SMART FASHION

RECOMMENDER APPLICATION

PROJECT REPORT

Submitted by

GEETHA S	**LEADER**
KARTHIKA B	**MEMBER 1**
MENAGA S	**MEMBER 2**
NITHYA JAYASHRI P	**MEMBER 3**
JANANI M	**MEMBER 4**

in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

INFORMATION TECHNOLOGY

SETHU INSTITUTE OF TECHNOLOGY

An Autonomous Institution PULLOOR,

KARIAPATTI – 626 115

NOVEMBER 2022

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.

ACKNOWLEDGEMENT

A Nalayathiran project of this magnitude and nature requires kind cooperation and support from many, for successful completion. We wish to express our sincere thanks to all those who were involved in the completion of this Nalayathiran project.

We wish to express our earnest gratefulness to our beloved **Founder & ChairmanMr. S. Mohamed Jaleel, B.Sc., B.L.,** for his sincere endeavor in educating us in hispremier institution.

We would like to bring forth our sincere thanks to our beloved **Principal Dr. A. Senthil Kumar, M.E., Ph.D.**

for providing all necessary facilities for the completion of the project.

We thank our Dean **Dr. S. Siva Ranjani, M.E., Ph.D.** for her keen assistance throughout our project.

We wish to convey our thanks and gratitude to our head of the department, **Dr.M. Poomani@punitha, M.E., Ph.D.,** Department of Information Technology, for her support and for providing us ample time to complete our project.

Last, we thank our parents and friends for providing their extensive moral support and encouragement during the project

Project Report Format

1. **INTRODUCTION**

    1.1 Project Overview

    1.2 Purpose

2. **LITERATURE SURVEY**

    2.1 Existing problem

    2.2 References

    2.3 Problem Statement Definition

3. **IDEATION & PROPOSED SOLUTION**

    3.1 Empathy Map Canvas

    3.2 Ideation & Brainstorming

    3.3 Proposed Solution

    3.4 Problem Solution fit

4. **REQUIREMENT ANALYSIS** -

Functional requirement

Non-Functional requirements

5. **PROJECT DESIGN**

Data Flow Diagrams

Solution & Technical Architecture

User Stories

6. **PROJECT PLANNING & SCHEDULING**

    Sprint Planning & Estimation

    Sprint Delivery Schedule

Reports from JIRA

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

Feature 1

Feature 2

Database Schema (if Applicable)

8. TESTING

Test Cases

User Acceptance Testing

9. RESULTS

Performance Metrics

10. ADVANTAGES & DISADVANTAGES

11. CONCLUSION

12. FUTURE SCOPE

13. APPENDIX

Source Code

GitHub

1. INTRODUCTION

Project Overview

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.

Purpose of the Project

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. LITERATURE SURVEY

Existing Problem

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.

References

i) Paper Title: A COMPREHENSIVE REVIEW ON ONLINE FASHION RECOMMENDATION Publication: December 2020

Author name: Samit Chakraborty

Methodology: Auto Regression (AR) and Linear Regression Model. Auto Regression (AR) and Linear Regression Model Using photos pulled from social media, online fashion magazines, well-known e-commerce sites, fashion site blogs, and discussion forums, (Ngai et al., 2018) employed the autoregressive (AR) model (or ARMAX) to forecast style or trends. Due to the data patterns being obtained over a set amount of time, it makes precise trend prediction possible (Fung, Wong, Ho, & Mignolet, 2003). These forecasting models' detailed theoretical contents were demonstrated in two separate studies by Liu et al. (2013) and Nenni, Giustiniano, & Pirolo (2013), which also included several general approach forms. Because they were straightforward, quick, wellinformed, and simple to understand, statistical techniques including auto-regression, exponential smoothing, ARIMA, and SARIMA were frequently employed to assess the sales of clothing. A technique for forecasting retail products was proposed by Demerit (2018). weekly using linear regression models in multi-processing groups with both positive and negative commodities. The introduction of dynamic pricing models to support markdown choices in multi-item group predictions has since followed. In order to prevent overfitting, grouping items in predictive models can be seen as a way of variable selection. They then exhibited regression results from multiple-item groupings on the real-world dataset provided by a clothing company in addition to the findings from the single-item regression model. They also revealed the results of markdown optimization for single items and groups of multiple items that serve as the foundation for multi-item forecasting models. The results suggested that regression models provide better estimates in many categories than the one-item model.

ii) Paper Title: Fashion Recommendation Systems

Author name: Samit Chakraborty , Md. Saiful Hoque, Naimur Rahman Jeem, Manik Chandra Biswas, Deepayan Bardhan and Edger Lobaton.

Methodology: Fast fashion has grown significantly over the past few years, which has had a significant impact on the textile and fashion industries.

 An effective recommendation system is needed in e-commerce platforms where there are many options available to sort, order, and effectively communicate to user's pertinent product content or information.

Fast fashion retailers have paid a lot of attention to image-based fashion recommendation systems (FRSs), which offer customers a customised purchasing experience.

There aren't many academic studies on this subject, despite its enormous potential. The studies that are now accessible do not conduct a thorough analysis of fashion recommendation systems and the accompanying filtering methods.

This review also looks at many potential models that might be used to create future fashion suggestion systems.

iii) Paper Title: A Review on Clothes Matching and Recommendation System Based on User Attributes.

Author name: Atharv Pandit , Kunal Goel , Manav Jain , Neha Katre

 Methodology: It's crucial to dress adequately while venturing out into the real world. The confidence of the individual is raised and a very positive impression is made when they are dressed appropriately in clothing that exhibits some degree of style and is worn in a way that complies with societal norms. The goal of the study is to make it easier for customers to locate the best-fitting outfits by taking into account fine elements like style, patterns, colors, and textures, as well as user characteristics like age, skin tone, and favorite colors. It seeks to assist the user in organizing their closet and making stylish clothing selections.

 It makes an effort to assist the user in dressing appropriately for the occasion and in finding clothing that compliments their personal style.

 In order to create a robust system that discovers the user's matching outfits and provides recommendations, an in-depth analysis of numerous systems that are built for various aspects is undertaken in this research. Systems created to propose clothing using various methodologies have been researched, with both their benefits and drawbacks highlighted.

 It has also been investigated how to make clothing detecting systems user-friendly while accepting feedback from the user.

iv) Paper Title: Individualized fashion recommender system Year: 10 October 2020

Author name: M Sridevi, N ManikyaArun, MSheshikala and E Sudarshan

Methodology: This design seeks to use an image of a product provided by the stoner as input to prompt recommendations because people frequently see things that they're interested in and tend to look for products that are similar to those. We reuse the Deep Fashion Dataset (DFD) photos using neural networks, and we generate the final suggestions using a closest neighbor backed recommender.

v) Paper Title: Image-based fashion recommender system. Publication: Year (2021).

 Author name: Shaghayegh Shirkhani.

 Methodology: Collaborative filtering, the iterative filtering process, matrix factorization, and content-based systems. Systems for collaborative filtering make product recommendations based on user similarity metrics and/or by grouping things from similar users' purchases.

 Despite the variety of collaborative filtering methods, many widely used systems can be distilled down to just two steps: 1. Seek out users who have similar rating tendencies to the active user (the user whom the prediction is for). 2. To establish a prediction for the active user, utilize the ratings from the users who shared your interests in step one.

Problem statement

Problem Statement Definition

Create a problem statement to understand your customer's point of view. The Customer Problem Statement template helps you focus on what matters to create experiences people will love. A well-articulated customer problem statement allows you and your team to find the ideal solution for the challenges your customers face.

Throughout the process, you'll also be able to empathize with your customers, which helps you better understand how they perceive your product or service.

How to write a problem statement?

A good problem statement can be created by identifying and answering several questions related to the problem,

1. Identify the Problem
2. Begin were statement with where ideal situation
3. Describe current gaps
4. State the consequence of the problem
5. Propose addressing the problem

KEY ELEMENTS OF PROBLEM STATEMENT:

- Reality - It will also describe when and where the problem was identified.
- Consequences - Problem statements should identify the consequences of the problem.
- Proposal - The proposal section of a problem statement may contain several possible solutions to the problem

I am	Describe customer with 3-4 key characteristics - *who are they?*	Describe the customer and their attributes here
I'm trying to	List their outcome or "job" the care about - *what are they trying to achieve?*	List the thing they are trying to achieve here
but	Describe what problems or barriers stand in the way – *what bothers them most?*	Describe the problems or barriers that get in the way here
because	Enter the "root cause" of why the problem or barrier exists – *what needs to be solved?*	Describe the reason the problems or barriers exist
which makes me feel	Describe the emotions from the customer's point of view – *how does it impact them emotionally?*	Describe the emotions the result from experiencing the problems or barriers

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	I am a student	I am trying to buy a formal dress for college event	I can't find perfect match	Either the brand or colour doesn't suit	I don't have any choice in choosing right brand.
PS-2	I am a bride	Looking for wedding and reception dress	Could not able to choose right dress	Availability of wide variety of products	Like I'm overthinking regarding how my dress should looks like.
PS-3	I am an Average Man	Buy clothes to attend family function.	could not find dress of my desire	Product price not within the budget	Give up and move for other ecommerce site

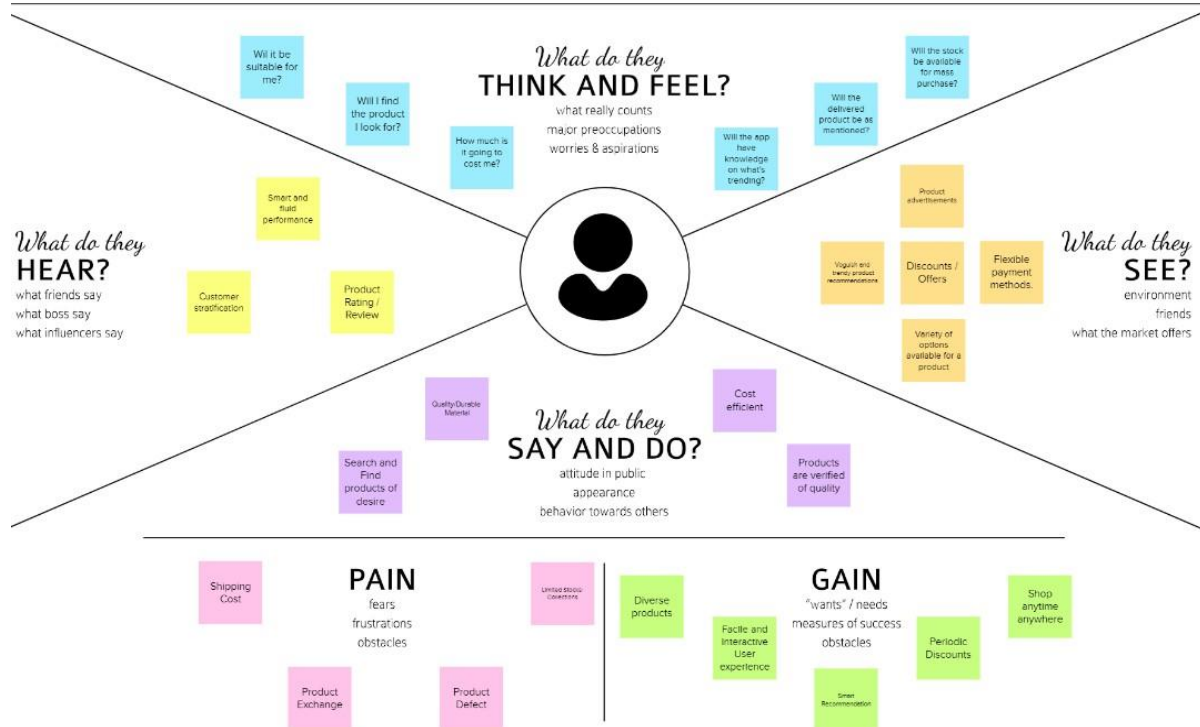## 3. IDEATION & PROPOSED SOLUTION

Empathy Map

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviors and attitudes. It is a useful tool to helps teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it.

The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

Benefits of empathy map

- *It is simple and quick to complete an empathy map. There is no need to spend a lot of time or money on the undertaking. All you require is a team and basic tools. As an alternative, you can use expensive but cutting-edge software, even on a tight budget.*
- *The visualizing technique allows for widespread participation. You invite managers, designers, tech and non-tech experts, and product owners. The more opinions and ideas you have, the better.*
- *It aids creators in expanding their thinking and understanding that their ideal product vision may vary from the viewpoint of the customers.*

- *These resources are all-purpose and can be used in any field, sector, or type of business.*
- *Companies that engage in empathy mapping outperform those that do not.*

What do they
THINK AND FEEL?
what really counts
major preoccupations
worries & aspirations

Wil it be suitable for me?
Will I find the product I look for?
How much is it going to cost me?
Will the app have knowledge on what's trending?
Will the delivered product be as mentioned?
Will the stock be available for mass purchase?

What do they
HEAR?
what friends say
what boss say
what influencers say

Smart and fluid performance
Customer stratification
Product Rating / Review

What do they
SEE?
environment
friends
what the market offers

Product advertisements
Yoguti and trendy product recommendations
Discounts / Offers
Flexible payment methods.
Variety of options available for a product

What do they
SAY AND DO?
attitude in public
appearance
behavior towards others

Quality/Durable Material
Search and Find products of desire
Cost efficient
Products are verified of quality

PAIN
fears
frustrations
obstacles

Shipping Cost
Limited stock covertions
Product Exchange
Product Defect

GAIN
"wants" / needs
measures of success
obstacles

Diverse products
Facile and Interactive User experience
Smart Recommendation
Periodic Discounts
Shop anytime anywhere

Brainstorming- Idea prioritization

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.

