Dr. B. B. Hegde First Grade College, Kundapura

(A Unit of Coondapura Education Society (R.))

Attendance Certificate

This is to certify that **Sumanth, Koushik** and **Shubha** of sixth semester BCA Degree has got adequate attendance in the Project work as stipulated by Mangalore University in BCA regulations.

Principal

ACKNOWLEDGMENT

ACKNOWLEDGEMENT

It gives us an immense pleasure to present the project report on **"Chic Choice".**

Our sincere thanks to **Ms. Megha** our project guide who helped us in developing this project **Chic Choice.**

We would like to express our gratitude to **Mr. Mahesh Kumar Head of the department of computer science** for his kind concern and encouragement.

We are sincerely thankful to **Prof. Kottadi Umesh Shetty**, Principal of Dr. B. B Hegde First Grade College, Kundapur and **Mr. Chetan Kumar Shetty**, Vice Principal of Dr. B. B Hegde First Grade College, Kundapura, for granting an opportunity to work on our Project.

Our Sincere thanks to all the faculty members of the Computer Science Department. We are thankful to our parents for their encouragement towards the project. Last but not the least, we whole heartedly appreciated the cooperation of our friends.

Thank you,

Project Team,

- Sumanth
- Koushik
- Shubha

TABLE OF CONTENTS

SYNOPSIS

1.1 Title of the Project

Chic Choice

1.2 Objectives of the Project

The “Chic Choice” is a web-based platform that provides a personalized shopping experience to customers by offering a wide range of trendy and high-quality clothing, footwear, and accessories. To sell clothing and accessories to customers online, with the ultimate goal of increasing sales revenue and profitability. Providing excellent customer service and providing business deals to the fashion designers. The website can build long-term relationships with customers and increase the likelihood of repeat purchases.

1.3 Project category

Web Application

1.4 Languages to be used

Frontend: HTML, CSS, Java script, Bootstrap

Backend: Python

Database: sqlite3

Framework: Django

1.5 Structure of the Project

1.5.1 Analysis

The main object of the project Shopping website system is to sell the fashion related products in the online. It’s an E-Commerce web application that covers all the common functionality of a shopping web application. It manages all the information about products and user can order the product they wish. The project is totally built at administrative end and thus only administrator is guaranteed the access. The purpose of the project is to build an application program to reduce the manual work.

1.5.2 Module description

- ❖ **Register Module:**

  This module allows user to register themselves. Once the user get registered he/she can login to the system with his username and password.

- ❖ **Login Module:**

  This module allows registered users and login themselves.

- **Forgot Password**
    - Email
        - New password
        - Confirm password

❖ Contact Us:

This module contains the contact information of the admin.

❖ Registered User:

After successful login to the system, the user can do the following things-

- **Login:** In login page, the user can login to the system with his/her username and password.
- **Search Item:** The users can search for product according to their brand name/product name.
- **Cart:** Here, the users can view the details of the specified product.
    - ✓ **Add to Cart**
    - ✓ **Increase Quantity**
    - ✓ **Remove from the Cart**
- **Profile:** The user can view their order history and status of the product.
    - ✓ **Add/Delete profile:** The user can view and delete the profile settings.
    - ✓ **My Orders:** User can view their order.
        - **Tracking**
        - **Cancel Order**
    - ✓ **Change password:** User can change their password through this option
    - ✓ **Logout:** The user can logout from the website using logout option.

❖ Admin Module:

This module performs function such as managing products, user details and many other. The administrator has the full authority to make modification.

- **Login:** In login page, the admin can login to the system with his/her username and password.
- **All Products:** Here admin can view all available products.

        - ✓ **Add Products:** Here admin can add the products.
        - ✓ **Edit:** Here admin can modify the already added products.
        - ✓ **Delete:** Here admin can delete the already added products.
    - ○ **Add Designer:** In this page, admin can add the designer.
        - ✓ **Edit:** Here admin can modify the already added designer.
        - ✓ **Delete:** Here admin can delete the already added designer.
    - ○ **View Orders:** This part of the module allows the admin to view customer products ordering details.
        - ✓ **Change Status:** Here admin can change the order status.
    - ○ **Logout:** The admin can logout from the website using logout option.

❖ Cart Module:

This module allows user to store the product which they want to buy.

- ○ **Remove:** Here user can remove the products added to the cart.
- ○ **Order:** By the giving the details of user they can place the order for the products in the cart.

❖ Designer Module:

This module allows admins to add fashion designers details. So the user can contact the fashion designers through admin.

- ○ **Login:** In login page, the designer can login to the system with his/her username and password
    - ✓ Username
    - ✓ Password
- ○ **Logout:** The designer can logout from the website using logout option.
- ○ **Order Details:** The designer can view the orders.
    - ✓ **Accept**
    - ✓ **Reject**

1.6 Data Structure:

1.6.1 Admin:

- Username
- Password

1.6.1.1 Add/Update/Delete Products:

- Product name
- Selling price
- Discounted Price
- Description
- Brand
- Product category
- Photo

1.6.1.2 Add/Update/Delete Designers

- User Name
- Designer Code
- Full name
- Email
- Phone
- Work Experience
- Address
- Zip code
- State
- Designer Image

1.6.1.3 View Orders:

- Order Id
- User Name
- Customer Info
- Product Info
- Quantity
- Order Date
- Status

1.6.2 Users:

- Username
- Email Id
- Password
- Confirm Password

1.6.2.1 Profile:

1.6.2.2.1 Add/Delete Profile

- Full name
- Phone number
- Locality

- City
- State
- Zip code

1.6.2.2.2 My Orders:

- Product Name
- Quantity
- Price
- Total Price
- Order Status
- Cancel Order

1.6.2.2.3 Change password

- Email
  - Old password
  - New password

1.6.3 Cart:

- **Add**
  - Product name
  - Description
  - Quantity
  - Price
  - Place Order
  - Remove Item
- **Remove**
  - Remove Item

1.6.4 Checkout Order:

- Address
- Total Cost
- Payment mode
  - Cash on Delivery
  - Paypal

1.6.5 Contact Us:

View

1.6.6 Designer:

- User Name
- Password

- Designer code

1.6.6.1 Change password:

  - Old password
  - New password
  - Confirm password

1.6.6.2 Order Details

  - Accept
  - Reject

1.7 Existing System:

In the existing system of the fashion shopping website, customers primarily have access to browse and purchase fashion products online. They can explore various categories, view product details, add items to their cart, and proceed to checkout for payment and delivery. However, there is no provision for directly hiring or connecting with fashion designers through the website. Customers seeking personalized fashion design services would typically need to engage in offline communication or find designers through other platforms

1.8 Proposed System:

The proposed system aims to enhance the fashion shopping website by introducing a designer module, which allows customers to directly hire and connect with fashion designers through the platform. Designers can create profiles on the website, customers will have the ability to search for designers based on various criteria such as style, experience, location. By introducing the designer module into the fashion shopping website, customers will have a comprehensive platform that combines the convenience of online shopping with the opportunity to access personalized fashion design services. It will enhance customer satisfaction, provide more opportunities for designers to showcase their talent, and create a unique selling point for the website

1.9 Future Scope:

- We can create android application for this project.

- Augmented reality: With the help of augmented reality, customers can try on clothes virtually and get a better sense of how they will look before making a purchase. This technology could also be used to show customers how clothes will fit on different body types.

- Social media integration: Social media has become an important part of the fashion industry, and e-commerce websites can take advantage of this by integrating social media platforms into their websites. This could include allowing customers to share their purchases on social media, or using social media influencers to promote products.

- Drone Delivery**:** Another popular online shopping trends 2023 is the use of drones for delivery. This is a popular trend that can be popular while businesses try to make contactless deliveries.

SOFTWARE REQUIREMENT SPECIFICATION

2.1. Introduction:

"CHIC CHOICE" is a web application that allow the users to fulfill all their fashion needs. Our application has best variety of collection to attract the users. It also has designers club where users can get their needs of their fashion fulfilled.

2.1.1. Purpose:

The purpose of our application Chic Choice is to enable users to have one stop solution for their shopping problems, where they can purchase clothes as well as get their dress designed from the best of the designers.

2.1.2. Scope:

SRS provides a reference for validation of the final product. A high-quality SRS lead to the high-quality software at low cost with the small cycle time.

- Multiple users can access this system in networking environment.
- This software reduces the manual effort.
- Easy to manage the details of registered users by the admin.
- Web-platform means that the system will be available for access 24/7 except when there is a temporary server issue which is expected to be minimal.

2.1.3. Definition, Acronyms, Abbreviations:

This subsection provides the definition of all the terms, acronyms and abbreviation used in this document to understand the SRS properly.

- CFD – Context Flow Diagram
- DFD – Data Flow Diagram
- ERD – Entity Relationship Diagram
- SRS – Software Requirement Specification
- HTML-Hyper Text Mark-up Language
- CSS-Cascading Style Sheet
- GUI- Graphical User Interface

2.1.4 Overview:

Mainly software requirement system gives overview of the functionality of the software. Software requirement specification establishes the basis for agreement between the user and the developer on what software product will do.

Software requirement specification helps the developer to understand their needs. This project is related to trending fashion accessories. The customer can view the Products and order the facilities without any intermediary service over the internet.

2.2 Overall Description:

2.2.1 Product Perspective:

Product perspective is essentially the relationship of the product to other product defining if the product independent or is a part of longer product function. This project is a standalone product where in user interface is designed by organising all the function to make them understandable to any user of the software.

Product of this project is a software that reduces the manual effort for ordering clothes, shoes, specs, hats, watch, etc. Admin has far more features than the user. This software product provides an easy way of maintaining details of users by admin.

2.2.2.1 Login / Register

This module allows system users to register themselves. Once the user gets registered, he can login to the system with his Username and password.

2.2.2.2 Contact Us

This module contains the contact information of the admin. The module also allows users to send queries regarding the website or booking details to the admin.

2.2.2.3 Registered User module

After successful login to the system, the user can do the following things –

- **Login:** In Login page, the user can login to the system with his Username and password.
- **Forgot password:** Here, the user can update his password by mentioning the new password.
- **Search Item:** The users can search for Products according to the product name, category, status.
- **Cart:** Here, the users can view the details of the specified product.
  - **Add to Cart**

        - **Increase Quantity**
        - **Remove from the Cart**
    - **Profile:** The user can view their order history and status of the product.

        - ✓ **Add/Delete profile:** The user can view and delete the profile settings.
        - ✓ **Orders:** User can view their order.
            - **Tracking**
            - **Cancel Order**
        - ✓ **Change password:** User can change their password through this option
        - ✓ **Logout:** The user can logout from the website using logout option.

2.2.2.4 Admin Module:

This module performs function such as managing products, user details and many other. The administrator has the full authority to make modification.

- **Login:** In login page, the admin can login to the system with his/her username and password.
- **All Products:** Here admin can view all available products.
    - ✓ **Add Products:** Here admin can add the products.
    - ✓ **Edit:** Here admin can modify the already added products.
    - ✓ **Delete:** Here admin can delete the already added products.
- **Add Designer:** In this page, admin can add the designer.
    - ✓ **Edit:** Here admin can modify the already added designer.
    - ✓ **Delete:** Here admin can delete the already added designer.
- **View Orders:** This part of the module allows the admin to view customer products ordering details.
    - ✓ **Change Status:** Here admin can change the order status.
- **Logout:** The admin can logout from the website using logout option.

2.2.2.5 Cart module:

This module allows the user to store the product which they want to buy.

- **Remove:** Here user can remove the products added to the cart.

- **Order:** By giving the details of user they can place the order for the products in the cart.

2.2.2 Product function:

The product function is a general abstract description of functions to be performed by a product.

- Provides user friendly GUI by means of HTML and JavaScript.
- More efficient and fast access of web page.
- All manually handling procedure for maintaining records is replaced by the software.
- It is easily operable.

2.2.3 User characteristics:

The user of the software need not be a computer proficient. The GUI of the software makes it user friendly. User has to have minimum knowledge of windows such as keyboard typing and mouse clicking.

2.2.4 General constraints:

The developed system should run on any of the operating systems that supports MySQL, Apache server and Apache maven.

2.2.5 Assumption and dependencies:

- The admin has high value of authority compared to the customer. He has an authority of creating his own password, adding and viewing products.
- Here the user has less authority compared to the administrator. The user has the authority of creating his own password.
- The user of the system is assumed to have only basic computer knowledge and can use GUI that includes menus, menu options, buttons etc.

2.3 Specific requirement

2.3.1 External interface requirements:

2.3.1.1 User interface:

User interface is the front end where user can interact with the system. The user can perform tasks by selecting any one of the menus depending on privilege given to them. The software provides GUI, there are many controls like textboxes and command buttons etc. user input will be via keyboard and mouse point click.

2.3.1.2 Hardware interfaces:

- Processor: Intel Core i3 / AMD RADEON R2 or above
- RAM: 4GB or above
- Hard Disk: 250GB

2.3.1.3 Software interface:

- Language: Python
- Frame work: Django
- Database: sqlite3
- User Interface: HTML, JS, CSS, Bootstrap.
- **Web Browser:** Google Chrome, Mozilla, OPERA
- Operating system: windows 7 or above

2.3.1.4 Communication Interface:

This proposed system shall use the HTTP protocol for communication over the internet and for the internet communication.

2.3.2 Functional requirements

This section gives the functional capabilities of the system that requires specifying the input, desired outputs and processing requirement. Functional requirements specify which outputs should be produced from the given inputs. They describe the relationship between the input and output of the system.

2.3.2.1 Register module

- **Input:**
  - Full Name
  - Email Address
  - Password
  - Confirm password
- **Process Definition:**

  All the information provided by the user will be stored in the database for logging in to the system.
- **Output:**

  User details will be stored in user table.

2.3.2.2 Login module:

This module allows the users to login to the system by entering the Username and password specified during registration.

- **Input:**
    - User Name
    - Password

2.3.2.3 Registered User module:

2.3.2.3.1 Login:

- **Input:**
    - Username
    - Password
- **Process Definition:** Login section allows user to login to the system using Email Id and password.
- **Output:**
    - **Login Successful:** If User is authenticated successfully, the user dashboard will be displayed.
    - **Invalid User:** If the user authentication fails, then the alert message will be displayed.

2.3.2.3.2 Forgot password:

- **Input:**
    - Email Id
        - New Password
        - Confirm password
- **Process Definition:** In this section user can update his password
- **Output:**
    - **Password updated:** If User already exists and new password and confirm password is same then password is updated
    - **Password doesn't match:** If user does not exists and new password and confirm password is not same then alert message will be displayed.

2.3.2.3.3 Profile

- **Input:**
  - Full name
  - Phone number
  - Locality
  - City
  - State
  - Zip code
- **Process Definition:**

  All the information provided by the user will be updated in the database.
- **Output:**
  - **Profile Update Successful:** If User password is correct then only profile updated

2.3.2.3.4 Search Item:

The users can search for Products according to the product name, category, status.

2.3.2.3.5 Cart:

Here, the users can view the details of the specified product. He can do below listed operation.

- Increase Quantity
- Remove from the Cart

2.3.2.3.6 View Details:

Here, the users can view the details of the specified product.

- Product name
- Description
- Price
- Add to cart
- Buy now

2.3.2.3.7 Order:

User can view their order.

- Tracking
- Cancel order

2.3.2.3.8 Logout

The User can logout from the website using logout option.

2.3.2.4 Admin module

Admin module is the main module which has entire control over the project. The authenticity of the administrator is checked before entering into the system. Admin module performs the following functions –

2.3.2.4.1 Login:

- ➢ **Input:**
  - Username
  - Password
- ➢ **Process Definition:** Login section allows user to login to the system using user name and password.
- ➢ **Output:**
  - ✚ **Login Successful:** If admin is authenticated successfully, the admin dashboard will be displayed.
  - ✚ **Invalid User:** If the user authentication fails, then the alert message will be displayed.

2.3.2.4.2 Add products:

- ➢ **Input:**
  - Product name
  - Selling price
  - Discounted Price
  - Description
  - Brand
  - Product category
  - Photo
- ➢ **Process Definition:**

  All the information provided by the admin will be added to the database.
- ➢ **Output:**
  - ✚ **Product added successfully:**

    Product added to the database.

2.3.2.4.3.1.1 Add designer:

- **Input:**
  - User Name
  - Designer Code
  - Full name
  - Email
  - Phone
  - Work Experience
  - Address
  - Zip code
  - State
  - Designer Image
- **Process Definition:** All the information provided by the admin will be added to the database.
- **Output:**
  - **Designer added successfully:**
    Designer added to the database

2.3.2.4.4 View orders:

In this section, admin can view the order details of all the users.

- Order Id
- User Name
- Customer Info
- Product Info
- Quantity
- Order Date
- Status

2.3.2.5 Cart module:

This module allows the user to store the product which they want to buy.

- **Remove:** Here user can remove the products added to the cart.
- **Order:** By giving the details of user they can place the order for the products in the cart.
  - **Input:**
    - **Address**
    - **Payment mode**

    - **Cash on delivery**
    - **PayPal**
- **Process Definition:**

  The user will first need to login to the system using the Email Id and password. Then order a product by entering certain details.
- **Output:**

  Product will be ordered.

2.3.2.6 Designer Module

Designer module is the unique module. Designer module performs the following functions –

2.3.2.4.1 Login:

- **Input:**
    - Username
    - Password
    - Designer code
- **Process Definition:** Login section allows user to login to the system using user name and password.
- **Output:**
    - **Login Successful:** If admin is authenticated successfully, the admin dashboard will be displayed.
    - **Invalid User:** If the user authentication fails, then the alert message will be displayed.

2.3.2.5.2 View orders:

In this section, Designer can view the order details of all the users.

- Accept
- Reject

2.3.2.7Contact us module

The system user can send queries regarding the order details or any other reviews to the admin.

2.3.3 Performance requirement

2.3.3.1 Static requirement:

Static requirements are those that do not impose constraints on the execution characteristics of the system. These include requirements like number of terminals supported, number of simultaneous users supported etc.

2.3.3.2 Dynamic requirements:

Dynamic requirements specify constraints on the execution behaviour of the system. These typically include response time and throughput constraints on the system.

2.3.4 Design constraints:

Design constraints specify all the constraints imposed on design. These constraints are typically imposed by the customer, by the development organization, or by external regulations. The constraints may be imposed on the hardware, software, data, operational procedures, interfaces, or any other part of the system.

2.3.4.1 Security:

This application will allow only valid users to access the system. User can login to the system with his user id and password. This software provides a higher level security for data. Admin can view all the records of the user.

2.3.4.2 Hardware constraints:

Hardware constraints specify the hardware which is required for designing the software.

- Processor: Pentium IV dual core and above
- Hard disk: standard 500 GB
- Primary memory: standard 2GB RAM

2.3.4.3 Software constraints:

Software constrains specify the software required

- Windows 10 and above
- Python
- Sqlite3

2.3.4.4 Fault tolerance:

This software is also fault tolerant. Each data input is validated using validation. If inputted data has any mistakes in its validation, then appropriate error message will be displayed.

2.3.5 System attribute:

System attribute specifies over all attributes that the system should have:

- Reliability: The product is reliable without any misinterpretation.
- Scalability: It is a standalone system. It is a client-server model and multiple user can use this system.
- Portability: User can use this application in any operating system that supports sqlite3
- Maintainability: The product is flexible for further modification in future.

2.3.6 Other requirements:

Not applicable

SYSTEM DESIGN

3.1 Introduction:

System design is the process of defining the architecture, components, modules, interface and data for a system to satisfy specified requirements. The purpose of this phase is planning the solution of the problem specified by the requirement documents. This phase is the first step in moving from problem domain to solution domain. Design document includes system specification as well as design specific task, test specification and actual program.

System design some time called top level design. Aims to identify the module that should be in the system and specification of the module and how they interact with each other to produce the desired result.

3.2 Applicable Documents:

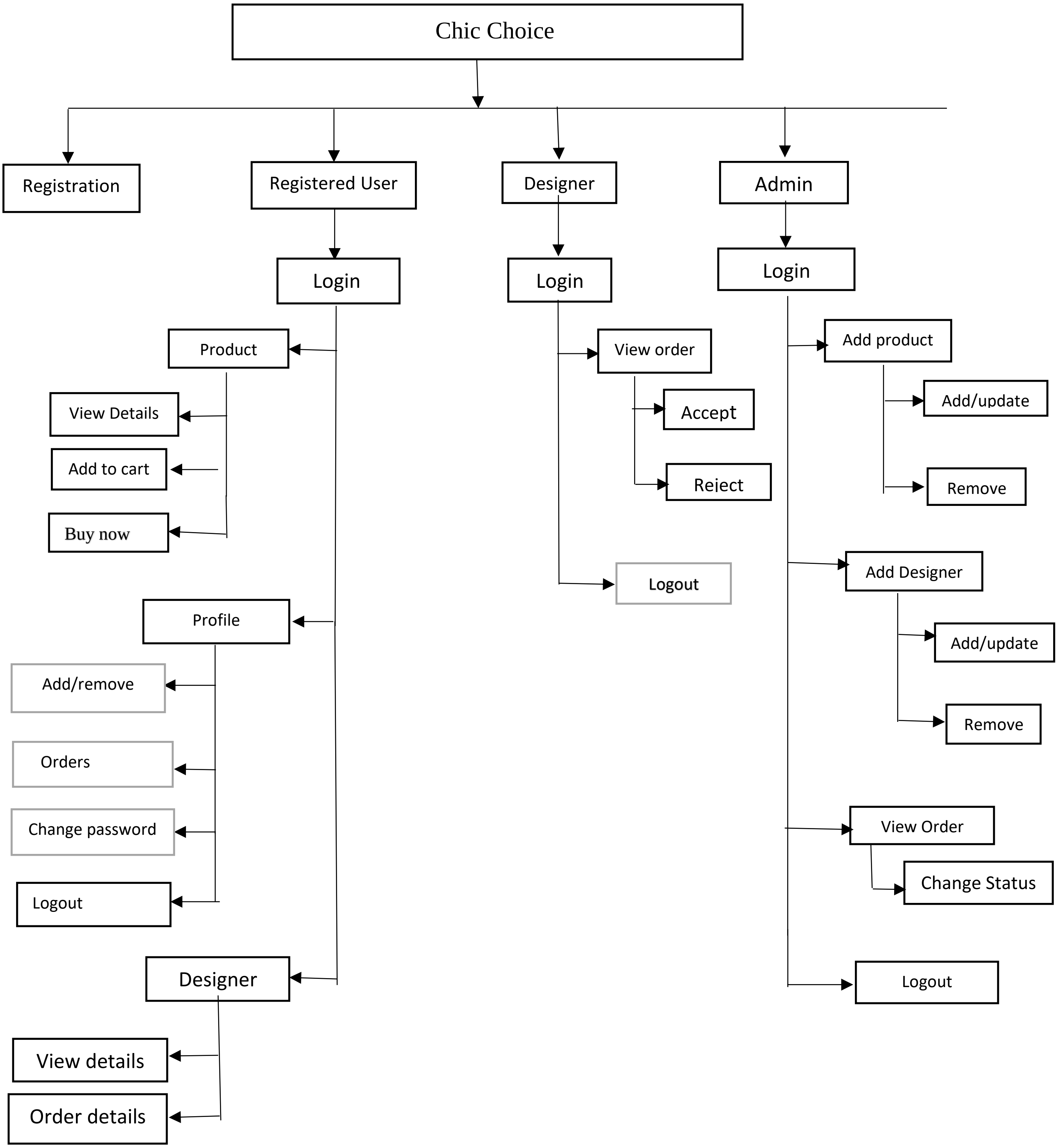
- Software requirement specification document
- Synopsis

3.3 Functional decomposition:

As name indicate here the functions are arranged in the hierarchical manner by using chart. Functional decomposition is the process of identifying the overall functions or all of the necessary sub-tasks to complete the project.

The functional component of the software package is:

- Login
- Registration
- Registered User
- Admin
- Cart
- Designer
- Search
- Contact us


Chic Choice
Registration
Registered User
Designer
Admin
Login
Login
Login
Product
View Details
Add to cart
Buy now
Profile
Add/remove
Orders
Change password
Logout
Designer
View details
Order details
View order
Accept
Reject
Logout
Add product
Add/update
Remove
Add Designer
Add/update
Remove
View Order
Change Status
Logout

3.4 Functional component of the project:

3.4.1 Login module:

This module allows the user to login to the system with username and Password.

3.4.2 Registration module:

This module is used to register new user.

3.4.3 Registered user module:

3.4.3.1 Login

3.4.3.2 Search Items

3.4.3.3 Cart

- Add to cart
- Increase Quantity
- Remove from the cart

3.4.3.4 Profile

- Add/Delete profile
- Order
    - Tracking
    - Cancel order
- Change password
- Logout

3.4.4 Admin module

3.4.4.1 Login

3.4.4.2 All products

- Add products
- Edit
- Delete

3.4.4.3 Add designer

- Edit
- Delete

3.4.4.4 View Orders

- Change Status

3.4.4.5 Logout

3.4.5 Search module:

This module will be helps to search a product.

3.4.6 Designer module:

- Login
- View order details
    - Accept
    - Reject
- Logout

3.4.7 Contact Us module:

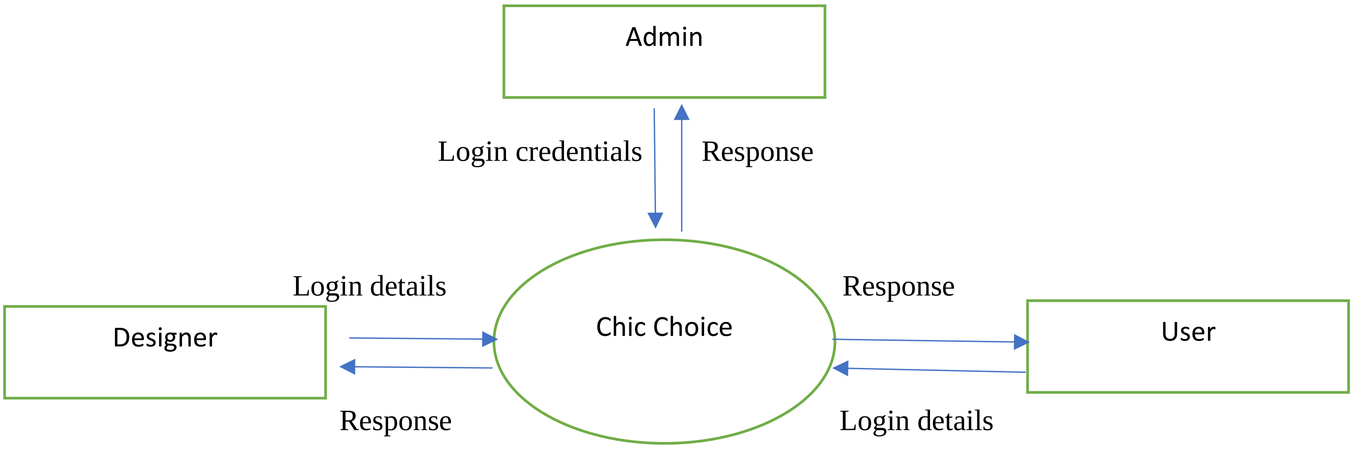
This module contains the contact information of the admin.

