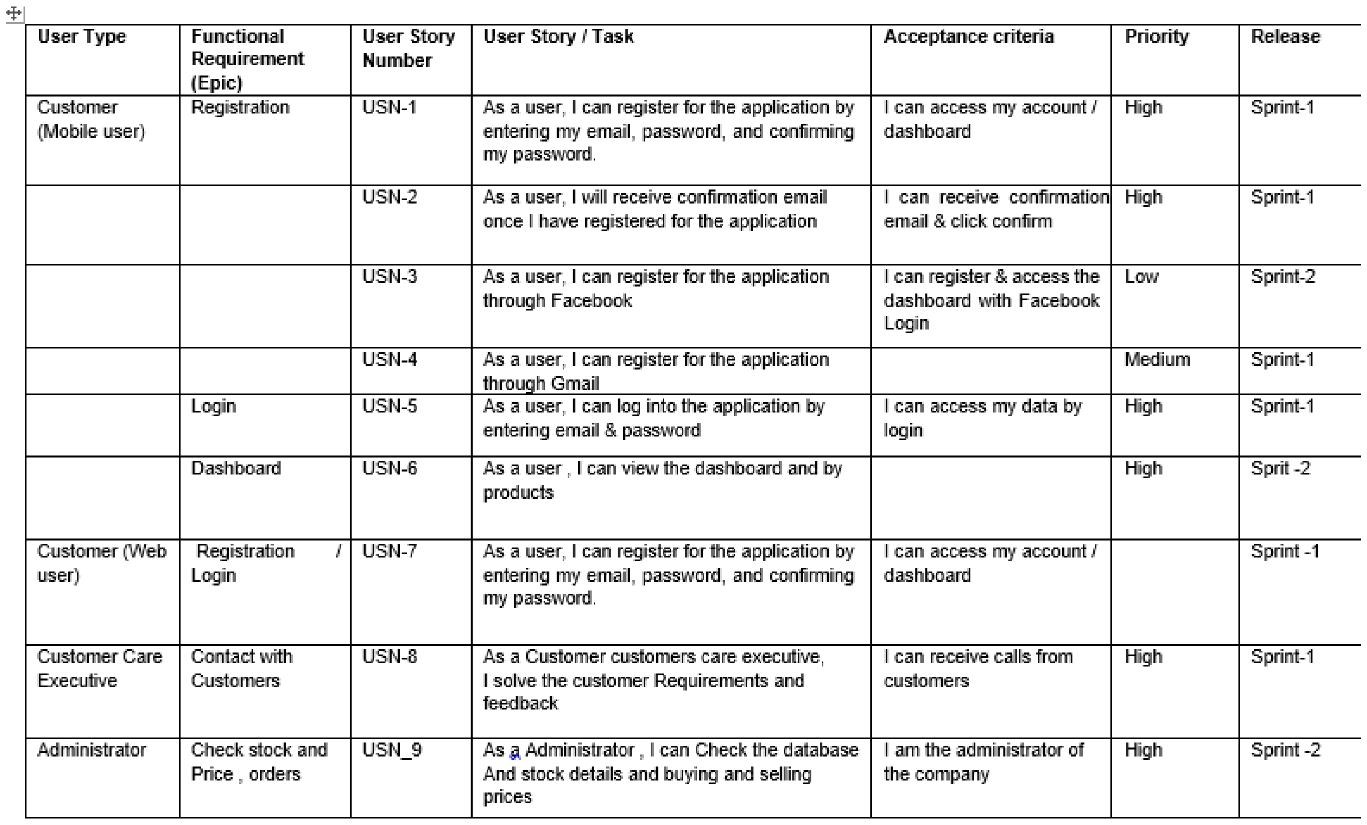

Stores the user data
IBM
DB2
Cluster
Worker Node
Kubernetes Cluster
Application
User
Chatbot for recommendation of products
Watson Assistant
Container Registry

`

5.2.2 TechnicalArchitecture



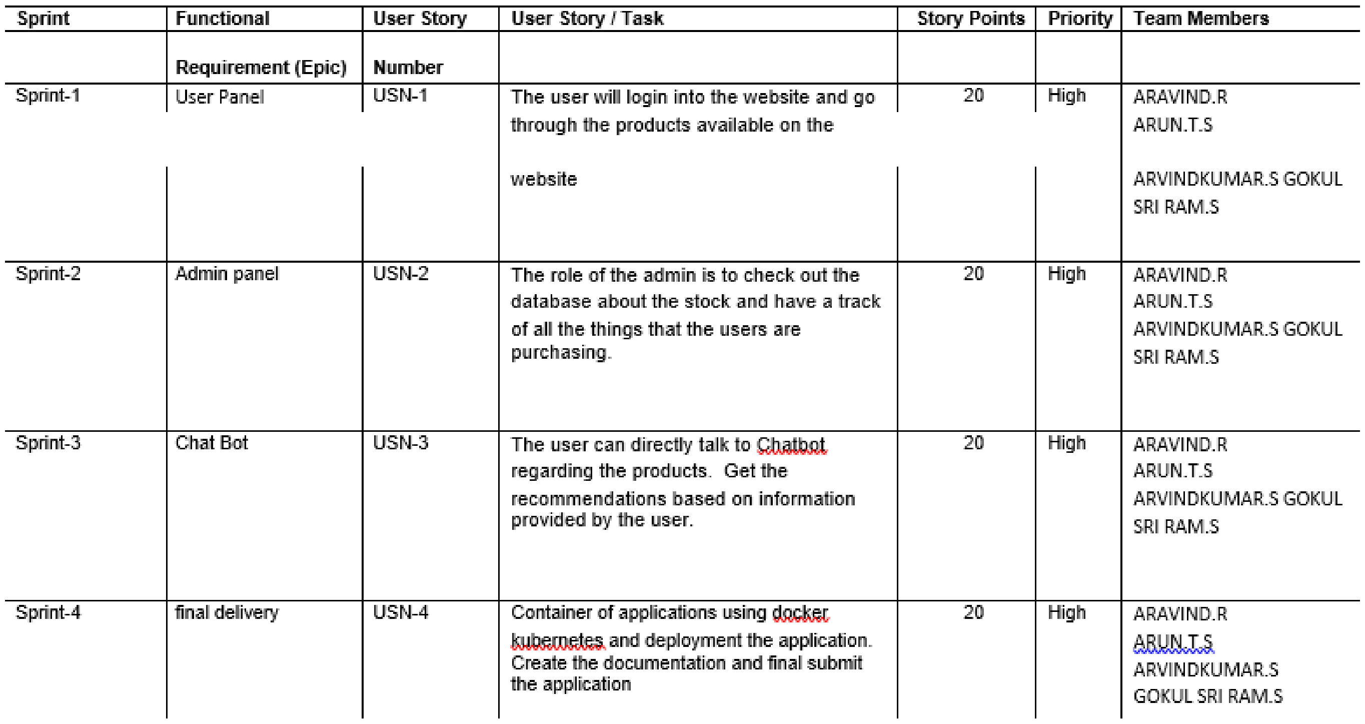
5.3 UserStories



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 my data by login	High	Sprint-1
	Dashboard	USN-6	As a user , I can view the dashboard and by products		High	Sprit -2
Customer (Web user)	Registration / Login	USN-7	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard		Sprint -1
Customer Care Executive	Contact with Customers	USN-8	As a Customer customers care executive, I solve the customer Requirements and feedback	I can receive calls from customers	High	Sprint-1
Administrator	Check stock and Price , orders	USN_9	As a Administrator , I can Check the database And stock details and buying and selling prices	I am the administrator of the company	High	Sprint -2