There are some brainstorming approaches:

Step 1: Prepare the Group

Step 2: Present the Problem

Step 3: Guide the Discussion

Step 1: Prepare the Group

How much background knowledge or planning is required for your team to generate problem-solving ideas? It's vital to keep in mind that preparation is necessary, but too much of it might hinder or even ruin a brainstorming session.

Once everyone is present, designate one person to take notes on the suggestions made during the session.

It's difficult to record and contribute at the same time, thus this individual shouldn't necessarily be the team manager.

Use a computer with a data projector, flip charts, or whiteboards to post notes where everyone can see them.

Step 2: Present the Problem

Lay up any requirements that must be met as well as the problem that you are trying to solve.

Make it crystal clear that the goal of the meeting is to create as many ideas as you can.

Allow everyone plenty of quiet time to come up with as many original ideas as they can at the beginning of the session.

Ask them to share or present their thoughts after that, while ensuring that everyone has an equal chance to contribute.

Step 3: Guide the Discussion

Start a group conversation once everyone has given their views a chance to be heard in order to develop existing ideas and generate new ones.

One of the most beneficial features of group brainstorming is building on others' ideas.

Encourage everyone—even the most reserved individuals—to participate and develop ideas, and forbid anybody from critiquing others' ideas.

If you have any ideas, you should share them as the group facilitator, but otherwise focus your time and efforts on helping your team and facilitating the conversation.

Keep each conversation to itself and bring the group back on track if it wanders off.

Don't spend too much time thinking along the same lines. Make sure to come up with lots of different ideas and thoroughly examine each one.

Give a team member the freedom to pursue an idea alone if they need to "tune out" for it.

Step-1: Team Gathering, Collaboration and Select the Problem Statement

Step-2: Brainstorm, Idea Listing and Grouping

Step-3: Idea Prioritization

Benefits of using brainstorming:

- *There are several advantages to brainstorming.*
- *Building engagement, dedication, devotion, and passion through brainstorming.*
- *People's creative talents are stimulated and unlocked by participating in the workshops.*
- *Because participants are solicited for their input and ideas during a brainstorming session, this activity also boosts self-esteem.*
- *You can foster more teamwork and cooperation using brainstorming.*
- *People who participate in brainstorming exercises together are more likely to form stronger bonds and communicate more effectively.*
- *The main benefit is that you will generate many excellent ideas, some of which may even change the course of the company.*
- *I've found that brainstorming activities have produced a number of excellent ideas to..*
- *Encourages Critical Thinking*
- *When you practice brainstorming as a group, you take team ownership of a campaign, product or event.*
- *This means that one person isn't left feeling like he is carrying the workload for the entire company, and also cultivates a feeling of team ownership.*

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)	- Lack of interaction between application and user - User need to navigate across multiple pages to choose right product - Confusion in choosing product - Lack of sales - Complex User Interface. - Lack of proper guidance.
2.	Idea / Solution description	By using Smart fashion recommender application: - Improve customer relationship, interactivity and services. - Effective recommendation of products. - Recommendation within a single page via chat-bot - Collect feedback instantly. - Reduce human error - Proper guidance in accessing application.
3.	Novelty / Uniqueness	Chat-bot asks and learns from user preference which recommends appropriate products to the user without making them to search through various filters. Reduces time in choosing right product thus increases sales.
4.	Social Impact / Customer Satisfaction	Feedback from the user at the end of session or after placing order is one of the most important factor in deriving customer satisfaction and providing better services.
5.	Business Model (Revenue Model)	The application can be developed at minimum cost with high performance and interactive user interface.
6.	Scalability of the Solution	The solution can be made scalable by using micro service architecture provided that each server responsible for certain functionality of the application. Storing user preferences along with product in browser cookie will enable to provide response instantly and allows for fetching related products.

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.

• 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.

Problem-Solution fit canvas 2.0 — Smart Fashion Recommender Application

1. CUSTOMER SEGMENT(S) — CS
Who is your customer?
i.e. working parents of 0-5 y.o. kids

- Customers are those who want to purchase fashion items in a short time

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.

- Most of the solution available in the internet hosts a lot of adds limiting its usability.
- Needs a proper 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 dothese solutions have? i.e. pen and paper is an alternative to digital notetaking

- Smart Fashion Recommender which are supported in many browsers
- Smart Fashion Recommender Chatbot is developed in thisproject.

2. JOBS-TO-BE-DONE / PROBLEMS — J&P
Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.

- To collect data about our visitors and leverage it to make better product suggestions and recommendations
- Understanding customer inquiries, their needs, and preferences can allow you to personalize product pages and build customer loyalty and affinity.

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.

- For No-Pressure Shopping Experiences
- Customer service will be available for 24/7
- Chatbot can help with recovering abandoned carts

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)

- Seamless Real-Life Interaction
- Customer Data Security
- Reduce Customer Frustration

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

- Improve Lead Generation.
- Reduce Customer Service Costs.
- Monitor Consumer Data to Gain Insights.

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.

- Took longer time to process and respond to the query

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.

- Instead of navigating to several screens for booking products online, the user can directly talk to Chatbot regarding the products.

8. CHANNELS of BEHAVIOUR — CH
8.1 ONLINE
What kind of actions do customers take online? Extract online channels from #7

- Able to serve customers with a consistent level of quality in a short period of time across different channels,

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

- Make sure they are aware of the usage of the chatbots

4. REQUIREMENT ANALYSIS:

Functional Requirements:

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 Requirements	Sub Registration
FR-1	Registration	Registration can be done using mobile number or gmail and needed some user information
FR-2	Login	User only log in by user id and password,Which is given during registration
FR-3	Delivery confirmation	Confirmation via email and phone number
FR-4	Assistance	Bot is integrated with the application to make the usability simple

Non-Functional Requirements

Following are the Non-Functional requirements of the proposed solution.

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	A user-friendly interface with chat bot to make usability efficient
NFR-2	Security	Secured connection HTTPS should be established for transmitting requests and responses
NFR-3	Reliability	The system should handle excepted as well as unexpected errors and exceptions to avoid termination of the program
NFR-4	Performance	The system shall be able to handle multiple requests at any given point in time and generate an appropriate response.
NFR-5	Availability	It is a cloud based web application so user can access without any platform limitations ,just using a browsers with a internet connection is enough for use the application
NFR-6	Scalability	It has a quick request and response time, high throughput, enough network resources and so on.

5. PROJECT DESIGN

Data Flow Diagram

A data flow diagram (DFD) is a graphical or visual representation that describes how data is moved through an organization's operations using a standardized set of symbols and notations.
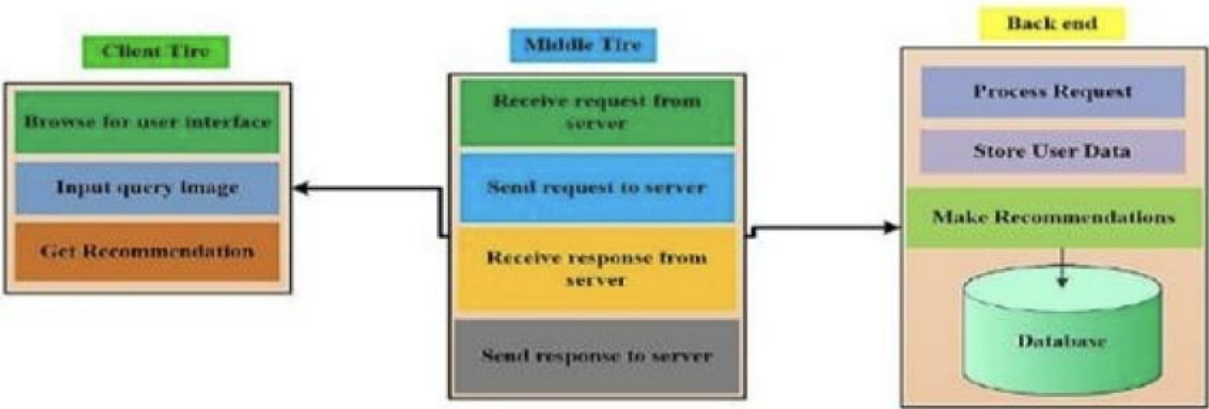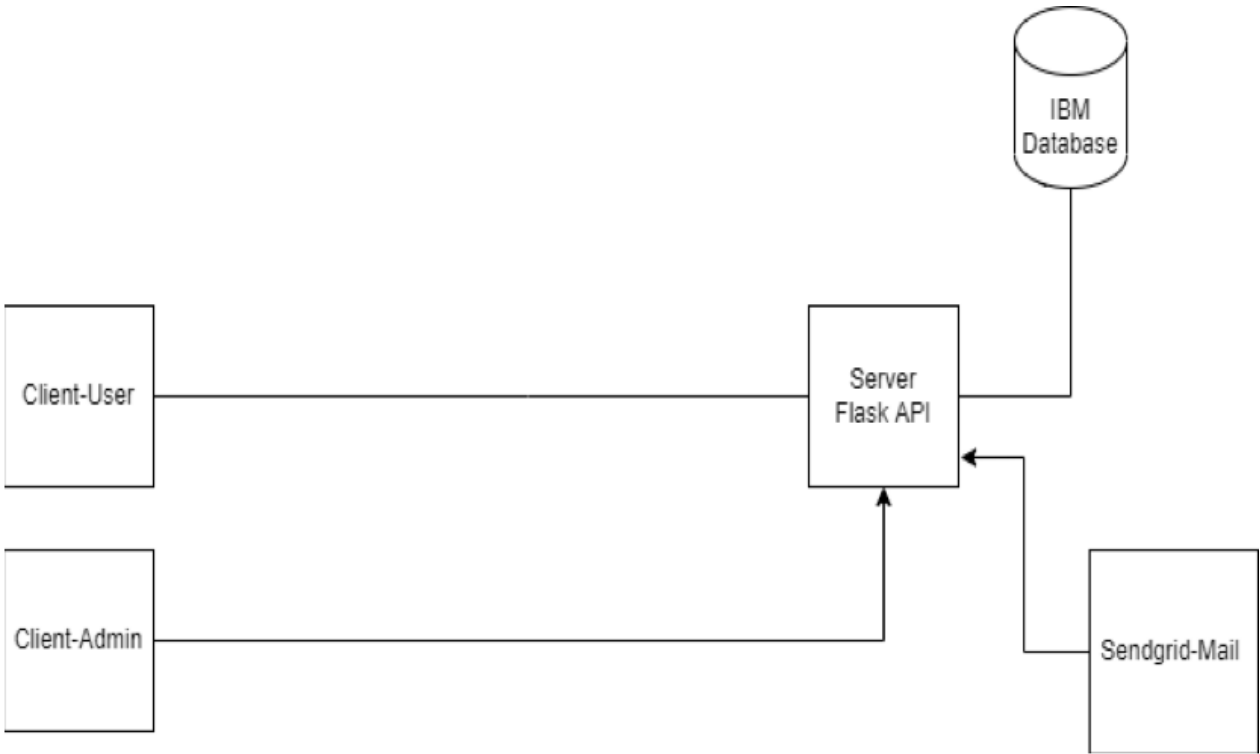
In formal methodologies like the Structured Systems Analysis and Design Method, they are frequently included. DFDs may initially resemble flowcharts or the Unified Modeling Language.
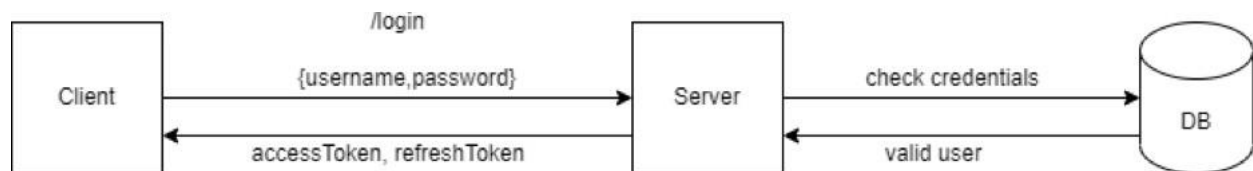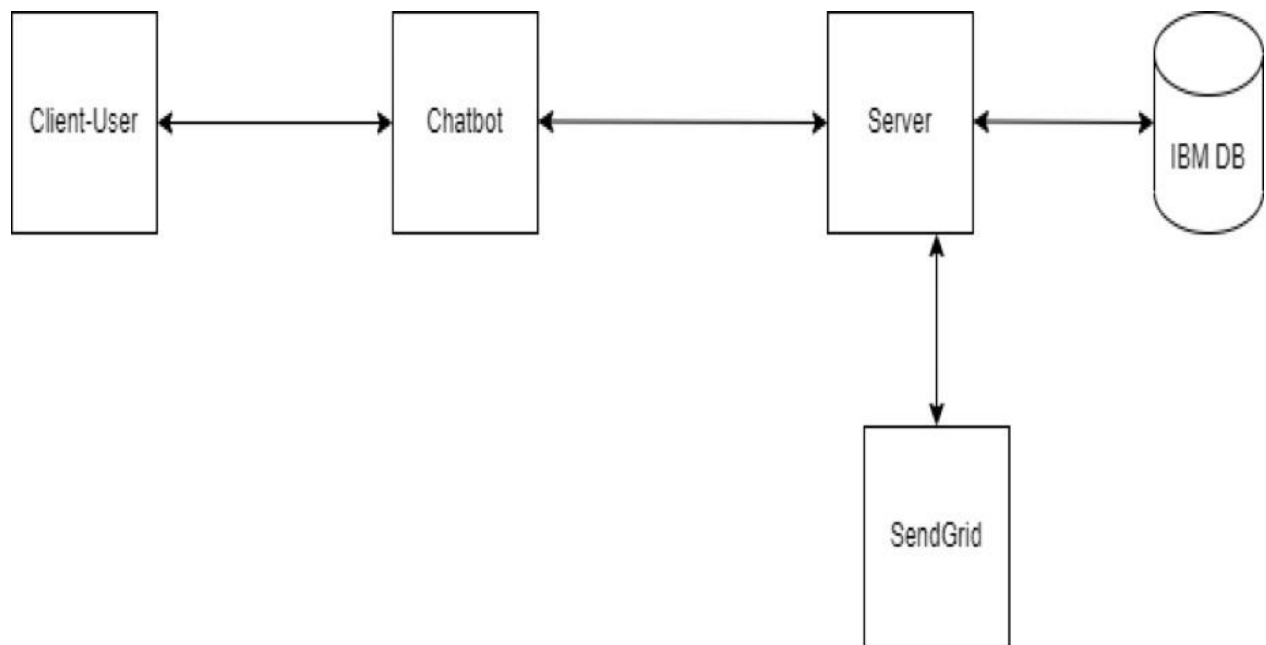
1. Advantages of data flow diagram:

○ It aids in describing the boundaries of the system.