3.5 Description of the program:

3.5.1 CFD (Context Flow Diagram):

Context Flow Diagrams show input and output of the system. It shows all the external entities that interact with the system and how the data flows between these external entities and the system.

The Context Flow Diagram for Agriculture Management System is shown in the below figure. The input of this system



3.5.2 DFD (Data flow diagram):

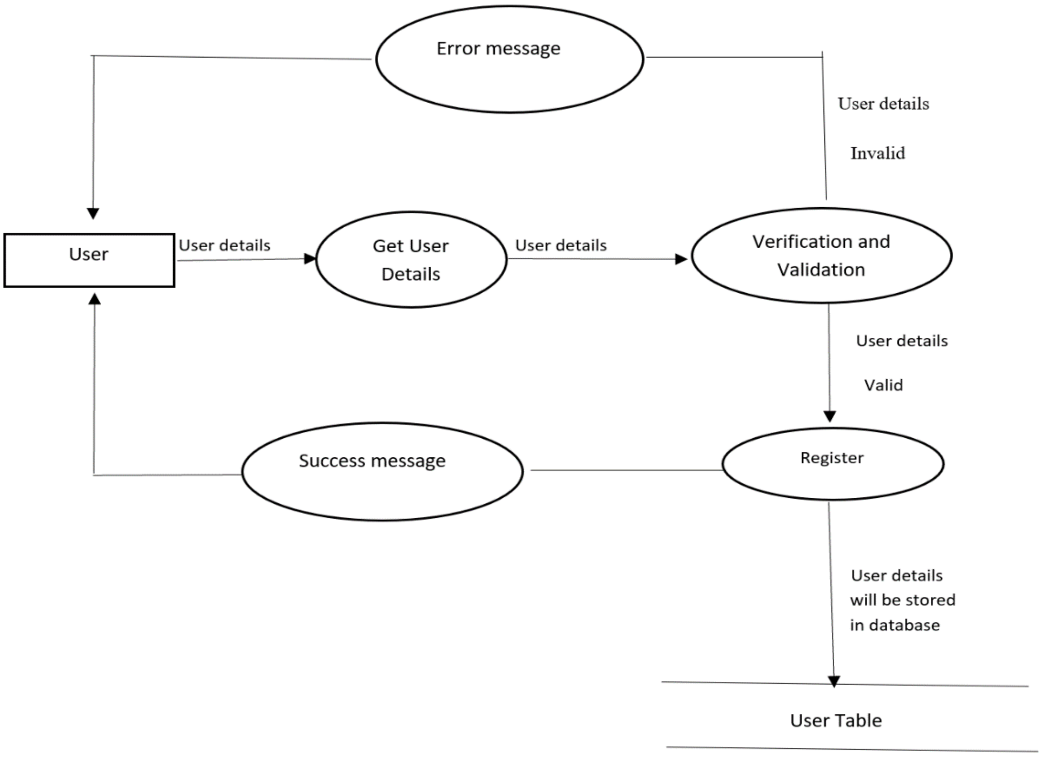
A data flow diagram shows the flow of the data through the system. It views the system as a function that transform the input to desire output. The data flow diagram aims to capture information that take place with system to the input. So, that eventually the output data is proceeded.

Symbols used in DFD:

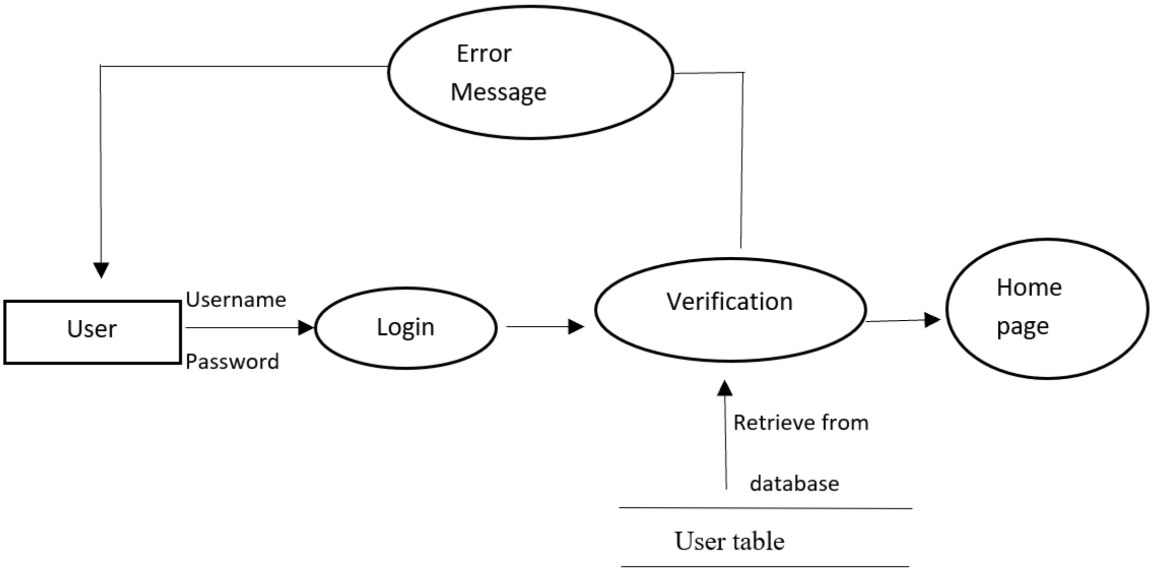
NOTATION	NAME	DESCRIPTION
⬭	Process	It performs the transformation of data from one start to another
▭	Source or Sink	It represents the external entity that may be either source or sink.
⟶	Data Flow	It represents the flow of data from source to destination.
═	Data store	Store to database

3.5.2.1 DFD for User:

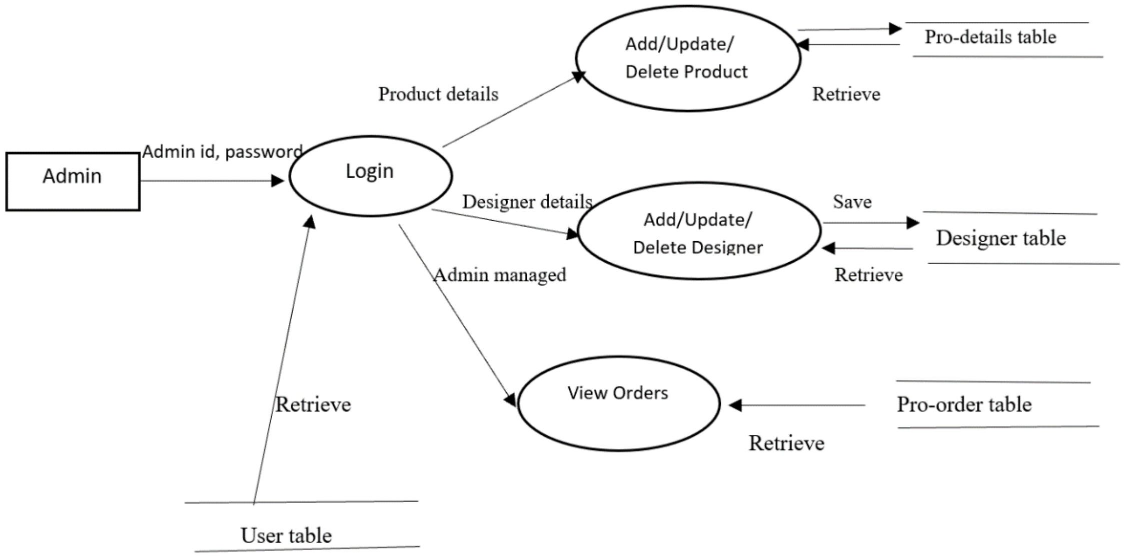
3.5.2.1 DFD for user Registration:



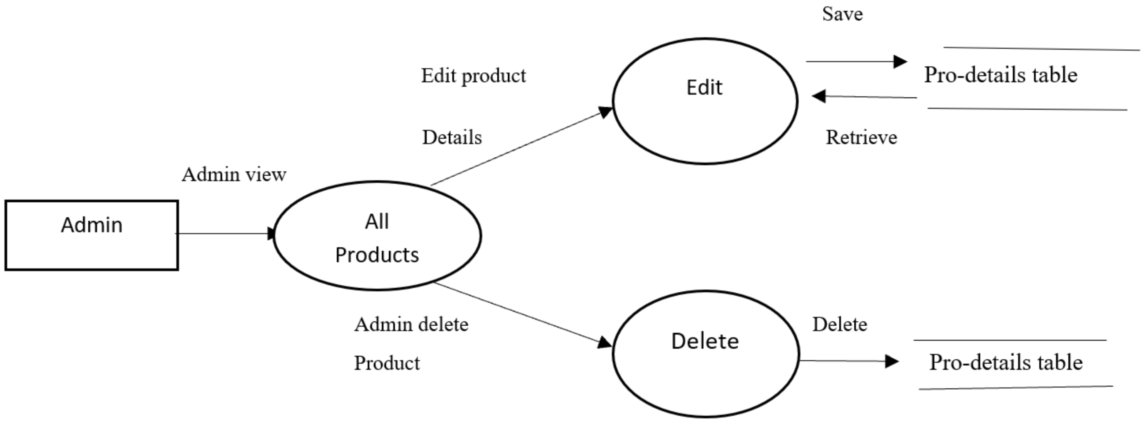
3.5.2.2 DFD for Login module



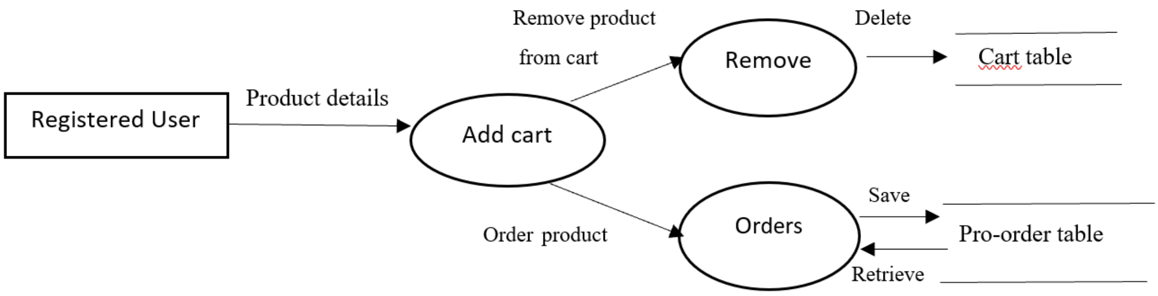
3.5.2.2 DFD for Admin:



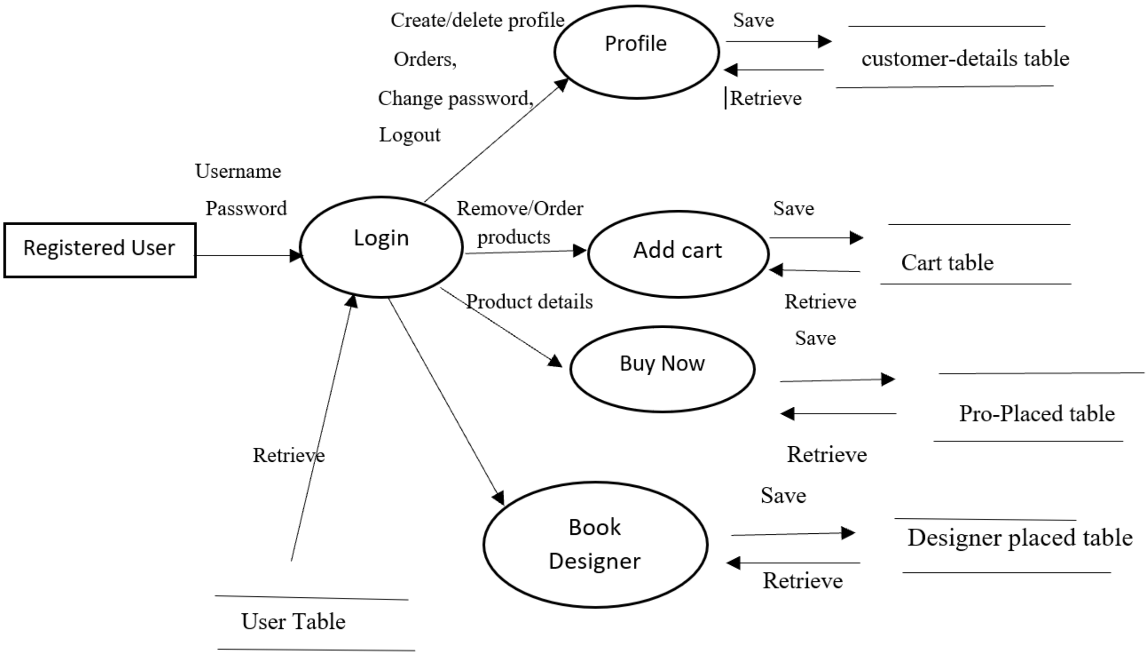
3.5.2.3 DFD for Products:



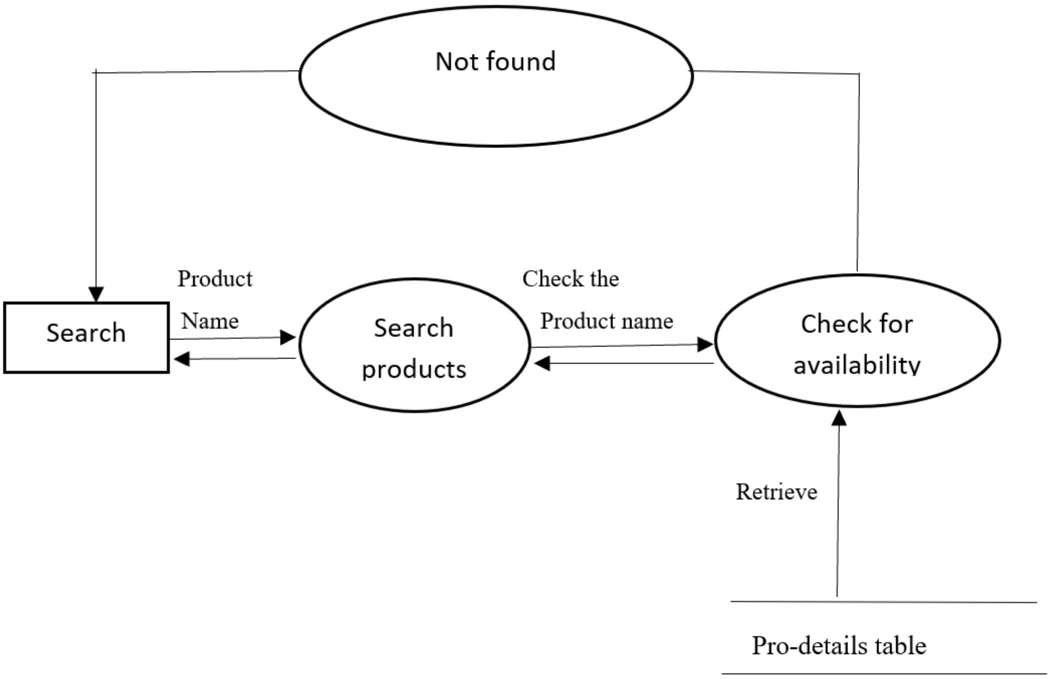
3.5.2.4 DFD for Cart Module:



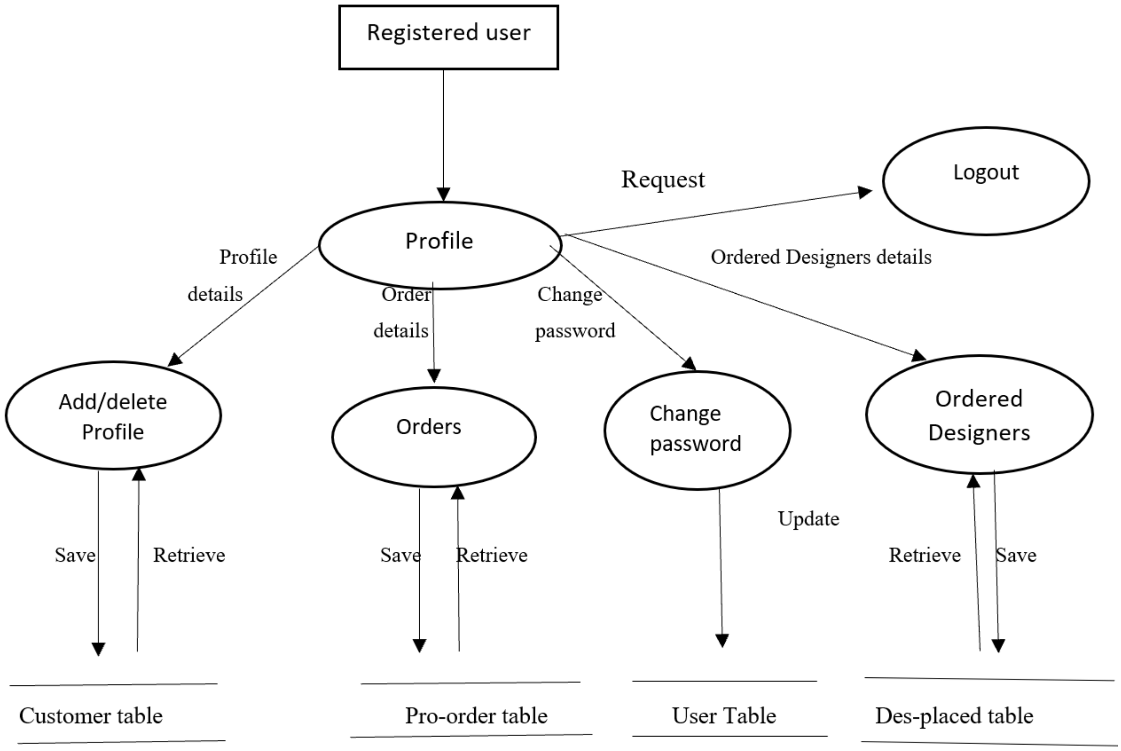
3.5.2.5 DFD for Registered user:



3.5.2.6 DFD for Search module:



3.5.2.7 DFD for Profile:



3.6 Description of components:

3.6.1 Registration

- **Input:**
  - Username
  - Email address
  - Password
  - Confirm password
- **Process Definition:**

  All the information provided by the user will be stored in the database for logging in to the system.
- **Output:**

  New system user will get his username and password for login purpose.
- **Interfaces with other functional components:**

  Login
- **Resource allocation:**

  User table.
- **User Interface:**

  User interface contains text boxes, buttons, checkbox.

3.6.2 Login:

- **Input:**
  - Username
  - Password
- **Process Definition:**

  Login module allows user to login the system using user name and password.
- **Output:**

  If login is success displays related page, otherwise displays an error message.
- **Interfaces with other functional components:**

  Login
- **Resource allocation:**

  User table.
- **User Interface:**

  User interface contains text boxes and buttons.

3.6.2.1 Add to cart:

3.6.2.1.1 Remove:

- **Input:**

  Click on Remove button
- **Process Definition:**

  Product will be Remove from cart.
- **Output:**

  Products is removed
- **Interfaces with other functional components:**

  Remove products
- **User Interface:**

  User interface contains buttons

3.6.2.1.2 Order

- **Input:**
  - Full name
  - Phone number
  - Locality
  - City
  - State
  - Zip code
- **Process Definition:**

  Product is to be ordered
- **Output:**

  Show message order successfully
- **Interfaces with other functional components:**

  Order products
- **Resource allocation:**

  Order table
- **User Interface:**

  User interface contains text boxes and buttons, dropdown list

3.6.2.2 View Details

- **Input:**

  Click on Details button

- **Process Definition:**

  The user can view details of the products.

- **Output:**

  Show all product details of user already purchased

- **Interfaces with other functional components:**

  View details of products

- **Resource allocation:**

  Pro-order table

- **User Interface:**

  User interface contains buttons

3.6.2.3 Profile:

3.6.2.3.1 Add/delete profile :

- **Input:**

  - Full name
  - Phone number
  - Locality
  - City
  - State
  - Zip code

- **Process Definition:**

  User name and password is valid user can edit his profile

- **Output:**

  Profile is added

- **Interfaces with other functional components:**

  Update user details

- **Resource allocation:**

  User table

- **User Interface:**

  User interface contains textboxes and buttons.

3.6.2.3.2 My orders

- **Input:**

  Click on My orders

- **Process Definition:**

  we can view the My orders

- **Output:**

  Show all product details of user already purchased

- **Interfaces with other functional components:**

  View order details

- **Resource allocation:**

  Pro-orders table

- **User Interface:**

  User interface contains buttons.

3.6.2.4 Forgot password

- **Input:**
  - Email address
    - New password
    - Conform password
- **Process Definition:**

  The user can create new password.

- **Output:**

  password will be updated

- **Interfaces with other functional components:**

  Login

- **Resource allocation:**

  User table

- **User Interface:**

  User interface contains text boxes and buttons

3.6.3 Admin

3.6.3.1 Login:

- **Input:**
  - User Name
  - Password
- **Process Definition:**

  Login section allows user to login to the system using user name and password.
- **Output:**
  - **Login Successful:** If admin is authenticated successfully, the respective page will be displayed.
  - **Invalid User:** If the user authentication fails, then the alert message will be displayed.
- **Interfaces with other functional components:**

  Login
- **Resource allocation:**

  Admin table.
- **User Interface:**

  User interface contains text boxes and buttons.

3.6.3.2 Add Products:

- **Input:**
  - Product name
  - Selling price
  - Discounted Price
  - Description
  - Brand
  - Product category
  - Photo
- **Process Definition:**

  Admin can Add new products
- **Output:**

  Updated products details will be stored in the database.
- **Interfaces with other functional components:**

View Products details

- **Resource allocation:**

  Pro-details table.

- **User Interface:**

  User interface contains text boxes and buttons

3.6.3.3 Add designer:

3.6.3.3.1 Add/Update/delete designer:

- **Input:**
  - User Name
  - Designer Code
  - Full name
  - Email
  - Phone
  - Work Experience
  - Address
  - Zip code
  - State
  - Designer Image
- **Process Definition:**

  Admin can edit products details.

- **Output:**

  Changes made will be stored in the database.

- **Interfaces with other functional components:**

  View updated product details

- **Resource allocation:**

  Pro-details table

- **User Interface:**

  User interface contains text boxes, buttons input type=file.

3.6.3.3.3 View Orders:

- **Input:**

  Click view Order button

- **Process Definition:**

  Admin can view the user’s order.

- **Output:**

Display user Order

- **Interfaces with other functional components:**

  View the product order

- **Resource allocation:**

  Order table

- **User Interface:**

  User interface contains labels buttons.

3.6.4 Contact Us module

- **Input:**

  Click on contact us

- **Process Definition:**

  User can view the details of the admin

- **Output:**

  Admin contact details will be show

- **Interfaces with other functional components:**

  Independent module

- **Resource allocation:**

  Admin table

- **User Interface:**

  User interface contains buttons

DATABASE DESIGN

4.1 Introduction:

Database is a collection of related data. Relational database stores data in a table or relations. The data stored in a relation are arranged in records. Each record consists of set of attributes or fields which can be referred to as characteristics of relation.

This document describes the tables that are used to design software, its attributes, data types, constraints and relationship among these tables. Database is a collection of data in tables or relational database stores data in tables or relations. The data stored in a relation are arranged in tables or records. Each record consists of set of attributes or fields that can be referred to as characteristics of records.