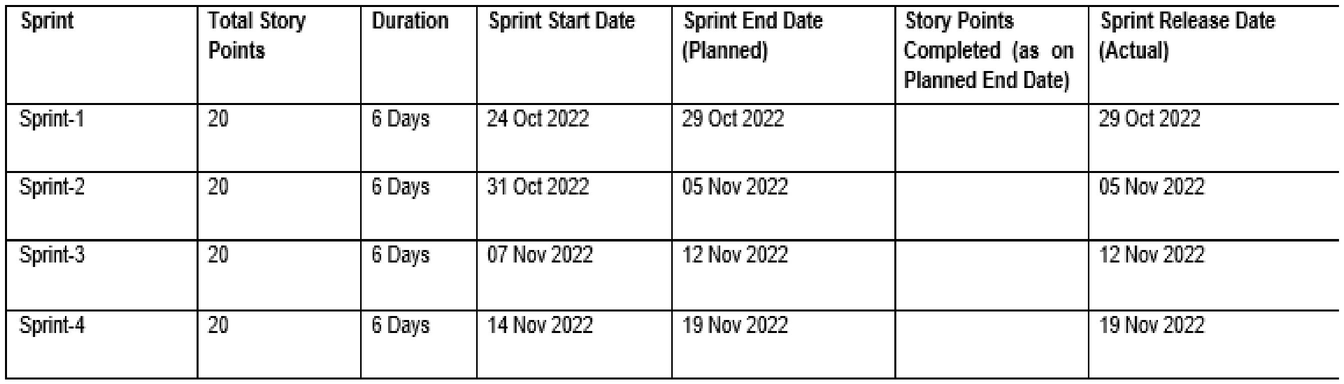
6. PROJECTPLANNING&SCHEDULING

6.1 SprintPlanning&Estimation



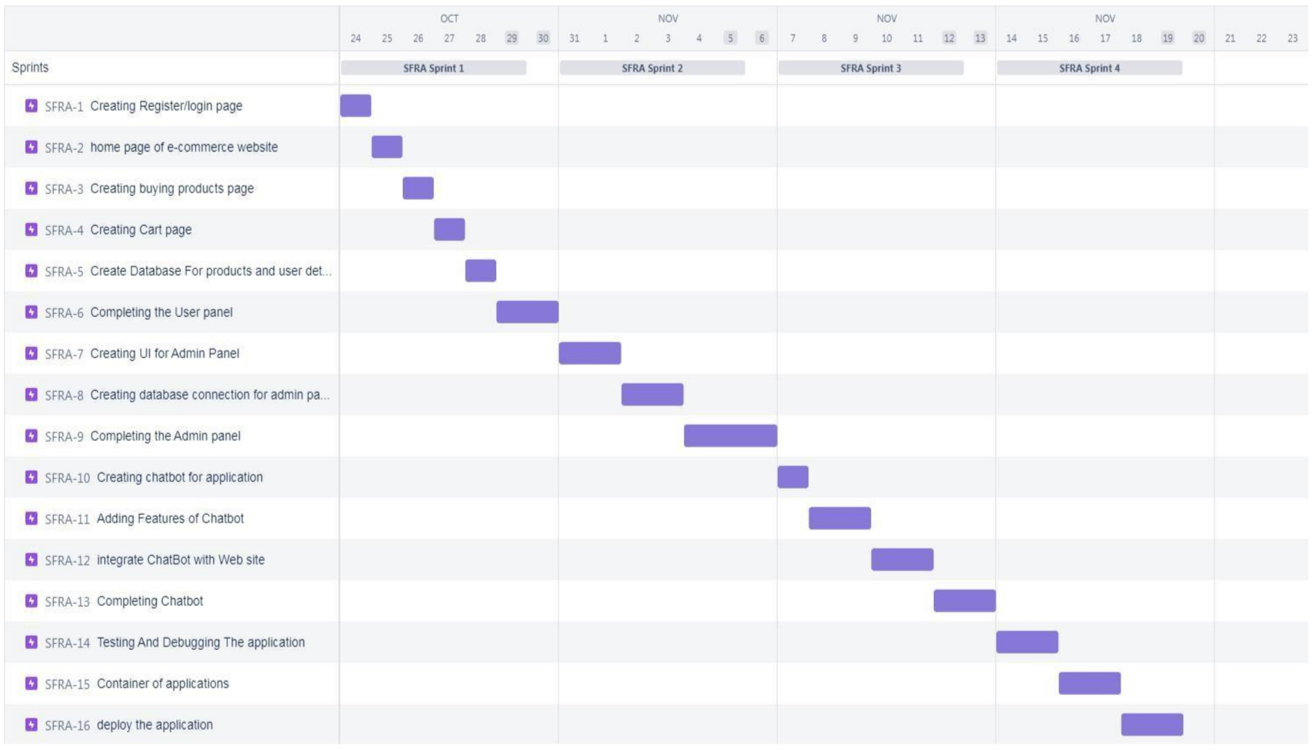
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	User Panel	USN-1	The user will login into the website and go through the products available on the website	20	High	ARAVIND.R ARUN.T.S ARVINDKUMAR.S GOKUL SRI RAM.S
Sprint-2	Admin panel	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	ARAVIND.R ARUN.T.S ARVINDKUMAR.S GOKUL SRI RAM.S
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	ARAVIND.R ARUN.T.S ARVINDKUMAR.S GOKUL SRI RAM.S
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	ARAVIND.R ARUN.T.S ARVINDKUMAR.S GOKUL SRI RAM.S

6.2 SprintDeliverySchedule



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

6.3 Reportsfrom JIRA



Sprints

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

7. CODING&SOLUTIONING

7.1 Feature1

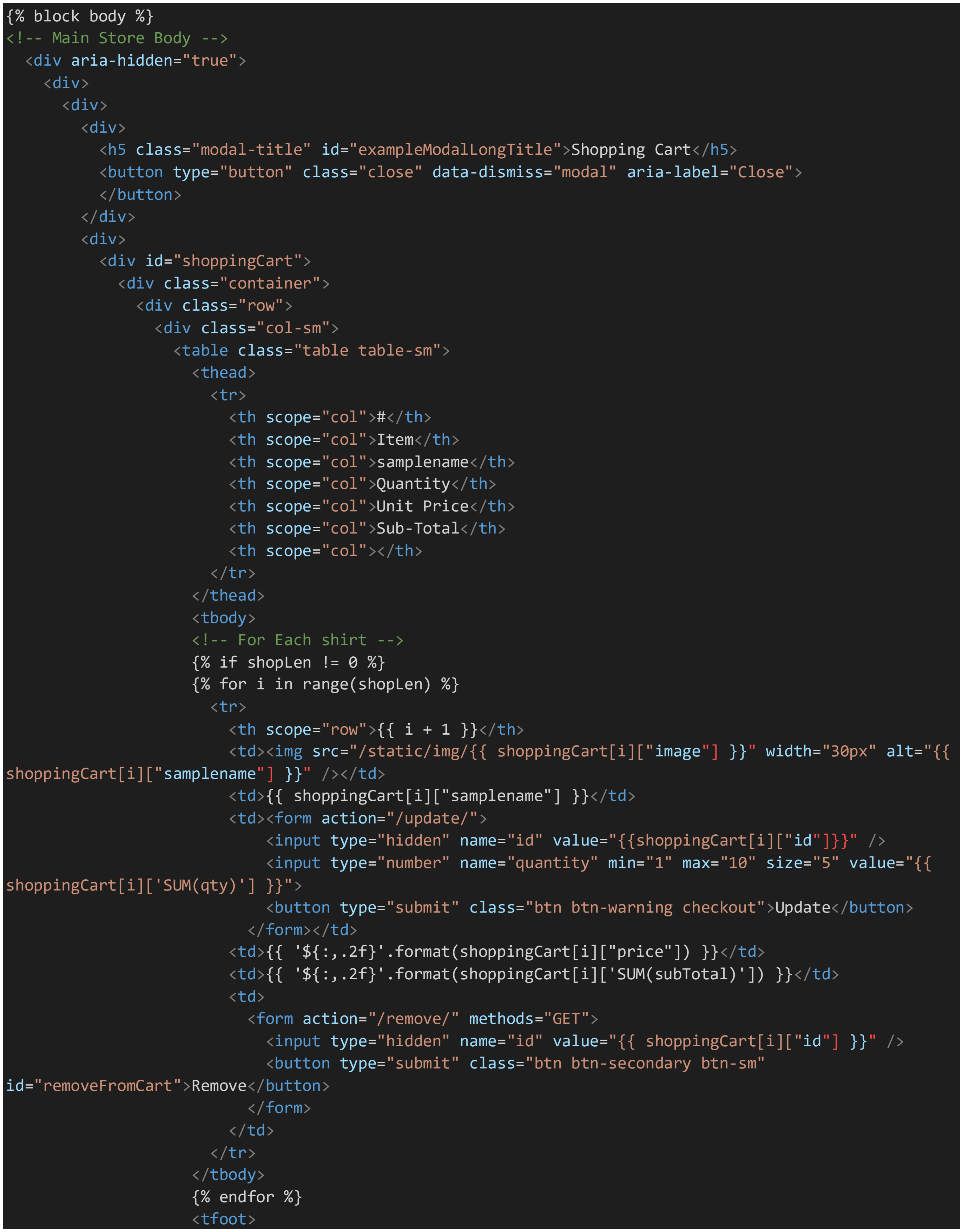
Homepage:

base.html

```html
<!DOCTYPE html>
<html lang="en">

    <head>
        <meta content="text/html;charset=utf-8" http-equiv="Content-Type">
        <meta content="utf-8" http-equiv="encoding">
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
        <meta name="theme-color" content="#000000">
        <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
            crossorigin="anonymous">
        <link href="{{ url_for('static',filename='css/custom.css') }}" rel="stylesheet"
type="text/css" />
        <script defer src="https://use.fontawesome.com/releases/v5.0.6/js/all.js"></script>
        <script src="https://code.jquery.com/jquery-1.11.0.min.js"></script>

        <title>{% block title %}{% endblock %}</title>
    </head>
    <body>
  <!-- Modal -->
  <div class="modal fade" id="modalCenter" tabindex="-1" role="dialog" aria-
labelledby="exampleModalCenterTitle" aria-hidden="true">
    <div class="modal-dialog modal-dialog-centered modal-lg" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title" id="exampleModalLongTitle">Shopping Cart</h5>
          <button type="button" class="close" data-dismiss="modal" aria-label="Close">
            <span aria-hidden="true">&times;</span>
          </button>
        </div>
        <div class="modal-body">
          <div id="shoppingCart">
            <div class="container">
              <div class="row">
                <div class="col-sm">
                  <table class="table table-sm">
                    <thead>
                      <tr>
                        <th scope="col">#</th>
                        <th scope="col">Item</th>
                        <th scope="col">Name</th>
                        <th scope="col">Quantity</th>
                        <th scope="col">Unit Price</th>
                        <th scope="col">Sub-Total</th>
                        <th scope="col"></th>
                      </tr>
                    </thead>
                    <tbody>
                    <!-- For Each shirt -->
                    {% if shopLen != 0 %}
                    {% for i in range(shopLen) %}
                      <tr>
                        <th scope="row">{{ i + 1 }}</th>
                        <td><img src="/static/img/{{ shoppingCart[i]["image"] }}" width="30px" alt="{{
shoppingCart[i]["samplename"] }}" /></td>
                        <td>{{ shoppingCart[i]["samplename"] }}</td>
                        <td>{{ shoppingCart[i]['SUM(qty)'] }}</td>
                        <td>{{ '${:,.2f}'.format(shoppingCart[i]["price"]) }}</td>
                        <td>{{ '${:,.2f}'.format(shoppingCart[i]['SUM(subTotal)']) }}</td><!--
                        <td>
                          <form action="/remove/" methods="GET">
                            <input type="hidden" name="id" value="{{ shoppingCart[i]["id"] }}" />
                            <button type="submit" class="btn btn-secondary btn-sm"
id="removeFromCart">Remove</button>
                          </form>
                        </td>-->
                      </tr>
                    </tbody>
                    {% endfor %}
                    <tfoot>
                      <tr>
                        <td colSpan="7">Total: {{ '${:,.2f}'.format(total) }}<br /><br />
                          <div class="modal-footer">
                            <a href="/cart/"><button type="button" class="btn btn-primary
checkout">Make Changes</button></a>
                            <button type="button" class="btn btn-primary checkout" data-
dismiss="modal">Continue Shopping</button>
                            <a href="/checkout/"><button type="button" class="btn btn-success
checkout">Quick Checkout</button></a>
                          </div>
                        </td>
                      </tr>
                    </tfoot>
                    {% else %}
                      <tr>
                        <td colSpan="7"><h3>Your cart is empty :\</h3></td>
                      </tr>
                    </tbody>
                    <tfoot>
                      <tr>
                        <td colSpan="7">Get some shirts now!<br />
                          <div class="modal-footer">
                            <button type="button" class="btn btn-primary" data-
dismiss="modal">Continue Shopping</button>
                          </div>
                        </td>
                      </tr>
                    </tfoot>
                    {% endif %}
                  </table>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
  <header>
    <nav class="navbar fixed-top navbar-dark bg-dark navbar-expand-sm box-shadow">
      <a href="/" class="navbar-brand d-flex align-items-center">
          <strong><i class="fa  fa-cart-plus"></i> Smart Fashion Recommender Application</strong>
      </a>
      {% if session %}
      <ul class="navbar-nav mr-auto">
        <li class="nav-item"><a href="/logout/" class="nav-link">Logout</a></li>
        <li class="nav-item"><a href="/history/" class="nav-link">You Bought</a></li>
      {% else %}
      <ul class="navbar-nav mr-auto">
        <li class="nav-item"><a href="/new/" class="nav-link">Register</a></li>
        <li class="nav-item"><a href="/login/" class="nav-link">Login</a></li>

      {% endif %}
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" id="navbardrop" data-toggle="dropdown">
            Filter By
          </a>
          <div class="dropdown-menu">
              <a class="dropdown-item" href="/">All</a>
              <a class="dropdown-item" href="/filter/?typeClothes=shirt">Shirts</a>
              <a class="dropdown-item" href="/filter/?typeClothes=pant">Trousers</a>
              <a class="dropdown-item" href="/filter/?typeClothes=shoe">Shoes</a>
              <a class="dropdown-item" href="/filter/?kind=casual">Casual Clothing</a>
              <a class="dropdown-item" href="/filter/?kind=formal">Formal Clothing</a>
              <a class="dropdown-item" href="/filter/?sale=1">On Sale</a>
              <a class="dropdown-item" href="/filter/?price=1">Price $0-$000</a>
          </div>
        </li>
      </ul>
      <div>
        <button class="navbar-toggler" style="display:inline" type="button" data-toggle="modal" data-
target="#modalCenter">
          <span class="glyphicon glyphicon-shopping-cart" data-toggle="modal" data-target="">
            <i class="fas fa-shopping-cart"></i>
            <span class="counter">No. of Items: {{ totItems }}</span>
            <span class="counter">Total: ${{ '{:,.2f}'.format(total) }}</span>
          </span>
        </button>
      </div>
    </nav>
  </header><br />
  <main>
    <div class="container">
      {% if display == 1 %}
      <div class="alert alert-success flashMessage" style="text-align:center">
        <strong>Your item was successfully removed from shopping cart!</strong>
      </div>
      {% endif %}
    {% block body %}{% endblock %}
    <footer>
        <div class="container">
            <div class="row">
                <div class="col-md">
                    <hr />
                    <p>&#169; <a href="/">Smart Fashion Recommender Application</a></p>
                </div>
            </div>
        </div>
    </footer>
    <script>
      window.watsonAssistantChatOptions = {
        integrationID: "614a4315-ff80-4187-8fe4-2fd9b506b723", // The ID of this integration.
        region: "au-syd", // The region your integration is hosted in.
        serviceInstanceID: "9670dcf8-789f-4609-8d7a-6e25c412a9ec", // The ID of your service instance.
        onLoad: function(instance) { instance.render(); }
      };
      setTimeout(function(){
        const t=document.createElement('script');
        t.src="https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +
(window.watsonAssistantChatOptions.clientVersion || 'latest') + "/WatsonAssistantChatEntry.js";
        document.head.appendChild(t);
      });
    </script>

        <!-- jQuery first, then Popper.js, then Bootstrap JS -->
        <script src="https://code.jquery.com/jquery-1.11.0.min.js"></script>
        <!-- <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
            crossorigin="anonymous"></script>-->
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
            crossorigin="anonymous"></script>
        <!-- Custom JS Scripts -->
        <script src="{{ url_for('static',filename='js/myscripts.js') }}"></script>
        <script src="{{ url_for('static',filename='js/validate.js') }}"></script>
    </body>
</html>
```