○ It is beneficial for communicating existing system knowledge to the users.

○ A straightforward graphical technique which is easy to recognise.

○ DFDs can provide a detailed representation of system components.

○ It is used as the part of system documentation file.

○ DFDs are easier to understand by technical and nontechnical audiences

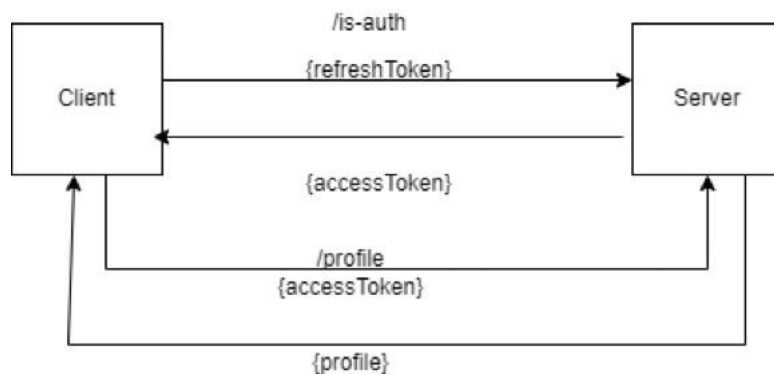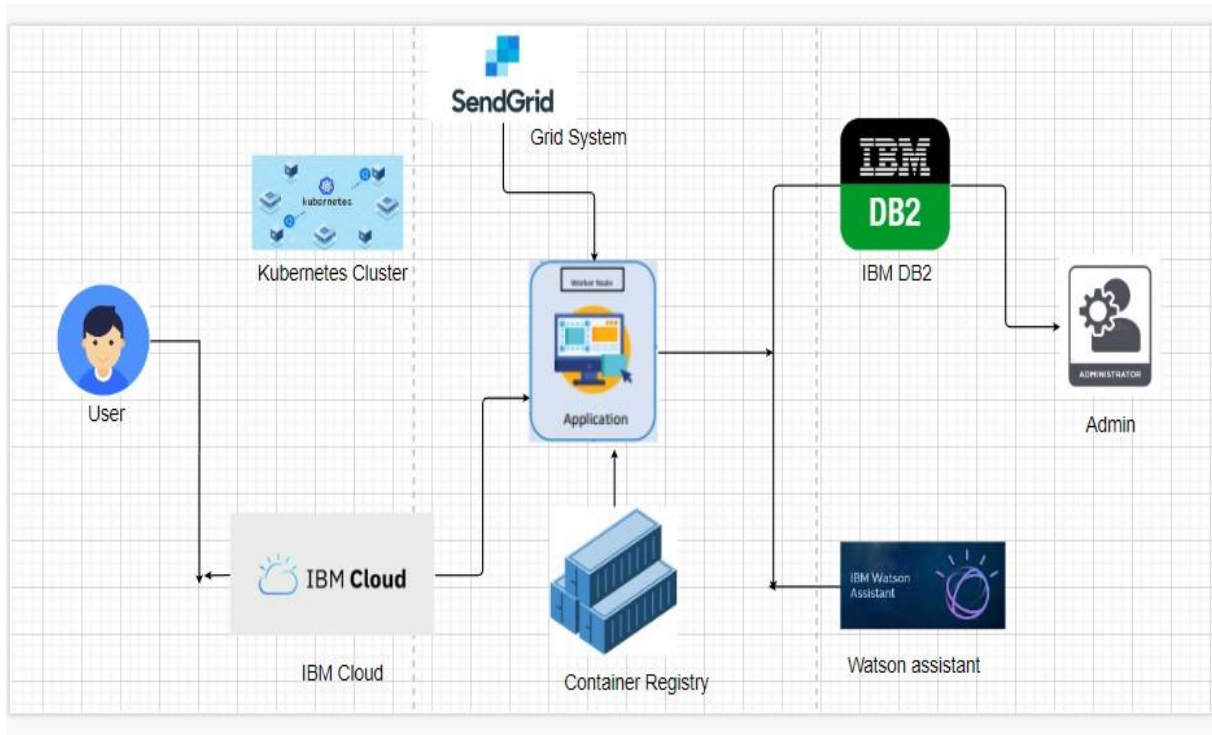○ It supports the logic behind the data flow within the system.
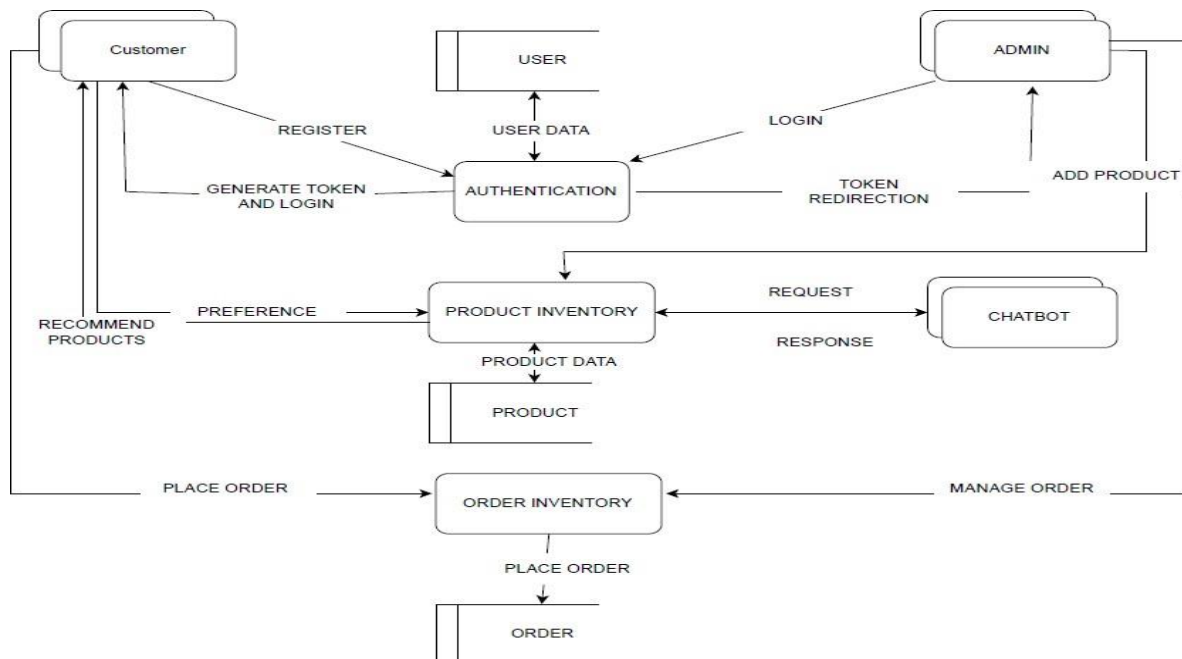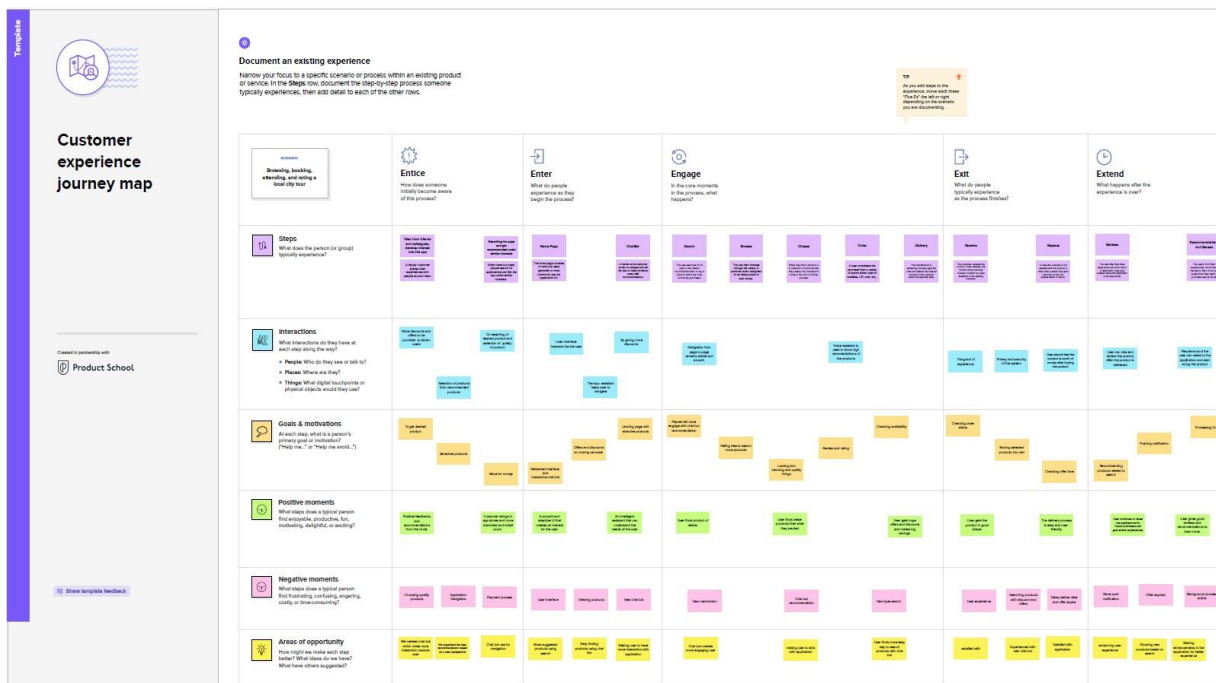
Solution & Technical Architecture

Client-User ⟷ **Chatbot** ⟷ **Server** ⟷ **IBM DB**

Server ⟷ **SendGrid**

/login

Client → **Server**
{username,password}

Server → **DB**
check credentials

Server ← **DB**
valid user

Client ← **Server**
accessToken, refreshToken

On New Session

/is-auth

Client → **Server**
{refreshToken}

Client ← **Server**
{accessToken}

/profile
{accessToken}

{profile}

User Stories

6. PROJECT PLANNING & SCHEDULING

Sprint Planning & Estimation

Use the below template to create product backlog and sprint schedule

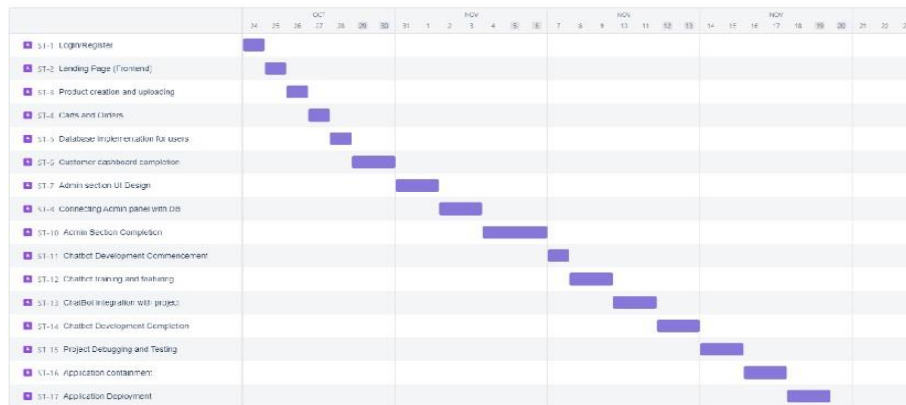| Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	User Interface (Front End)	USN-1	The user gets visualizations of the landing page and register pages through which they can sign up or login.	20	High	Geetha S Karthika B Menaga S Nithya Jayashri P Janani M
Sprint-2	Administrator Handle	USN-2	The administrator can login to the database and upload the products and shipper details	20	High	Geetha S Karthika B Menaga S Nithya Jayashri P Janani M
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	Geetha S Karthika B Menaga S Nithya Jayashri P Janani M
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	Geetha S Karthika B Menaga S Nithya Jayashri P Janani M

Sprint Delivery Schedule

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

Reports from JIRA

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.

Components and technology stack

S.NO	Component	Technology	Description
1.	User Interface	HTML, CSS, JavaScript/Angular JS / React JS.	User can interact with the application through chatbot for good Human-computer interface.
2.	Application Logic-1	Java Python	The application will have the login /signup page where the user can login into the main dashboard or they can register into the application.
3.	Application Logic-2	IBM Watson STT service	The application contains a Chatbot where the user needs to give their details like ✓ gender, ✓ age, ✓ type of product these were they wish to buy using Watson assistant through chatbot.
4.	Application Logic-3	IBM Watson Assistant	User's will get the recommendations based on their interests, can get the details about offers, discounts and chatbot will send a notification to customers if the order is confirmed.
5.	Database	MySQL, NoSQL,	Customer's details and order are stored in the database and whenever we can be fetch and retrieve data from database.
6.	Cloud Database	IBM DB2, IBM Cloud account	With use of Database Service on Cloud, user can access all the data stored in the cloud over a network from any device and user's data are stored in a well secure manner.
7.	File Storage	IBM Block Storage or Other Storage or Other Storage Service or Local Filesystem	Previously ordered product details and other customer details can be stored in the IBM Block Storage as the data kept inside are highly protected.
8.	Infrastructure (Server/Cloud)	Local, Cloud Foundry, Kubernetes, Docker	Chatbot with updated services can be deployed in an IBM cloud by using Watson assistant.