4.2 Description of Table and fields:

User table:

SL. No	Field Name	Data Type	Size	Constraints	Description
1.	id	Integer Field	10	Not NULL, PRIMARY KEY	User Id
2.	username	Char Field	50	Not NULL	User Name
3.	email address	Email Field	50	Not NULL	User Email
4.	first name	Char Field	50	Not NULL	User First name
5.	last name	Char Field	50	Not NULL	User Last name
6.	Password	Char Field	50	Not NULL	User Password

Customer table:

SL. No	Field Name	Data Type	Size	Constraints	Description
1	id	Integer Field	10	PRIMARY KEY	Customer Id
2	user	User Instance	10	FOREIGN KEY	User Foreign Key
3	Full name	Char Field	200	Not NULL	Customer Full name

4	phone	Integer Field	20	Not NULL	Customer phone number
5	locality	Char Field	200	Not NULL	Customer locality
6	city	Char Field	50	Not NULL	Customer city
7	zip code	Integer Field	10	Not NULL	Customer zip code
8	state	Char Field	50	Not NULL	Customer state

Products table:

SL. NO	Field Name	Data Type	Size	Constraints	Description
1.	id	Integer Field	4	PRIMARY KEY	Product Id
2.	title	Char Field	100	Not NULL	Product name
3.	selling price	Float Field	10	Not NULL	Product selling price
4.	discounted price	Float Field	10	Not NULL	Product discounted price
5.	description	Text Field	100	Not NULL	Product description
6.	brand	Char Field	50	Not NULL	Product brand
7.	category	Char Field	50	Not NULL	Product category
8.	product Image	Image Field		Not NULL	Product image

Order Placed Table:

SL. No	**Field Name**	**Data Type**	**Size**	**Constraints**	**Description**
1	id	Integer Field	4	PRIMARY KEY	Product Id
2.	user	User Instance	10	FOREIGN KEY	User Foreign Key
3.	customer	Customer Instance	10	FOREIGN KEY	Customer Foreign Key
4.	customer Info	Char Field	45	Not NULL	Customer Information
5.	product Info	Char Field	45	Not NULL	Product Information
6.	product	Product Instance	10	FOREIGN KEY	Product Foreign Key
7.	quantity	Integer Field	4	Not NULL	Product Quantity
8.	ordered date	Datetime Field	45	Not NULL	Product ordered date
9.	status	Char Field	45	Not NULL	Product Status

Cart table:

SL. No	**Field Name**	**Data Type**	**Size**	**Constraints**	**Description**
1.	id	Integer Field	4	PRIMARY KEY	Cart Id
2.	user	User Instance	10	FOREIGN KEY	User Foreign Key
3.	product	Product Instance	10	FOREIGN KEY	Product Foreign Key
4.	quantity	Integer Field	10	Not NULL	Product Quantity

Designer table:

SL. No	Field Name	Data Type	Size	Constraints	Description
1.	id	Integer Field	4	PRIMARY KEY	Designer Id
2.	username	User Instance	10	FOREIGN KEY	User Foreign Key
3.	designer code	Integer Field	10	Not NULL	designer code
4.	full name	Char Field	100	Not NULL	designer full name
5.	email	Email Field	100	Not NULL	designer email
6.	phone	Char Field	100	Not NULL	designer phone
7.	work experience	Integer Field	4	Not NULL	designer work experience
8.	address	Text Field	100	Not NULL	designer address
9.	zip code	Integer Field	4	Not NULL	designer zip code
10.	state	Char Field	50	Not NULL	designer state
11.	designer image	Image Field		Not NULL	designer image

Designer Placed table:

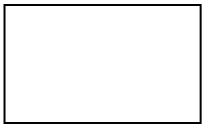
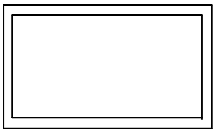
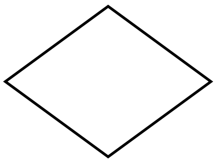
SL. No	Field Name	Data Type	Size	Constraints	Description
1.	id	Integer Field	4	PRIMARY KEY	Designer Placed Id
2.	user	User Instance	4	FOREIGN KEY	User Foreign Key
3.	customer	Customer Instance	4	FOREIGN KEY	Customer Foreign Key
4.	designer	Designer Instance	4	FOREIGN KEY	Designer Foreign Key

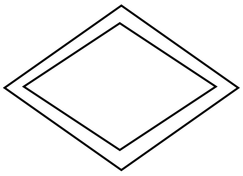
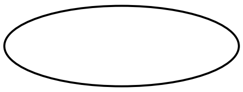
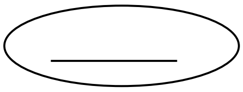
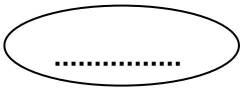
5.	Ordered date	DateTime Field		Not NULL	Ordered date
6.	status	Char Field	50	Not NULL	status

4.3 Entity Relationship Diagram (ER Diagram)

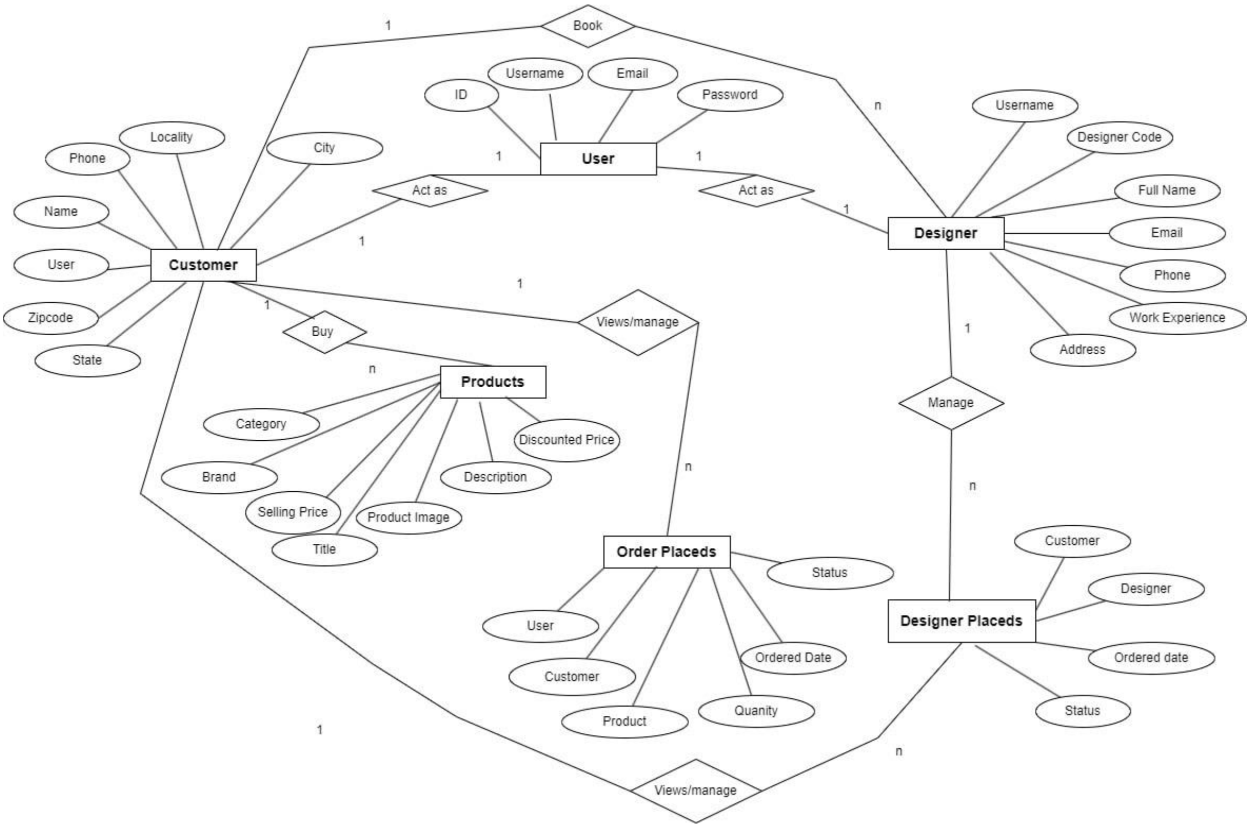
The ER Diagram is diagrammatical representation pf data model on real world entities. It is widely used on database design. A basic ER model is composed of entity types and specifies relationships that can exist between entities. An entity may be physical object as a house or a car, an event such as house sale or a service or a concept such as customer.

Some of the conventions while designing the ER diagram are shown below:

Symbol	**Convention**
	Entity
	Weak Entity
	Relationship

	Identifying relationship
	Attribute
	Key Attribute
	Foreign Key

ER DIAGRAM


Book
1
n
ID
Username
Email
Password
User
1
1
Act as
Act as
1
1
Locality
Phone
City
Name
User
Zipcode
State
Customer
Designer
Username
Designer Code
Full Name
Email
Phone
Work Experience
Address
1
1
Buy
n
Views/manage
Products
Category
Brand
Selling Price
Title
Product Image
Description
Discounted Price
n
1
Manage
n
Order Placeds
Status
User
Customer
Product
Quanity
Ordered Date
Designer Placeds
Customer
Designer
Ordered date
Status
1
n
Views/manage

DETAILED DESIGN

5.1 Introduction:

During the detail design, the internal logic of each module specified in the system design is decided. Detailed design focuses on the detailed explanation of each module in the software. It illustrates the entire requirement for the software. The logic of a module is usually specified in a high-level description language, which is independent of the target language in which the software will eventually be implemented.

5.2 Applicable documents:

- Synopsis
- Software requirement specification
- System design

5.3 Structure of the software package:

- Registration
- Login
- Registered User
    - Cart
        - Add to cart
        - Increase Quantity
        - Remove from the cart
    - Search
    - Profile
        - Add/Delete Profile
        - My Orders
            - Tracking
            - Cancel orders
        - Change password
        - Logout
    - Contact Us

- Admin
    - Login
    - All Products
        - Add products
        - Edit

        - Delete
    - Add Designer
        - Edit
        - Delete
    - View Orders
    - Logout
- Cart
    - Remove
    - Order
- Designer
    - Login
    - Order details
        - Accept
        - Reject
    - Logout

5.4 Modular decomposition of documents:

5.4.1 Login:

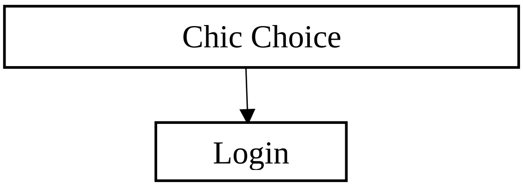
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Allows the user to enter the User id and password.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  User table

- **Design logic (Structured English):**

  Structured English is the use of English language with the syntax of structured programming to communicate the design of the computer program to non-technical user by breaking it down into logical steps using straight forward English words. Structured English aims to get the benefits to both programming logic and natural English. Programming logic helps to attain precision whilst natural language helps with familiarity of the spoken words.

```
IF Username, password is valid THEN
        Respective page will be loaded
ELSE
        Error message will be displayed
END IF
```

5.4.1.1 Forgot password:

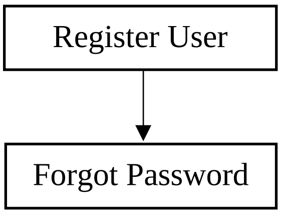
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  It will check the authenticity before changing the password. User must specify the already registered User Id.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  user table

- **Design logic (Structured English):**

```
IF details valid THEN
        Update password in the database
ELSE
        Display error message
END IF
```

5.4.2 Registration:

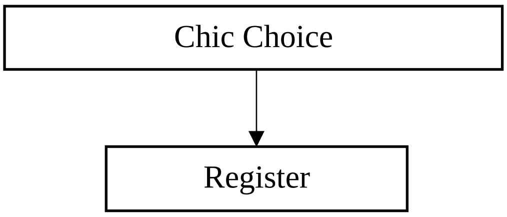
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  This module allows the system user to register themselves.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

User table

- **Design logic (Structured English):**

```
IF user details are valid THEN
        Store to database
        Display success message
ELSE
        Display error message
END IF
```

5.4.3 Admin

5.4.3.1 All products:

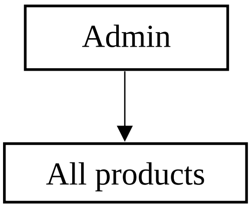
- **Design assumptions:**

All the fields are mandatory.

- **Identifications of the modules:**

This module allows the admin to view, edit and delete products.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

product table

- **Design logic (Structured English):**

```
IF product details valid THEN
        Display the product
ELSE
        Edit or delete the product
END IF
```

5.4.3.1.1 Add/Update/Delete Products:

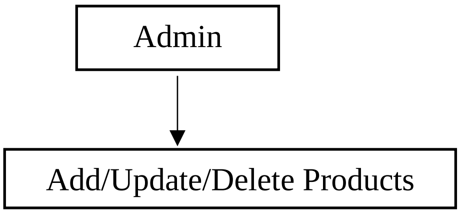
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  This module allows the admin to add, update and delete product.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  Product table

- **Design logic (Structured English):**

```
IF Product is valid THEN
        Store it to the database
ELSE
        Display error message
END IF
```

5.4.3.1.2 Add/Update/Delete Designer:

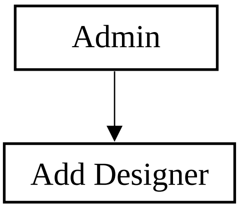
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  This module allows the admin to add, update and delete designer.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  Designer table

- **Design logic (Structured English):**

```
IF Designer is valid THEN
```

```
        Store it to the database
    ELSE
        Display error message
    END IF
```

5.4.3.1.3 View Orders:

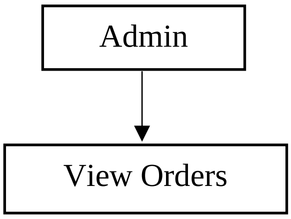
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Admin can view the order details of all the users.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  pro_order table

- **Design logic (Structured English):**

```
IF User Orders present THEN
        Display order details
END IF
```

5.4.4 Registered user:

5.4.4.1 Profile

5.4.4.1.1Add/Delete Profile:

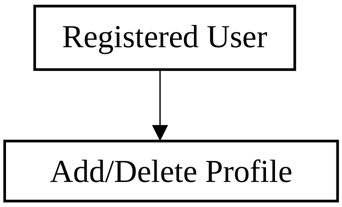
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Here, users can add or delete the address or profile.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

user table

- **Design logic (Structured English):**

```
IF user details available THEN
    Display user details
END IF.
```

5.4.4.1.2 My Order:

5.4.4.1.2.1 Tracking Product:

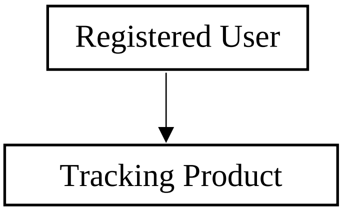
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Here, users can view and track the order product.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  Order placed table

- **Design logic (Structured English):**

```
IF order product available THEN
    Display user details
END IF.
```

5.4.4.1.2.1 Cancel order:

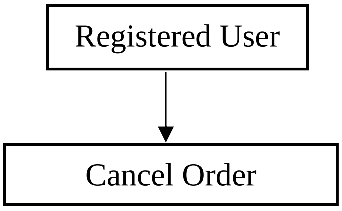
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Here, users can cancel ordered product.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  Order placed table

- **Design logic (Structured English):**

```
IF order product available THEN
    Display user details
```

5.4.4.1.3 Change Password:

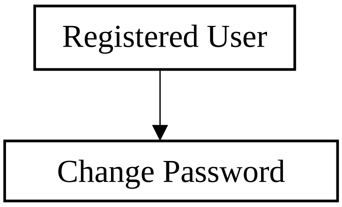
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Here, users can change the password.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  user table

- **Design logic (Structured English):**

```
IF user details available THEN
    Update to database
    Display success message
END IF.
```

5.4.4.1.4Cart:

5.4.4.3.1 Add to cart:

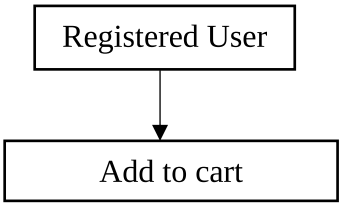
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Here, users can add the particular products to cart for ordering.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  cart table

- **Design logic (Structured English):**

```
IF user wants to order product THEN
    Add product to cart.
END IF
```

5.4.4.3.2 Increase Quantity:

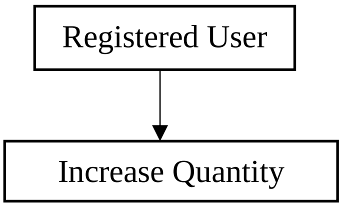
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Here, users can increase the product quantity.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  cart table

- **Design logic (Structured English):**

```
IF user wants to increase product quantity THEN
    Increase quantity of  the product.
END IF
```

5.4.4.3.3 Remove from the cart:

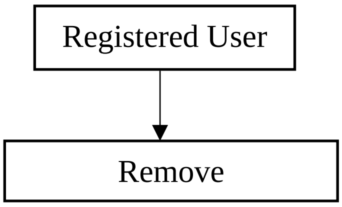
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Here, users can remove the product from cart table.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  cart table

- **Design logic (Structured English):**

```
IF user wants to remove product THEN
```

```
            Increase quantity of  the product.
        END IF
```

5.4.5 Search:

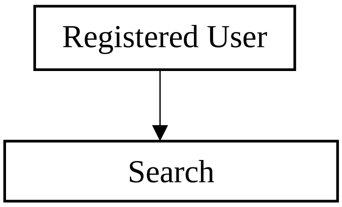
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Here, users can search the particular product.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  pro_details table

- **Design logic (Structured English):**

```
IF particular product available THEN
        Display that particular product.
END IF
```

5.4.6 Cart:

5.4.6.1 Remove:

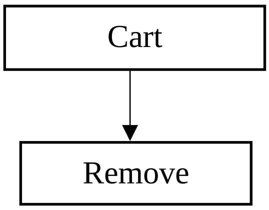
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  The user can remove the product which they don’t want to order.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  cart table

- **Design logic (Structured English):**

```
IF user don't want product THEN
        Remove that product
END IF
```

5.4.6.2 Order Product:

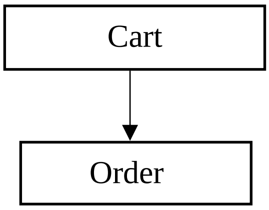
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  The user will first need to login to the system using the username and password. Then order a product by entering certain details.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  Order placed table

- **Design logic (Structured English):**

```
IF order details are valid THEN
        Store to database
        Display success message
ELSE
        Display error message
END IF
```

5.4.7 Designer:

5.4.7.1 Order details:

5.4.7.1.1 Accept:

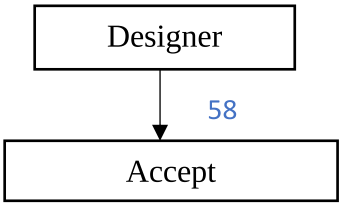
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Here, designer can accept the order.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  user table

- **Design logic (Structured English):**

```
IF designer have order THEN
    Accept the order
END IF
```

5.4.7.1.2 Reject:

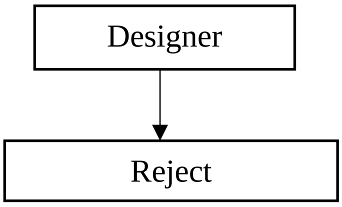
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Here, designer can reject the order.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  user table

- **Design logic (Structured English):**

```
IF designer have order THEN
    Reject the order
END IF
```

5.4.8 Contact Us:

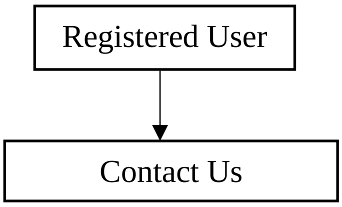
- **Design assumptions:**

  All the fields are mandatory.

- **Identifications of the modules:**

  Here, users can contact admin regarding any queries.

- **Structure chart showing the hierarchy of modules:**



- **Data structures shared among modules:**

  user table

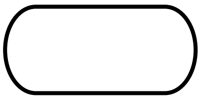
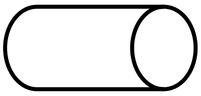
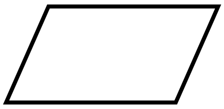
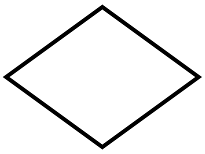
- **Design logic (Structured English):**

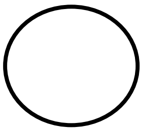
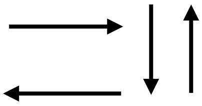
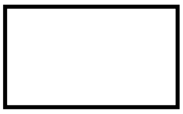
```
IF user have any queries THEN
        Contact admin.
END IF
```

5.5 Flowchart and Structured chart:

Flowchart is a pictorial representation of modules or graphs. It is a type of diagram that represents an algorithm, work flow or process, showing the steps as basis of the various kinds and their order by connecting them with arrows. The diagrammatic representation illustrates a solution to model to a given problem. Flowcharts are used in analysing, designing and documenting all the activities of the software product.

Symbols used in the flowchart:

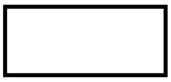
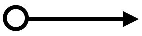
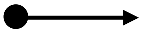
Symbols	**Name**	**Description**
	Terminator	It includes the beginning and the end.
	Direct Access Storage	To represent data stored in the databases.
	Input / Output	It represents the user inputs and the outputs produced
	Decision	It represents sequence in process where end users choose an option and then branches an alternate path.

	Connector	It indicates the continuity of a next step in another page.
	Flow	It represents the flow of control.
	Process	It indicates process functions like calculation, data manipulation, information processing and assignment.

Structured chart:

A structured chart is a graphical depiction of the decomposition of the problem. It is a tool used in software design. A structure chart is a top-down modular design tool, constructed of square representing different modules in a system and lines that connect them.

Symbols used in the structured chart:

Symbols	**Name**	**Description**
	Module	It represents sub-ordinates and superior-ordinate modules.
	Data Flow	It indicates the direction of flow of data.
	Control flow	It indicates the direction of flow of control.

⟶	Invocation	It represents the sub-ordinate modules being invoked by superior ordinate modules.

5.5.1 Module design of components:

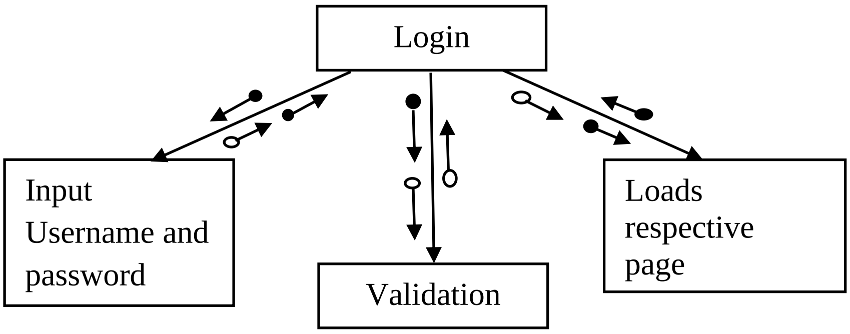
5.5.1.1 LOGIN:

5.5.1.1.1 Input:

- Username
- Password

5.5.1.1.2 Procedural details:

❖ **Structured chart**

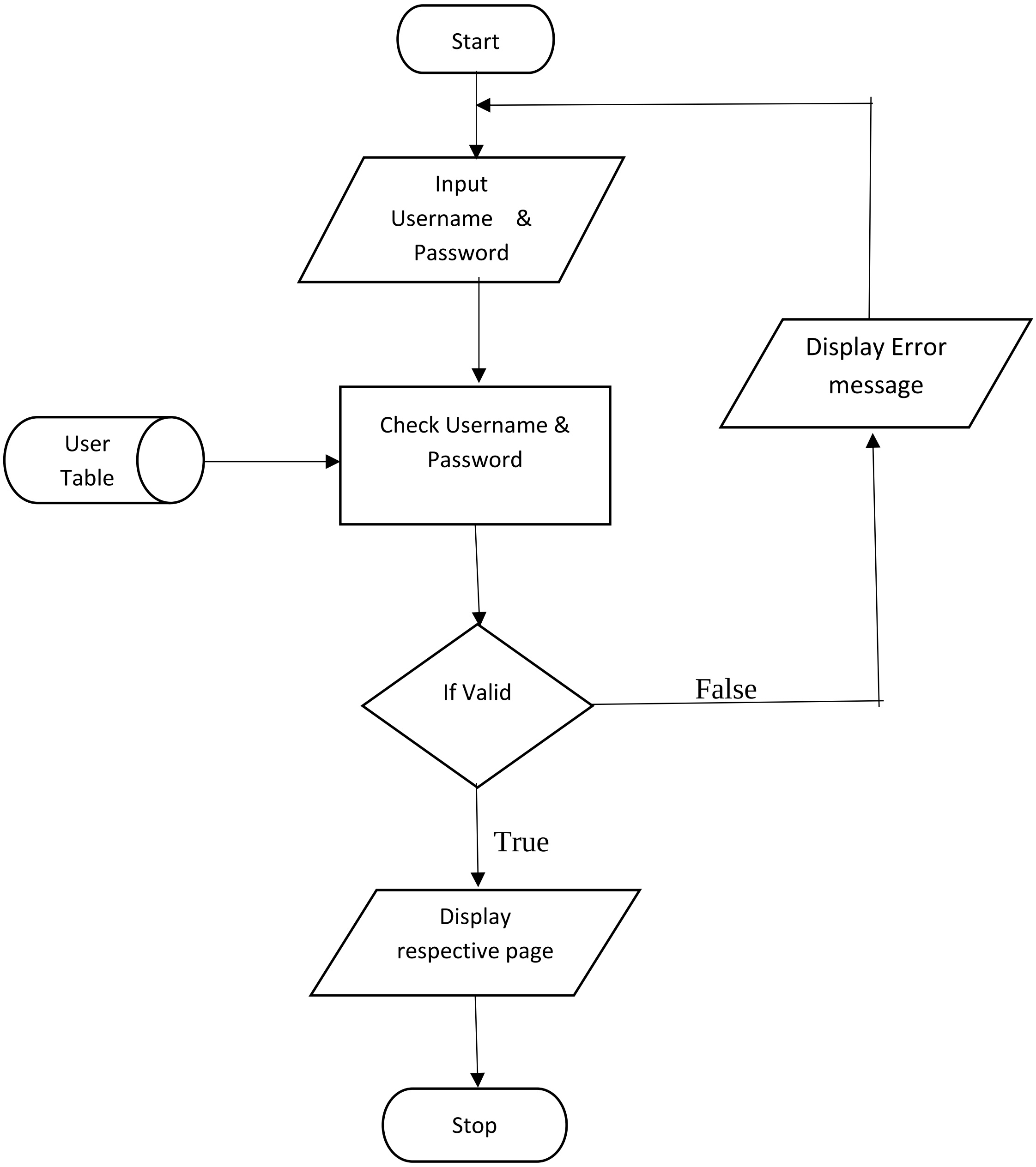


❖ **Algorithm: User Login**

An algorithm is a finite sequence of explicit instructions. Algorithm is provided with a set of input and then terminates.

```
Step1: Start
Step2: Input Username and password
Step3: Verify Username and password
Step4: If data is valid then
                Display the respective page.
        Else
                Display error message.
        End if
Step5: Exit
```

- **Flowchart:**



5.5.1.3.2.1.3 File I/O interfaces:

User table

5.5.1.3.2.1.4 Output:

Product details will be updated.

5.5.1.3.2.1.5 Implementation aspects:

User interface contains labels, textboxes and button.