Cart.html:

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

{% block title %}
Smart Fashion Recommender Application - Home
{% endblock %}
```



```
{% block body %}
<!-- Main Store Body -->
  <div aria-hidden="true">
    <div>
      <div>
        <div>
          <h5 class="modal-title" id="exampleModalLongTitle">Shopping Cart</h5>
          <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          </button>
        </div>
        <div>
          <div id="shoppingCart">
            <div class="container">
              <div class="row">
                <div class="col-sm">
                  <table class="table table-sm">
                    <thead>
                      <tr>
                        <th scope="col">#</th>
                        <th scope="col">Item</th>
                        <th scope="col">samplename</th>
                        <th scope="col">Quantity</th>
                        <th scope="col">Unit Price</th>
                        <th scope="col">Sub-Total</th>
                        <th scope="col"></th>
                      </tr>
                    </thead>
                    <tbody>
                    <!-- For Each shirt -->
                    {% if shopLen != 0 %}
                    {% for i in range(shopLen) %}
                      <tr>
                        <th scope="row">{{ i + 1 }}</th>
                        <td><img src="/static/img/{{ shoppingCart[i]["image"] }}" width="30px" alt="{{
shoppingCart[i]["samplename"] }}" /></td>
                        <td>{{ shoppingCart[i]["samplename"] }}</td>
                        <td><form action="/update/">
                            <input type="hidden" name="id" value="{{shoppingCart[i]["id"]}}" />
                            <input type="number" name="quantity" min="1" max="10" size="5" value="{{
shoppingCart[i]['SUM(qty)'] }}">
                            <button type="submit" class="btn btn-warning checkout">Update</button>
                          </form></td>
                        <td>{{ '${:,.2f}'.format(shoppingCart[i]["price"]) }}</td>
                        <td>{{ '${:,.2f}'.format(shoppingCart[i]['SUM(subTotal)']) }}</td>
                        <td>
                          <form action="/remove/" methods="GET">
                            <input type="hidden" name="id" value="{{ shoppingCart[i]["id"] }}" />
                            <button type="submit" class="btn btn-secondary btn-sm"
id="removeFromCart">Remove</button>
                          </form>
                        </td>
                      </tr>
                    </tbody>
                    {% endfor %}
                    <tfoot>
```

```
                      <tr>
                        <td colSpan="7">Total: {{ '${:,.2f}'.format(total) }}<br /><br />
                          <div class="modal-footer">
                            <a href="/"><button type="button" class="btn btn-primary
checkout">Continue Shopping</button></a>
                            <a href="/checkout/"><button type="button" class="btn btn-success
checkout">Proceed to Checkout</button></a>
                          </div>
                        </td>
                      </tr>
                    </tfoot>
                    {% else %}
                      <tr>
                        <td colSpan="7"><h3>Your cart is empty :\</h3></td>
                      </tr>
                    </tbody>
                    <tfoot>
                      <tr>
                        <td colSpan="7">Get some shirts now!<br />
                          <div>
                            <a href="/"><button type="button" class="btn btn-secondary" data-
dismiss="modal">Continue Shopping</button></a>
                          </div>
                        </td>
                      </tr>
                    </tfoot>
                    {% endif %}
                  </table>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
    </div>
  </main>

{% endblock %}
```

history.html :

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

{% block title %}
Smart Fashion Recommender Application - Home
{% endblock %}

{% block body %}
<!-- Main Store Body -->
      <div class="row">
```

```
        <div class="col-sm">
          <h2>Your Shopping History</h2>
          <p>Items you've bought in the past.</p>
          <table class="table table-sm">
            <thead>
              <tr>
                <th scope="col">#</th>
                <th scope="col">Item</th>
                <th scope="col">Name</th>
                <th scope="col">Quantity</th>
                <th scope="col">Date</th>
                <th scope="col"></th>
              </tr>
            </thead>
            <tbody>
            <!-- For Each shirt -->
            {% for i in range(myShirtsLen) %}
              <tr>
                <th scope="row">{{ i + 1 }}</th>
                <td><img src="/static/img/{{ myShirts[i]["image"] }}" width="30px" alt="{{
myShirts[i]["samplename"] }}" /></td>
                <td>{{ myShirts[i]["samplename"] }}</td>
                <td>{{ myShirts[i]["quantity"] }}</td>
                <td>{{ myShirts[i]["date"] }}</td>
                <td><a href="/filter/?id={{ myShirts[i]["id"] }}"><button type="button" class="btn
btn-warning">Buy Again</button></a></td>
              </tr>
            </tbody>
            {% endfor %}
            <tfoot>
            </tfoot>
          </table>
        </div>
      </div>
    </div>
  </main>

{% endblock %}
```

index.html:

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

{% block title %}
Smart Fashion Recommender Application - 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">
                    <form action="/buy/" methods="POST">
                        <h5 class="card-title">{{shirts[i]["typeClothes"].capitalize()}}</h5>
                      <img src="/static/img/{{shirts[i]["image"]}}" class="shirt" alt="" />
                      <h5 class="card-text">{{shirts[i]["samplename"]}}</h5>
                      {% if shirts[i]["onSale"] %}
                        <img src="/static/img/sale-icon.png" width="26px" />
                        <h4 class="card-text price" style="color:red; display:inline">{{
'{:,.2f}'.format(shirts[i]["onSalePrice"]) }}</h4>
                      {% else %}
                        <h4 class="card-text price">{{ '{:,.2f}'.format(shirts[i]["price"]) }}</h4>
                      {% endif %}
                      <div class="stepper-input">
                          <span class="decrement target">-</span>
                          <input class="quantity" name="quantity" value='0' />
                          <span class="increment target">+</span>
                      </div>
                      <input type="hidden" name="id" value="{{shirts[i]["id"]}}" />
                      {% if not session %}
                      <input type="hidden" name="loggedin" value="0" />
                      {% else %}
                      <input type="hidden" name="loggedin" value="1" />

                      {% endif %}
                      <input type="submit" class="btn btn-primary addToCart" value="Add To Cart" /><br
/><br />
                      <div class="alert alert-danger flashMessage" style="text-align: center;
display:none; font-size:0.9em;"></div>
                    </form>
                  </div>
              </div>
          </div>
      {% endfor %}
      </div>
    </div>
  </main>

{% endblock %}
```

login.html:

```
<!DOCTYPE html>
<html lang="en">
```

```
    <head>
        <meta content="text/html;charset=utf-8" http-equiv="Content-Type">
        <meta content="utf-8" http-equiv="encoding">
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
        <meta name="theme-color" content="#000000">
        <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
            crossorigin="anonymous">
        <link href="{{ url_for('static',filename='css/custom.css') }}" rel="stylesheet"
type="text/css" />
        <script defer src="https://use.fontawesome.com/releases/v5.0.6/js/all.js"></script>
        <script src="https://code.jquery.com/jquery-1.11.0.min.js"></script>

        <title>Smart Fashion Recommender Application - Log In</title>
    </head>
    <body>
    <header>
        <nav class="navbar fixed-top navbar-dark bg-dark navbar-expand-sm box-shadow">
          <a href="/" class="navbar-brand d-flex align-items-center">
              <strong><i class="fa  fa-cart-plus"></i> Smart Fashion Recommender Application</strong>
          </a>
        </nav>
    </header><br />
    <main>
        <div class="container">
            <div class="row">
                <div class="col-sm">
                    <h2>Log In to Buy</h2>
                    <p>{{ msg }}</p>
                    <div>
                        <form action="/logged/" class="form" method="post">
                            <div>
                                <input type="text" name="username" autofocus placeholder="Username">
                                <input type="password" name="password" placeholder="Password">
                                <button type="submit" class="btn btn-primary">Login</button>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </main>
    </body>
</html>
```

new.html:

```
<!DOCTYPE html>
<html lang="en">
    <head>
```

```
        <meta content="text/html;charset=utf-8" http-equiv="Content-Type">
        <meta content="utf-8" http-equiv="encoding">
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
        <meta name="theme-color" content="#000000">
        <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
            crossorigin="anonymous">
        <link href="{{ url_for('static',filename='css/custom.css') }}" rel="stylesheet"
type="text/css" />
        <script defer src="https://use.fontawesome.com/releases/v5.0.6/js/all.js"></script>
        <script src="https://code.jquery.com/jquery-1.11.0.min.js"></script>