Components and technology stack

26

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 sy

stem.

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.
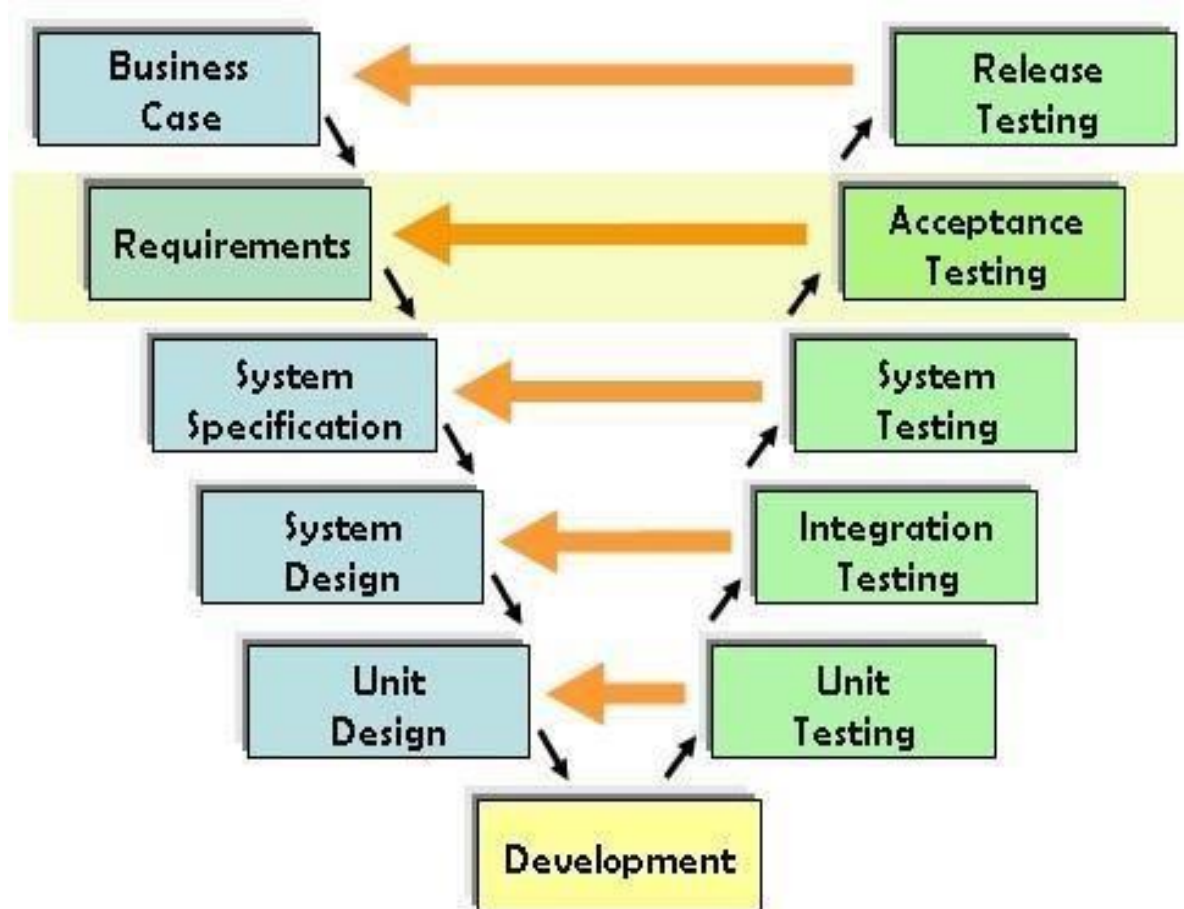
Fig 7.1) Testing architecture

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 [ProductName] project at the time of the release to User Acceptance Testing (UAT).

2. Defect Analysis

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

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	5	4	2	3	14
Duplicate	1	0	3	0	4
External	0	1	0	1	2
Fixed	11	2	4	20	37
Not Reproduced	0	0	1	0	1
Skipped	0	0	0	1	1
Won't Fix	0	0	1	1	2
Totals	17	7	11	26	61

3. Test Case Analysis

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

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	7	0	0	7
Client Application	51	0	0	51
Security	2	0	0	2
Outsource Shipping	3	0	0	3
Exception Reporting	9	0	0	9
Final Report Output	4	0	0	4
Version Control	2	0	0	2

7. CODING & SOLUTIONING

Feature 1 - Backend

i) app.py

```
from datetime import datetime

from flask import Flask,request,send_from_directory from .lib import db,validation_error
import ibm_db

from flask_cors import CORS

import os

from .api import auth_bp,category_bp,product_bp,cart_bp,order_bp app = Flask( name )
app.config['MAX_CONTENT_LENGTH'] = 1024 * 1024

app.config['UPLOAD_EXTENSIONS'] = ['.jpg', '.png', '.gif'] app.config['UPLOAD_PATH'] = 'uploads'


CORS(app) # This will enable CORS for all routes


db.get_db()


app.register_blueprint(auth_bp.auth_bp,url_prefix="/api/v1/auth")
app.register_blueprint(category_bp.category_bp,url_prefix="/api/v1/category")
app.register_blueprint(product_bp.product_bp,url_prefix="/api/v1/product") app.register_blueprint(cart_bp.cart_bp,url_prefix="/api/v1/cart")

app.register_blueprint(order_bp.order_bp,url_prefix="/api/v1/order")
```

The API source files are given as follows:

ii) auth_bp.py

```python
from flask import Blueprint,jsonify,g,request

import ibm_db

from passlib.hash import sha256_crypt

import jwt


from ..lib import validation_error

from ..lib import exception

from ..lib import db


auth_bp = Blueprint("auth",__name__)




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

def check():

    print(g.get("db"))

    return jsonify({"msg":"hi"})



@auth_bp.route('/register',methods=['POST'])

def reg():

    try:
```

```python
        data = request.get_json()

        name=data['name']

        email=data['email']

        password=data['password']

        mobile_no=data['mobileNo']

        print(email,password,name,mobile_no)

        insert_sql="INSERT INTO USER(name,email,password,role,mobilenumber) VALUES(?,?,?,?,?)"

        prep_stmt = ibm_db.prepare(db.get_db(), insert_sql)

        ibm_db.bind_param(prep_stmt,1,name)

        ibm_db.bind_param(prep_stmt,2,email)

        ibm_db.bind_param(prep_stmt,3,sha256_crypt.encrypt(password))

        ibm_db.bind_param(prep_stmt,4,"user")

        ibm_db.bind_param(prep_stmt,5,mobile_no)

        ibm_db.execute(prep_stmt)

        return {"message":'Created'},201



    except Exception as e:

        return exception.handle_exception(e)




@auth_bp.route('/me',methods=['GET'])

def getMe():

    try:

        token = request.headers['Authorization']

        if (not token):
```

```python
        return validation_error.throw_validation("Please login",401)

    decoded = jwt.decode(token,"secret",algorithms=["HS256"])

    select_sql = "SELECT * FROM USER WHERE ID=?"

    prep_stmt = ibm_db.prepare(db.get_db(), select_sql)

    ibm_db.bind_param(prep_stmt,1,decoded['id'])

    ibm_db.execute(prep_stmt)

    isUser=ibm_db.fetch_assoc(prep_stmt)

    return  isUser

  except Exception as e:

      return exception.handle_exception(e)




@auth_bp.route('/login',methods=['POST'])

def auth_log():

  try:

    data = request.get_json()

    print(data)

    email=data['email']

    password=data['password']

    select_sql = "SELECT * FROM USER WHERE EMAIL=?"

    prep_stmt = ibm_db.prepare(db.get_db(), select_sql)

    ibm_db.bind_param(prep_stmt,1,email)

    ibm_db.execute(prep_stmt)

    isUser=ibm_db.fetch_assoc(prep_stmt)

    print(isUser)
```

```python
        if not isUser:

            return validation_error.throw_validation("Invalid Credentials",400)

        if not sha256_crypt.verify(password,isUser['PASSWORD']):

            return validation_error.throw_validation("Invalid Credentials",400)

        encoded_jwt = jwt.encode({"id":isUser['ID'],"role":isUser['ROLE']},"secret",algorithm="HS256")

        isUser["token"] = encoded_jwt

        return  isUser

    except Exception as e:

        return exception.handle_exception(e)
Footer
```

iii) cart_bp.py

```python
from flask import Blueprint,request

import ibm_db

from ..lib import validation_error

from ..lib.auth import check_auth

from ..lib import exception

from ..lib import db




cart_bp = Blueprint("cart",__name__)




@cart_bp.route("/",methods=['POST'])
```

```python
def add_cart():

 try:

   user_id =check_auth(request)

   data=request.get_json()

   product=data['product']

   select_sql = "SELECT * FROM PRODUCT WHERE ID=?"

   prepare_select =ibm_db.prepare(db.get_db(),select_sql)

   ibm_db.bind_param(prepare_select,1,product)

   ibm_db.execute(prepare_select)

   is_product = ibm_db.fetch_assoc(prepare_select)


   print(is_product)


   if not is_product:

     return validation_error.throw_validation("No Product found",404)


   if(is_product['STOCK']<=0):

     return validation_error.throw_validation("No Stock found",404)


   print("Hey")

   insert_sql="INSERT INTO CART(user,product) VALUES(?,?)"

   prep_stmt = ibm_db.prepare(db.get_db(), insert_sql)

   ibm_db.bind_param(prep_stmt,1,user_id)

   ibm_db.bind_param(prep_stmt,2,product)

   ibm_db.execute(prep_stmt)
```

```python
    print("heyy")


    update_sql="UPDATE PRODUCT SET stock=? WHERE ID=?"

    update_stmt = ibm_db.prepare(db.get_db(), update_sql)

    ibm_db.bind_param(update_stmt,1,is_product['STOCK']-1 or 0)

    ibm_db.bind_param(update_stmt,2,product)

    ibm_db.execute(update_stmt)




  print("sdd")

  return {"message":'Created'},201

 except Exception as e:

  return exception.handle_exception(e)



@cart_bp.route("/",methods=['DELETE'])

def delete_user_cart():

 try:

  user_id =check_auth(request)

  insert_sql="DELETE FROM CART WHERE USER=?"

  prep_stmt = ibm_db.prepare(db.get_db(), insert_sql)

  ibm_db.bind_param(prep_stmt,1,user_id)



  ibm_db.execute(prep_stmt)

  return {"message":'Deleted'},201
```

```python
        except Exception as e:

          return exception.handle_exception(e)




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

def get_cart():

 try:

    user_id =check_auth(request)

    insert_sql="SELECT PRODUCT.ID AS product_id,cart_id,
category,category_name,product_name,description,price,stock,image,brand,specificity,CART.user as user
FROM CART JOIN PRODUCT ON CART.PRODUCT=PRODUCT.ID JOIN CATEGORY ON
PRODUCT.CATEGORY = CATEGORY.ID WHERE CART.USER=?"

    prep_stmt = ibm_db.prepare(db.get_db(), insert_sql)

    ibm_db.bind_param(prep_stmt,1,user_id)



    ibm_db.execute(prep_stmt)

    products=[]

    product=ibm_db.fetch_assoc(prep_stmt)

    while(product != False):

    products.append(product)

      product = ibm_db.fetch_assoc(prep_stmt)

    print(products)

    return products or [],200



 except Exception as e:
```

```python
        return exception.handle_exception(e)




@cart_bp.route("/<product>/<id>",methods=['DELETE'])

def delete_cart(product,id):

 try:

   user_id =check_auth(request)

   print(product,id,user_id)


   select_sql = "SELECT * FROM PRODUCT WHERE ID=?"

   prepare_select =ibm_db.prepare(db.get_db(),select_sql)

   ibm_db.bind_param(prepare_select,1,product)

   ibm_db.execute(prepare_select)

   is_product = ibm_db.fetch_assoc(prepare_select)


   print(is_product)


   if not is_product:

     return validation_error.throw_validation("No Product found",404)


   print("ff")

   insert_sql="DELETE FROM CART WHERE CART_ID=? AND user=?"

   prep_stmt = ibm_db.prepare(db.get_db(), insert_sql)

   ibm_db.bind_param(prep_stmt,1,id)

   ibm_db.bind_param(prep_stmt,2,user_id)
```

```python
    ibm_db.execute(prep_stmt)

    print("aa")

    update_sql="UPDATE PRODUCT SET stock=? WHERE ID=?"

    update_stmt = ibm_db.prepare(db.get_db(), update_sql)

    ibm_db.bind_param(update_stmt,1,is_product['STOCK']+1)

    ibm_db.bind_param(update_stmt,2,product)

    ibm_db.execute(update_stmt)

    return {"message":'Deleted'},200

  except Exception as e:

    return exception.handle_exception(e)
Footer
```