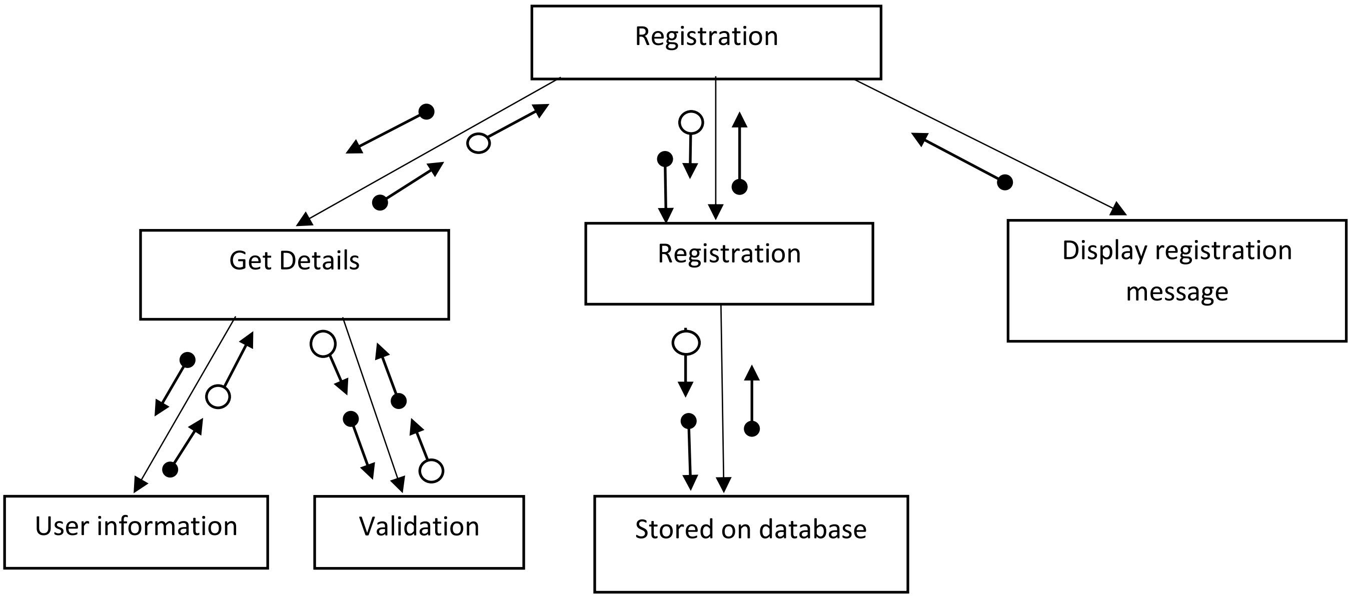
5.5.1.2 REGISTRATION:

5.5.1.2.1 Input:

- Full name
- Email address
- Password
- Confirm Password

5.5.1.2.2 Procedural details:

- **Structured chart:**



Algorithm: User Registration

Step1: Start

Step2: Input user information

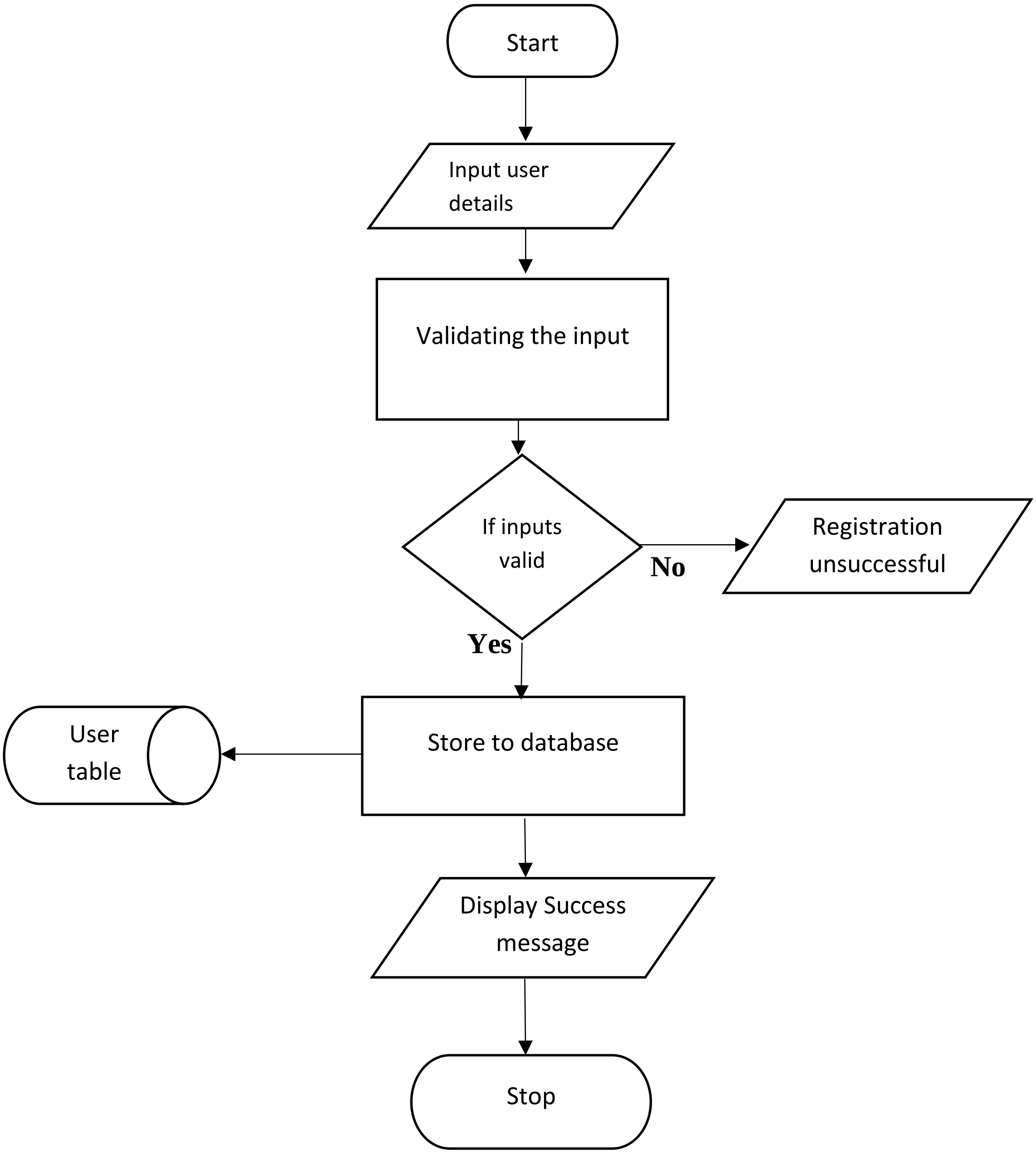
Step3: Validating the input

Step4: If input is valid then

Display registration successful message and store user details to the database

```
        Else
                Display error message
        End if
Step5: End
```

- **Flowchart:**



5.5.1.2.3 File I/O interfaces:

User Table

5.5.1.2.4Output:

Stores user details in the database.

5.5.1.2.5 Implementation aspects:

Users interface contains textboxes, checkbox and a button.

5.5.1.3 ADMIN

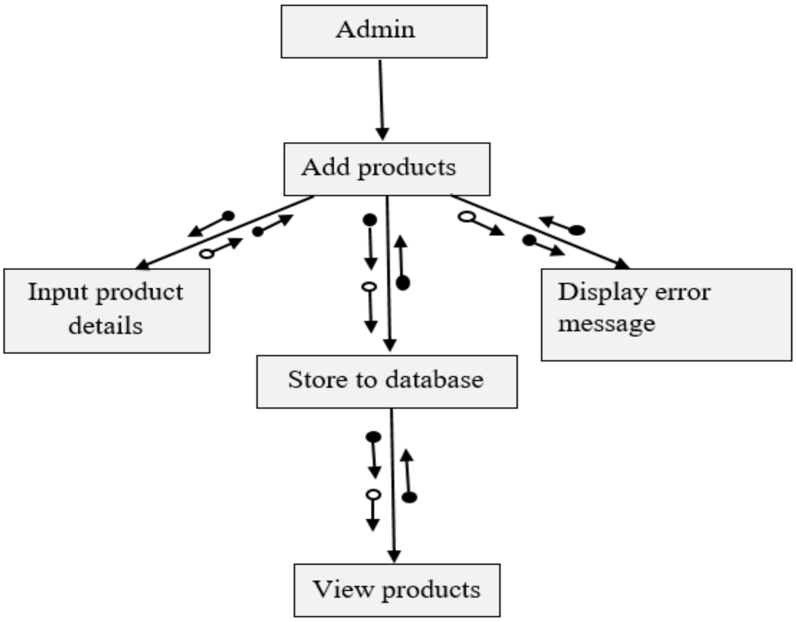
5.5.1.3.1 Add products:

5.5.1.3.1.1 Input:

- Product Name
- Description
- Price
- Product Category
- Product Status
- Upload Photo

5.5.1.3.1.2 Procedural details:

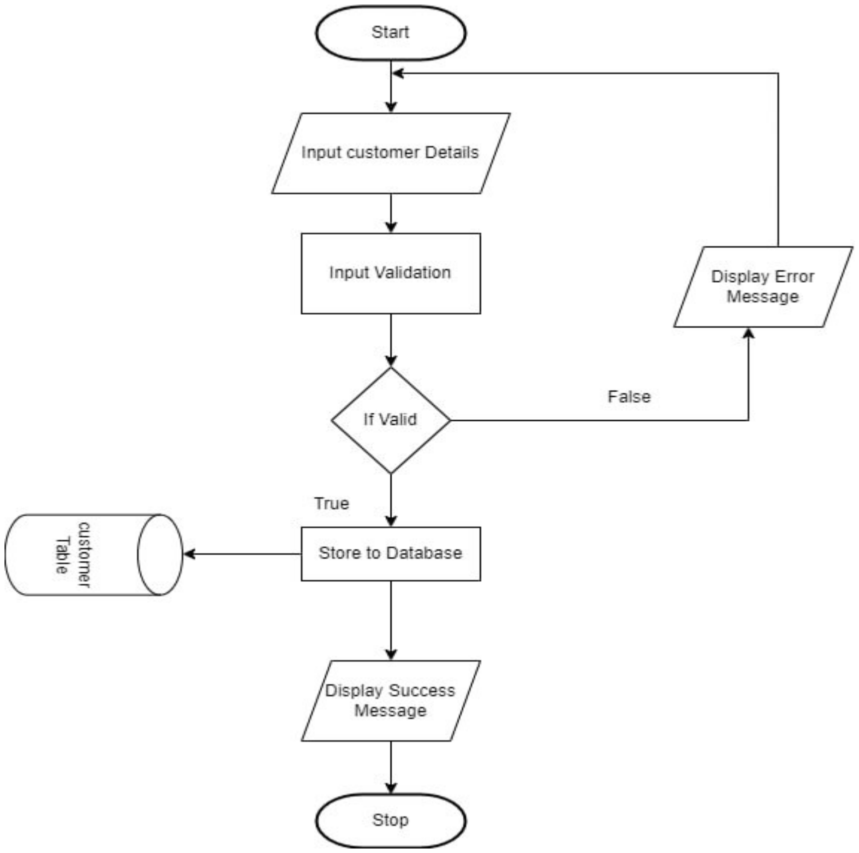
❖ **Structured chart:**



Algorithm: Add products

```
Step1: Start
Step2: Input product details
Step3: IF details are valid then
            a. Add product details to the database
            b. Display Success message
        Else
            Display error message
        End if
Step4: Exit
```

❖ **Flowchart:**



5.5.1.3.1.3File I/O interfaces:

Pro_details table

5.5.1.3.1.4Output:

Product details will be stored in the database.

5.5.1.3.1.5Implementation aspects:

User interface contains text boxes, image, drop down lists and buttons.

5.5.1.3.2 Add Designer:

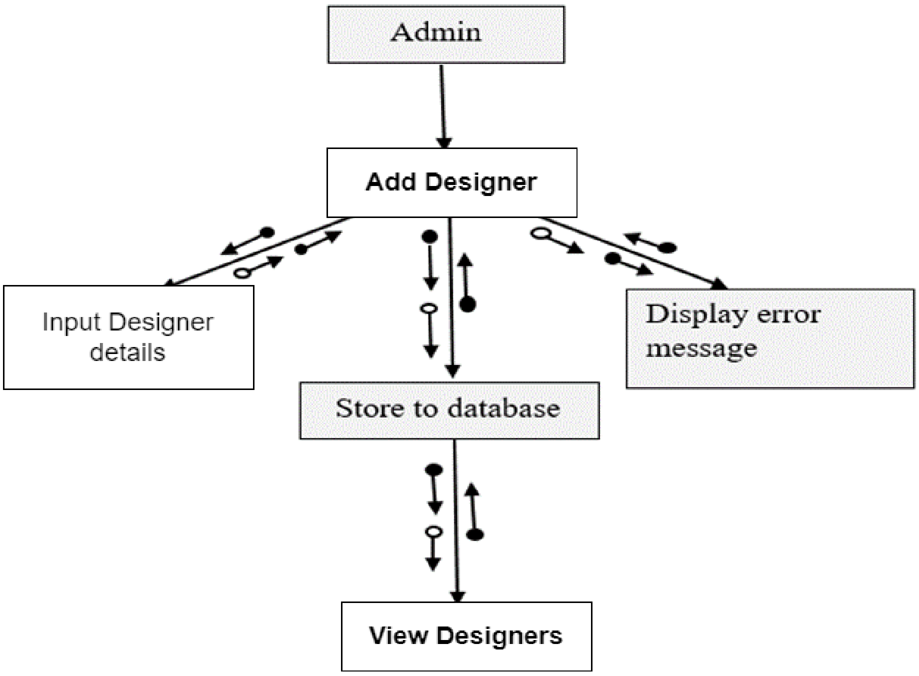
5.5.1.3.2.1 Edit/Delete Designer:

5.5.1.3.2.1.1 Input:

- User Name
- Designer code
- Full Name
- Email
- Phone number
- Work Experience
- Address
- Zip code
- State
- Designer image

5.5.1.3.2.1.2 Procedural details:

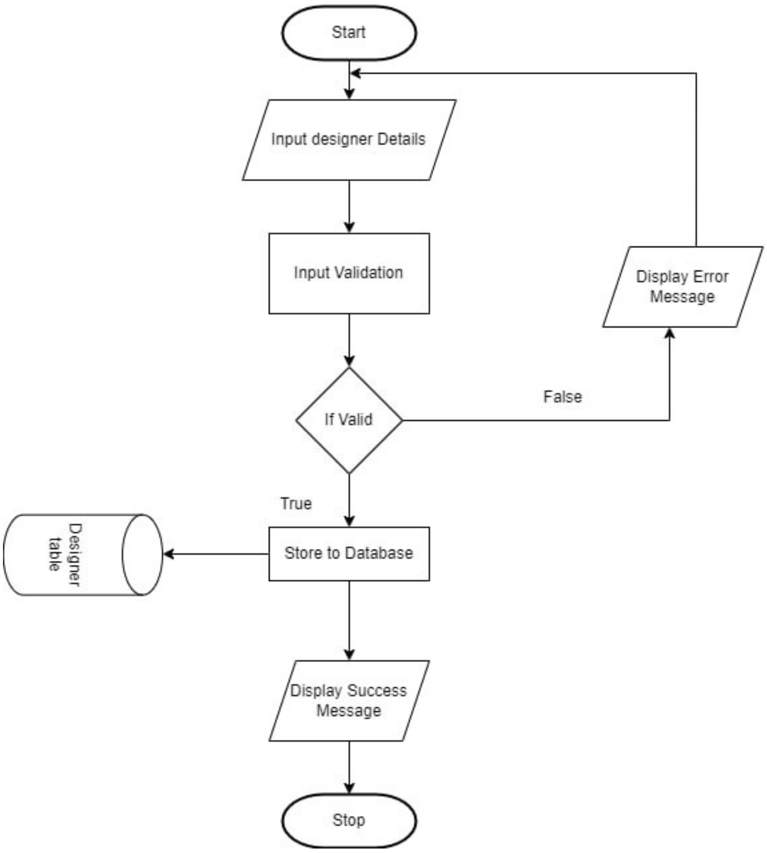
- **Structured chart:**



❖ **Algorithm: Add products**

```
Step1: Start
Step2: Input designer details
Step3: IF details are valid then
                    a. Add designer  details to the database
                    b. Display Success message
         Else
                    Display error message
         End if
  Step4: Exit
```

- **Flowchart:**



5.5.1.3.2.1.3 File I/O interfaces:

Pro_details table

5.5.1.3.2.1.4 Output:

Product details will be updated.

5.5.1.3.2.1.5 Implementation aspects:

User interface contains labels, textboxes and button.

5.5.1.4 REGISTERED USER:

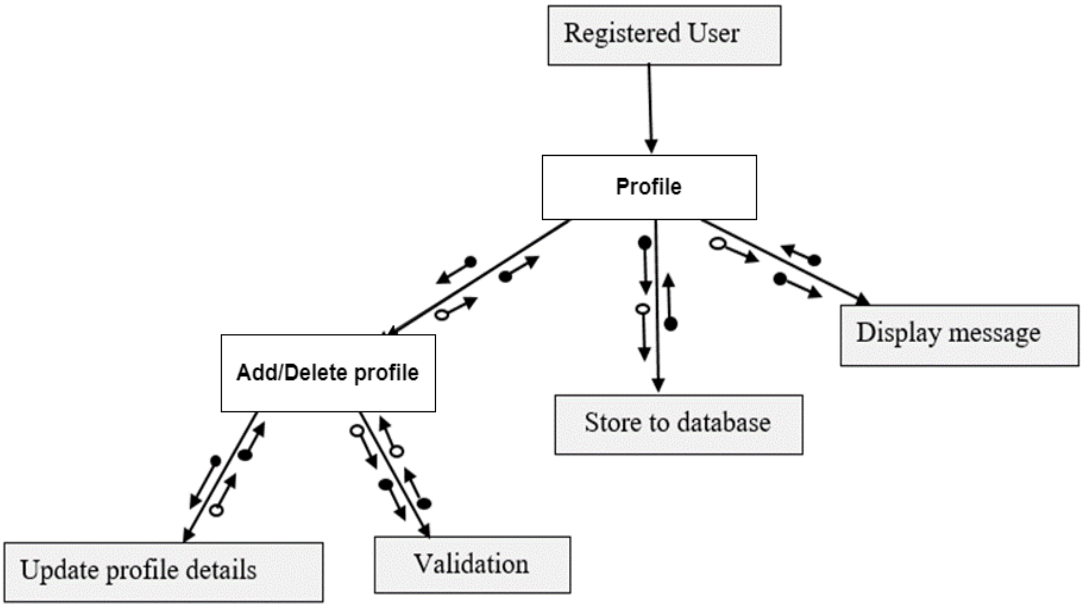
5.5.1.4.1 Add/Delete Profile:

5.5.1.4.1.1 Input:

- Name
- Phone number
- Locality
- City
- State
- Zip code

5.5.1.4.1.2 Procedural details:

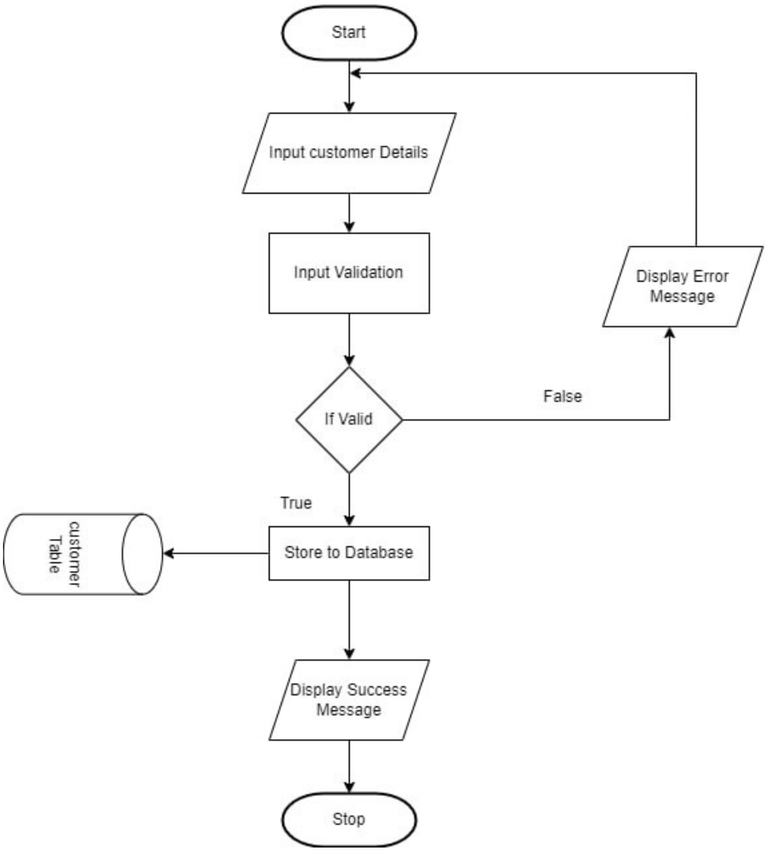
❖ Structured chart:



❖ Algorithm: Add/Delete profile:

```
Step1: Start
Step2: Input User details
Step3: If customer details invalid then
            Display error message
       Else
            user information will be updated and stored to the
            database.
       End if
  Step4: Exit
```

- ❖ **Flowchart:**



5.5.1.4.1.3 File I/O interfaces:

User table

5.5.1.4.1.4 Output:

Profile settings will be updated.

5.5.1.4.1.5 Implementation aspects:

User interface contains labels, textboxes and button.

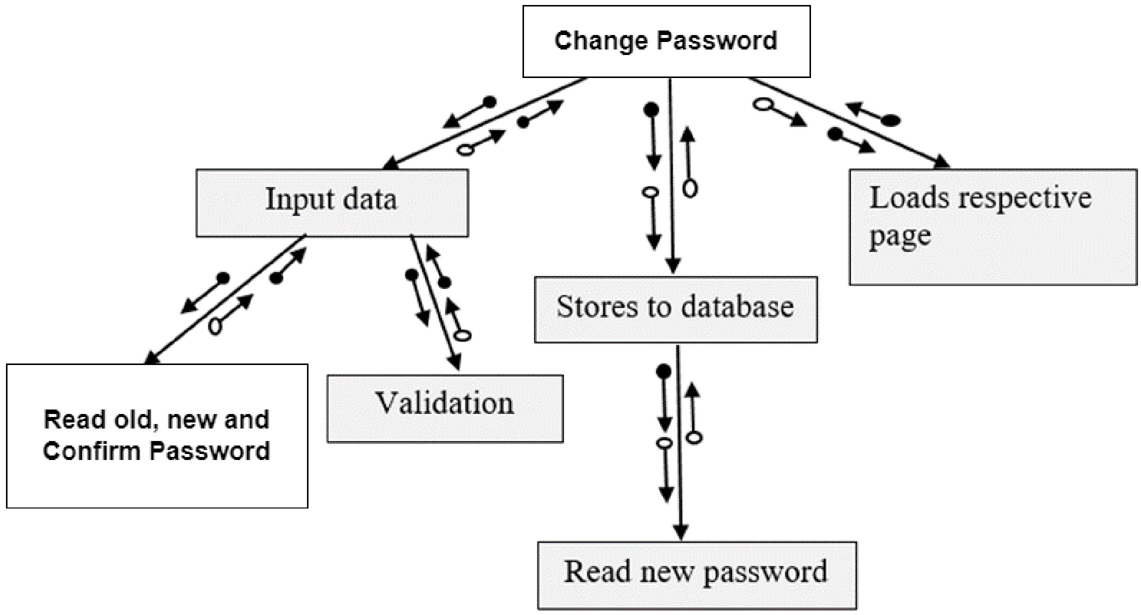
5.5.1.4.2 Change password:

5.5.1.4.2.1 Input:

- Old Password
- New Password
- Confirm Password

5.5.1.4.2.2 Procedural details:

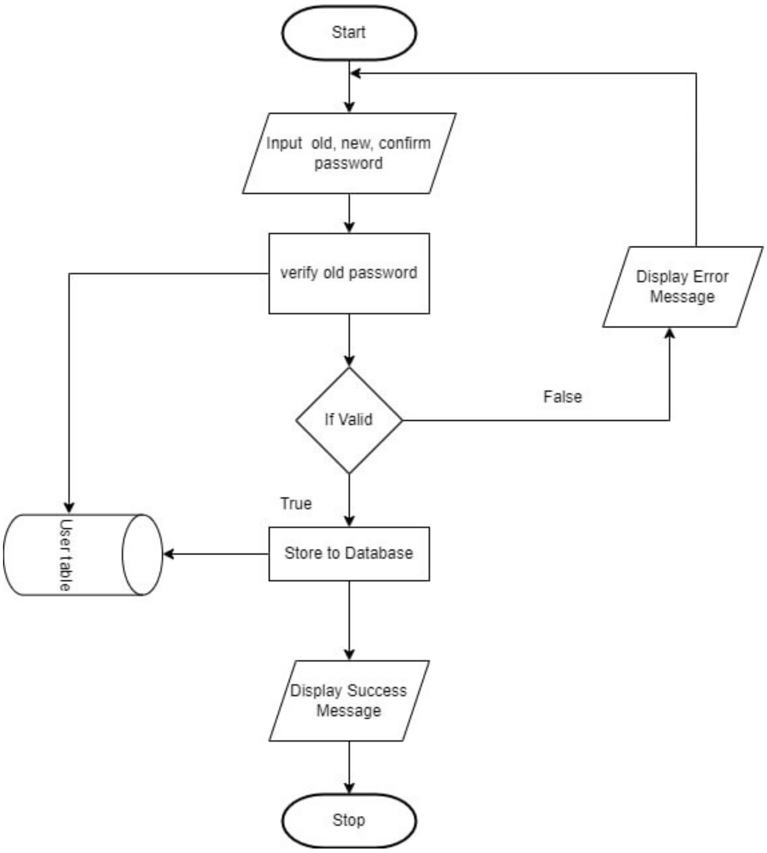
- ❖ **Structured chart:**



❖ **Algorithm: Forgot Password**

```
Step1: Start
Step2: Input old password, new password, confirm password
Step3: Compare the new password with confirm password.
Step4: IF passwords are similar then
                a. Read new password then
                b. Store to database
        Else
                Display error message
        End if
  Step5: Exit.
```

- **Flow chart:**



5.5.1.4.2.3 File I/O interfaces:

User table

5.5.1.4.2.4 Output:

password will be updated.

5.5.1.4.2.5 Implementation aspects:

User interface contains labels, textboxes and button.