        <title>Smart Fashion Recommender Application - Register</title>
    </head>
    <body>
        <header>
            <nav class="navbar fixed-top navbar-dark bg-dark navbar-expand-sm box-shadow">
              <a href="/" class="navbar-brand d-flex align-items-center">
                  <strong><i class="fa fa-shopping-bag"></i> Smart Fashion Recommender
Application</strong>
              </a>
            </nav>
        </header><br />
        <main>
            <div class="container">
                <div class="row">
                    <div class="col-sm">
                        <h2>Register</h2>
                        <p>{{msg}}</p>
                        <form action="/register/" class="form" method="post">
                            <input type="text" name="username" id="username" placeholder="Username"
autofocus required > <span id="user-msg" class="alert alert-danger"></span><br /><br />
                            <input type="password" name="password" id="password"
placeholder="Password" required > <span id="password-msg" class="alert alert-danger"></span><br /><br
/>
                            <input type="password" name="confirm" id="confirm" placeholder="Confirm
Password" required> <span id="confirm-msg" class="alert alert-danger"></span><br /><br />
                            <input type="text" name="fname" id="fname" placeholder="First Name"
required> <span id="fname-msg" class="alert alert-danger"></span><br /><br />
                            <input type="text" name="lname" id="lname" placeholder="Last Name"
required> <span id="lname-msg" class="alert alert-danger"></span><br /><br />
                            <input type="email" name="email" id="email" placeholder="Email" required>
<span id="email-msg" class="alert alert-danger"></span><br /><br /><br />
                            <button type="reset" class="btn btn-secondary">Clear</button>
                            <button type="submit" id="submit" class="btn btn-
primary">Register</button>
                        </form>
                    </div>
                </div>
            </div>
        </main>
    <!-- Custom JS Scripts -->
```

```
        <script src="{{ url_for('static',filename='js/validate.js') }}"></script>
    </body>
</html>
```

custom.css:

```
.card:hover {
    border-color:red;
    box-shadow: 1px 2px red;
}

.card {
    margin-bottom: 1em;
    background-color: pink;
}

.price {
    color: seagreen;
    font-weight: bold;
}

.price:before {
    content: '$';
}

.shirt {
    margin-bottom: 10px;
    width: 200px;
    height: 200px;
}

.stepper-input{
    display: flex;
    display: -webkit-flex;
    color: #222;
    max-width: 120px;
    margin: 10px auto;
    text-align: center;
}

header {
    margin-bottom: 50px;
}

.shirtCart {
    width: 25px;
}

.add {
    text-transform: uppercase;
    font-size: 0.8em;
    font-weight: bold;
    color: white;
```

```css
}

.checkout {
    text-transform: uppercase;
    font-size: 0.8em;
    font-weight: bold;
}

.add:hover {
    background-color: sandybrown;
    border-color: sandybrown;
}

tr {
    text-align: center;
}

.modal-header {
    border-bottom: 0px;
}

.counter {
    font-size: 0.6em;
    margin-left: 1em;
    font-weight: bold;
}


.increment,
.decrement{
        height: 24px;
        width: 24px;
        border: 1px solid #222;
        text-align: center;
        box-sizing: border-box;
        border-radius: 50%;
        text-decoration: none;
        color: #222;
        font-size: 24px;
        line-height: 22px;
        display: inline-block;
        cursor: pointer;
    }

.decrement:hover,
.increment:hover {
    color: green;
}

.decrement:active,
.increment:active {
    background-color: green;
    color: white;
}
```

```css
.quantity{
        height: 24px;
        width: 48px;
        text-align: center;
        margin: 0 12px;
        border-radius: 2px;
        border: 1px solid #222;

}

body {
    margin: 0;
    font-family: -apple-system,BlinkMacSystemFont,"Segoe UI",Roboto,"Helvetica Neue",Arial,sans-
serif,"Apple Color Emoji","Segoe UI Emoji","Segoe UI Symbol";
    font-size: 1rem;
    font-weight: 400;
    line-height: 1.5;
    color: #212529;
    text-align: left;
    background-color: beige;
}

.bg-dark {
    background-color: grey!important;
}
```

myscript.js :

```javascript
$(".target").on("click", function() {
    let $button = $(this);
    let oldVal = parseInt($button.parent().find("input").val());
    let newVal = 0;

    if ($button.text() == '+') {
        newVal = oldVal + 1;
    }

    else {
        if (oldVal > 0) {
            newVal = oldVal - 1;
        }
        else {
            newVal = 0;
        }
    }

    $button.parent().find("input").val(newVal);
});
```

```
$('.addToCart').on("click", function(event) {
    console.log('hello');
    if($(this).prev().prev().prev().find("input").val() == '0') {
        event.preventDefault();
        $(this).next().next().next().html("You need to select at least one clothing.");
        $(this).next().next().next().css("display", "block");
        $(this).next().next().next().delay(3000).slideUp();
    }

    if ($(this).prev().val() == "0") {
            event.preventDefault();
            $(this).next().next().next().html("You need to log in to buy.");
            $(this).next().next().next().css("display", "block");
            $(this).next().next().next().delay(3000).slideUp();
        }
});


$(".flashMessage").delay(3000).slideUp();
```

Validate.js :

```
// The submit button
const SUBMIT = $( "#submit" );

// Each of the fields and error message divs
const USERNAME = $( "#username" );
const USERNAME_MSG = $( "#user-msg" );

const PASSWORD = $( "#password" );
const PASSWORD_MSG = $( "#password-msg" );

const CONFIRM = $( "#confirm" );
const CONFIRM_MSG = $( "#confirm-msg" );

const FNAME = $( "#fname" );
const FNAME_MSG = $( "#fname-msg" );

const LNAME = $( "#lname" );
const LNAME_MSG = $( "#lname-msg" );

const EMAIL = $( "#email" );
const EMAIL_MSG = $( "#email-msg" );

/**
 * Resets the error message fields and makes the submit
 * button visible.
 */
function reset_form ( )
{
    USERNAME_MSG.html( "" );
    USERNAME_MSG.hide();
    PASSWORD_MSG.html( "" );
    PASSWORD_MSG.hide();
    CONFIRM_MSG.html( "" );
    CONFIRM_MSG.hide();
    LNAME_MSG.html( "" );
    LNAME_MSG.hide();
    FNAME_MSG.html( "" );
    FNAME_MSG.hide();
    EMAIL_MSG.html( "" );
    EMAIL_MSG.hide();
    SUBMIT.show();
}

/**
 * Validates the information in the register form so that
 * the server is not required to check this information.
 */
function validate ( )
{
    let valid = true;
    reset_form ( );


    // This currently checks to see if the username is
    // present and if it is at least 5 characters in length.
    if ( !USERNAME.val() || USERNAME.val().length < 5  )
    {
        // Show an invalid input message
        USERNAME_MSG.html( "Username must be 5 characters or more" );
        USERNAME_MSG.show();
        // Indicate the type of bad input in the console.
        console.log( "Bad username" );
        // Indicate that the form is invalid.
        valid = false;
    }
    // TODO: Add your additional checks here.


    if ( USERNAME.val() != USERNAME.val().toLowerCase())
    {
        USERNAME_MSG.html("Username must be all lowercase");
        USERNAME_MSG.show();
        valid = false;
    }

    if ( !PASSWORD.val() || PASSWORD.val().length < 8 )
    {
        PASSWORD_MSG.html("Password needs to be at least 8 characters long");
        PASSWORD_MSG.show();
        valid = false;
    }

    if ( !CONFIRM.val() || PASSWORD.val() != CONFIRM.val() )
    {
        CONFIRM_MSG.html("Passwords don't match");
        CONFIRM_MSG.show();
        valid = false;
    }

    if ( !FNAME.val() )
    {
        FNAME_MSG.html("First name must not be empty");
        FNAME_MSG.show();
        valid = false;
    }

    if ( !LNAME.val() )
    {
        LNAME_MSG.html("Last name must not be empty");
        LNAME_MSG.show();
        valid = false;
    }

    var x = EMAIL.val().trim();
    var atpos = x.indexOf("@");
    var dotpos = x.lastIndexOf(".");
    if ( atpos < 1 || dotpos < atpos + 2 || dotpos + 2 >= x.length ) {
        EMAIL_MSG.html("You need to enter a valid email address");
        EMAIL_MSG.show();
        valid = false;
    }

    // If the form is valid, reset error messages
    if ( valid )
    {
        reset_form ( );
    }
}

// Bind the validate function to the required events.
$(document).ready ( validate );
USERNAME.change ( validate );
PASSWORD.change ( validate );
CONFIRM.change ( validate );
LNAME.change ( validate );
FNAME.change ( validate );
EMAIL.change ( validate );
```

Python:

application.py:

```
from cs50 import SQL
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["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
Session(app)

# Creates a connection to the database
db = SQL ( "sqlite:///data.db" )


@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, session=session
)
    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=samplename, image=image,
price=price, subTotal=subTotal)
        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, session=session
)


@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=samplename, image=image,
price=price, subTotal=subTotal)
        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, session=session
)
    # 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"], samplename=item["samplename"],
image=item["image"], quantity=item["qty"] )
    # 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")


@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
    # 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, myShirtsLen=myShirtsLen)


@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
    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)
```

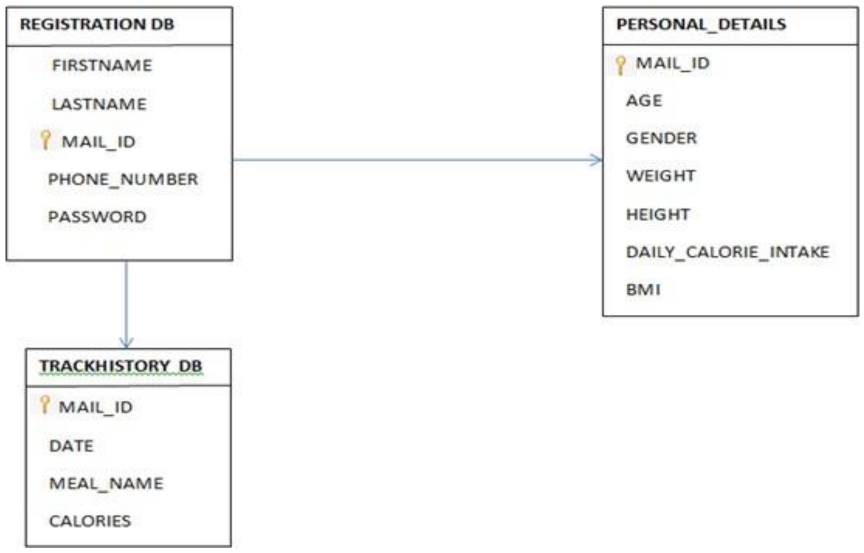
wsgi.py:

```
from application import app

if __name__ == "__main__":
    app.run()
```

CHATBOT INTEGRATION:

```
<script>
 Windows.watsonAssistantChatOptions = {
            integrationID: "614a4315-ff80-4187-8fe4-2fd9b506b723",
            region:"au-syd",
            serviceInstanceID: "9670dcf8-789f-4609-8d7a-6e25c412a9ec",
            onLoad: function(instance){ instance.render(); }
};
 setTimeout(function()){
   const t=document.createElement('script');

t.src=https://webchat.global.assistant.watson.appdomain.cloud/versions/+(window.watsonAssistantChatOptions.cli
entVersion || 'latest') +"/WatsonAssistantChatEntry.js";
document.head.appendChild(t);
});
</script>
```

7.2 DatabaseSchema



8. Testing

Test Cases:

A test case is a series of operations carried out on a system to see if it complies with software requirements and operates properly. A test case's objective is to ascertain whether various system features operate as anticipated and to check that the system complies with all applicable standards, recommendations, and user requirements. The act of creating a test case can also aid in identifying flaws or mistakes in the system.

A test case document includes test steps, test data, preconditions and the post conditions that verify requirements.

Why test case are so important:

Writing test cases is a significant task and is regarded as one of the most crucial components of software testing. The testing team, the development team, and management all use it. We can use a test case as a starting point document if an application doesn't have any documentation.

● In other words, test cases make clear what must be done to test a system. It provides us with the actions we take in a system, the values we provide as input data, and the anticipated outcomes when we run a certain test case.

● Test cases give a precise picture of the expectations that must be met.

● Test cases demonstrate how you addressed and tested each requirement for the product.

● Test cases assist new team members in quickly becoming engaged in the project, learning about your product and test management processes, and running prepared test cases when necessary.

● A solid foundation for developing automated scripts and adding automated testing to the QA process is detailed manual test cases.

1.User Acceptance Testing

User acceptance testing (UAT), also called application testing or end-user testing, is a phase of software development in which the software is tested in the real world by its intended audience.

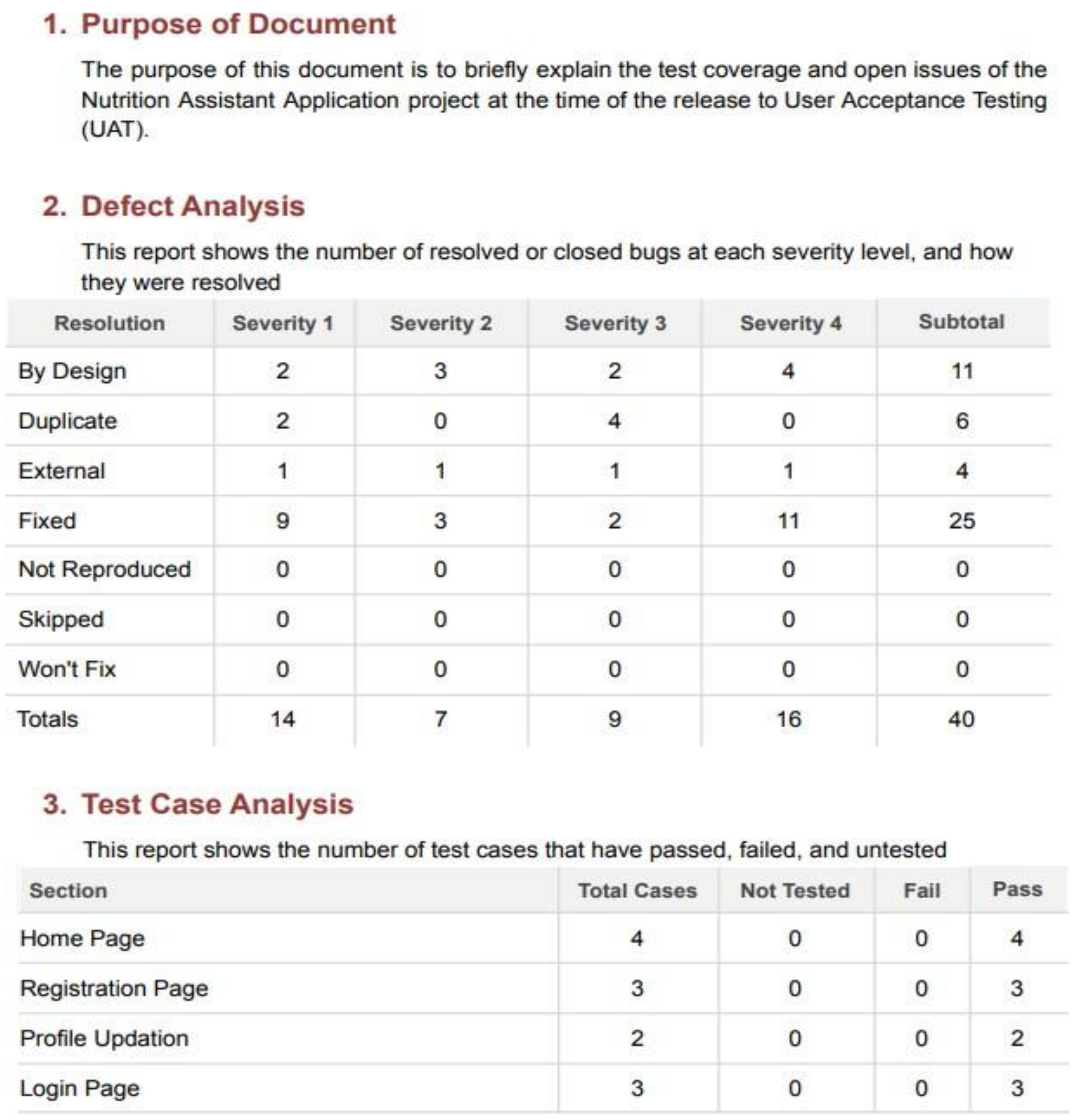
User Acceptance Testing (UAT) is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment. UAT is done in the final phase of testing after functional, integration and system testing is done.

UAT testing meaning can also be defined as the user methodology where the developed software is tested by the business user to validate if the software is working as per the specifications defined.

This type of testing is also known as beta testing, application testing, or more commonly end-user testing. The main Purpose of UAT is to validate end to end business flow.