iv) category_bp.py

```python
from flask import Blueprint,request,jsonify

import ibm_db

from ..lib import exception

from ..lib import db




category_bp = Blueprint("category",__name__)




@category_bp.route("/get",methods=["GET"])

def get_category():

 try:
```

```python
    select_sql = "SELECT * FROM CATEGORY WHERE"

    prep_stmt = ibm_db.prepare(db.get_db(), select_sql)

    ibm_db.execute(prep_stmt)

    categories=[]

    category=ibm_db.fetch_assoc(prep_stmt)

    while(category != False):

    categories.append(category)

      category = ibm_db.fetch_assoc(prep_stmt)

    print(categories)

    # return categories,200

    return jsonify(categories),200

  except Exception as e:

    return exception.handle_exception(e)


@category_bp.route("/create",methods=["POST"])

def add_category():

 try:

   data = request.get_json()

   category = data['category']

   insert_sql="INSERT INTO CATEGORY(category_name) VALUES(?)"

   prep_stmt = ibm_db.prepare(db.get_db(), insert_sql)

   ibm_db.bind_param(prep_stmt,1,category)

   ibm_db.execute(prep_stmt)

 return {"message":'Created'},201

 except Exception as e:
```

```python
    return exception.handle_exception(e)




@category_bp.route("/<id>",methods=["DELETE"])

def get_category_id(id):

 try:

   print(id)

   select_sql = "DELETE FROM CATEGORY WHERE ID=?"

   prep_stmt = ibm_db.prepare(db.get_db(), select_sql)

   ibm_db.bind_param(prep_stmt,1,id)

   ibm_db.execute(prep_stmt)


   return {"message":'Deleted'},200

 except Exception as e:

   return exception.handle_exception(e)
```

v) image_bp.py

```python
from datetime import datetime

from flask import Blueprint,request

import ibm_db

import os

from ..lib import exception

from ..lib import db
```

```python
image_bp = Blueprint("image",__name__)


@image_bp.route('/image/<id>',methods=['POST'])

def uploadImage(id):

 try:

   uploaded_file = request.files['file']+datetime.date

   if uploaded_file.filename != '':

      uploaded_file.save(os.path.join('/uploads', uploaded_file.filename))

   insert_sql="UPDATE PRODUCT SET image=? WHERE ID=?"

   prep_stmt = ibm_db.prepare(db.get_db(), insert_sql)

   ibm_db.bind_param(prep_stmt,1,uploaded_file)

   ibm_db.bind_param(prep_stmt,2,id)


   ibm_db.execute(prep_stmt)

 return {"message":'Updated'},200

 except Exception as e:

   return exception.handle_exception(e)


@image_bp.route('/<filename>')

def upload(filename):

 try:

   return send_from_directory("/uploads", filename)

 except Exception as e:

   return exception.handle_exception(e)
```

v) product_bp.py

```python
from flask import Blueprint,request,jsonify

import ibm_db

from ..lib import exception

from ..lib import db




product_bp = Blueprint("product",__name__)


@product_bp.route("/create",methods=['POST'])

def add_product():

 try:

    data = request.get_json()

    product_name=data['product_name']

    category=data['category']

    description = data['description']

    stock=data['stock']

    price = data['price']

    insert_sql="INSERT INTO PRODUCT(product_name,category,description,stock,price)
VALUES(?,?,?,?,?)"

    prep_stmt = ibm_db.prepare(db.get_db(), insert_sql)

    ibm_db.bind_param(prep_stmt,1,product_name)

    ibm_db.bind_param(prep_stmt,2,category)

    ibm_db.bind_param(prep_stmt,3,description)

    ibm_db.bind_param(prep_stmt,4,stock)
```

```python
        ibm_db.bind_param(prep_stmt,5,price)

        ibm_db.execute(prep_stmt)

        return {"message":'Created'},200

    except Exception as e:

        return exception.handle_exception(e)


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

def get_product():

    try:

        # select_sql = "SELECT PRODUCT.ID AS product_id,
category,category_name,product_name,description,price,stock,image FROM PRODUCT JOIN CATEGORY
ON CATEGORY.ID=PRODUCT.CATEGORY"

        select_sql = "SELECT * FROM PRODUCT WHERE"

        prep_stmt = ibm_db.prepare(db.get_db(), select_sql)

        ibm_db.execute(prep_stmt)

        products=[]

        product=ibm_db.fetch_assoc(prep_stmt)

        while(product != False):

        products.append(product)

            product = ibm_db.fetch_assoc(prep_stmt)

        print(products)

        return jsonify(products) or [],200

    except Exception as e:

        return exception.handle_exception(e)
```

```python
@product_bp.route("/<id>",methods=['GET'])

def get_product_id(id):

  try:

    # select_sql = "SELECT PRODUCT.ID AS product_id,
category,category_name,product_name,description,price,stock,image FROM PRODUCT JOIN CATEGORY
ON CATEGORY.ID=PRODUCT.CATEGORY WHERE PRODUCT.ID=?"

    select_sql = "SELECT * FROM PRODUCT WHERE PRODUCT.ID=?"

    prep_stmt = ibm_db.prepare(db.get_db(), select_sql)

    ibm_db.bind_param(prep_stmt,1,id)

    ibm_db.execute(prep_stmt)

    product=ibm_db.fetch_assoc(prep_stmt)

    print(product)

    return product or [],200

  except Exception as e:

    return exception.handle_exception(e)




@product_bp.route("/<id>",methods=['PUT'])

def update_product(id):

  try:

    data = request.get_json()

    product_name=data['product_name']

    category=data['category']

    description = data['description']

    stock=data['stock']

    price = data['price']
```

```python
        insert_sql="UPDATE PRODUCT SET product_name=?,category=?,description=?,stock=?,price=?
WHERE ID=?"

        prep_stmt = ibm_db.prepare(db.get_db(), insert_sql)

        ibm_db.bind_param(prep_stmt,1,product_name)

        ibm_db.bind_param(prep_stmt,2,category)

        ibm_db.bind_param(prep_stmt,3,description)

        ibm_db.bind_param(prep_stmt,4,stock)

        ibm_db.bind_param(prep_stmt,5,price)

        ibm_db.bind_param(prep_stmt,6,id)

        ibm_db.execute(prep_stmt)

        return {"message":'Updated'},200

    except Exception as e:

        return exception.handle_exception(e)




@product_bp.route("/<id>",methods=['DELETE'])

def delete_product(id):

    try:

        insert_sql="DELETE FROM PRODUCT WHERE ID=?"

        prep_stmt = ibm_db.prepare(db.get_db(), insert_sql)

        ibm_db.bind_param(prep_stmt,1,id)

        ibm_db.execute(prep_stmt)

        return {"message":'Deleted'},200

    except Exception as e:

        return exception.handle_exception(e)
```

The Library source code files for the application are given as follows

vi) auth.py

```python
import jwt

from . import validation_error


def check_auth(request):

  token = request.headers['Authorization']

  if (not token):

    return validation_error.throw_validation("Please login",401)

  decoded = jwt.decode(token,"secret",algorithms=["HS256"])

  return decoded['id']
```

vii) db.py

```python
import ibm_db


conn = None


def get_db():

    global conn

    print(conn)

    if conn == None:

        conn =
ibm_db.connect("DATABASE=bludb;HOSTNAME=9938aec0-8105-433e-8bf9-0fbb7e483086.c1ogj3sd0tgtu
0lqde00.databases.appdomain.cloud;PORT=32459;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRo
otCA.crt;UID=yvp21124;PWD=jNDAkHSrZNopa2oe","","")
```

```python
    return conn
```

viii) exception.py

```python
from flask import json


def handle_exception(e):

    exception_name = type(e).__name__

    exception_str = e.__str__()

    print(exception_name,exception_str.find('803'))

    status_code=500

    response={

        "status":"fail",

        "message":"Something went wrong"

    }

    if exception_name=='KeyError':

        status_code=400

        response["message"] = "Please fill all fields"



    if exception_name=='ValidationException':

        print(e.args)

        status_code=400

        response["message"] = "Please fill all fields"
```

```python
    if exception_str.find('803') >=0:

        print('Db')

        status_code=400

        response["message"] = "Invalid value provided"


    if exception_name == 'DecodeError':

        print("token error")

        status_code=401

        response['message']='Token expired.Please login'




    return json.dumps(response), status_code, {'ContentType':'application/json'}
```

ix) sendmail.py

```python
import os

from sendgrid import SendGridAPIClient

from sendgrid.helpers.mail import Mail

from ..lib import exception




def sendemail():
```

```python
    message = Mail(

        from_email='from_email@example.com',

        to_emails='to@example.com',

        subject='ORDER CONFIRMATION',

        html_content='<strong>Your order placed</strong>')

    try:

        sg = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))

        response = sg.send(message)

        print(response.status_code)

        print(response.body)

        print(response.headers)

    except Exception as e:

        return exception.handle_exception(e)
```

x) validation_error.py

```python
from flask import json


def throw_validation(msg,code):

 return json.dumps({"status":"fail","message":msg}),code,{'ContentType':'application/json'}
```

7.1 Feature 2 - Frontend

The main face of the application acts as a linker for all other frontend and backend sources is as follows.

i) index.html

```html
<!doctype html>
```

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

<head>

 <meta charset="utf-8">

 <title>SmartFashionRecommender</title>

 <base href="/">

 <meta name="viewport" content="width=device-width, initial-scale=1">

 <link rel="icon" type="image/x-icon" href="/assets/logo.png">

 <link rel="stylesheet"

href="https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined:opsz,wght,FILL,GRAD@20..48,
100..700,0..1,-50..200" />

 <link rel="preconnect" href="https://fonts.googleapis.com">

 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

 <link

href="https://fonts.googleapis.com/css2?family=Cinzel+Decorative:wght@700&family=Tangerine:wght@700
&display=swap"

   rel="stylesheet">

</head>

<body>

 <app-root></app-root>

</body>

</html>
```

ii) main.ts

```typescript
import { enableProdMode } from '@angular/core';

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';


import { AppModule } from './app/app.module';

import { environment } from './environments/environment';


if (environment.production) {

  enableProdMode();

}


platformBrowserDynamic().bootstrapModule(AppModule)

  .catch(err => console.error(err));
```

iii) style.css

```css
/* You can add global styles to this file, and also import other style files */

:root {

  --primary-color: #b0e6dc;

  --secondary-color: #c0c2c3;

}

html, body {

  height: 100%;

  margin: 0;

 }

.cursor-pointer {
```

```css
    cursor: pointer;

}

/* button styles */

.primary-btn {

    background-color: black;

    color: white;

    border: 0;

    padding: 0.7rem 2.5rem;

    border-radius: 10px;

}


.secondary-btn {

    background-color: white;

    color: black;

    border: 1px solid black;

    padding: 0.7rem 2.5rem;

    border-radius: 10px;

}


// login screen and signup screen

.login-container {

    background-image: url("/assets/login-banner.png");

    background-repeat: no-repeat;

    background-size: cover;

}
```

```css
.login-text {

    font-family: "Cinzel Decorative";

    letter-spacing: 0.7rem;

}

.login-logo-img {

    width: 4rem;

    height: 4rem;

}



.login-form-container {

    background: rgba(255, 255, 255, 0.2);

    border-radius: 16px;

    box-shadow: 0 4px 30px rgba(0, 0, 0, 0.1);

    backdrop-filter: blur(5px);

    -webkit-backdrop-filter:  blur(5px);

    border: 1px solid rgba(255, 255, 255, 0.3);

}



.login-input {

    background: rgba(255, 255, 255, 0.2);

    border-radius: 16px;

    box-shadow: 0 4px 30px rgba(0, 0, 0, 0.1);

    border: 1px solid rgba(255, 255, 255, 0.3);

    margin: 1rem 0;

    padding: 0 1rem;
```

```css
  height: 3rem;

  width: 25rem;

}



@media only screen and (max-width: "600px") {

  .login-input {

    width: 100%;

  }

}
```

iv)test.js

```js
import 'zone.js/testing';

import { getTestBed } from '@angular/core/testing';
```

```typescript
  platformBrowserDynamicTesting
} from '@angular/platform-browser-dynamic/testing';


declare const require: {
  context(path: string, deep?: boolean, filter?: RegExp): {
    <T>(id: string): T;
    keys(): string[];
  };
};


// First, initialize the Angular testing environment.
getTestBed().initTestEnvironment(
BrowserDynamicTestingModule,
platformBrowserDynamicTesting(),
);


// Then we find all the tests.
const context = require.context('./', true, /\.spec\.ts$/);
// And load the modules.
context.keys().forEach(context);
```

v)index.d.ts

```typescript
export {};
```

```
    watsonAssistantChatOptions: any; // ☐ turn off type checking

  }

}
```

vi) environment.prod.ts

```
export const environment = {

  production: true,

  API_HOST: '',


  API_BASE: '/api/v1',

};
```

vii) environment.ts

```
export const environment = {

  production: false,

  API_HOST: 'http://127.0.0.1:5000',

  // API_HOST: 'http://localhost:5000',

  API_BASE: '/api/v1',

};
```

3.admin pages

  i) **add-product-routing.module.ts**

```
    import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { AddProductComponent } from './add-product.component';
```

```
const routes: Routes = [{ path: '', component: AddProductComponent }];

@NgModule({

imports: [RouterModule.forChild(routes)],

  exports: [RouterModule]

})

export class AddProductRoutingModule { }
```

ii) **add-product-routing.module.ts**

```html
<section class="container">

  <div class="row">

    <h4>Add new product</h4>

    <hr>

    <form class="col-12 col-md-6" [formGroup]="addProductForm" (ngSubmit)="createProduct()">

      <input type="text" class="form-input w-100" placeholder="product name"
formControlName="product_name">

      <textarea type="text" class="form-input w-100" placeholder="Description"
formControlName="description"></textarea>

      <input type="number" class="form-input w-100" placeholder="Price" min="1"
formControlName="price">

      <input type="number" class="form-input w-100" placeholder="Stock" min="1"
formControlName="stock">

      <select name="category" class="form-input w-100 py-3" formControlName="category">

        <option value="">Select Category</option>

        <option [value]="c.ID" *ngFor="let c of categoryList">{{c.CATEGORY_NAME}}</option>