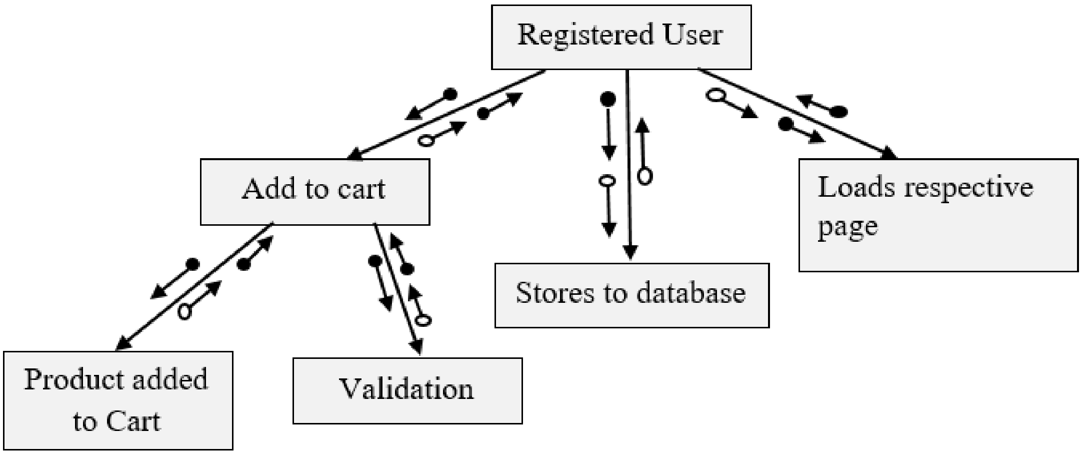
5.5.1.4.3 Add to cart:

5.5.1.4.3.1 Input:

- User Name
- Product
- Quantity

5.5.1.4.3.2 Procedural details:

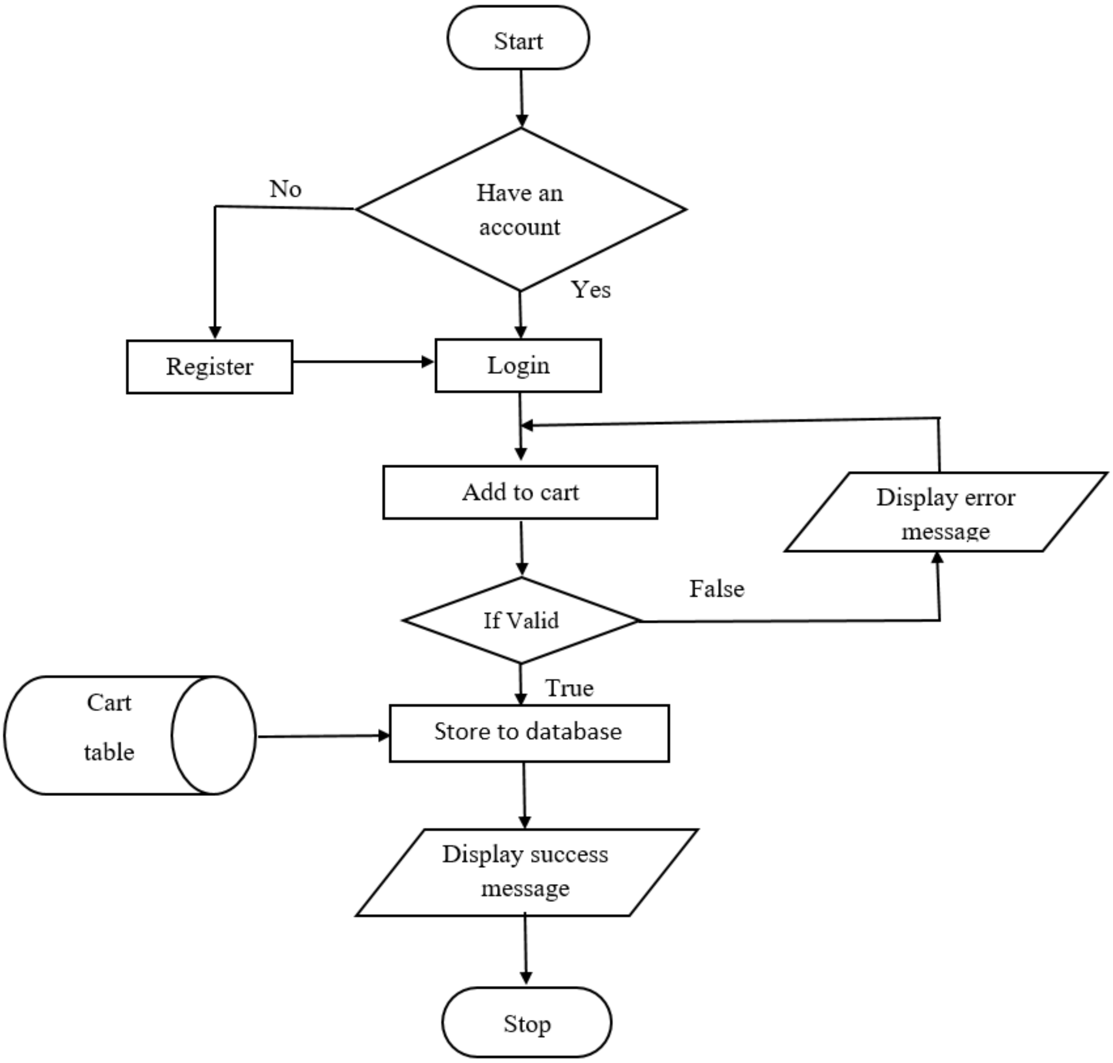
- **Structured chart:**



Algorithm: Add to cart

```
Step1: Start
Step2: Input product order details
Step3: If product status is active then
            Store the product details to the database
            Display success message
    Else
            Display error message
    End if
    Step4: Exit
```

- **Flowchart:**



5.5.1.4.3.3 File I/O interfaces:

Cart table

5.5.1.4.3.4 Output:

Particular product added to cart.

5.5.1.4.3.5 Implementation aspects:

User interface contains textboxes, tables and buttons.

5.5.1.5 Cart:

5.5.1.5.1 Order

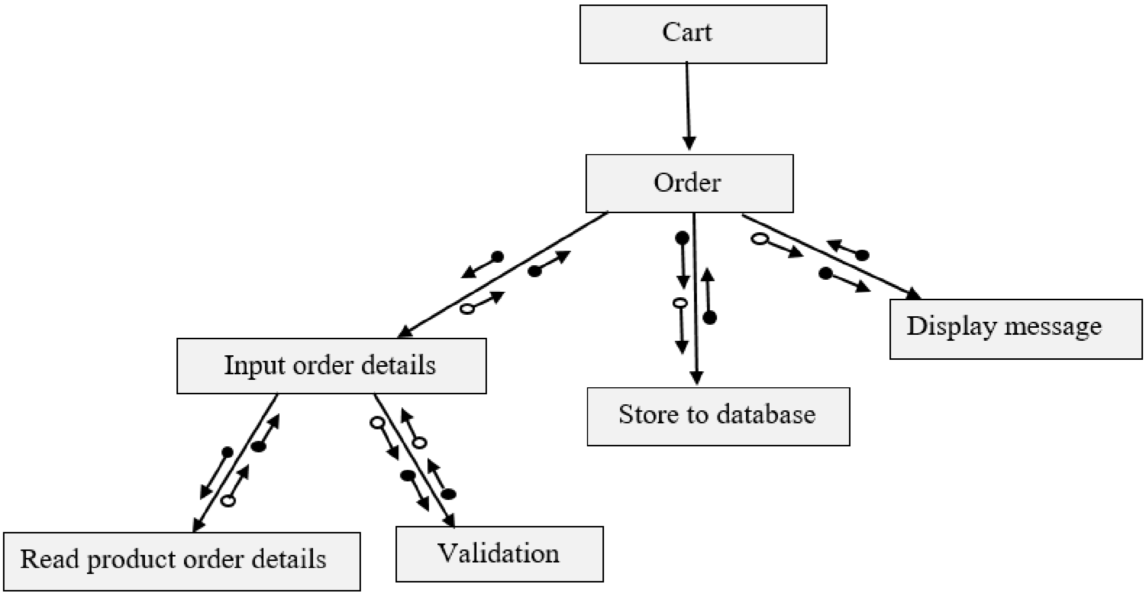
5.5.1.5.1.1 Input:

- **User**

- **Customer**
- **Product**
- **Quantity**
- **Ordered date**
- **Status**

5.5.1.5.1.2 Procedural details:

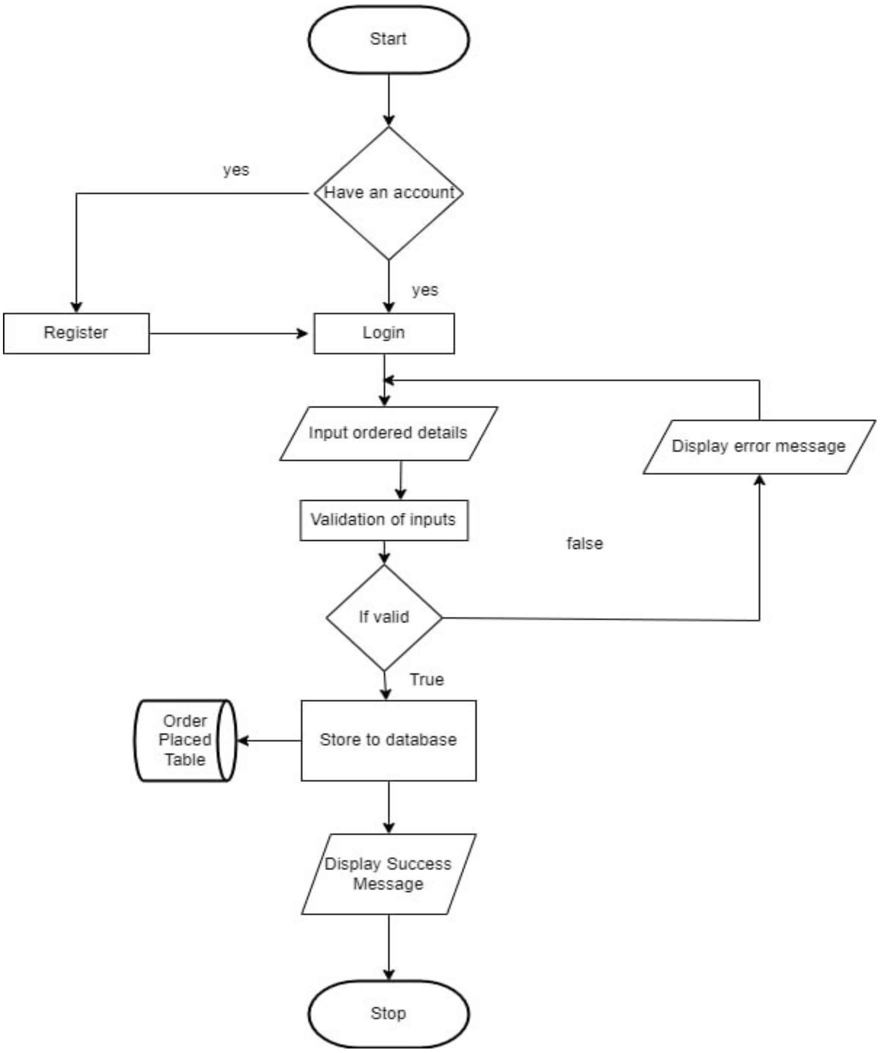
- **Structured chart**:



Algorithm: Order product

```
Step1: Start
Step2: Input product order details
Step3: If order details valid then
            Add details to the product order table
            Display success message
        Else
            Display error message
        End if
Step4: Exit
```

- **Flowchart:**



5.5.1.5.1.1.3 File I/O interfaces:

Order placed table

5.5.1.5.1.1.4 Output:

Product will be ordered.

5.5.1.5.1.1.5 Implementation aspects:

User interface contains textboxes and buttons.

CODING

Import files for views.py

```
from django.shortcuts import render, redirect, HttpResponse

from django.views import View

from .models import Customer, Product, Cart, OrderPlaced, BCart, Designer, DesignerPlaced, TempDesignerPlaced

from .forms import CustomerRegistrationForm, CustomerProfileForm

from django.contrib import messages

from django.contrib.auth.decorators import login_required

from django.utils.decorators import method_decorator

from django.db.models import Q

from django.http import JsonResponse
```

Code for Customer Registration:

```
class CustomerRegistrationView(View):
    def get(self, request):
        form = CustomerRegistrationForm()
        return render(request, 'app/customerregistration.html', {'form':form})
    def post(self, request):
        form = CustomerRegistrationForm(request.POST)
        if form.is_valid():
            messages.success(request, "Congratulations!! Registered Successfully")
            form.save()
            return redirect("login")
        return render(request, 'app/customerregistration.html', {'form':form})
```

Code for Customer Login:

```
class LoginForm(AuthenticationForm):
    username = UsernameField(widget=forms.TextInput(attrs={'autofocus':True, 'class':'form-control'}))
    password = forms.CharField(label=_("Password"), strip=False,widget=forms.PasswordInput
                    (attrs={'autocomplete':'current-password', 'class':'form-control'}))
```

Code for Product Detail View:

```
class ProductDetailView(View):
    def get(self,request,pk):
        product = Product.objects.get(pk=pk)
        totalitem = 0
        item_already_in_cart = False
        if request.user.is_authenticated:
            totalitem = len(Cart.objects.filter(user=request.user))
            item_already_in_cart = Cart.objects.filter(Q(product=product.id)
                        & Q(user=request.user)).exists()
        return render(request, "app/productdetail.html", {"product":product,
                        'item_already_in_cart':item_already_in_cart, 'totalitem':totalitem})
```

Code for Designer Detail View:

```
class DesignerDetailView(View):
    def get(self,request,pk):
        designer = Designer.objects.get(pk=pk)
        already_exist = DesignerPlaced.objects.filter(
                            user=request.user, designer=designer)
        if already_exist:
            return redirect('designer_orders')
        totalitem = 0
        if request.user.is_authenticated:
            totalitem = len(Cart.objects.filter(user=request.user))
        return render(request, "app/designerdetail.html",
                            {"designer":designer, "totalitem":totalitem})
```

Code for Show Cart:

```
@login_required
def show_cart(request):
    totalitem = 0
    if request.user.is_authenticated:
        totalitem = len(Cart.objects.filter(user=request.user))
        user = request.user
        carts = Cart.objects.filter(user=user)
        amount = 0.0
        shipping_amount = 70
        total_amount = 0.0
        cart_product = [p for p in Cart.objects.all() if p.user == user]
        if cart_product:
            for p in cart_product:
                tempamount = (p.quantity * p.product.discounted_price)
                amount+= tempamount
                total_amount = amount + shipping_amount
            return render(request, 'app/addtocart.html',
                          {'carts':carts, 'totalamount':total_amount,
                         'amount':amount, 'totalitem':totalitem} )
        else:
            return render(request,'app/emptycart.html')
```

Code for Orders:

```
@login_required
def orders(request):
    totalitem = 0
    if request.user.is_authenticated:
        totalitem = len(Cart.objects.filter(user=request.user))
    op = OrderPlaced.objects.filter(user=request.user)
    if op:
        return render(request, 'app/orders.html',
                                {'order_placed':op, 'totalitem':totalitem})
    else:
        return render(request,'app/emptyorder.html')
```

Code for Plus Cart:

```
def plus_cart(request):
    if request.method == 'GET':
        prod_id = request.GET['prod_id']
        user = request.user
        c = Cart.objects.get(Q(product=prod_id)& Q(user=request.user))
        c.quantity+=1
        c.save()
        amount = 0.0
        shipping_amount = 70
        cart_product = [p for p in Cart.objects.all() if p.user == user]
        for p in cart_product:
            tempamount = (p.quantity * p.product.discounted_price)
            amount+= tempamount
        data = {
            'quantity': c.quantity,
            'amount': amount,
            'totalamount': amount + shipping_amount
            }
        return JsonResponse(data)
```

Code for Minus Cart:

```
def minus_cart(request):
    if request.method == 'GET':
        prod_id = request.GET['prod_id']
        user = request.user
        c = Cart.objects.get(Q(product=prod_id)& Q(user=request.user))
        c.quantity-=1
        c.save()
        amount = 0.0
        shipping_amount = 70
        cart_product = [p for p in Cart.objects.all() if p.user == user]
        for p in cart_product:
            tempamount = (p.quantity * p.product.discounted_price)
            amount+= tempamount
        data = {
            'quantity': c.quantity,
            'amount': amount,
            'totalamount': amount + shipping_amount
            }
        return JsonResponse(data)
```

Code for Delete Item:

```
def delete_item(request):
    if request.method == 'GET':
        prod_id = request.GET['prod_id']
        c = OrderPlaced.objects.get(Q(product=prod_id)& Q(user=request.user))
        c.delete()
    return redirect('orders')
```

Code for Delete Profile:

```
def delete_profile(request):
    if request.method == 'GET':
        add_id = request.GET['add_id']
        c = Customer.objects.get(Q(id=add_id)& Q(user=request.user))
        c.delete()
    return redirect('address')
```

Code for Remove Cart:

```
def remove_cart(request):
    if request.method == 'GET':
        prod_id = request.GET['prod_id']
        user = request.user
        c = Cart.objects.get(Q(product=prod_id)& Q(user=request.user))
        c.delete()
        amount = 0.0
        shipping_amount = 70
        cart_product = [p for p in Cart.objects.all() if p.user == user]
        for p in cart_product:
            tempamount = (p.quantity * p.product.discounted_price)
            amount+= tempamount
        data = {
            'amount': amount,
            'totalamount': amount + shipping_amount
            }
        return JsonResponse(data)
```

Code for Profile View:

```
@method_decorator(login_required, name='dispatch')
class ProfileView(View):
    totalitem = 0
    def get(self, request):
        form = CustomerProfileForm()
        totalitem = len(Cart.objects.filter(user=request.user))
        return render(request, 'app/profile.html', {'form':form,
                                    'active':'btn-primary', 'totalitem':totalitem})
    def post(self, request):
        form = CustomerProfileForm(request.POST)
        if form.is_valid():
            usr = request.user
            name = form.cleaned_data['name']
            phone = form.cleaned_data['phone']
            locality = form.cleaned_data['locality']
            city = form.cleaned_data['city']
            state = form.cleaned_data['state']
            zipcode = form.cleaned_data['zipcode']
            reg = Customer(user=usr, name=name, phone=phone,
                             locality=locality, city=city, state=state, zipcode=zipcode)
            reg.save()
            messages.success(request, 'Congratulations!!! Profile updated successfully')
            form = CustomerProfileForm()
        return render(request, 'app/profile.html', {'form':form, 'active':'btn-primary'})
```

Code for Address:

```
@login_required
def address(request):
    totalitem = 0
    add = Customer.objects.filter(user=request.user)
    if request.user.is_authenticated:
        totalitem = len(Cart.objects.filter(user=request.user))
    return render(request, 'app/address.html', {'add':add, 'active':'btn-primary',
                    'totalitem':totalitem})
```

Code for Checkout:

```
@login_required
def checkout(request):
    totalitem = 0
    user = request.user
    add = Customer.objects.filter(user=user)
    cart_items = Cart.objects.filter(user=user)
    amount = 0.0
    shipping_amount =70
    totalamount = 0.0
    if request.user.is_authenticated:
        totalitem = len(Cart.objects.filter(user=request.user))
    cart_product = [p for p in Cart.objects.all() if p.user == user]
    if add:
        if cart_product:
            for p in cart_product:
                tempamount = (p.quantity * p.product.discounted_price)
                amount+= tempamount
            totalamount = amount + shipping_amount
            dlr = int(totalamount/82.35)
        return render(request, 'app/checkout.html', {'add':add, 'totalamount':totalamount,
                        'cart_items':cart_items, 'dlr':dlr, 'totalitem':totalitem})
    else:
        return redirect('profile')
```

Code for Add to Cart:

```
@login_required
def add_to_cart(request):
    user = request.user
    product_id = request.GET.get('prod_id')
    product = Product.objects.get(id=product_id)
    Cart(user=user, product=product).save()
    return redirect('/cart')
```

Code for Buynow Checkout:

```
@login_required
def buynow_checkout(request):
    user = request.user
    product_id = request.GET.get('prod_id')
    pro_duct = Product.objects.get(id=product_id)
    BCart(user=user, product=pro_duct).save()
    add = Customer.objects.filter(user=user)
    amount = 0.0
    shipping_amount =70
    totalamount = 0.0
    # cart_product = [p for p in Product.objects.all() if p.user == user]
    if add:
        if pro_duct:
            amount = pro_duct.discounted_price
            image = pro_duct.product_image
            totalamount = amount + shipping_amount
        dlr = int(totalamount/82.35)
        return render(request, 'app/buynow_checkout.html', {'add':add,"amount":amount,
                        "image":image, 'totalamount':totalamount,
                                'pro_duct':pro_duct, 'dlr':dlr})
    else:
        return redirect('profile')
```

Code for Payment Done:

```
@login_required
def payment_done(request):
    user = request.user
    custid = request.GET.get('custid')
    customer = Customer.objects.get(id=custid)
    cart = Cart.objects.filter(user=user)
    for c in cart:
        OrderPlaced(user=user, customer=customer,
                          product=c.product, quantity=c.quantity).save()
        c.delete()
    return redirect("orders")
```

Code for About, Contact and Help:

```
@login_required
def about(request):
    return render(request, "app/about.html")

@login_required
def contact(request):
    return render(request, "app/contact_page.html")

@login_required
def help(request):
    return render(request, "app/chatbot.html")
```

Code for Adding Product:

```
@login_required
def mshirts(request, data=None):
    if data == None:
        mshirts = Product.objects.filter(category="MST")
    elif data == "Nap_Chief" or data == "Ben_Martin" or data == "Park_Avenue":
        mshirts = Product.objects.filter(category="MST").filter(brand=data)
    return render(request, 'app/mshirts.html', {"mshirts":mshirts})
```

Code for Search:

```
def search(request):
    query = request.GET.get('query')
    results = []
    if query:
        query1 = query.title()
        results = Product.objects.filter(title=query1) or Product.objects.filter(brand=query1)
    context = {
        'query1': query1,
        'results': results,
    }
    return render(request, 'app/search.html', context)
```

Code for Product View:

```
class ProductView(View):
    def get(self, request):
        totalitem = 0
        blazers = Product.objects.filter(category="MB")
        topwear = Product.objects.filter(category="TW")
        boy = Product.objects.filter(category="B")
        mshirt = Product.objects.filter(category="MST")
        mshoes = Product.objects.filter(category="MS")
        mhats = Product.objects.filter(category="MH")
        if request.user.is_authenticated:
            totalitem = len(Cart.objects.filter(user=request.user))
        return render(request,"app/home.html",{"blazers":blazers,
                     "topwear":topwear, "boy":boy, "mshirt":mshirt,
                     "mshoes":mshoes, "mhats":mhats, 'totalitem':totalitem })
```

Code for Designer Login:

```
def designer_login(request):
    if request.method == "POST":
        username = request.POST.get("username")
        password = request.POST.get("password")
        designer_code = request.POST.get("designer_code")
        user = authenticate(request, username = username, password = password)
        temp = Designer.objects.filter(designer_code=designer_code)
        if user and temp:
            login(request, user)
            designer = Designer.objects.filter(username=request.user,
                                               designer_code=designer_code)
            if designer:
                return redirect( 'designer_home')
            else:
                messages.error(request,"Invalid Username or Password or Designer code")
        else:
            messages.error(request,"Invalid Username or Password or Designer code")
    return render(request, 'app/designer_login.html')
```

Code for Designer Homepage:

```
def designer_home(request):
    designer = Designer.objects.filter(username=request.user)
    return render(request, 'app/designer_home.html', {'designer':designer})
```

Code for Designer Checkout:

```
@login_required
def designer_checkout(request):
    user = request.user
    designer_id = request.GET.get('designer_id')
    designer = Designer.objects.get(id=designer_id)
    add = Customer.objects.filter(user=user)
    TempDesignerPlaced(user=user, designer=designer).save()
    already_exist = DesignerPlaced.objects.filter(user=user, designer=designer)
    totalitem = 0
    if request.user.is_authenticated:
            totalitem = len(Cart.objects.filter(user=request.user))
    if add:
        if designer:
            return render(request, 'app/designer_checkout.html',
                                    {'add':add,'designer':designer, "totalitem":totalitem})
    else:
        return redirect('profile')
```

Code for Payment done Designer:

```
@login_required
def payment_done_designer(request):
    user = request.user
    custid = request.GET.get('custid')
    customer = Customer.objects.get(id=custid)
    tdp = TempDesignerPlaced.objects.filter(user=user)
    loop_executed = False
    for c in tdp:
        if not loop_executed:
            DesignerPlaced(user=user, customer=customer,designer=c.designer).save()
            TempDesignerPlaced.objects.all().delete()
            loop_executed = True
    return redirect("designer_orders")
```

Code for Buynow Payment:

```
def payment_done_buynow(request):
    user = request.user
    custid = request.GET.get('custid')
    customer = Customer.objects.get(id=custid)
    bcart = BCart.objects.filter(user=user)
    loop_executed = False
    for c in bcart:
        if not loop_executed:
                OrderPlaced(user=user, customer=customer,
                                  product=c.product, quantity=c.quantity).save()
                BCart.objects.all().delete()
                loop_executed = True
    return redirect("orders")
```

Code for Designer Orders:

```
@login_required
```

```
def designer_orders(request):
    op = DesignerPlaced.objects.filter(user=request.user)
    totalitem = 0
    if request.user.is_authenticated:
        totalitem = len(Cart.objects.filter(user=request.user))
    if op:
        return render(request, 'app/designer_orders.html',
                                        {'order_placed':op, "totalitem":totalitem})
    else:
        return render(request,'app/emptyorder.html')
```

Code for Delete Designer:

```
def delete_designer(request):
    if request.method == 'GET':
        des_id = request.GET['des_id']
        c = DesignerPlaced.objects.get(Q(designer=des_id)& Q(user=request.user))
        c.delete()
    return redirect('designer_orders')
```

Code for Designer Order View:

```
@login_required
def view_orders_designer(request):
    designer = Designer.objects.get(username=request.user)
    op = DesignerPlaced.objects.filter(designer=designer)
    if op:
        return render(request, 'app/view_orders_designer.html',
                                        {'order_placed':op})
    else:
        return render(request,'app/designer_emptyorder.html')
```

Code for Designer Status Update:

```
def update_status(request, designer_placed_id):
    designer_placed = get_object_or_404(DesignerPlaced, id=designer_placed_id)
    if request.method == 'POST':
        form = StatusUpdateForm(request.POST)
        if form.is_valid():
            designer_placed.status = form.cleaned_data['status']
            designer_placed.save()
            return redirect('view_orders_designer')
    else:
        form = StatusUpdateForm(initial={'status': designer_placed.status})
    return render(request, 'app/update_status.html', {'form': form})
```