It does not focus on cosmetic errors, spelling mistakes or system testing. User Acceptance Testing is carried out in a separate testing environment with production-like data setup.

It is a kind of black box testing where two or more end- users will be involved.



1. Purpose of Document

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

2. Defect Analysis

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

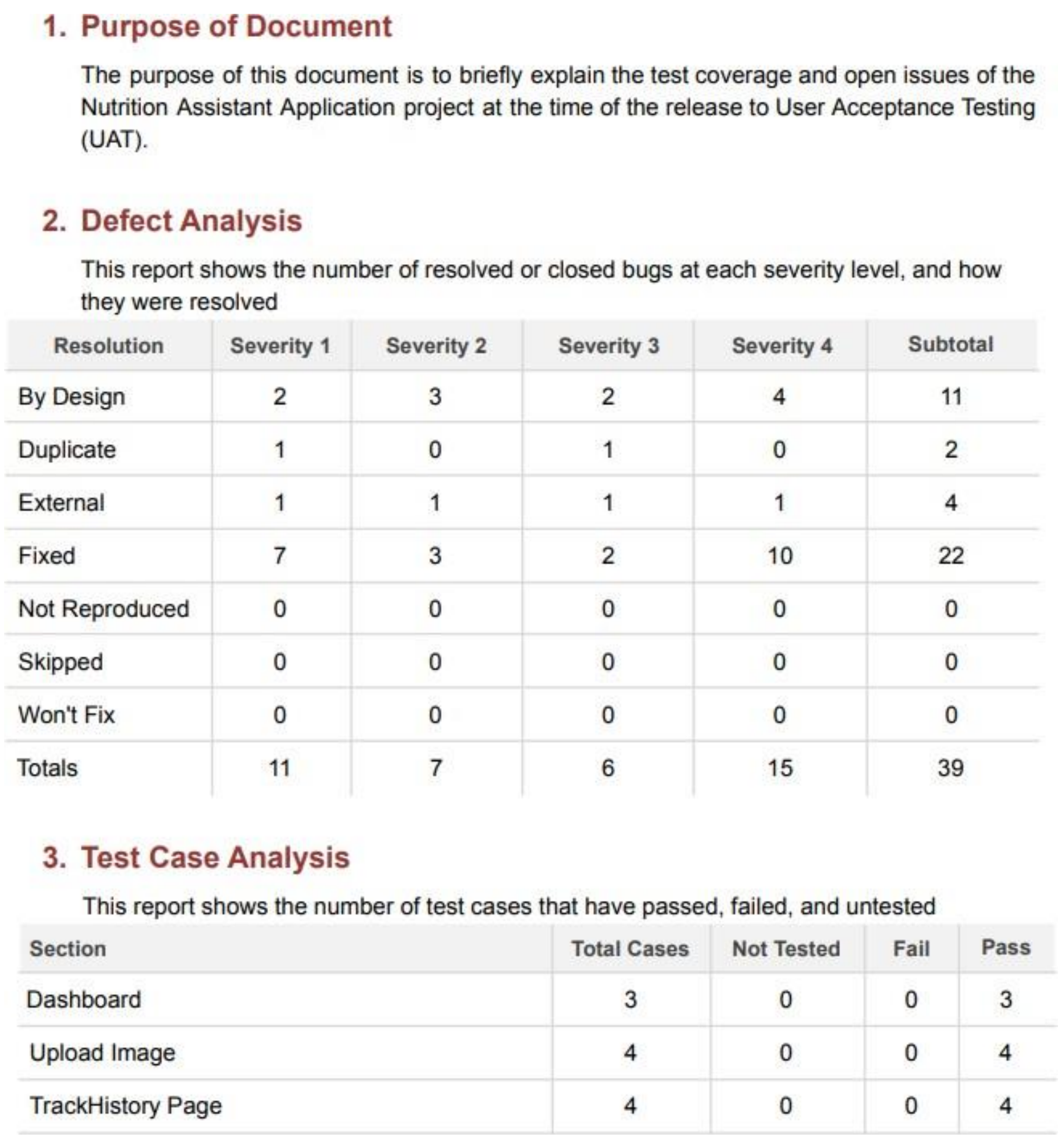
Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	2	3	2	4	11
Duplicate	2	0	4	0	6
External	1	1	1	1	4
Fixed	9	3	2	11	25
Not Reproduced	0	0	0	0	0
Skipped	0	0	0	0	0
Won't Fix	0	0	0	0	0
Totals	14	7	9	16	40

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
Home Page	4	0	0	4
Registration Page	3	0	0	3
Profile Updation	2	0	0	2
Login Page	3	0	0	3

Sprint2:



1. Purpose of Document

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

2. Defect Analysis

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

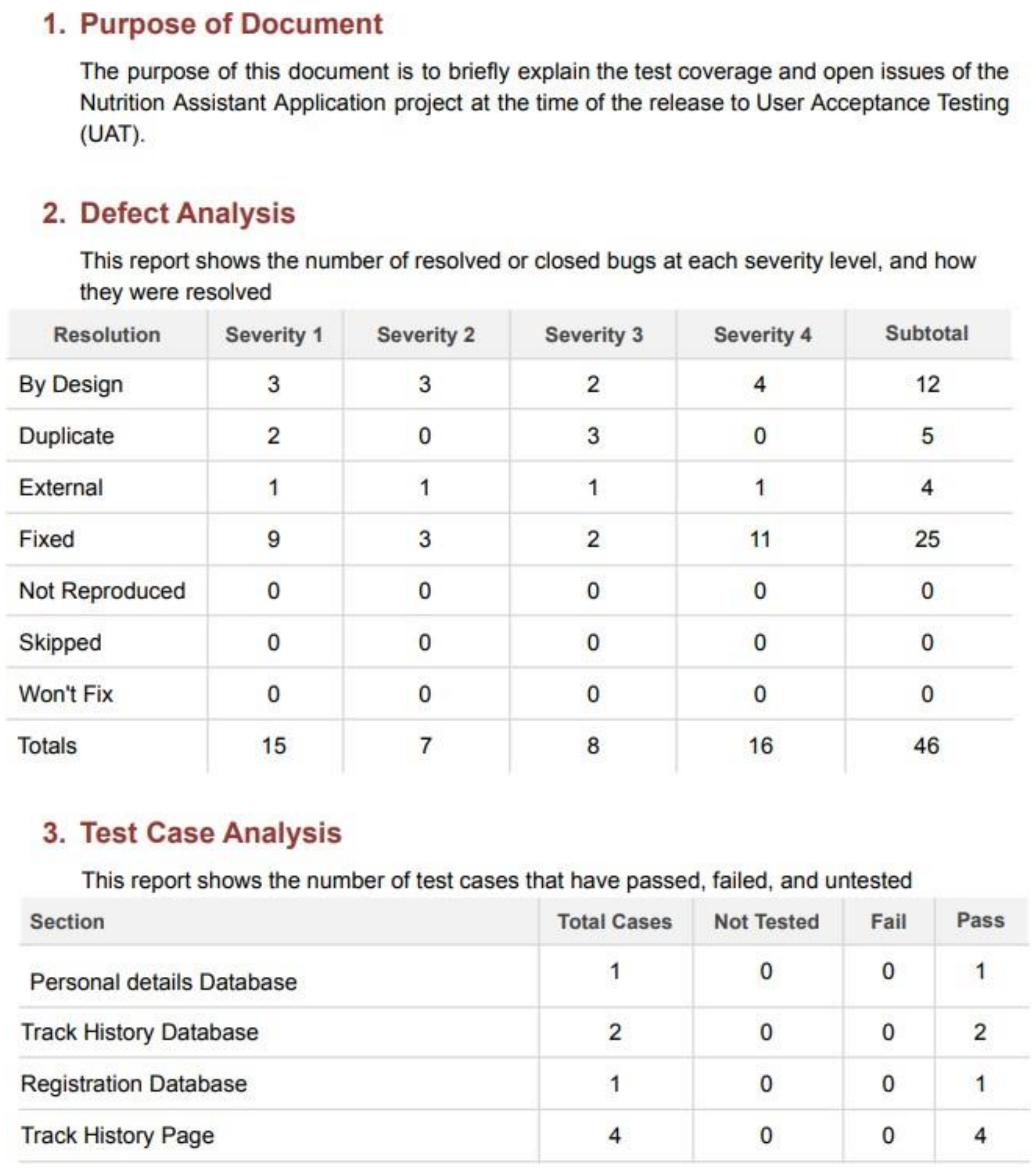
Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	2	3	2	4	11
Duplicate	1	0	1	0	2
External	1	1	1	1	4
Fixed	7	3	2	10	22
Not Reproduced	0	0	0	0	0
Skipped	0	0	0	0	0
Won't Fix	0	0	0	0	0
Totals	11	7	6	15	39

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
Dashboard	3	0	0	3
Upload Image	4	0	0	4
TrackHistory Page	4	0	0	4

Sprint3:



1. Purpose of Document

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

2. Defect Analysis

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

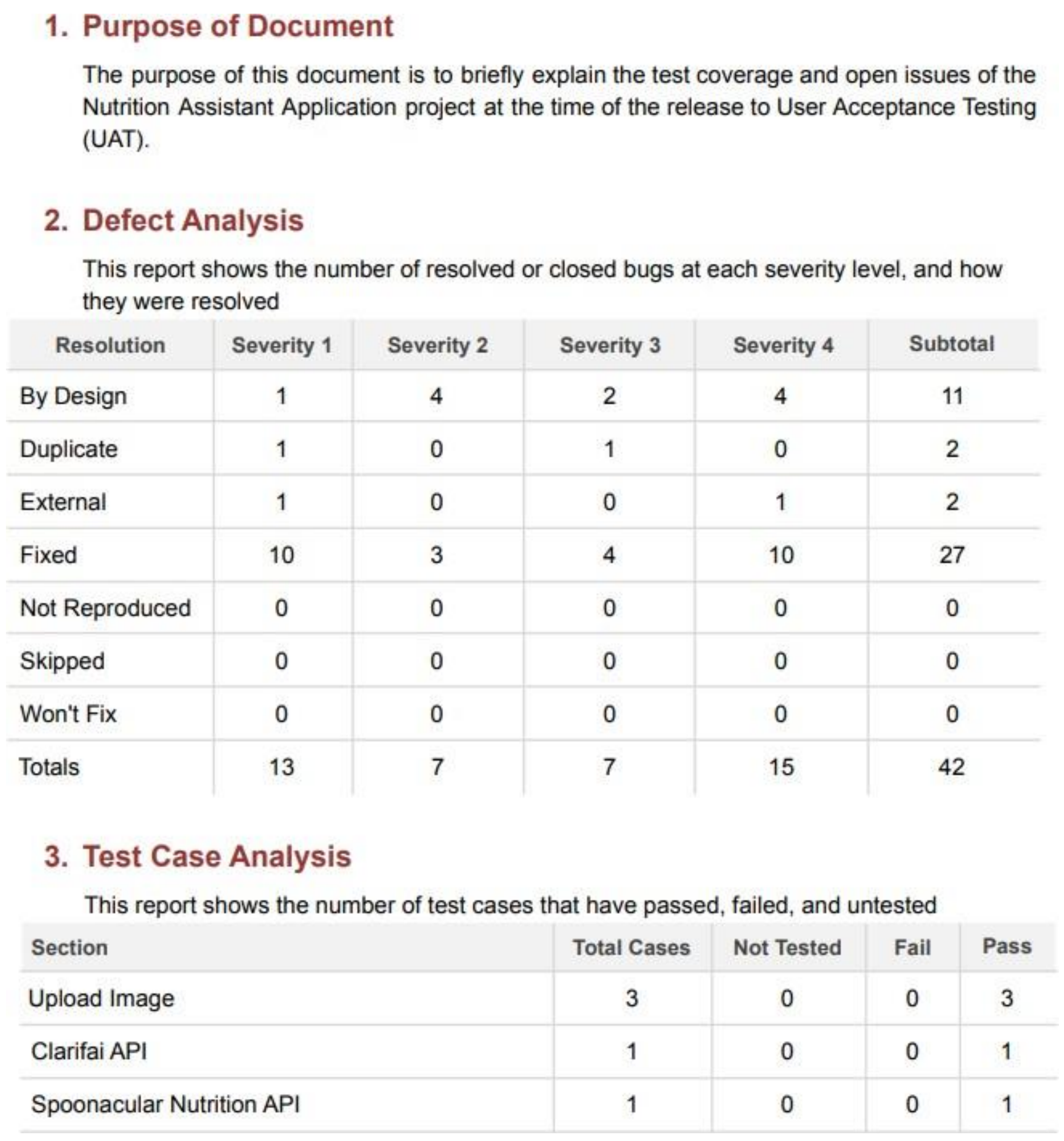
Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	3	3	2	4	12
Duplicate	2	0	3	0	5
External	1	1	1	1	4
Fixed	9	3	2	11	25
Not Reproduced	0	0	0	0	0
Skipped	0	0	0	0	0
Won't Fix	0	0	0	0	0
Totals	15	7	8	16	46

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
Personal details Database	1	0	0	1
Track History Database	2	0	0	2
Registration Database	1	0	0	1
Track History Page	4	0	0	4

Sprint4:



1. Purpose of Document

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

2. Defect Analysis

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

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	1	4	2	4	11
Duplicate	1	0	1	0	2
External	1	0	0	1	2
Fixed	10	3	4	10	27
Not Reproduced	0	0	0	0	0
Skipped	0	0	0	0	0
Won't Fix	0	0	0	0	0
Totals	13	7	7	15	42

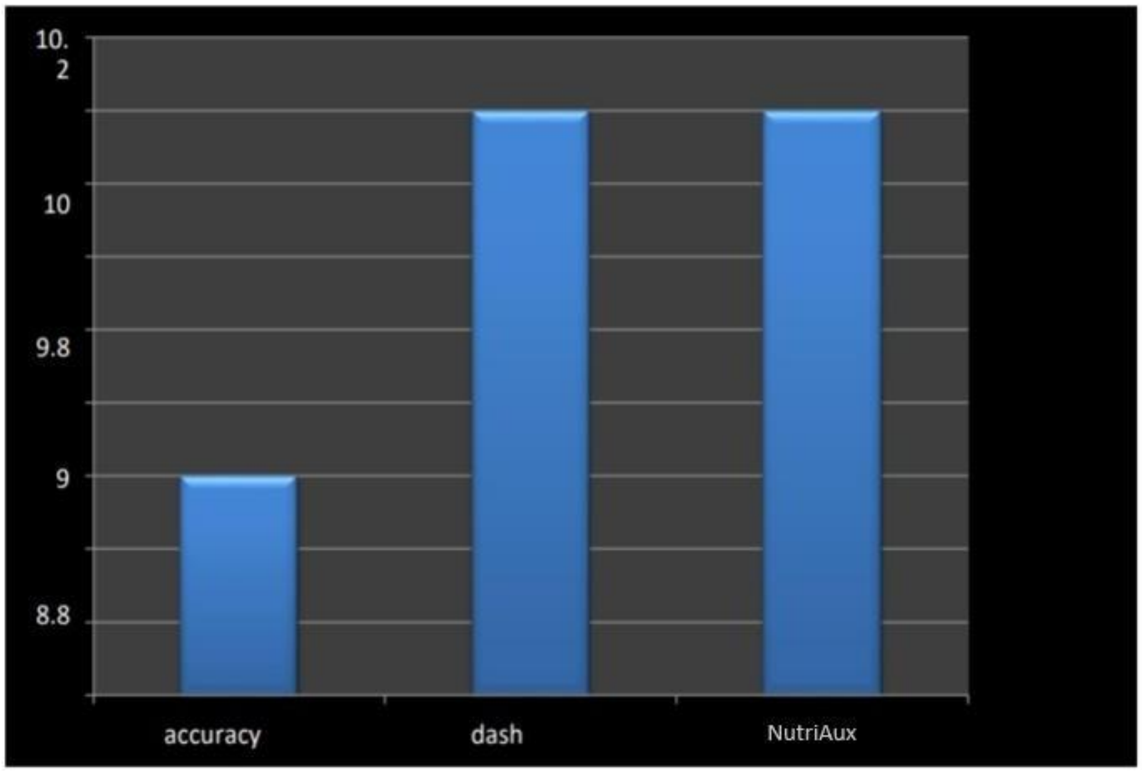
3. Test Case Analysis

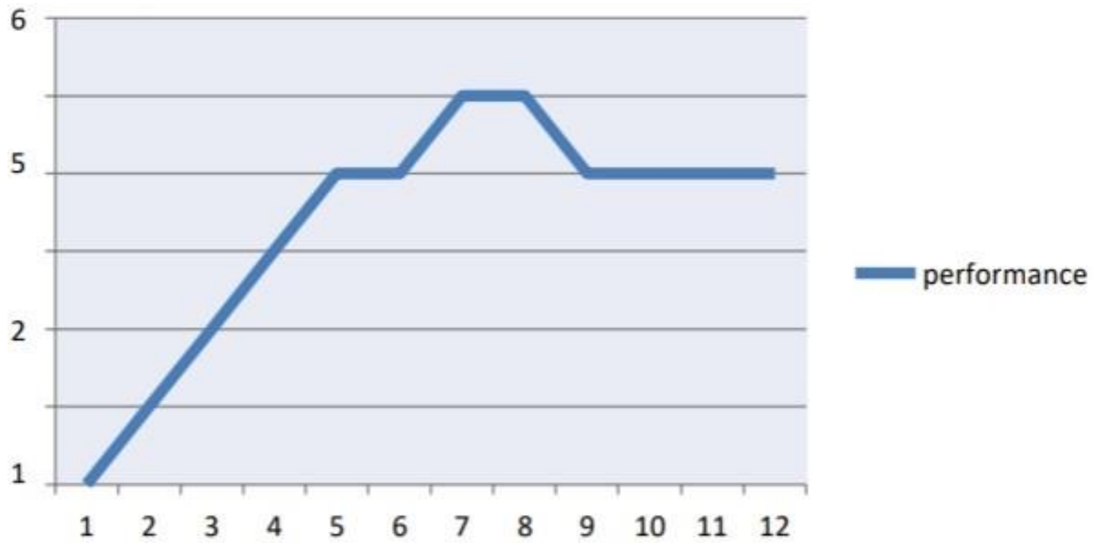
This report shows the number of test cases that have passed, failed, and untested

Section	Total Cases	Not Tested	Fail	Pass
Upload Image	3	0	0	3
Clarifai API	1	0	0	1
Spoonacular Nutrition API	1	0	0	1

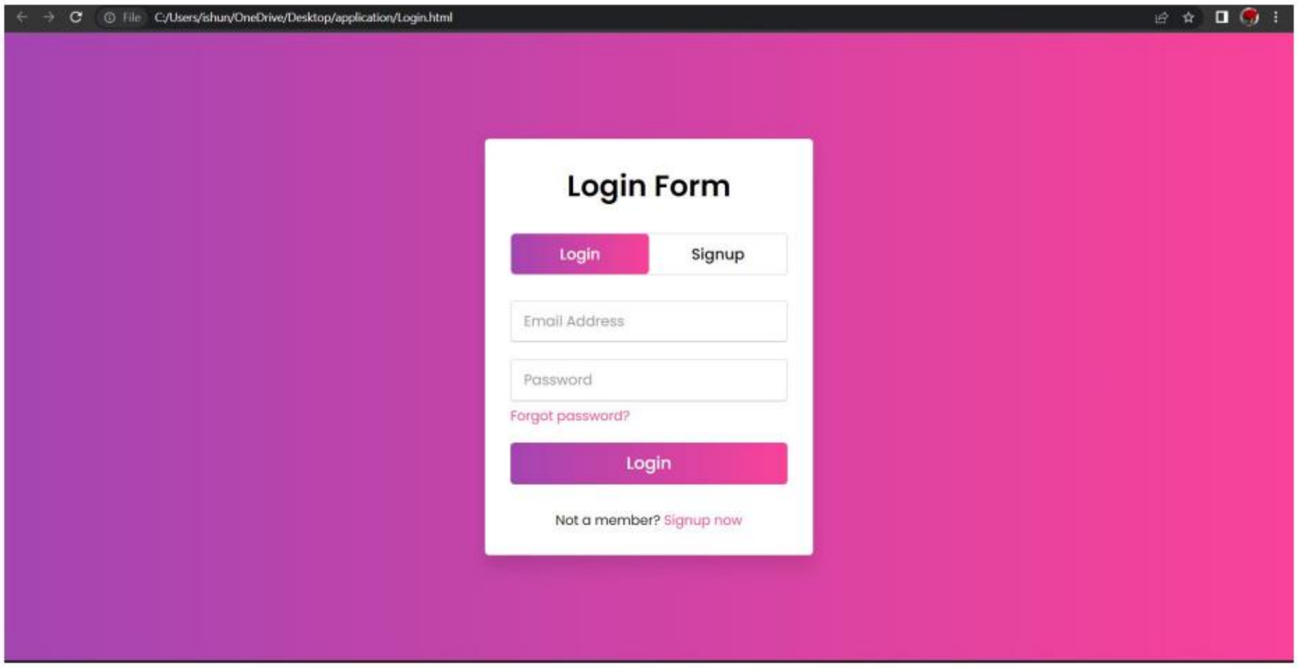
9. RESULTS

9.1 PerformanceMetrics

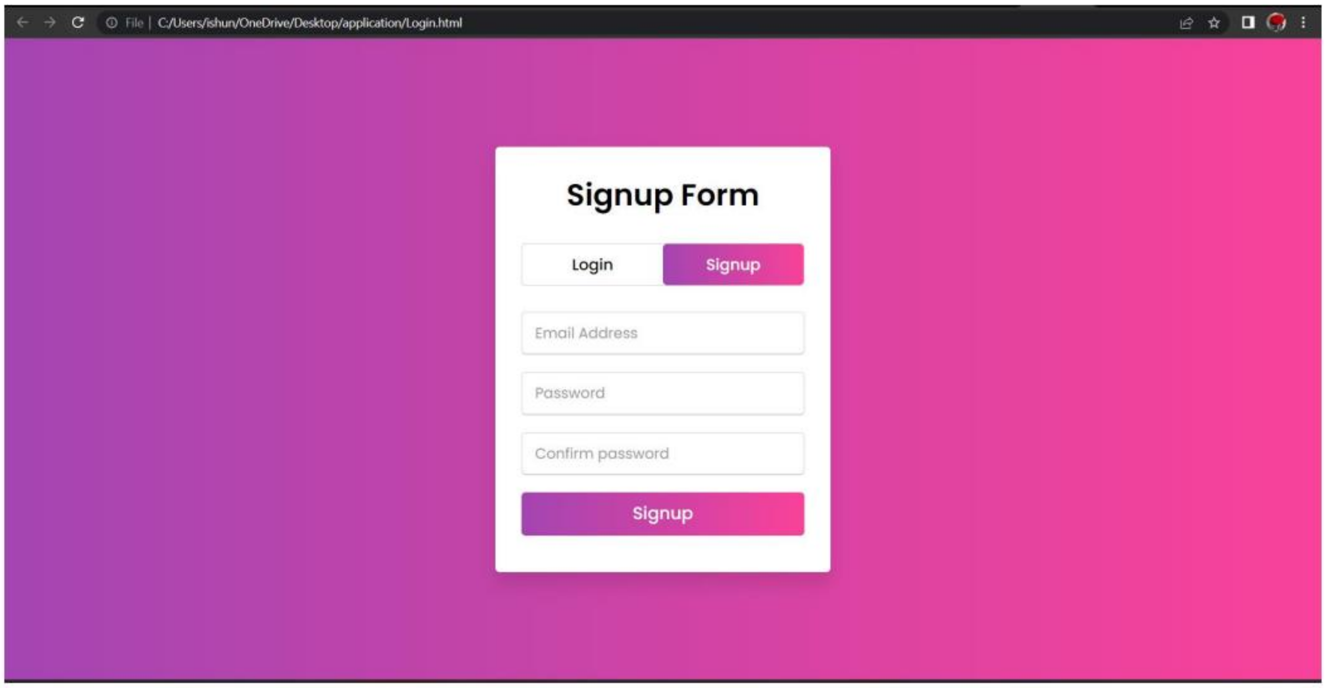




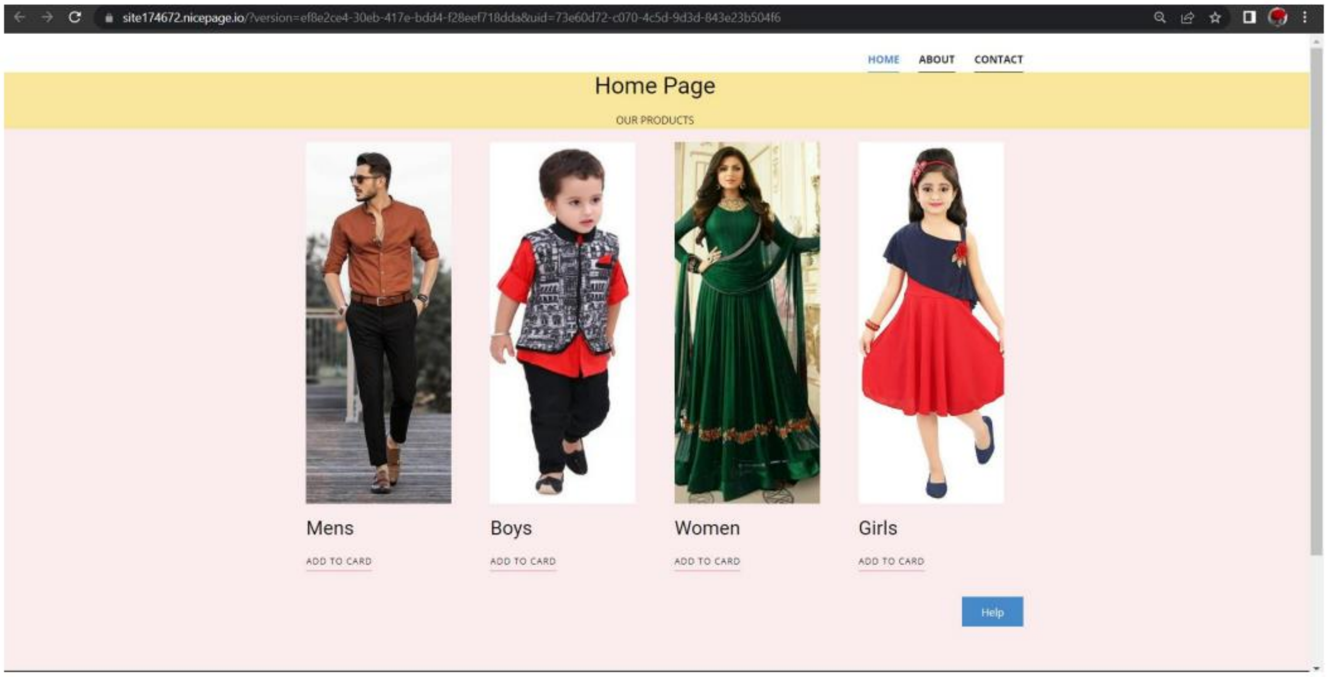
LOGIN PAGE :



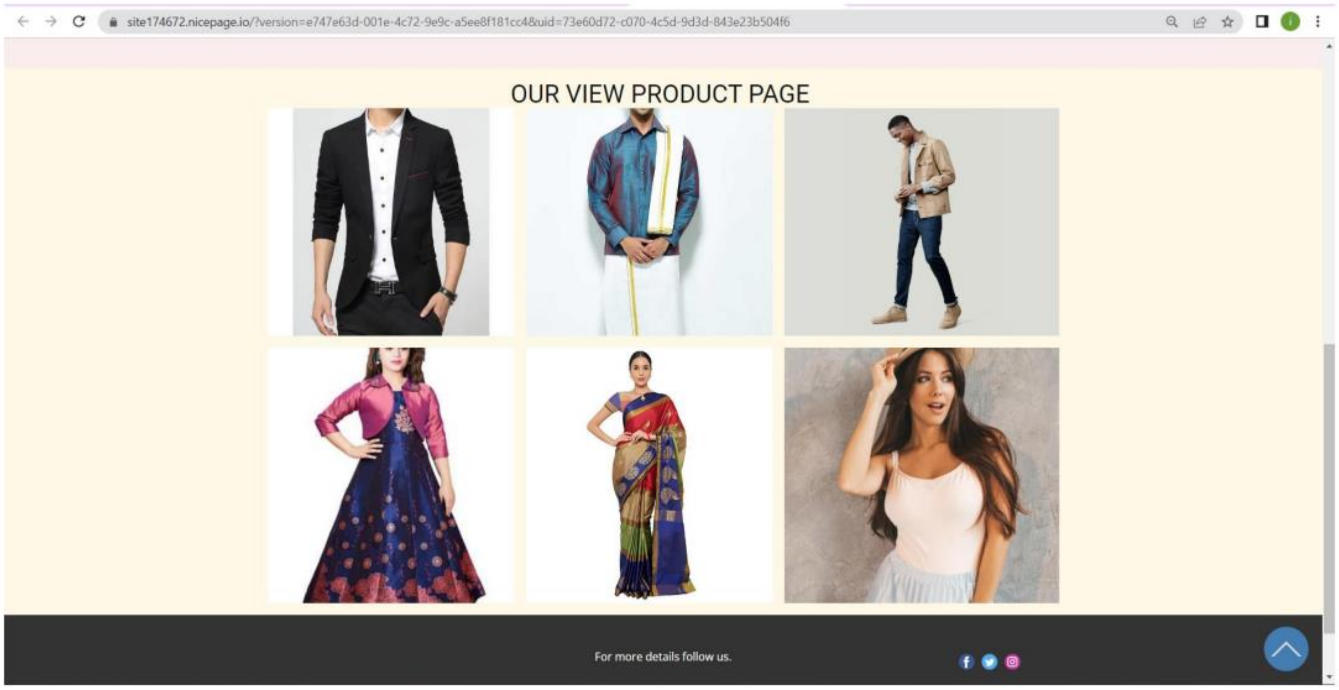
SIGNUP PAGE :



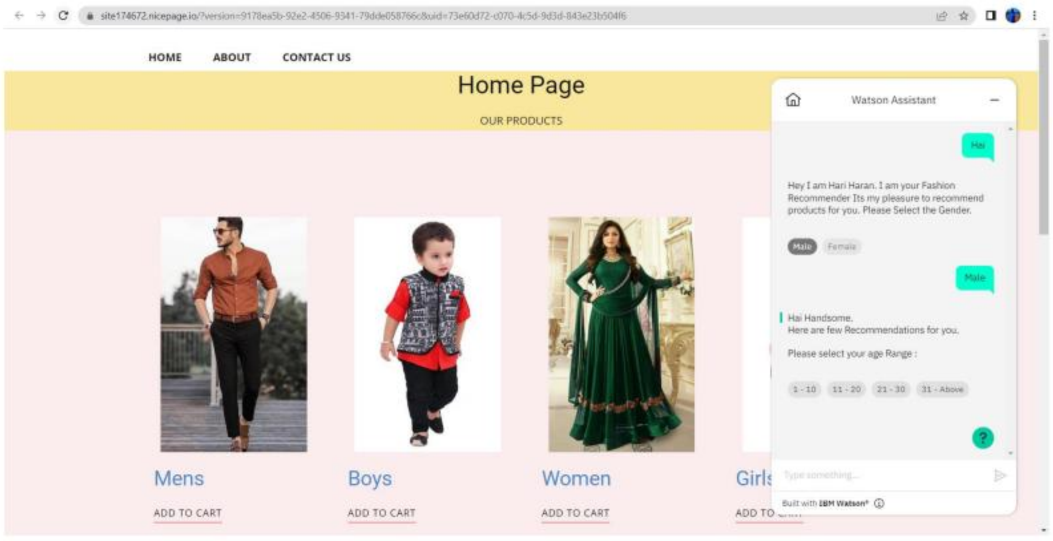
PRODUCT PAGE :


site174672.nicepage.io/?version=ef8e2ce4-30eb-417e-bdd4-f28eef718dda&uid=73e60d72-c070-4c5d-9d3d-843e23b504f6