      </select>

      <div>

        <button class="primary-btn" type="submit">Add Product</button>
```

```html
          </div>

        </form>

        <!-- <div class="col-12 col-md-6 text-center">

          <p>select product image</p>

          <img src="/assets/products/4.jpg" alt="Product image" class="product-img">

        </div> -->

      </div>

</section>
```

iii) **add-product.component.scss**

```scss
.form-input {

    border: 2px solid black;

    padding: .5rem .3rem;

    border-radius: 10px;

    margin: .3rem 0;

}



.product-img {

    width: 15rem;

    height: 15rem;

}
```

iv) **add-product.component.spec.ts**

```typescript
import { ComponentFixture, TestBed } from '@angular/core/testing';
```

```typescript
import { AddProductComponent } from './add-product.component';

describe('AddProductComponent', () => {

  let component: AddProductComponent;

  let fixture: ComponentFixture<AddProductComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      declarations: [ AddProductComponent ]

    })

    .compileComponents();

    fixture = TestBed.createComponent(AddProductComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});
```

v)   **add-product.component.ts**

```typescript
import { Component, OnInit } from '@angular/core';
```

```typescript
import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { Router } from '@angular/router';

import { CategoryService } from 'src/app/services/category/category.service';

import { ProductService } from 'src/app/services/product/product.service';



@Component({

  selector: 'app-add-product',

  templateUrl: './add-product.component.html',

  styleUrls: ['./add-product.component.scss']

})

export class AddProductComponent implements OnInit {

  categoryList: any = [];

  addProductForm: FormGroup;

  constructor(

    private categoryServe: CategoryService,

    private fb: FormBuilder,

    private productServe: ProductService,

    private router: Router,

  ) {

    this.addProductForm = this.fb.group({

      product_name: ['', [Validators.required]],

      category: ['', [Validators.required]],

      description: ['', [Validators.required]],

      stock: ['', [Validators.required]],

      price: ['', [Validators.required]],
```

```typescript
  })

}


ngOnInit(): void {

  this.getCategoryList();

}


// get list

async getCategoryList(): Promise<void> {

  try {

    this.categoryList = await this.categoryServe.getCategory();

  } catch (error) {

    console.log(error);



  }

}


// create product

async createProduct(): Promise<void> {

  try {

    const data = await this.productServe.createProduct(this.addProductForm.value);

    console.log(data);

    this.router.navigate(['/admin'])

  } catch (error) {

    console.log(error);
```

```
    }

  }

}
```

vi) **add-product.module.ts**

```typescript
import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';


import { AddProductRoutingModule } from './add-product-routing.module';

import { AddProductComponent } from './add-product.component';

import { ReactiveFormsModule } from '@angular/forms';


@NgModule({

  declarations: [

  AddProductComponent

  ],

  imports: [

    CommonModule,

    AddProductRoutingModule,

    ReactiveFormsModule

  ]

})

export class AddProductModule { }
```

Category type:

i)category-routing.module.ts

```typescript
import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { CategoryComponent } from './category.component';


const routes: Routes = [

  { path: '', component: CategoryComponent },

];


@NgModule({

  imports: [RouterModule.forChild(routes)],

  exports: [RouterModule]

})

export class CategoryRoutingModule { }
```

ii)category.component.html

```html
<section class="container">

  <div class="row">

    <div class="d-flex justify-content-between mb-3">

      <h5>CATEGORY LIST</h5>

      <div class="position-relative">

        <button class="primary-btn" (click)="toggleAddCategory()">Add

          Category</button>

        <span class="position-absolute add-category-card shadow border text-success"
```

```html
    *ngIf="successMessage !== ''">{{successMessage}}</span>

    <!-- add category -->

    <form class="position-absolute add-category-card shadow border"
*ngIf="showAddCategory"

        [formGroup]="addCategory" (ngSubmit)="createCategory()">

        <input type="text" placeholder="New Category Name" class="form-input"
formControlName="category">

        <div class="text-center">

            <button class="secondary-btn" type="submit" style="background-color:
var(--primary-color);"

                [disabled]="this.addCategory.invalid">Create</button>

        </div>

    </form>

  </div>

</div>

<hr>

<!-- orders list -->

<div *ngIf="categoryList.length === 0">

    Loading

    <div class="spinner-border text-dark ms-3" style="vertical-align: bottom;" role="status">

        <span class="visually-hidden">Loading...</span>

    </div>

</div>

<div class="col-12" *ngIf="categoryList.length > 0">

    <div class="table-responsive">

        <!-- table -->
```

```html
<table class="table table-striped table-hover table-bordered w-auto">

  <thead>

    <tr>

      <th scope="col">#</th>

      <th scope="col">Category</th>

      <th scope="col">Delete</th>

    </tr>

  </thead>

  <tbody class="align-middle">

    <tr *ngFor="let t of categoryList, index as i">

      <th scope="row">{{i+1}}</th>

      <td>{{t.CATEGORY_NAME}}</td>

      <td>

        <span class="material-symbols-outlined cursor-pointer text-danger"

          (click)="deleteCategory(t.ID)">

          delete

        </span>

      </td>


    </tr>

  </tbody>

</table>

</div>

</div>

</div>
```

```
</section>
```

iii) **category.component.scss**

```scss
.form-input {

    border: 2px solid black;

    padding: .7rem .3rem;

    border-radius: 10px;

    margin: .3rem 0;

}


.add-category-card {

    z-index: 3;

    background-color: white;

    padding: 1rem;

    left: -100%;

    top: 4rem;

}
```

iv) **category.component.spec.ts**

```ts
import { ComponentFixture, TestBed } from '@angular/core/testing';


import { CategoryComponent } from './category.component';


describe('CategoryComponent', () => {

  let component: CategoryComponent;
```

```typescript
  let fixture: ComponentFixture<CategoryComponent>;


  beforeEach(async () => {

    await TestBed.configureTestingModule({

      declarations: [ CategoryComponent ]

    })

    .compileComponents();


    fixture = TestBed.createComponent(CategoryComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });


  it('should create', () => {

    expect(component).toBeTruthy();

  });

});
```

v) category.component.ts

```typescript
import { Component, OnInit } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { CategoryService } from 'src/app/services/category/category.service';
```

```typescript
@Component({

  selector: 'app-category',

  templateUrl: './category.component.html',

  styleUrls: ['./category.component.scss']

})

export class CategoryComponent implements OnInit {

  showAddCategory = false;

  addCategory: FormGroup;

  successMessage = '';

  categoryList: any = [];

  constructor(

    private fb: FormBuilder,

    private categoryServe: CategoryService,

  ) {

    this.addCategory = this.fb.group({

      category: ['', [Validators.required]],

    })

  }


  ngOnInit(): void {

    this.getCategorys();

  }



  async createCategory(): Promise<void> {
```

```typescript
  try {

    await this.categoryServe.createCategory(this.addCategory.value);

    this.setMessage('successfully added');

    this.showAddCategory = false;

    this.getCategorys();

    this.addCategory.reset();

  } catch (error) {

    console.log(error);

    this.setMessage('fail to added');

  }

}

// set error messages

setMessage(message: any): void {

this.successMessage = message;

setTimeout(() => {

  this.successMessage = '';

 }, 2000);

};


toggleAddCategory(): void {

 this.showAddCategory = !this.showAddCategory;

}


// get category list
```

```typescript
async getCategorys(): Promise<void> {

  try {

    this.categoryList = await this.categoryServe.getCategory();

  } catch (error) {

    console.log(error);

  }

}


// delete category

async deleteCategory(id: any): Promise<void> {

  try {

    await this.categoryServe.deleteCategory(id);

    this.getCategorys();

    this.setMessage("successfully Deteted");

  } catch (error) {

    console.log(error);

    this.setMessage("Fail to delete")

  }

}
```

vi) category.module.ts

```typescript
import { NgModule } from '@angular/core';
```

```typescript
import { CommonModule } from '@angular/common';


import { CategoryRoutingModule } from './category-routing.module';

import { CategoryComponent } from './category.component';

import { ReactiveFormsModule } from '@angular/forms';



@NgModule({

 declarations: [

 CategoryComponent,

 ],

 imports: [

   CommonModule,

   CategoryRoutingModule,

   ReactiveFormsModule

 ]

})

export class CategoryModule { }

```

Orders

i)orders-routing.module.ts

```typescript
import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { OrdersComponent } from './orders.component';
```

```
const routes: Routes = [{ path: '', component: OrdersComponent }];



@NgModule({

  imports: [RouterModule.forChild(routes)],

  exports: [RouterModule]

})

export class OrdersRoutingModule { }
```

ii)orders.component.html

```html
<section class="container">

  <div class="row">

    <div class="d-flex justify-content-between">

      <h5>ORDERS LIST</h5>

      <p class="fw-bold">Total orders {{'343'}}</p>

    </div>

    <hr>

    <!-- orders list -->

    <div class="col-12">

      <div class="table-responsive">

        <table class="table table-striped table-hover table-bordered">

          <thead>
```

```html
          <tr>

            <th scope="col">#</th>

            <th scope="col">User Name</th>

            <th scope="col">Product Name</th>

            <th scope="col">Product Price</th>

            <th scope="col">Order Date</th>

          </tr>

        </thead>

        <tbody class="align-middle">

          <tr *ngFor="let t of [1,1,1,1,1,1,1,1,1], index as i">

            <th scope="row">{{i+1}}</th>

            <td>Devendran</td>

            <td>Dress</td>

            <td>{{'322' | currency: 'INR'}}</td>

            <td>{{'27-08-2022'}}</td>

          </tr>

        </tbody>

      </table>

    </div>

  </div>

</div>

</section>
```

iii)orders.component.spec.ts

```typescript
import { ComponentFixture, TestBed } from '@angular/core/testing';
```

```typescript
import { OrdersComponent } from './orders.component';

describe('OrdersComponent', () => {

  let component: OrdersComponent;

  let fixture: ComponentFixture<OrdersComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      declarations: [ OrdersComponent ]

    })

    .compileComponents();

    fixture = TestBed.createComponent(OrdersComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});
```

iv)orders.component.ts

```
import { Component, OnInit } from '@angular/core';


@Component({

  selector: 'app-orders',

  templateUrl: './orders.component.html',

  styleUrls: ['./orders.component.scss']

})

export class OrdersComponent implements OnInit {


  constructor() { }


  ngOnInit(): void {

  }

}
```

v)orders.module.ts

```
import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';


import { OrdersRoutingModule } from './orders-routing.module';

import { OrdersComponent } from './orders.component';



@NgModule({

  declarations: [
```

```
    OrdersComponent

  ],

  imports: [

    CommonModule,

    OrdersRoutingModule

  ]

})

export class OrdersModule { }
```

Product list

i)product-list-routing.module.ts

```
import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { ProductListComponent } from './product-list.component';


const routes: Routes = [{ path: '', component: ProductListComponent }];


@NgModule({

  imports: [RouterModule.forChild(routes)],

  exports: [RouterModule]

})

export class ProductListRoutingModule { }
```

ii) product-list.component.html

```html
<section class="container-fluid">

  <div class="row">

    <div class="d-flex justify-content-between">

      <h5>PRODUCTS LIST</h5>

      <p class="fw-bold">Total products {{'343'}}</p>

    </div>

    <hr>

      <div  *ngIf="productsList.length === 0">

        Loading

        <div class="spinner-border text-dark ms-3" style="vertical-align: bottom;" role="status">

          <span class="visually-hidden">Loading...</span>

        </div>

      </div>

    <!-- list of products -->

    <div class="col-12" *ngIf="productsList.length > 0">

      <div class="table-responsive">

        <table class="table table-striped table-hover table-bordered">

          <thead>

            <tr>

              <th scope="col">#</th>

              <th scope="col">Image</th>

              <th scope="col">Product Name</th>

              <th scope="col">Description</th>
```

```html
            <th scope="col">Price</th>

            <th scope="col">Delete</th>

          </tr>

        </thead>

        <tbody class="align-middle">

          <tr *ngFor="let p of productsList, index as i">

            <th scope="row">{{i+1}}</th>

            <td><img src="{{BASE_URL}}{{p.IMAGE}}" alt="Product image"
class="product-img" /></td>

            <td>{{p.PRODUCT_NAME}}</td>

            <td>{{p.DESCRIPTION}}</td>

            <td>{{p.PRICE | currency: 'INR'}}</td>

            <td>

              <span class="material-symbols-outlined cursor-pointer text-danger">

                delete

              </span>

            </td>

          </tr>

        </tbody>

      </table>

    </div>

  </div>

</div>

</section>
```

iii) product-list.component.scss

```scss
.product-img {

    width: 4rem;

    height: 4.5rem;

}
```

iv) product-list.component.spec.ts

```typescript
import { ComponentFixture, TestBed } from '@angular/core/testing';



import { ProductListComponent } from './product-list.component';



describe('ProductListComponent', () => {

  let component: ProductListComponent;

  let fixture: ComponentFixture<ProductListComponent>;



  beforeEach(async () => {

    await TestBed.configureTestingModule({

      declarations: [ ProductListComponent ]

    })

    .compileComponents();



    fixture = TestBed.createComponent(ProductListComponent);

    component = fixture.componentInstance;
```

```
    fixture.detectChanges();

  });



  it('should create', () => {

    expect(component).toBeTruthy();

  });

});
```

v) product-list.component.ts

```typescript
import { Component, OnInit } from '@angular/core';

import { ProductService } from 'src/app/services/product/product.service';



@Component({

  selector: 'app-product-list',

  templateUrl: './product-list.component.html',

  styleUrls: ['./product-list.component.scss']

})
export class ProductListComponent implements OnInit {

productsList:any = [];