Import files for urls.py file

```
from django.urls import path
from app import views
from django.conf import settings
from django.conf.urls.static import static
from django.contrib.auth import views as auth_views
from .forms import LoginForm, MyPasswordChangeForm, MyPasswordResetForm,
MySetPasswordForm
from .views import update_status
```

urls patterns

```
urlpatterns = [
    path("",views.ProductView.as_view(),name="home"),
    path('product-detail/<int:pk>', views.ProductDetailView.as_view(), name='product-detail'),
    path('add-to-cart/', views.add_to_cart, name='add-to-cart'),
    path('cart/', views.show_cart, name='showcart'),
    path('pluscart/', views.plus_cart, name='pluscart'),
    path('minuscart/', views.minus_cart, name='minuscart'),
    path('removecart/', views.remove_cart, name='removecart'),
    path('delete_item/', views.delete_item, name='delete_item'),
    path('delete_profile/', views.delete_profile, name='delete_profile'),
    path('checkout/', views.checkout, name='checkout'),
    path('buynow_checkout/', views.buynow_checkout, name='buynow_checkout'),
    path('paymentdone/', views.payment_done, name='paymentdone'),
    path('payment_done_buynow/', views.payment_done_buynow, name='payment_done_buynow'),
    path('buy/', views.buy_now, name='buy-now'),
    path('profile/', views.ProfileView.as_view(), name='profile'),
    path('address/', views.address, name='address'),
    path('orders/', views.orders, name='orders'),
    path('accounts/login/', auth_views.LoginView.as_view(next_page='home', template_name='app/login.html', authentication_form=LoginForm), name="login"),
    path('logout/', auth_views.LogoutView.as_view(next_page='login'), name='logout'),
    path('passwordchange/', auth_views.PasswordChangeView.as_view(template_name="app/passwordchange.html", form_class=MyPasswordChangeForm, success_url='/passwordchangedone/'), name ="passwordchange" ),
    path('passwordchangedone/', auth_views.PasswordChangeView.as_view(template_name="app/passwordchangedone.html"),name= "passwordchangedone"),
    path('password-reset/', auth_views.PasswordResetView.as_view(template_name='app/password_reset.html',form_class=MyPasswordResetForm), name='password_reset'),
    path('password-reset/done/', auth_views.PasswordResetDoneView.as_view(template_name='app/password_reset_done.html'), name='password_reset_done'),
    path('password-reset-confirm/<uidb64>/<token>/', auth_views.PasswordResetConfirmView.as_view(template_name='app/password_reset_confirm.html' , form_class=MySetPasswordForm), name='password_reset_confirm'),
    path('password-reset-complete/', auth_views.PasswordResetCompleteView.as_view(template_name='app/password_reset_complete.html'), name='password_reset_complete'),
    path("registration/", views.CustomerRegistrationView.as_view(), name="customerregistration"),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
    path('help/', views.help, name='help'),
    #men
    path('mblazers/', views.mblazers, name='mblazers'),
    path('mblazers/<slug:data>', views.mblazers, name='mblazersdata'),
    path('mshirts/', views.mshirts, name='mshirts'),
    path('mshirts/<slug:data>', views.mshirts, name='mshirtsdata'),
    path('mpants/', views.mpants, name='mpants'),
    path('mpants/<slug:data>', views.mpants, name='mpantsdata'),
    path('mshoes/', views.mshoes, name='mshoes'),
    path('mshoes/<slug:data>', views.mshoes, name='mshoesdata'),
    path('mhats/', views.mhats, name='mhats'),
    path('mhats/<slug:data>', views.mhats, name='mhatsdata'),
    path('meyewear/', views.meyewear, name='meyewear'),
    path('meyewear/<slug:data>', views.meyewear, name='meyeweardata'),
    #women
    path('topwear/', views.topwear, name='topwear'),
    path('topwear/<slug:data>', views.topwear, name='topweardata'),
    path('Chudidar/', views.Chudidar, name='Chudidar'),
    path('Chudidar/<slug:data>', views.Chudidar, name='Chudidardata'),
    path('bottomwear/', views.bottomwear, name='bottomwear'),
    path('bottomwear/<slug:data>', views.bottomwear, name='bottomweardata'),
    path('wshoes/', views.wshoes, name='wshoes'),
    path('wshoes/<slug:data>', views.wshoes, name='wshoesdata'),
    path('jewellery/', views.jewellery, name='jewellery'),
    path('jewellery/<slug:data>', views.jewellery, name='jewellerydata'),
    path('saree/', views.saree, name='saree'),
    path('saree/<slug:data>', views.saree, name='sareedata'),
    #kids
    path('boy/', views.boy, name='boy'),
    path('boy/<slug:data>', views.boy, name='boydata'),
    path('girl/', views.girl, name='girl'),

    path('girl/<slug:data>', views.girl, name='girldata'),

    #search

    path('search/', views.search, name='search'),

    #designer

    path('designer_login/',views.designer_login, name='designer_login'),

    path('designer_home/',views.designer_home, name='designer_home'),

    path('des_passwordchange/', auth_views.PasswordChangeView.as_view(template_name="app/des_passwordchange.html", form_class=MyPasswordChangeForm, success_url='/des_passwordchangedone/'), name ="des_passwordchange" ),

    path('des_passwordchangedone/', auth_views.PasswordChangeView.as_view(template_name="app/des_passwordchangedone.html"),name="des_passwordchangedone"),

    path('designers/', views.designers, name='designers'),

    # path('designers/<slug:data>', views.designers, name='designersdata'),

    path('designer-detail/<int:pk>', views.DesignerDetailView.as_view(), name='designer-detail'),

    path('designer_checkout/', views.designer_checkout, name='designer_checkout'),

    path('payment_done_designer/', views.payment_done_designer, name='payment_done_designer'),

    path('designer_orders/', views.designer_orders, name='designer_orders'),

    path('delete_designer/', views.delete_designer, name='delete_designer'),

    path('view_orders_designer/', views.view_orders_designer, name='view_orders_designer'),

    path('designer_placed/<int:designer_placed_id>/update_status/', update_status, name='update_status'),


] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Import files for models.py file:

```
from django.db import models

from django.contrib.auth.models import User
```

Customer Table:

```
class Customer(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=200)
    phone = models.IntegerField()
    locality = models.CharField(max_length=200)
    city = models.CharField(max_length=50)
    zipcode = models.IntegerField()
    state = models.CharField(choices=STATE_CHOICES, max_length=50)

    def __str__(self):
        return str(self.id)
```

Product table:

```
class Product(models.Model):
    title = models.CharField(max_length=100)
    selling_price = models.FloatField()
    discounted_price = models.FloatField()
    description = models.TextField()
    brand = models.CharField(max_length=100)
    category = models.CharField(choices=CATEGORY_CHOICES,max_length=4)
    product_image = models.ImageField(upload_to="productimg")

    def __str__(self):
        return str(self.id)
```

Cart Table:

```
class Cart(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField(default=1)
    def __str__(self):
        return str(self.id)
```

Order Placed Table:

```
class OrderPlaced(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField(default=1)
    ordered_date = models.DateTimeField(auto_now_add=True)
    status = models.CharField(max_length=50,choices=STATUS_CHOICES,
default="Pending")
```

Designer Table:

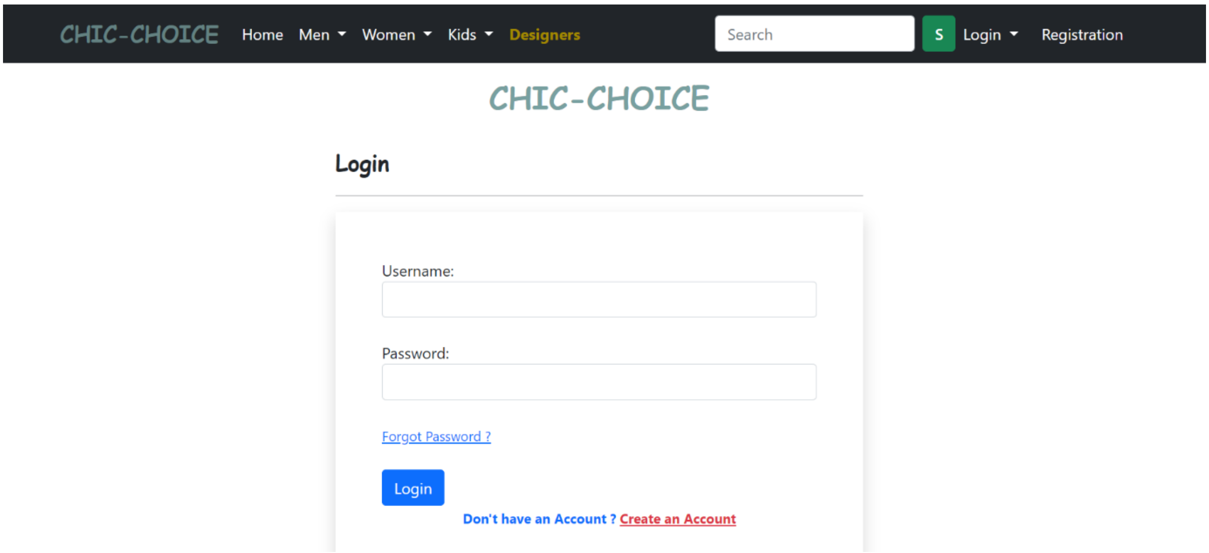
```
class Designer(models.Model):
    username = models.ForeignKey(User, on_delete=models.CASCADE)
    # category = models.CharField(choices=CATEGORY_CHOICES,max_length=4)
    designer_code = models.IntegerField(default=generate_random_number, unique=True)
    full_name = models.CharField(max_length=100, null=True)
    email = models.EmailField(max_length=100, null=True)
    phone = models.CharField(max_length=100, null=True)
    work_exp = models.IntegerField()
    address = models.TextField(max_length=100, null=True)
    zipcode = models.IntegerField()
    state = models.CharField(choices=STATE_CHOICES, max_length=50)
    # des_category =
models.CharField(choices=DES_CATEGORY_CHOICES,max_length=4)
    designer_image = models.ImageField(upload_to="designerimg")
    def __str__(self):
        return str(self.id)
```

Designer Placed Table:

```
class DesignerPlaced(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    designer = models.ForeignKey(Designer, on_delete=models.CASCADE)
    ordered_date = models.DateTimeField(auto_now_add=True)
    status = models.CharField(max_length=50,choices=DESIGNER_STATUS_CHOICES,
default="Pending")
```

USER INTERFACE

Customer Login Page



CHIC-CHOICE

Login

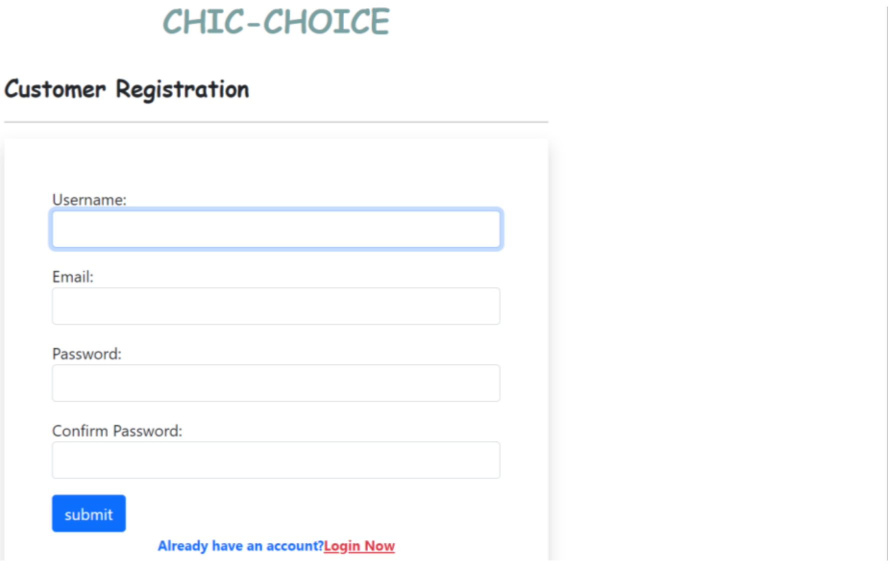
Username:

Password:

Forgot Password ?

Don't have an Account ? Create an Account

Customer Registration Page



CHIC-CHOICE

Customer Registration

Username:

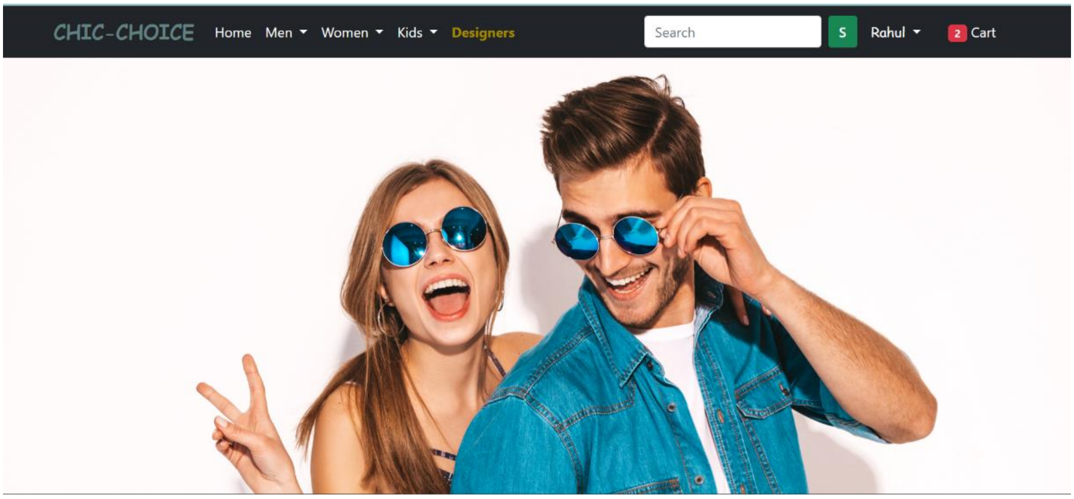
Email:

Password:

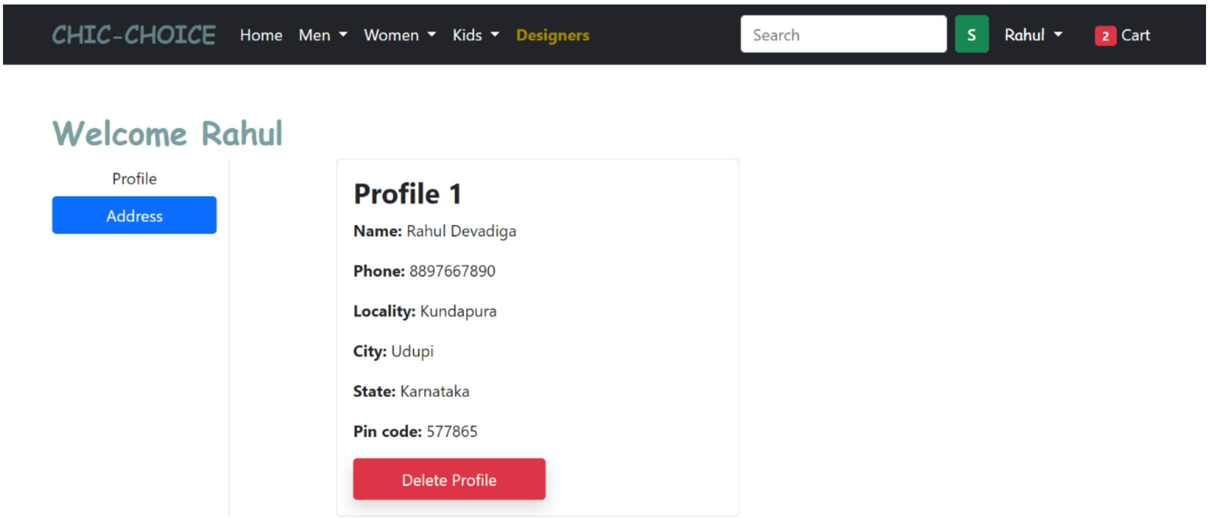
Confirm Password:

Already have an account?Login Now

Customer Home Page



Customer Profile Page



Welcome Rahul

Profile 1

Name: Rahul Devadiga

Phone: 8897667890

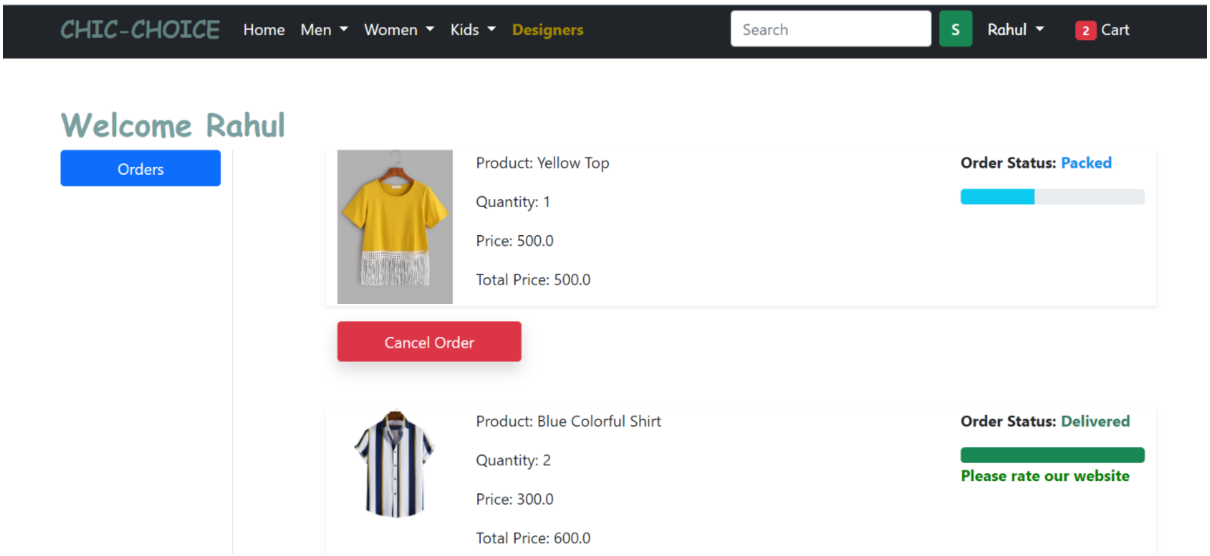
Locality: Kundapura

City: Udupi

State: Karnataka

Pin code: 577865

Customer Orders Page



Welcome Rahul

Product: Yellow Top

Quantity: 1

Price: 500.0

Total Price: 500.0

Order Status: Packed

Product: Blue Colorful Shirt

Quantity: 2

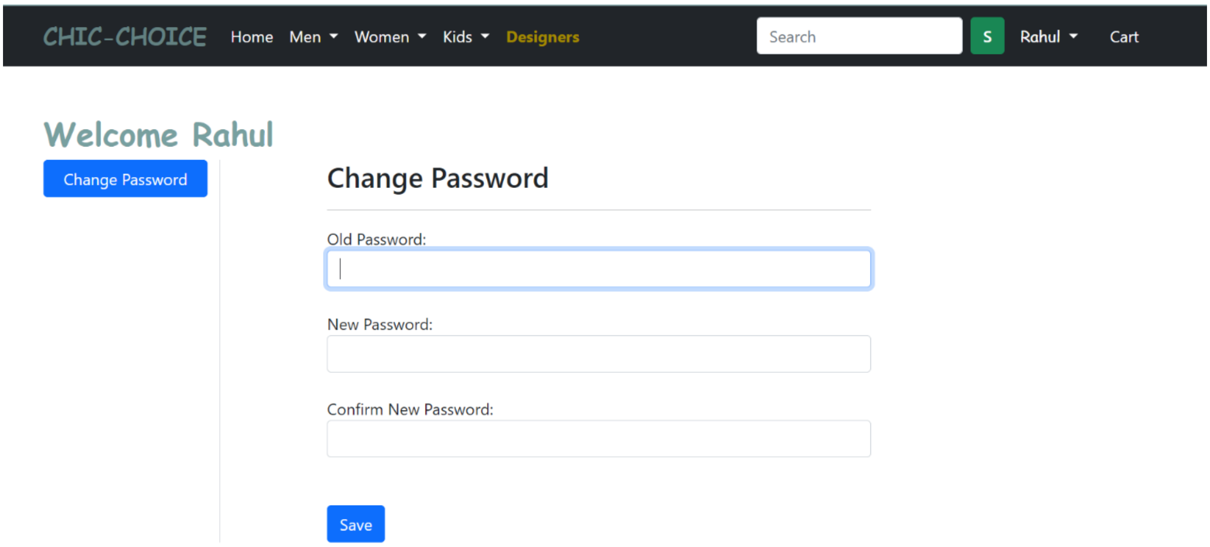
Price: 300.0

Total Price: 600.0

Order Status: Delivered

Please rate our website

Customer Password Change Page



Welcome Rahul

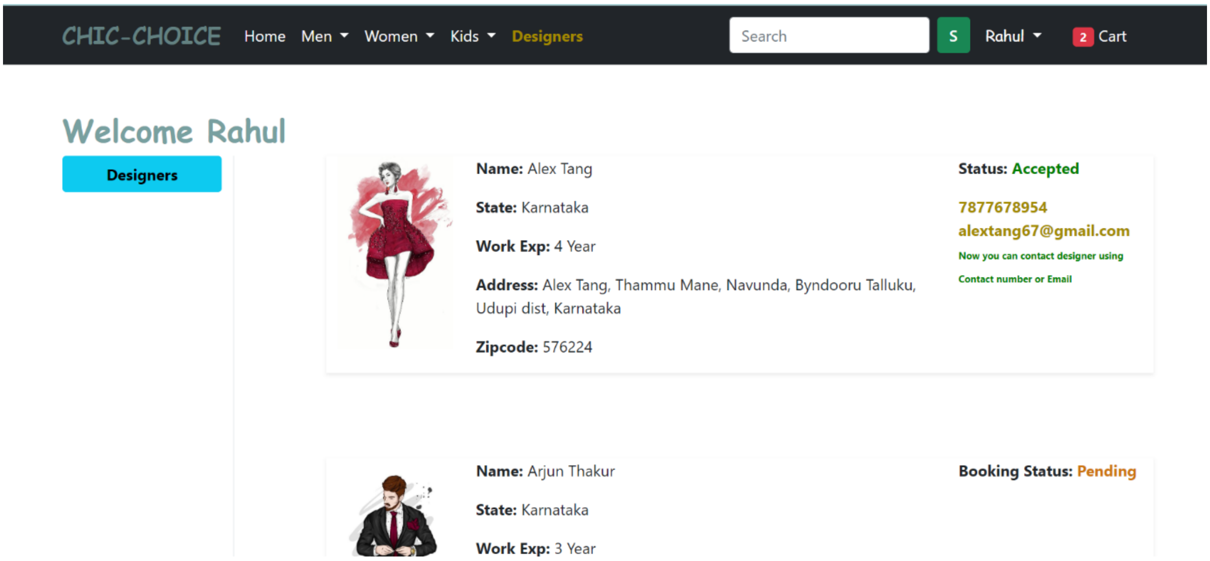
Change Password

Old Password:

New Password:

Confirm New Password:

Customer Designer Booked Page



Welcome Rahul

Designers

Name: Alex Tang

State: Karnataka

Work Exp: 4 Year

Address: Alex Tang, Thammu Mane, Navunda, Byndooru Talluku, Udupi dist, Karnataka

Zipcode: 576224

Status: **Accepted**

7877678954

alextang67@gmail.com

Now you can contact designer using Contact number or Email

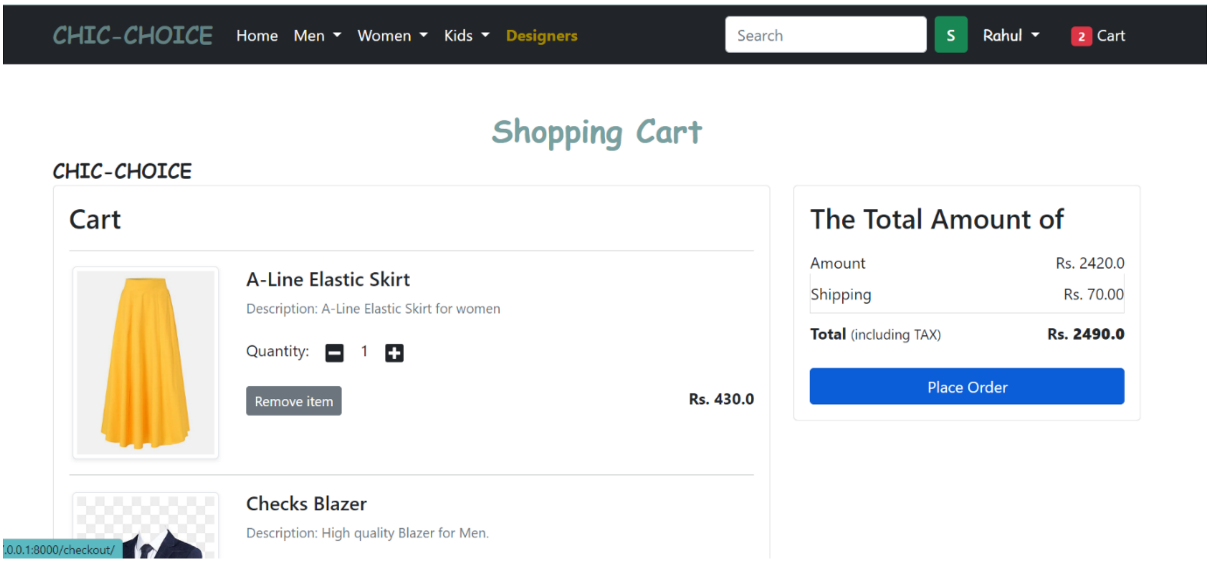
Name: Arjun Thakur

State: Karnataka

Work Exp: 3 Year

Booking Status: **Pending**

Customer Cart Page



Shopping Cart

CHIC-CHOICE

Cart

A-Line Elastic Skirt

Description: A-Line Elastic Skirt for women

Quantity: ⊟ 1 ⊞

Remove item

Rs. 430.0

Checks Blazer

Description: High quality Blazer for Men.

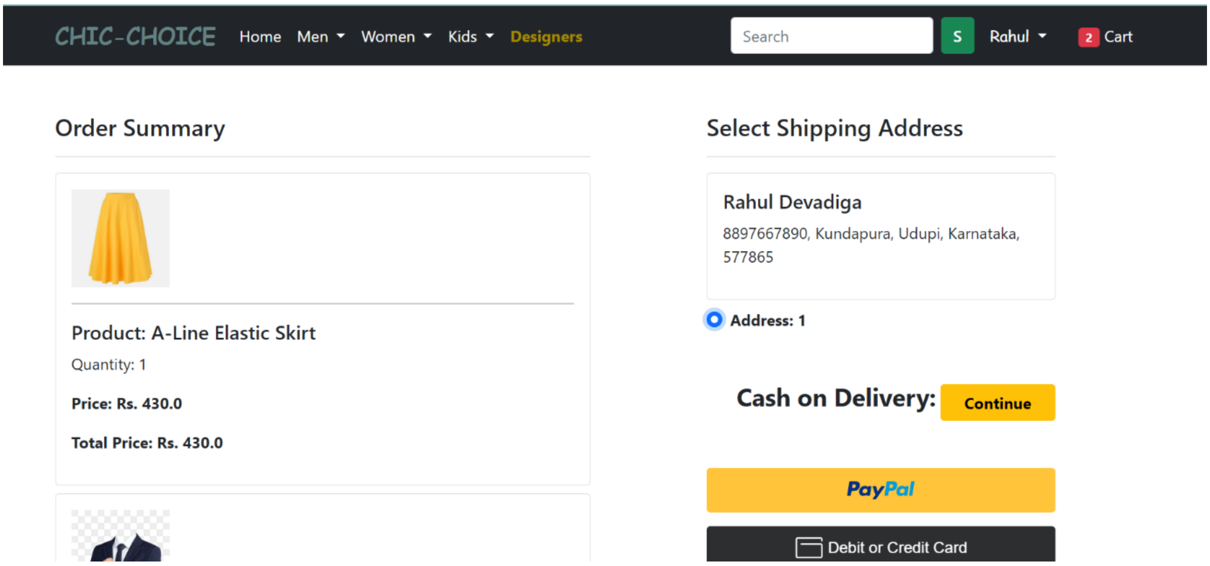
:0.0.1:8000/checkout/

The Total Amount of

Amount	Rs. 2420.0
Shipping	Rs. 70.00
Total (including TAX)	**Rs. 2490.0**

Place Order

Checkout Page



Order Summary

Product: A-Line Elastic Skirt

Quantity: 1

Price: Rs. 430.0

Total Price: Rs. 430.0

Select Shipping Address

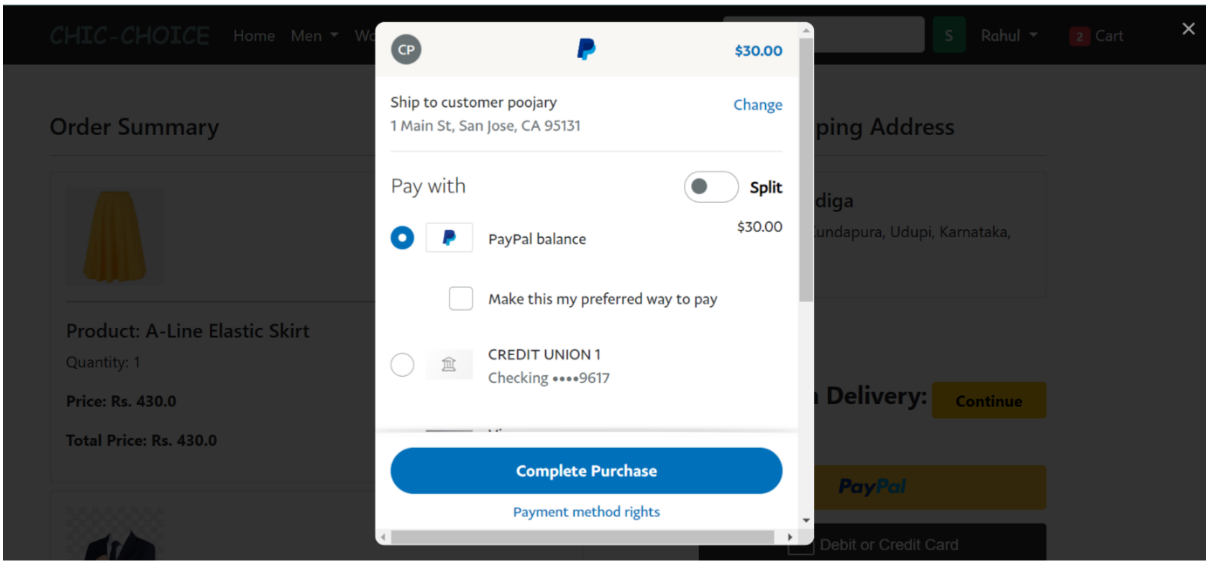
Rahul Devadiga

8897667890, Kundapura, Udupi, Karnataka, 577865

● **Address: 1**

Cash on Delivery: Continue

PayPal Payment



Ship to customer poojary

1 Main St, San Jose, CA 95131

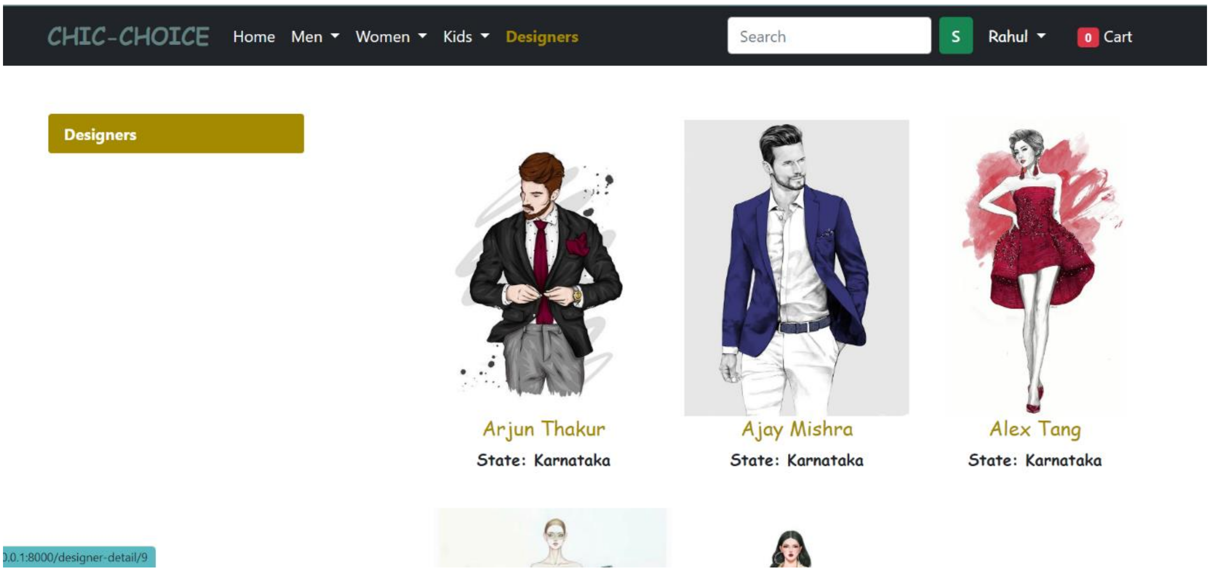
Pay with

● PayPal balance $30.00

□ Make this my preferred way to pay

○ CREDIT UNION 1
Checking ••••9617

Designers for Booking

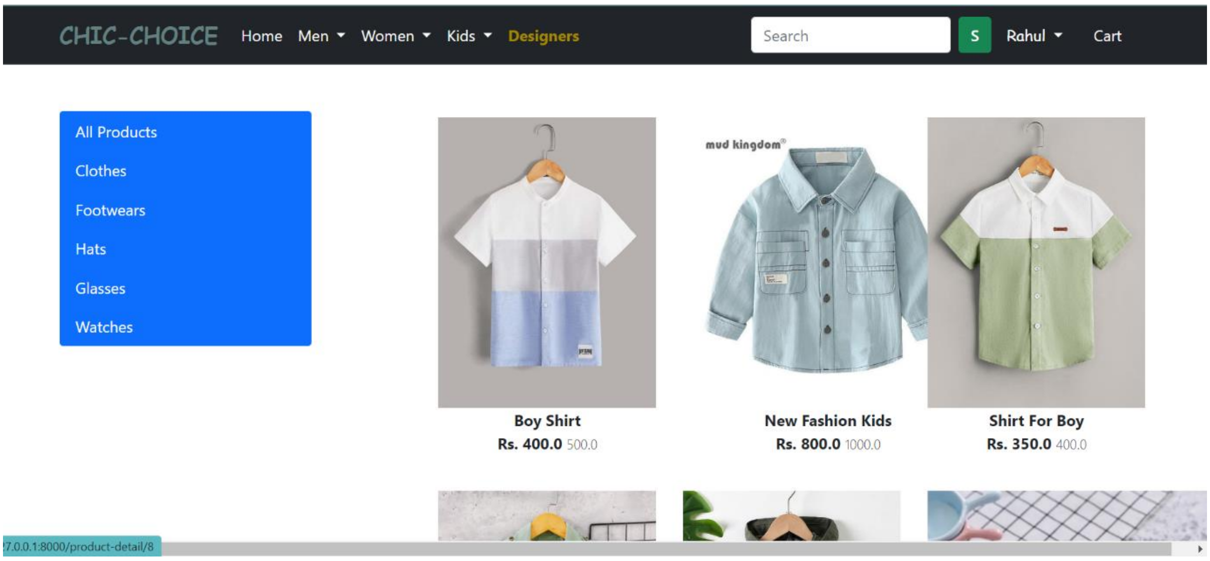


Arjun Thakur
State: Karnataka

Ajay Mishra
State: Karnataka

Alex Tang
State: Karnataka

Products for Buy

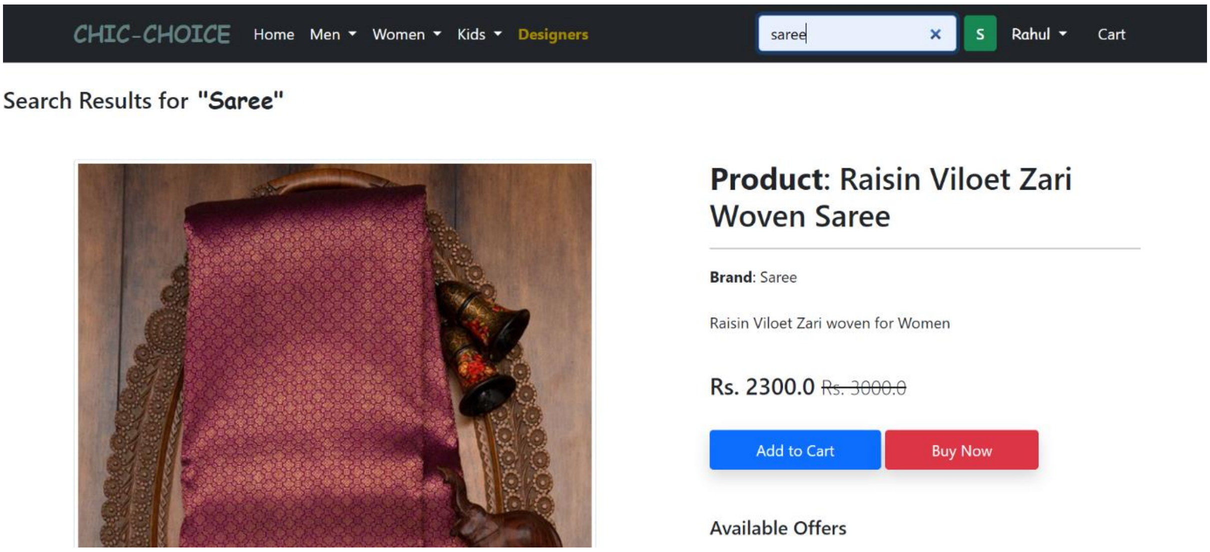


Boy Shirt
Rs. 400.0 500.0

New Fashion Kids
Rs. 800.0 1000.0

Shirt For Boy
Rs. 350.0 400.0

Search Page



Search Results for "Saree"

Product: Raisin Viloet Zari Woven Saree

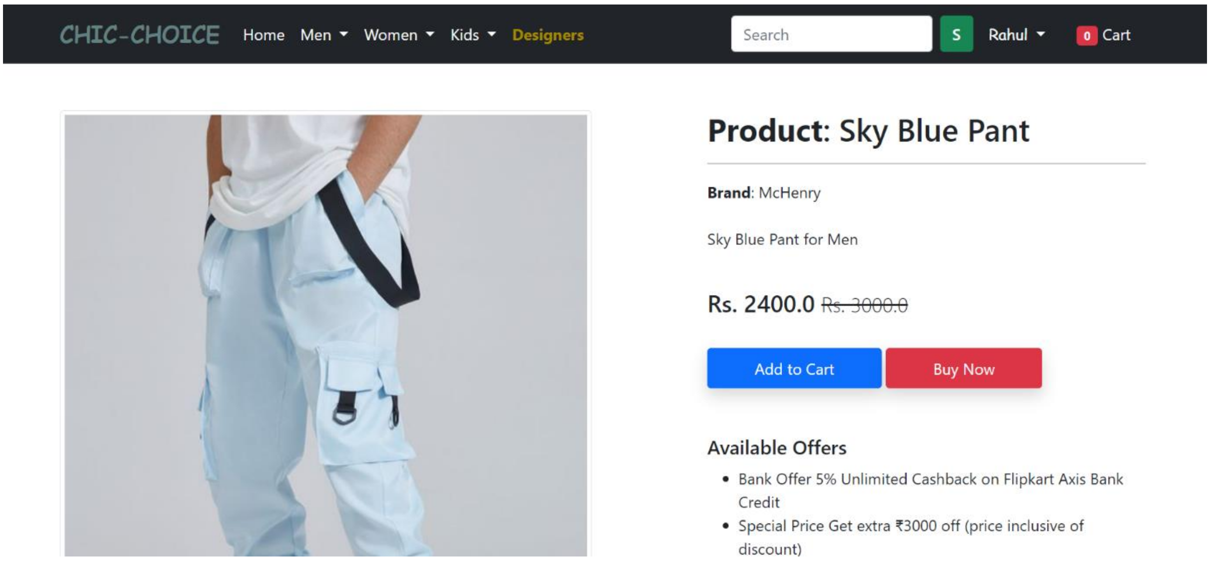
Brand: Saree

Raisin Viloet Zari woven for Women

Rs. 2300.0 ~~Rs. 3000.0~~

Available Offers

Product Details Page



Product: Sky Blue Pant

Brand: McHenry

Sky Blue Pant for Men

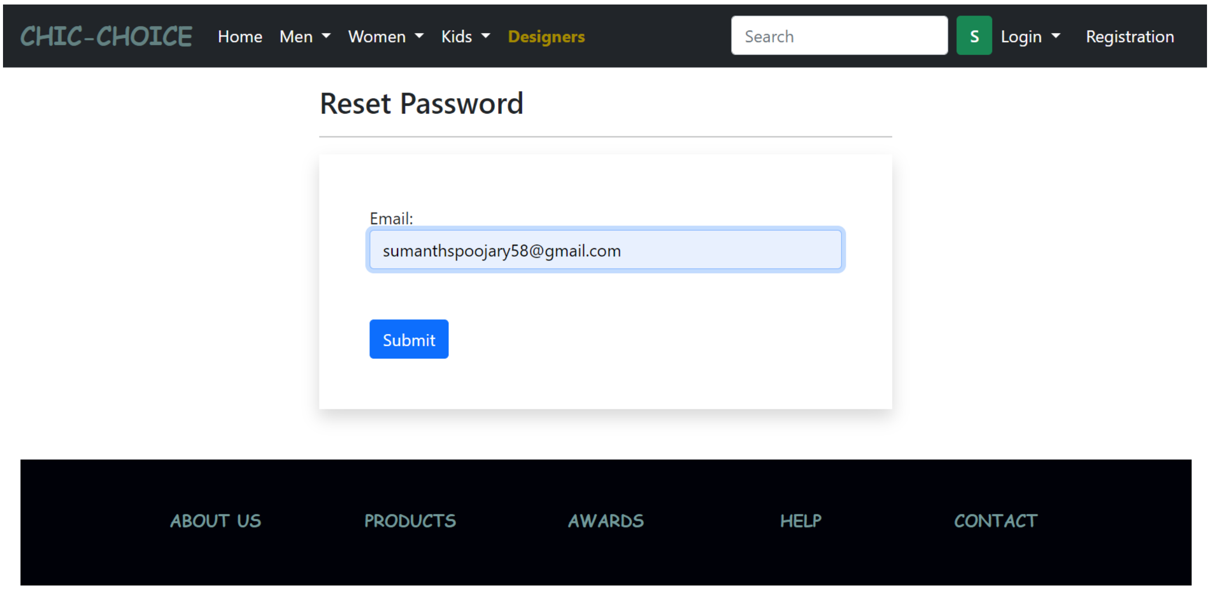
Rs. 2400.0 ~~Rs. 3000.0~~

Available Offers

- Bank Offer 5% Unlimited Cashback on Flipkart Axis Bank Credit
- Special Price Get extra ₹3000 off (price inclusive of discount)

Forgot Password

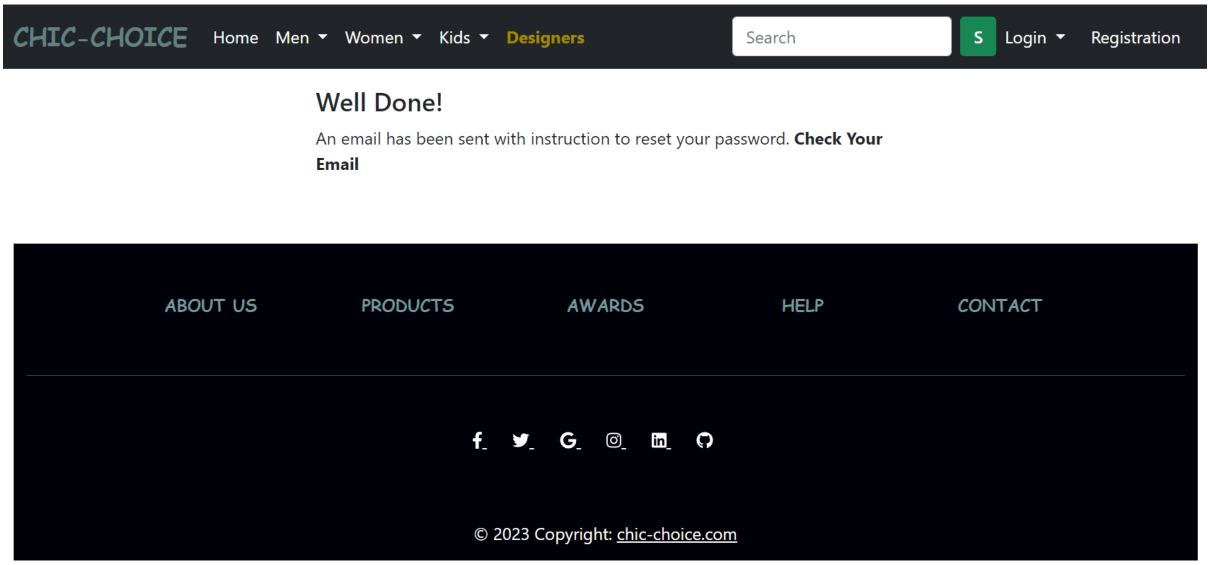
1) Enter your email



Reset Password

Email: sumanthspoojary58@gmail.com

2) Success message

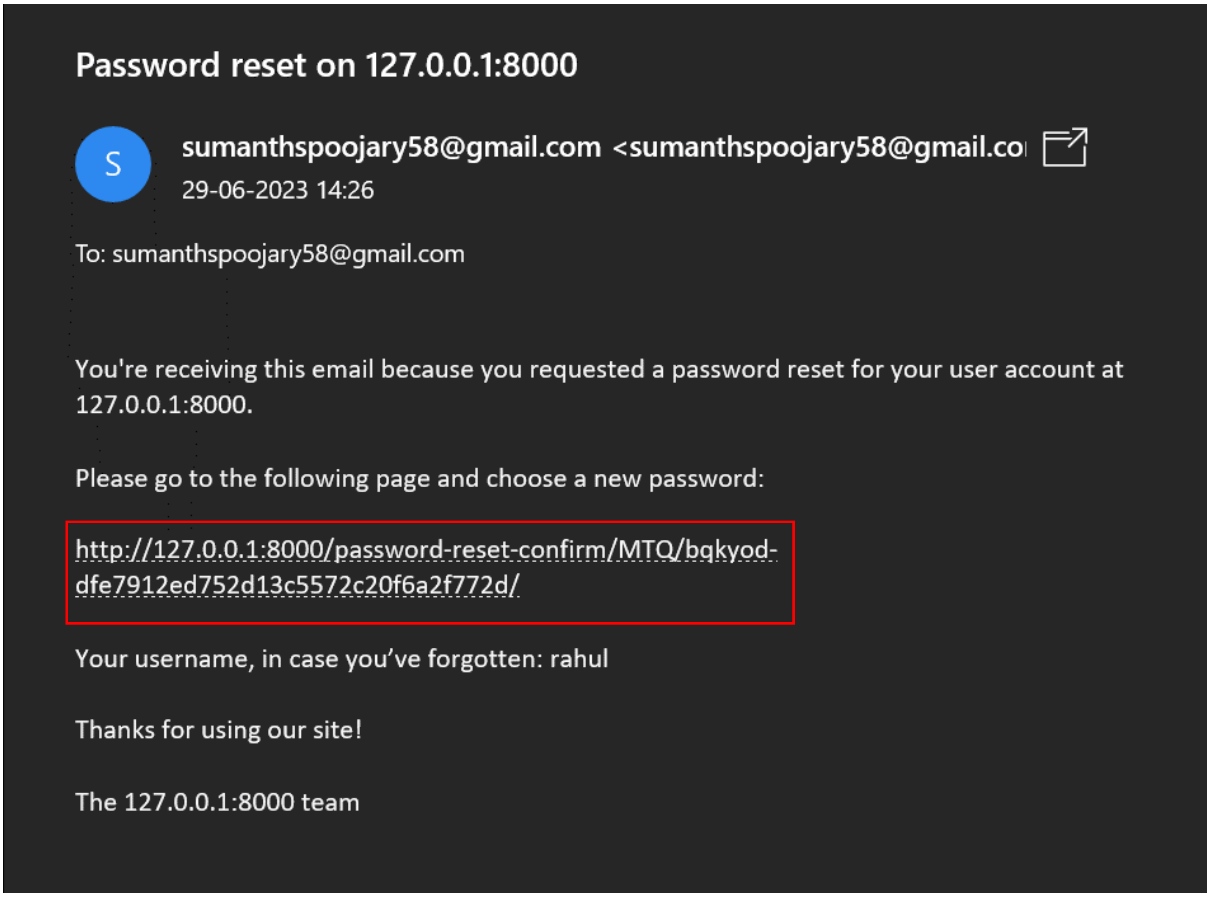


Well Done!

An email has been sent with instruction to reset your password. **Check Your Email**