HOME
ABOUT
CONTACT
Home Page
OUR PRODUCTS
Mens
ADD TO CARD
Boys
ADD TO CARD
Women
ADD TO CARD
Girls
ADD TO CARD
Help

VIEW PRODUCT PAGE :


site174672.nicepage.io/?version=e747e63d-001e-4c72-9e9c-a5ee8f181cc4&uid=73e60d72-c070-4c5d-9d3d-843e23b504f6
OUR VIEW PRODUCT PAGE
For more details follow us.

INTEGRATING CHATBOT WITH HTML PAGE:



Hai

Hey I am Hari Haran. I am your Fashion Recommender Its my pleasure to recommend products for you. Please Select the Gender.

Male Female

Male

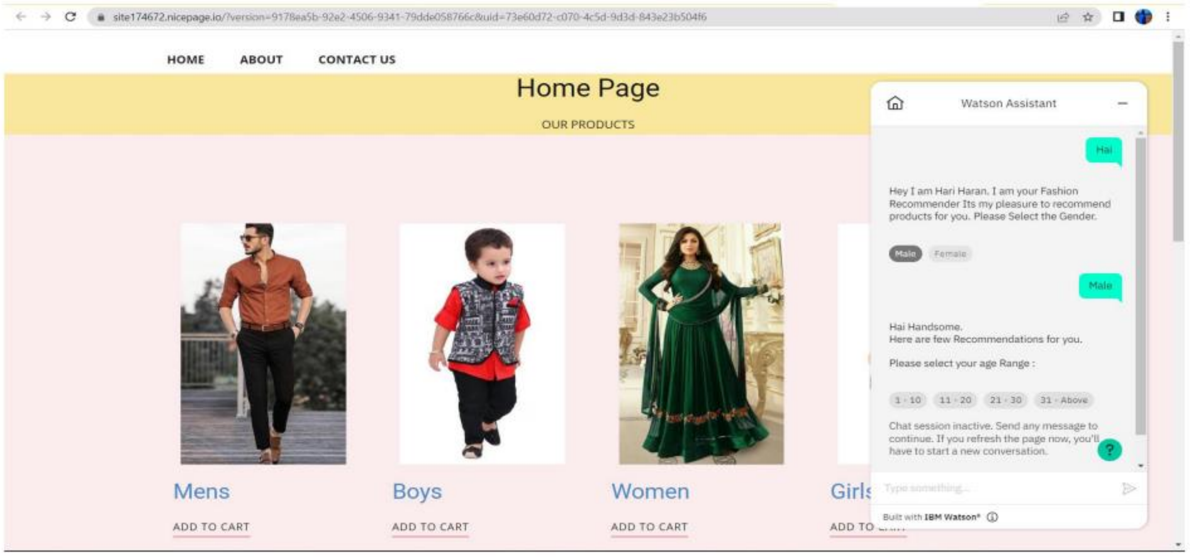
Hai Handsome.
Here are few Recommendations for you.

Please select your age Range :

1 - 10 11 - 20 21 - 30 31 - Above

BUILDING A CHATBOT:

Web link : https://site174672.nicepage.io/?version=9178ea5b-92e2-4506-9341-79dde058766c&uid=73e60d72-c070-4c5d-9d3d-843e23b504f6



Hai

Hey I am Hari Haran. I am your Fashion Recommender Its my pleasure to recommend products for you. Please Select the Gender.

Male Female

Male

Hai Handsome.
Here are few Recommendations for you.

Please select your age Range :

1 - 10 11 - 20 21 - 30 31 - Above

Chat session inactive. Send any message to continue. If you refresh the page now, you'll have to start a new conversation.

9. ADVANTAGES & DISADVANTAGES

Advantages:

- Products recommended based on the evaluation of experienced users.
- CBF does not need any information from other users, which makes this technique more feasible and less time consuming.
- CBF can attain the specific interest of a user and make recommendations accordingly.
- Provides a valuable explanation, which motivates users to make decisions.
- This method can allow users to discover new interests despite the absence of content in the user’s profile.
- Yields better results when it comes to customer satisfaction and needs.
- Enhanced customer experiences.
- Higher return on investment (ROI).
- Highly engaging social campaigns.

Disadvantages:

- As it is CBF domain-dependent, rigorous domain knowledge is required to make precise recommendations.
- The model only recommends products based on an existing database of previous users’ interest, which restricts its expansion.
- Due to cold start problem, cannot be applied to make recommendations to new users.
- This method suffers limited content analysis issues, meaning users are restricted to the items already recommended.

11. CONCLUSION

Recommendation systems have the potential to explore new opportunities for retailers by enabling them to provide customized recommendations to consumers based on information retrieved from the Internet. They help consumers to instantly find the products and services that closely match with their choices. Moreover, different statof-the-art algorithms have been developed to recommend products based on users’ interactions with their social groups. Therefore, research on embedding social media images within fashion recommendation systems has gained huge popularity in recent times. This paper presented a review of the fashion recommendation systems, algorithmic models and filtering techniques based on the academic articles related to this topic. The technical aspects, strengths and weaknesses of the filtering techniques have been discussed elaborately, which will help future researchers gain an in-depth understanding of fashion recommender systems.