BASE_URL: any = 'http://127.0.0.1:5000/uploads/';



  constructor(

    private productServe: ProductService,

  ) { }
```

```ts
ngOnInit(): void {

  this.getProducts();

}



async getProducts(): Promise<void>{

  try {

    this.productsList = await this.productServe.getProducts();

    console.log(this.productsList, '00000000000');



  } catch (error) {

    console.log(error);

  }

}


}
```

vi) **product-list.module.ts**

```ts
import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';



import { ProductListRoutingModule } from './product-list-routing.module';

import { ProductListComponent } from './product-list.component';
```

```
@NgModule({

 declarations: [

 ProductListComponent

 ],

 imports: [

   CommonModule,

   ProductListRoutingModule

 ]

})

export class ProductListModule { }
```

Component/admin-shell

i)admin-shell.component.html

```html
<div class="container-fluid h-100">

   <div class="row h-100 flex-nowrap position-fixed w-100">

     <!-- nav bar -->

     <div class="col-8 col-md-3 col-lg-2 h-100 nav-con" *ngIf="showNav">

       <!-- logo image -->

       <div class="fw-bold text-dark">

         <img class="logo-img" src="/assets/logo.png" />

         <span class="d-none d-md-inline">New Fashion</span>

         <div class="d-md-none">

           <span>New Fashion</span>
```

```html
        </div>

    </div>

    <hr>

    <nav class="nav-bar">

      <ul>

        <li class="mt-2"><a class="py-3 ps-3 nav-link" routerLinkActive="active"
routerLink="/admin"

              [routerLinkActiveOptions]="{exact:true}">Products

              List</a></li>

        <li class="mt-2"><a class="py-3 ps-3 nav-link" routerLinkActive="active"
routerLink="orders"

              [routerLinkActiveOptions]="{exact:true}">Orders

              List</a></li>

        <li class="mt-2"><a class="py-3 ps-3 nav-link" routerLinkActive="active"
routerLink="category"

              [routerLinkActiveOptions]="{exact:true}">Category

              List</a></li>

        <li class="mt-2"><a class="py-3 ps-3 nav-link" routerLinkActive="active"
routerLink="add-product"

              [routerLinkActiveOptions]="{exact:true}">Add

              Products</a></li>

        <li class="mt-2 position-fixed bottom-0 mb-4"><a class="py-3 px-3 px-md-5 nav-link
border border-dark" (click)="logout()"><span

              class="material-symbols-outlined" style="vertical-align: bottom;">

              logout

            </span> <span class="ms-3">Log Out</span></a></li>

      </ul>
```

```html
      </nav>

    </div>

    <!-- top nav content -->

    <div class="col-12 col-md-9 col-lg-10 p-0 flex-fill overflow-scroll">

      <div class="d-flex top-nav px-3 py-3 w-100 justify-content-between">

        <div class="material-symbols-outlined cursor-pointer" (click)="toggleNav()">

          menu

        </div>

        <div>

          profile

        </div>

      </div>

      <!-- main content of route -->

      <main class="p-1 p-md-3">

        <router-outlet></router-outlet>

      </main>

    </div>

  </div>

</div>
```

ii)admin-shell.component.scss

```scss
.nav-con, .top-nav {

  background-color: var(--primary-color);

}

a {
```

```css
    text-decoration: none;

    color: black;

}

.nav-bar > ul {

    list-style: none;

    font-weight: bold;

}


.nav-bar a {

    border-radius: 10px;

}


.top-nav {

    position: fixed;

    top: 0;

}

main {

    margin-top: 60px;

}

.nav-link:hover,

.nav-link.active {

    background-color: white;

}


.logo-img {
```

```
  width: 4rem;

  height: 4rem;

}
```

iii)admin-shell.component.spec.ts

```
import { ComponentFixture, TestBed } from '@angular/core/testing';



import { AdminShellComponent } from './admin-shell.component';



describe('AdminShellComponent', () => {

 let component: AdminShellComponent;

 let fixture: ComponentFixture<AdminShellComponent>;



 beforeEach(async () => {

  await TestBed.configureTestingModule({

   declarations: [ AdminShellComponent ]

  })

  .compileComponents();



  fixture = TestBed.createComponent(AdminShellComponent);

  component = fixture.componentInstance;

  fixture.detectChanges();

 });



 it('should create', () => {

  expect(component).toBeTruthy();
```

```
  });

});
```

iv)admin-shell.component.ts

```typescript
import { Component, OnInit } from '@angular/core';

import { AuthService } from 'src/app/services/auth/auth.service';


@Component({

  selector: 'app-admin-shell',

  templateUrl: './admin-shell.component.html',

  styleUrls: ['./admin-shell.component.scss']

})
export class AdminShellComponent implements OnInit {

  showNav = true;

  constructor(

    private authServe: AuthService

  ) { }


  toggleNav(): void {

    this.showNav = !this.showNav;

  }


  logout(): void {

    this.authServe.logout();

  }
```

```
  ngOnInit(): void {

  }


}
```

v)admin-shell.module.ts

```typescript
import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { AdminShellComponent } from './admin-shell.component';

import { RouterModule } from '@angular/router';




@NgModule({

 declarations: [

 AdminShellComponent

 ],

 imports: [

  CommonModule,

  RouterModule

 ]

})

export class AdminShellModule { }
```

Footer

i)footer.component.html

```html
<footer class="p-4 shadow-lg text-center bg-white">

  © Design and developed by NEW TEAM

</footer>
```

ii)footer.component.spec.ts

```typescript
import { ComponentFixture, TestBed } from '@angular/core/testing';


import { FooterComponent } from './footer.component';


describe('FooterComponent', () => {

  let component: FooterComponent;

  let fixture: ComponentFixture<FooterComponent>;


  beforeEach(async () => {

    await TestBed.configureTestingModule({

      declarations: [ FooterComponent ]

    })

    .compileComponents();


    fixture = TestBed.createComponent(FooterComponent);
```

```
    component = fixture.componentInstance;

    fixture.detectChanges();

  });



  it('should create', () => {

    expect(component).toBeTruthy();

  });

});
```

iii) footer.component.ts

```
import { Component, OnInit } from '@angular/core';



@Component({

  selector: 'app-footer',

  templateUrl: './footer.component.html',

  styleUrls: ['./footer.component.scss']

})
export class FooterComponent implements OnInit {



  constructor() { }



  ngOnInit(): void {

  }
```

```
}
```

iv) footer.module.ts

```typescript
import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FooterComponent } from './footer.component';




@NgModule({

 declarations: [FooterComponent],

 imports: [

  CommonModule

 ],

 exports: [

  FooterComponent

 ]

})

export class FooterModule { }
```

Navbar

i) nav-bar.component.html

```html
<section class="container-fluid nav-bar">

  <div class="row">

    <!-- nav bar -->

    <nav class="col-12 p-0 p-md-3 pb-0 shadow-sm">

      <section class="row">

        <div class="col-10 col-md-3 col-lg-4 fw-bold text-dark fs-4">

          <img class="logo-img" src="/assets/logo.png" />

          <span>New Fashion</span>

        </div>

        <div class="col-2 d-md-none align-self-center">

          <span class="material-symbols-outlined text-dark fw-bold cursor-pointer"

            (click)="toggleMobileNav()">

            menu_open

          </span>

        </div>

        <div class="col-12 col-md-9 col-lg-8 d-none d-md-block">

          <div class="row">

            <div class="nav-items col-7 align-self-center">

              <ul class="d-flex">

                <li class="nav-item px-3"><a class="nav-link" routerLinkActive="active" routerLink='/'

                  [routerLinkActiveOptions]="{exact:true}">HOME</a></li>

                <li class="nav-item px-3"><a class="nav-link" routerLinkActive="active"

                  routerLink="/products">PRODUCTS</a></li>

                <li class="nav-item px-3"><a class="nav-link" routerLinkActive="active"
```

```html
                        routerLink="/cart">CART</a></li>

                </ul>

            </div>

            <!-- login btn -->

            <div class="col-5 row justify-content-end">

                <button class="col-5 primary-btn" (click)="logOut()">Log Out</button>

                <!-- <button class="col-5 secondary-btn" routerLink="/signup">Sig up</button> -->

            </div>

        </div>

    </div>

    </section>

  </nav>

</div>

<!-- mobile side navbar -->

<section class="position-fixed side-nav-wrapper" [ngClass]="isMobileNavOpen ? 'showNav' :
'hideNav'">

    <div class="p-3 w-75 h-100 side-nav text-dark">

        <div class="text-end">

            <span class="material-symbols-outlined cursor-pointer fw-bold"
(click)="toggleMobileNav()">

                cancel

            </span>

        </div>

        <div class="nav-links-container h-100 justify-content-center" (click)="toggleMobileNav()">

            <a class="text-dark nav-link" routerLinkActive="active" [routerLink]="['/']"

                [routerLinkActiveOptions]="{exact:true}">Home</a>
```

```html
        <a class="text-dark nav-link" routerLinkActive="active"
[routerLink]="['/products']">Products</a>

        <a class="text-dark nav-link" routerLinkActive="active"
[routerLink]="['/contact']">Contact</a>

        <a class="text-dark nav-link" routerLinkActive="active" [routerLink]="['/cart']">Cart</a>

        <div>

          <button class="primary-btn mt-2" (click)="logOut()">Log Out</button>

        </div>

      </div>

    </div>

  </section>

</section>
```

ii) **nav-bar.component.scss**

```scss
.nav-items a,

ul {

  color: black;

  text-decoration: none;

  list-style-type: none;

  font-weight: bold;

  margin: 0;

}

.logo-img {

  width: 4rem;

  height: 4rem;

}
```

```css
.nav-bar {

    background-color: white;

    position: fixed;

    top: 0;

    width: 100%;

    z-index: 5;

}


/* nav bar */

.nav-link {

    line-height: 3;

    padding: 0 10px;

}


.nav-link:hover,

.nav-link.active {

    background-color: var(--primary-color);

    color: white;

    padding: 0 10px;

    border-radius: 10px;

}

.nav-links-container > .nav-link:hover,

.nav-links-container > .active {

    background-color: white;
```

```css
        padding: 0 10px;

        border-radius: 10px;

        font-weight: bold;

}


.side-nav-wrapper {

        top: 0;

        left: 0;

        overflow: hidden;

        width: 0;

        z-index: 5;

        background-color: rgba(128, 128, 128, 0.386);

}

.side-nav {

        background-color: var(--primary-color);

}


.hideNav {

        width: 0;

        overflow: hidden;

        transition: all 0.1s;

}

.showNav {

        height: 100vh;

        width: 100%;
```

```
    transition: all 0.2s;

}
```

iii) **nav-bar.component.spec.ts**

```typescript
import { ComponentFixture, TestBed } from '@angular/core/testing';


import { NavBarComponent } from './nav-bar.component';


describe('NavBarComponent', () => {
  let component: NavBarComponent;

  let fixture: ComponentFixture<NavBarComponent>;


  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ NavBarComponent ]
    })
    .compileComponents();


    fixture = TestBed.createComponent(NavBarComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();
  });


  it('should create', () => {
```

```
    expect(component).toBeTruthy();

  });

});
```

iv) nav-bar.component.ts

```typescript
import { Component, OnInit } from '@angular/core';

import { AuthService } from 'src/app/services/auth/auth.service';


@Component({

  selector: 'app-nav-bar',

  templateUrl: './nav-bar.component.html',

  styleUrls: ['./nav-bar.component.scss']

})

export class NavBarComponent implements OnInit {

  isMobileNavOpen = false;

  constructor(

    private authServe: AuthService,

  ) { }

  // mobile nav

  toggleMobileNav(): void {

    this.isMobileNavOpen = !this.isMobileNavOpen;

  }


  logOut(): void {
```

```
    this.authServe.logout();

 }

 ngOnInit(): void {

 }


}
```

v) nav-bar.module.ts

```
import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { NavBarComponent } from './nav-bar.component';

import { RouterModule } from '@angular/router';




@NgModule({

 declarations: [NavBarComponent],

 imports: [

  CommonModule,

  RouterModule

 ],

 exports: [

  NavBarComponent

 ]
```

```
})

export class NavBarModule { }
```

Product Card

i) product-card.component.html

```html
<section class="row">

  <div class="col-6 col-md-4 col-lg-3 col-xl-2" *ngFor="let p of product, index as i">

    <div class="p-2 shadow pro-card" [routerLink]="['/products/',p.IMAGE]">

      <div class="text-center">

        <img src="{{BASE_URL}}{{p.IMAGE}}" alt="product img" class="product-img"
loading="lazy" decoding="async">

      </div>

      <div>

        <p class="title-text">{{p.PRODUCT_NAME}}</p>

        <h5>{{p.PRICE | currency: 'INR'}}</h5>

        <p>{{'Free Delivery'}}</p>

        <!-- review section -->

        <div class="d-flex align-items-center">

          <div class="rating-star d-flex align-items-center bg-success text-white fw-bold px-2
rounded">

            <span>

              {{'4.2'}}

            </span>

            <span class="material-symbols-outlined text-white ms-2 fs-6">

              star
```

```
        </span>

      </div>

      <div class="title-text ms-1" style="font-size: 14px;">

        {{'63'}} Reviews

      </div>

    </div>

   </div>

  </div>

 </div>

</section>
```

ii) **product-card.component.scss**

```scss
.product-img {

  width: 100%;

  height: 16rem;

}


.pro-card {

  min-height: 440px;

  max-height: 440px;

}


.title-text {

  color: #b0b0b0;

  font-weight: bold;
```

```css
}


@media only screen and (max-width: 600px) {

  .product-img {

    width: 100%;

    height: 11rem;

  }

  .pro-card {

    min-height: 370px;

    max-height: 370px;