However, the proposed prototypes should be tested in commercial applications to understand their feasibility and accuracy in the retail market, because inaccurate recommendations can produce a negative impact on a customer. Moreover, future research should concentrate on including time series analysis and accurate categorization of product images based on the variation in color, trend and clothing style in order to develop an effective recommendation system. The proposed model will follow brand-specific personalization campaigns and hence it will ensure highly curated and tailored offerings for users. Hence, this research will be highly beneficial for researchers interested in using augmented and virtual reality features to develop recommendation systems.

12. FUTURE SCOPE

In this section, 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 laborcosting 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.

10.APPENDIX

SourceCode:

application.py:

```
from cs50 import SQL
from flask_session import Session
from flask import Flask, render_template, redirect, request, session, jsonify
from datetime import datetime

# # Instantiate Flask object named app
```

```python
app = Flask(__name__)

# # Configure sessions
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
Session(app)

# Creates a connection to the database
db = SQL ( "sqlite:///data.db" )


@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, session=session
)
    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=samplename, image=image,
price=price, subTotal=subTotal)
        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, session=session
)


@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=samplename, image=image,
price=price, subTotal=subTotal)
        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, session=session
)
    # 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"], samplename=item["samplename"],
image=item["image"], quantity=item["qty"] )
```

```
    # 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")


@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 )
```

```python
    # 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
    # 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, myShirtsLen=myShirtsLen)


@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
    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)
```

wsgi.py:

```
from application import app

if __name__ == "__main__":
    app.run()
```

GitHub&ProjectDemoLink

GitHub: https://github.com/IBM-EPBL/IBM-Project-8243-1658912566

Project Demo Link: https://drive.google.com/drive/folders/1-DOecAEosEuCBhgPvQdS3SDdoZhxNddv