  }

}



@media (min-width:720px) and (max-width:920px) {

  .product-img {

    width: 100%;

  }

}
```

iii) product-card.component.spec.ts

```typescript
import { ComponentFixture, TestBed } from '@angular/core/testing';
```

```typescript
import { ProductCardComponent } from './product-card.component';

describe('ProductCardComponent', () => {

  let component: ProductCardComponent;

  let fixture: ComponentFixture<ProductCardComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      declarations: [ ProductCardComponent ]

    })

    .compileComponents();

    fixture = TestBed.createComponent(ProductCardComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});
```

iv) **product-card.component.ts**

```typescript
import { Component, Input, OnInit } from '@angular/core';
```

```typescript
@Component({

  selector: 'app-product-card',

  templateUrl: './product-card.component.html',

  styleUrls: ['./product-card.component.scss']

})
export class ProductCardComponent implements OnInit {

  @Input() product: any;

  BASE_URL: any = 'http://127.0.0.1:5000/uploads/';

  constructor(

  ) {

  }




  ngOnInit(): void {

  }


}
```

v) product-card.module.ts

```
import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { ProductCardComponent } from './product-card.component';

import { RouterModule } from '@angular/router';




@NgModule({

  declarations: [ProductCardComponent],

  imports: [

    CommonModule,

    RouterModule

  ],

  exports: [

    ProductCardComponent

  ]

})
export class ProductCardModule { }
```

pages/cart:

i)cart-routing.module.ts

```typescript
import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { CartComponent } from './cart.component';


const routes: Routes = [{ path: '', component: CartComponent }];


@NgModule({

  imports: [RouterModule.forChild(routes)],

  exports: [RouterModule]

})

export class CartRoutingModule { }
```

ii)cart.component.html

```html
<section class="container mt-3 pt-3 pt-md-4">

  <div class="row">

    <div class="col-12 col-md-6">

      <div class="shadow p-2 mt-3" *ngFor="let t of [1,1,1,1,1], index as i">

        <div class="d-flex align-items-center">

          <img src="{{BASE_URL}}{{i + 1}}.jpg" alt="Product image" class="product-img" />

          <p class="fw-bold ms-2">Title of the image</p>

        </div>

        <p>Price : {{344 | currency:'INR'}}</p>

      </div>
```

```html
        </div>

    <div class="col-12 col-md-6">

        <h3>Total Cart Details</h3>

        <hr>

        <div>

            <p>Total items : {{5}}</p>

            <p *ngFor="let t of [1,1,1,1,1], index as i">Item : {{i + 1}} = {{63 | currency:'INR'}}</p>

            <p>TOTAL PRICE : {{5464 | currency:"INR"}}</p>

            <hr>

            <div class="col-12 col-md-6">

                <button class="primary-btn mt-2 w-100">PROCEDURE TO BUY</button>

            </div>

        </div>

    </div>

  </div>

</section>
```

iii)cart.component.scss

```scss
.product-img {

  width: 4rem;

  height: 4.5rem;

}
```

iv)cart.component.spec.ts

```typescript
import { ComponentFixture, TestBed } from '@angular/core/testing';
```

```typescript
import { CartComponent } from './cart.component';

describe('CartComponent', () => {

  let component: CartComponent;

  let fixture: ComponentFixture<CartComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      declarations: [ CartComponent ]

    })

    .compileComponents();

    fixture = TestBed.createComponent(CartComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});
```

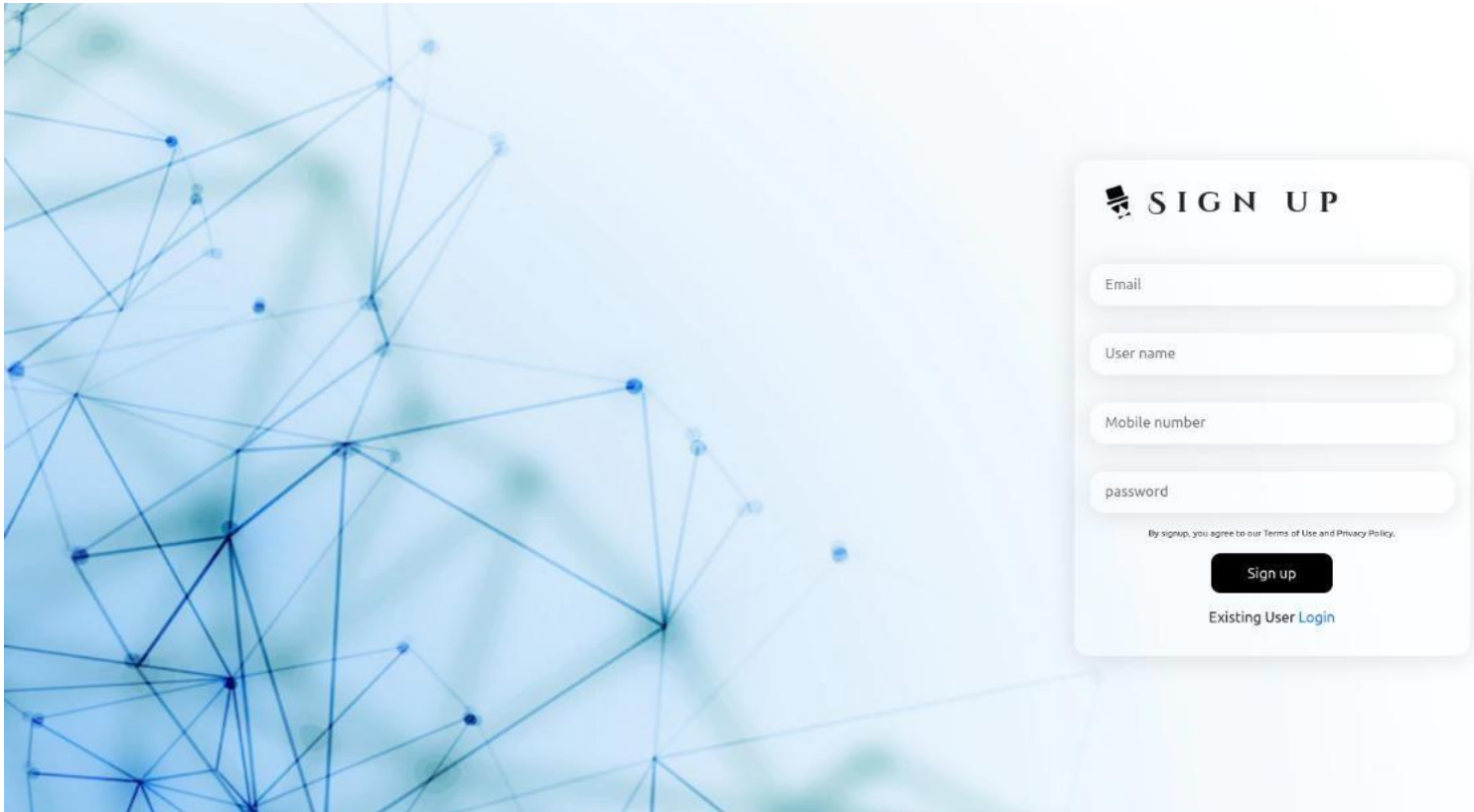**v)cart.component.ts**

```typescript
import { Component, OnInit } from '@angular/core';


@Component({

 selector: 'app-cart',

 templateUrl: './cart.component.html',

 styleUrls: ['./cart.component.scss']

})
export class CartComponent implements OnInit {

 BASE_URL: any = 'http://127.0.0.1:5000/uploads/';


 constructor() { }


 ngOnInit(): void {

 }


}
```

vi)cart.module.ts

```typescript
import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';


import { CartRoutingModule } from './cart-routing.module';

import { CartComponent } from './cart.component';
```

```
@NgModule({

 declarations: [

 CartComponent

 ],

 imports: [

   CommonModule,

   CartRoutingModule

 ]

})

export class CartModule { }
```

pages/home/home-slider

i)home-slider.component.html

```
<section class="container-fluid" *ngIf="slides.length > 0">

   <div class="row justify-content-center">

    <div class="col-12">

     <owl-carousel-o [options]="customOptions">

      <ng-container *ngFor="let slide of slides">

       <ng-template carouselSlide [id]="slide.id">

        <picture>

         <img class="img-fluid" width="1440" height="300" loading="lazy" decoding="async"
alt="banner image" [src]="slide?.src">

        </picture>

       </ng-template>
```

```html
        </ng-container>

      </owl-carousel-o>

    </div>

  </div>

</section>
```

ii) home-slider.component.scss

```scss
.img-fluid {

  height: 30rem;

}



@media only screen and (max-width: 600px) {

  .img-fluid {

    height: 10rem;

  }

}
```

iii) home-slider.component.spec.ts

```typescript
import { ComponentFixture, TestBed } from '@angular/core/testing';



import { HomeSliderComponent } from './home-slider.component';



describe('HomeSliderComponent', () => {

 let component: HomeSliderComponent;
```

```
  let fixture: ComponentFixture<HomeSliderComponent>;


  beforeEach(async () => {

    await TestBed.configureTestingModule({

      declarations: [ HomeSliderComponent ]

    })

    .compileComponents();


    fixture = TestBed.createComponent(HomeSliderComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });


  it('should create', () => {

    expect(component).toBeTruthy();

  });

});
```

iv) home-slider.component.ts

```
import { Component, OnInit } from '@angular/core';

import { OwlOptions } from 'ngx-owl-carousel-o';
```

Screenshots

🖼 **Wild spark**

Type something... ▷

Built with **IBM Watson**® ⓘ

Add new product

product name

Description

Price

Stock

Select Category

Add Product
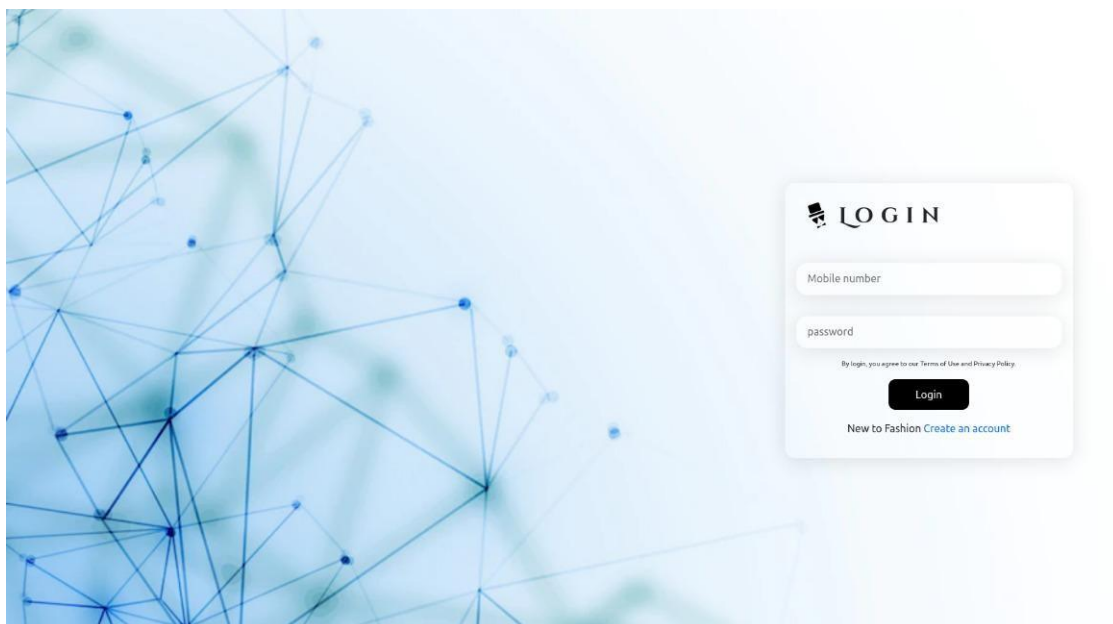
LOGIN

Mobile number

password

By login, you agree to our Terms of Use and Privacy Policy.

Login

New to Fashion Create an account

TRENDING ITEMS

See all

Stylish Elegant Men Tshirts

₹120.00

Free Delivery

4.0 ☆ 444 Reviews

MY Tshirts

₹320.00

10 day Delivery

3.2 ☆ 444 Reviews

Elegant Men Tshirts

₹820.00

Free Delivery

3.0 ☆ 4474 Reviews

Stylish Elegant Men Tshirts

₹120.00

Free Delivery

4.0 ☆ 444 Reviews

Stylish Elegant Men Tshirts

₹120.00

Free Delivery

4.0 ☆ 444 Reviews

Stylish Elegant Men Tshirts

₹120.00

Free Delivery

4.0 ☆ 444 Reviews

132

New Fashion

- **Products List**
- Orders List
- Category List
- Add Products

[→] **Log Out**

PRODUCTS LIST

Total products 343

#	Image	Product Name	Description	Price	Delete
1		Moda Rapido	Men Maroon Printed Cotton Pure Cotton T-shirt	₹319.00	🗑
2		WROGN	Men Rust Brown Slim Fit Polo Collar Cotton Pure Cotton T-shirt	₹674.00	🗑
3		HERE&NOW	Men Teal Green Typography Printed Pure Cotton T-shirt	₹443.00	🗑
4		Purple -women	Purple & Yellow Puff Sleeve Smocked A-Line Top	₹333.00	🗑
5		WEDZE By Decathlon	Women Black Solid Thermal Tops	₹431.00	🗑
6		WEDZE By Decathlon	Women Black Solid Thermal Tops	₹431.00	🗑

4.0 ☆ 333 Reviews

₹234.00
inclusive of all taxes

ADDRESS :
No:1/2 , NEW TEAM, near chennai
panimalar
poonamallee
chennai 602026

Change Address >

ADD TO CART

BUY NOW