© 2023 Copyright: chic-choice.com

3) Check your mail and click on the link



Password reset on 127.0.0.1:8000

sumanthspoojary58@gmail.com <sumanthspoojary58@gmail.co

29-06-2023 14:26

To: sumanthspoojary58@gmail.com

You're receiving this email because you requested a password reset for your user account at 127.0.0.1:8000.

Please go to the following page and choose a new password:

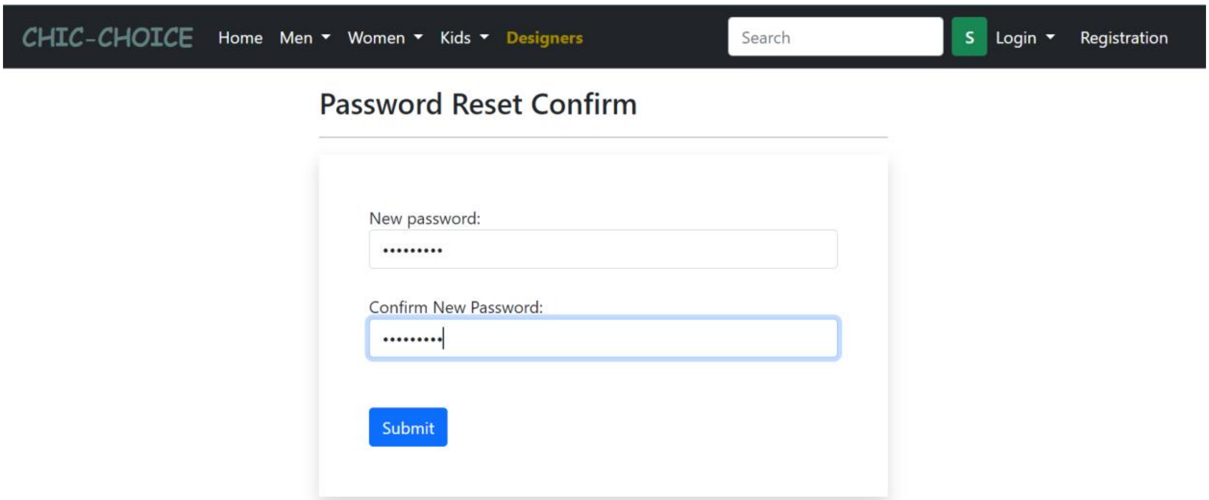
http://127.0.0.1:8000/password-reset-confirm/MTQ/bqkyod-dfe7912ed752d13c5572c20f6a2f772d/

Your username, in case you've forgotten: rahul

Thanks for using our site!

The 127.0.0.1:8000 team

4) Here create your new password

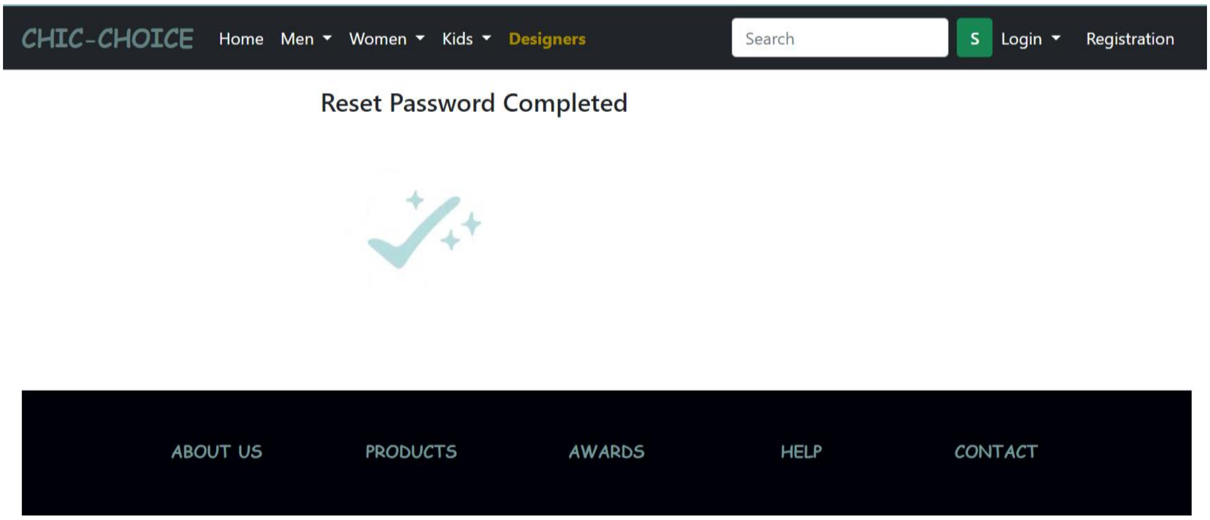


Password Reset Confirm

New password: ••••••••

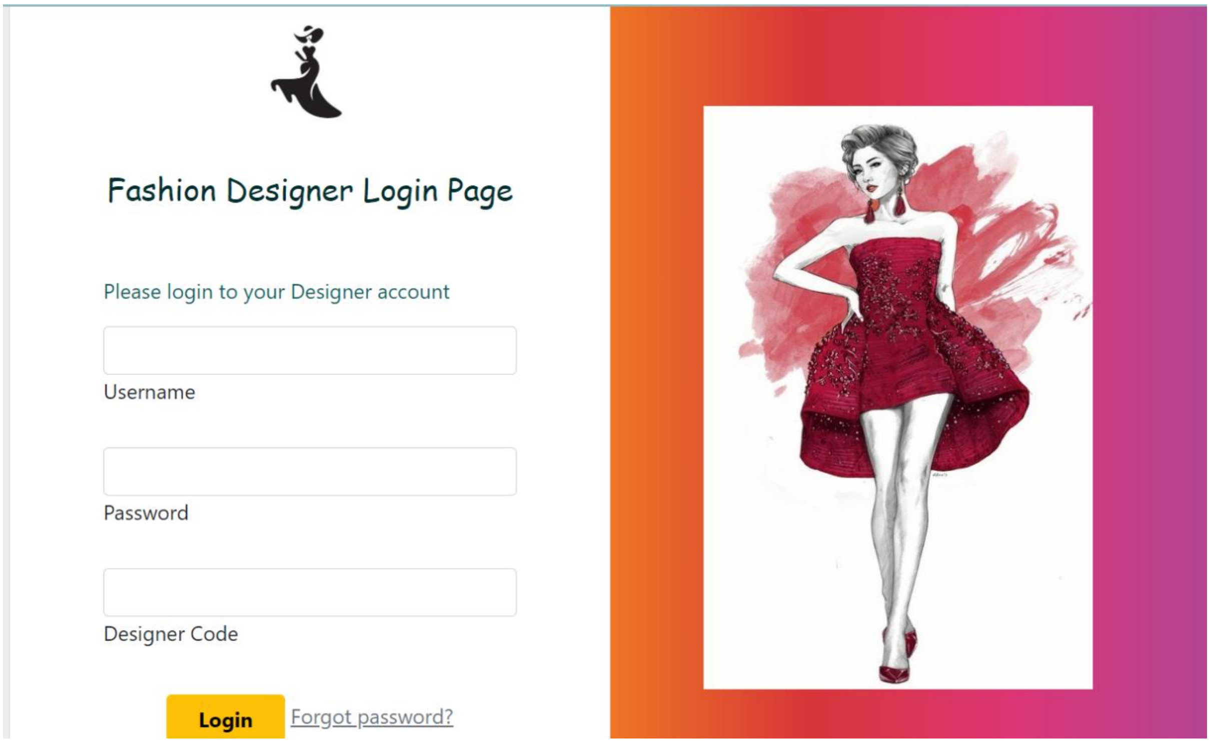
Confirm New Password: ••••••••

5) Password reset complete message



Reset Password Completed

Fashion Designer Login Page



Fashion Designer Login Page

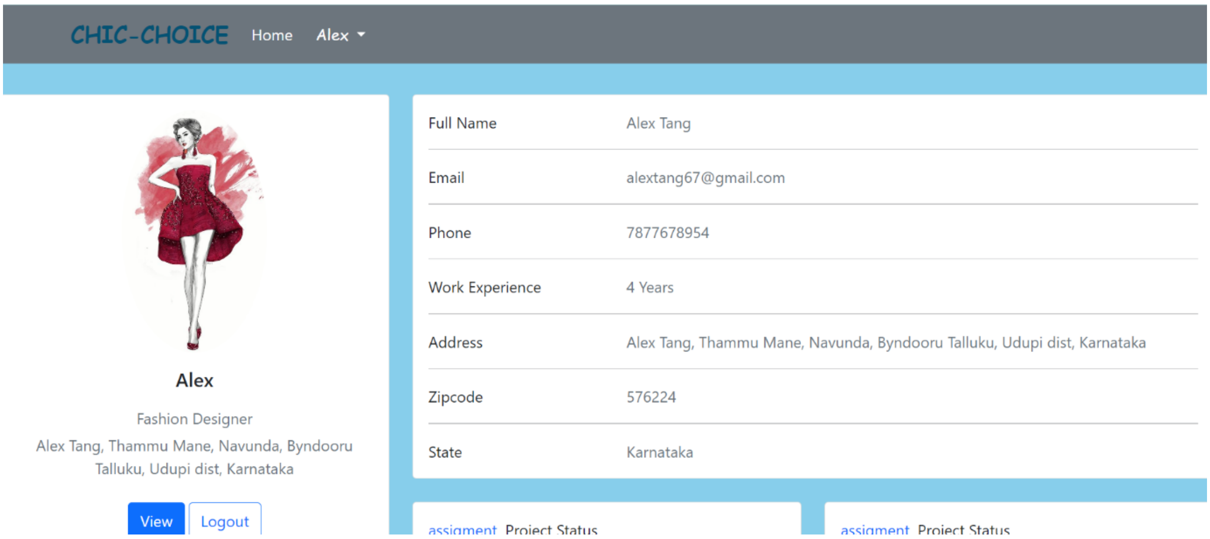
Please login to your Designer account

Username

Password

Designer Code

Fashion Designer Home Page



Alex

Fashion Designer

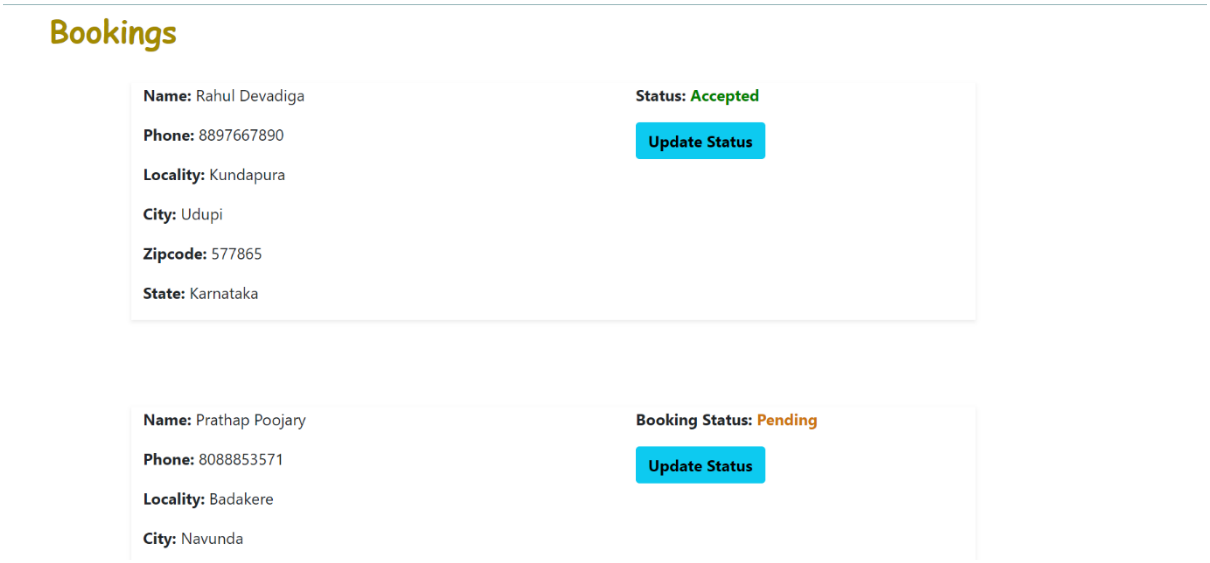
Alex Tang, Thammu Mane, Navunda, Byndooru Talluku, Udupi dist, Karnataka

Full Name	Alex Tang
Email	alextang67@gmail.com
Phone	7877678954
Work Experience	4 Years
Address	Alex Tang, Thammu Mane, Navunda, Byndooru Talluku, Udupi dist, Karnataka
Zipcode	576224
State	Karnataka

assigment Project Status

assigment Project Status

Fashion Designer's Bookings Page



Bookings

Name: Rahul Devadiga

Phone: 8897667890

Locality: Kundapura

City: Udupi

Zipcode: 577865

State: Karnataka

Status: Accepted

Name: Prathap Poojary

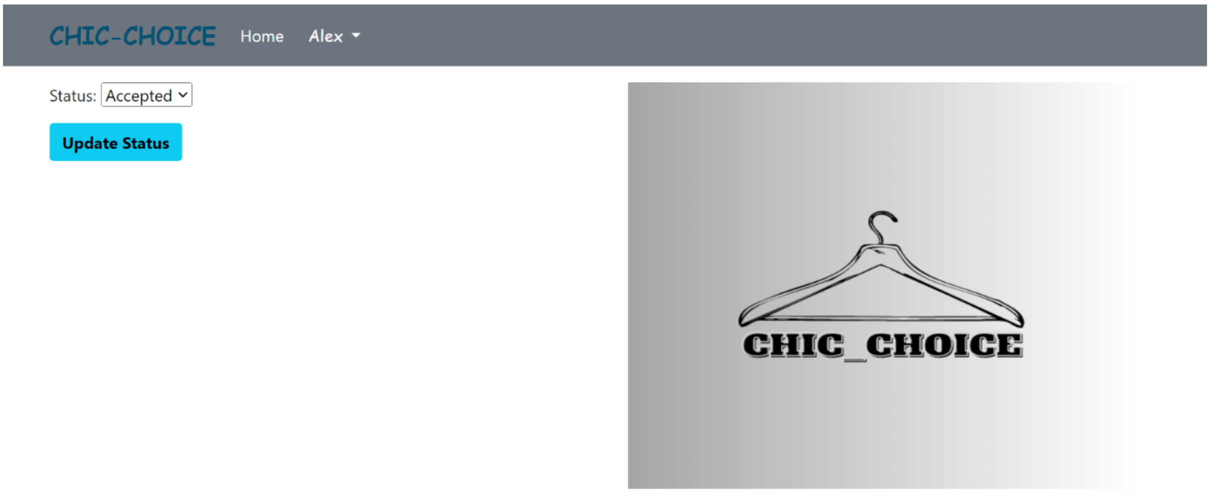
Phone: 8088853571

Locality: Badakere

City: Navunda

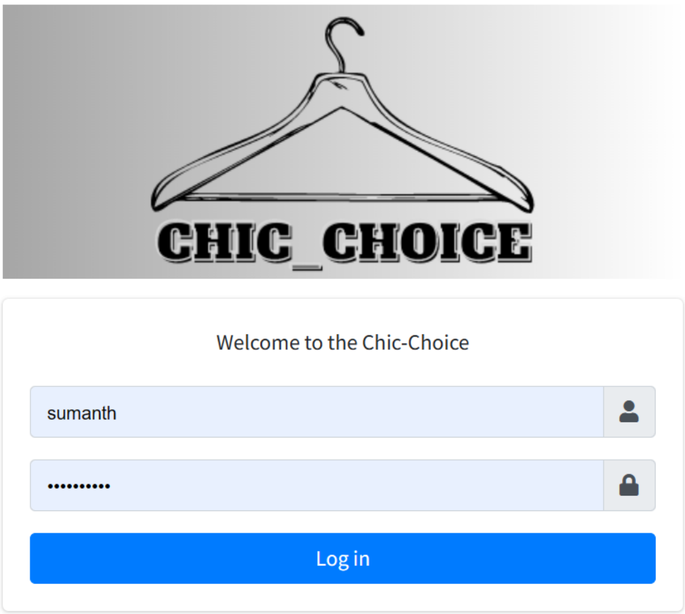
Booking Status: Pending

Fashion Designer’s Status Change Page



Status: Accepted

Admin Login Page

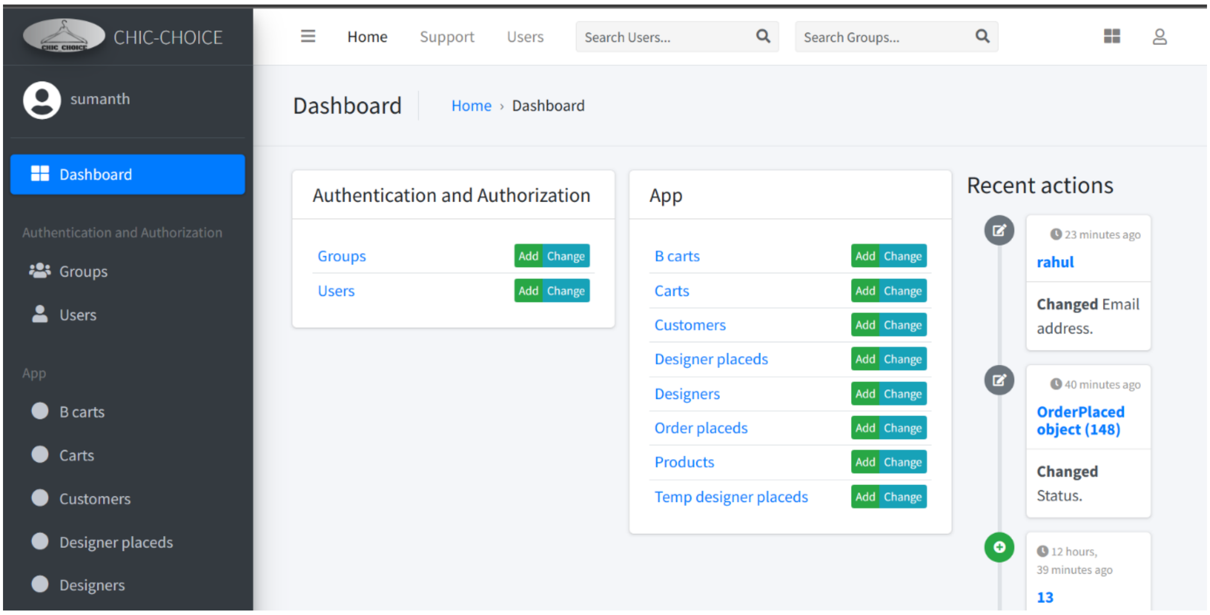


Welcome to the Chic-Choice

sumanth

••••••••••

Admin Home Page



Dashboard | Home › Dashboard

Authentication and Authorization

Groups	Add Change
Users	Add Change

App

B carts	Add Change
Carts	Add Change
Customers	Add Change
Designer placeds	Add Change
Designers	Add Change
Order placeds	Add Change
Products	Add Change
Temp designer placeds	Add Change

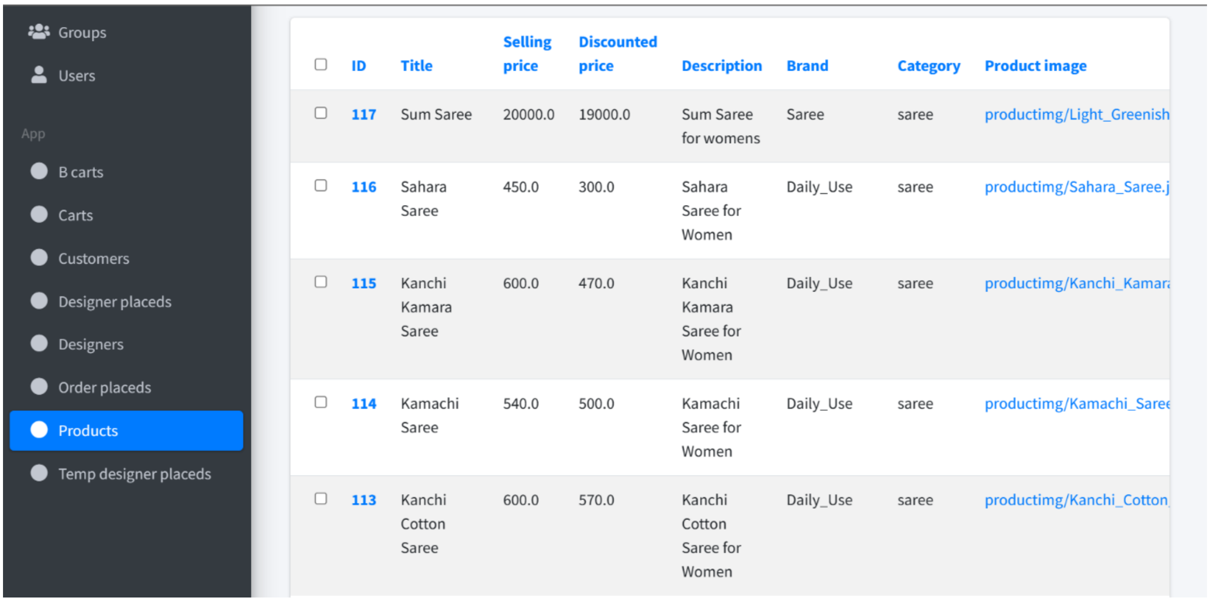
Recent actions

23 minutes ago
rahul
Changed Email address.

40 minutes ago
OrderPlaced object (148)
Changed Status.

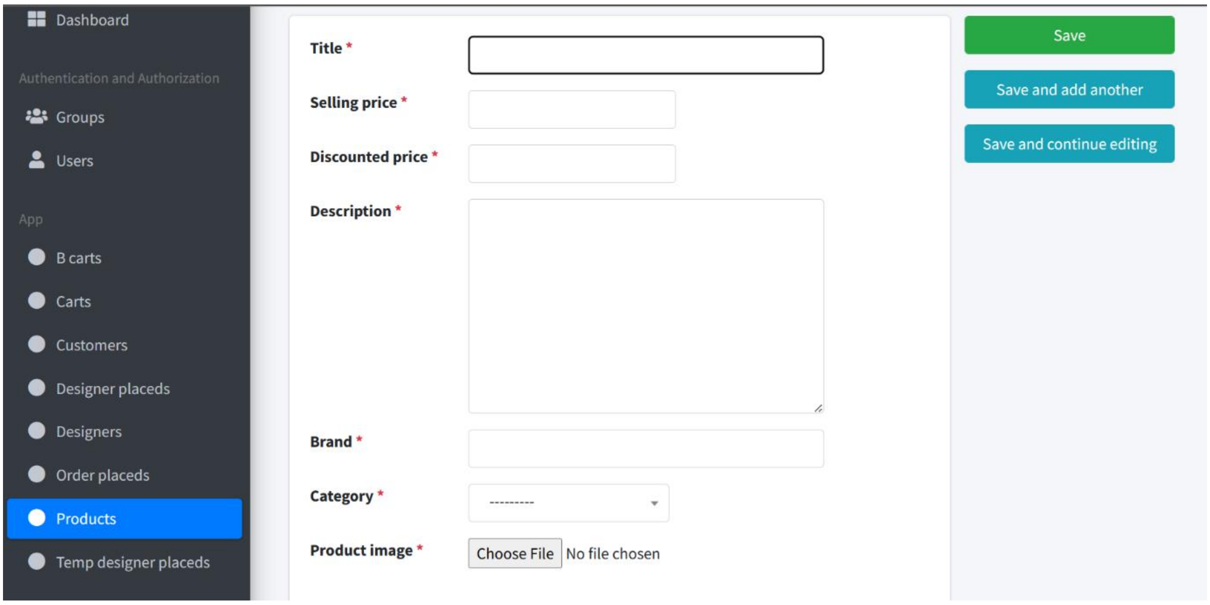
12 hours, 39 minutes ago
13

All Products



□	ID	Title	Selling price	Discounted price	Description	Brand	Category	Product image
□	117	Sum Saree	20000.0	19000.0	Sum Saree for womens	Saree	saree	productimg/Light_Greenish
□	116	Sahara Saree	450.0	300.0	Sahara Saree for Women	Daily_Use	saree	productimg/Sahara_Saree.j
□	115	Kanchi Kamara Saree	600.0	470.0	Kanchi Kamara Saree for Women	Daily_Use	saree	productimg/Kanchi_Kamara
□	114	Kamachi Saree	540.0	500.0	Kamachi Saree for Women	Daily_Use	saree	productimg/Kamachi_Saree
□	113	Kanchi Cotton Saree	600.0	570.0	Kanchi Cotton Saree for Women	Daily_Use	saree	productimg/Kanchi_Cotton_

Add Products Page



Title *

Selling price *

Discounted price *

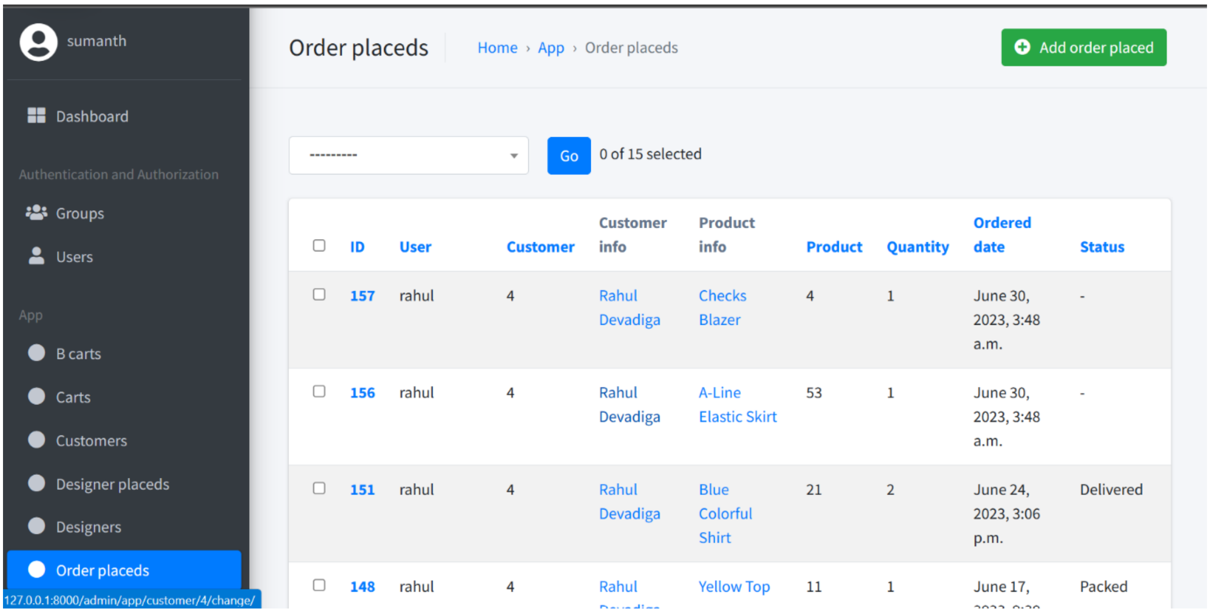
Description *

Brand *

Category * ---------

Product image * No file chosen

Orders Placed Page

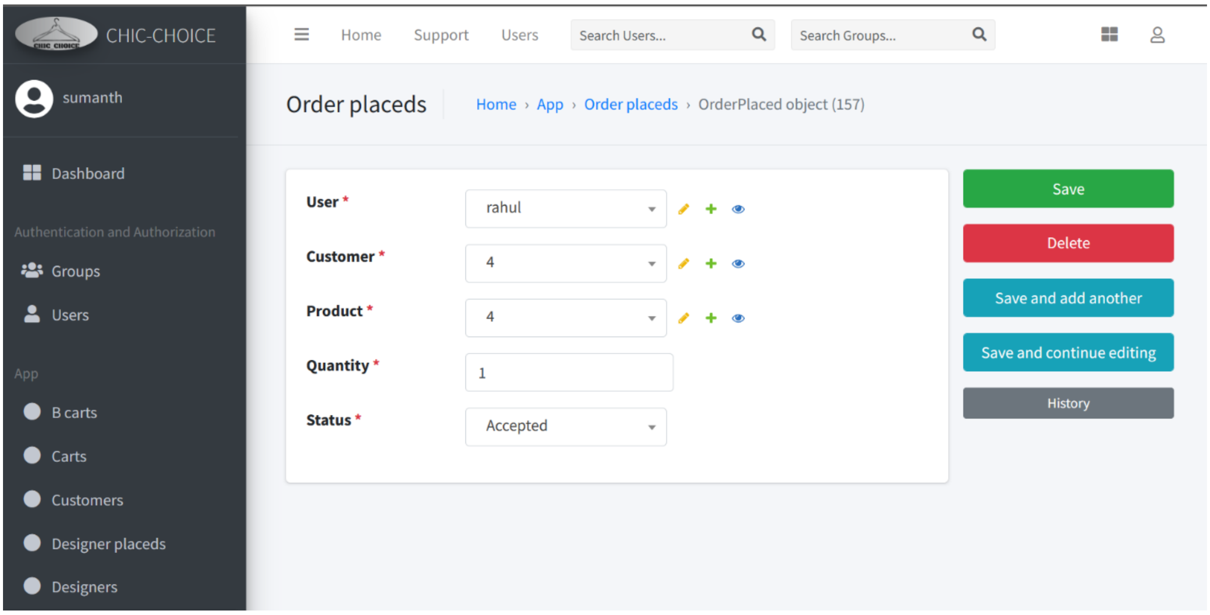


Order placeds

0 of 15 selected

☐	ID	User	Customer	Customer info	Product info	Product	Quantity	Ordered date	Status
☐	157	rahul	4	Rahul Devadiga	Checks Blazer	4	1	June 30, 2023, 3:48 a.m.	-
☐	156	rahul	4	Rahul Devadiga	A-Line Elastic Skirt	53	1	June 30, 2023, 3:48 a.m.	-
☐	151	rahul	4	Rahul Devadiga	Blue Colorful Shirt	21	2	June 24, 2023, 3:06 p.m.	Delivered
☐	148	rahul	4	Rahul	Yellow Top	11	1	June 17,	Packed

Managing Placed Orders



Order placeds | Home › App › Order placeds › OrderPlaced object (157)

User * rahul

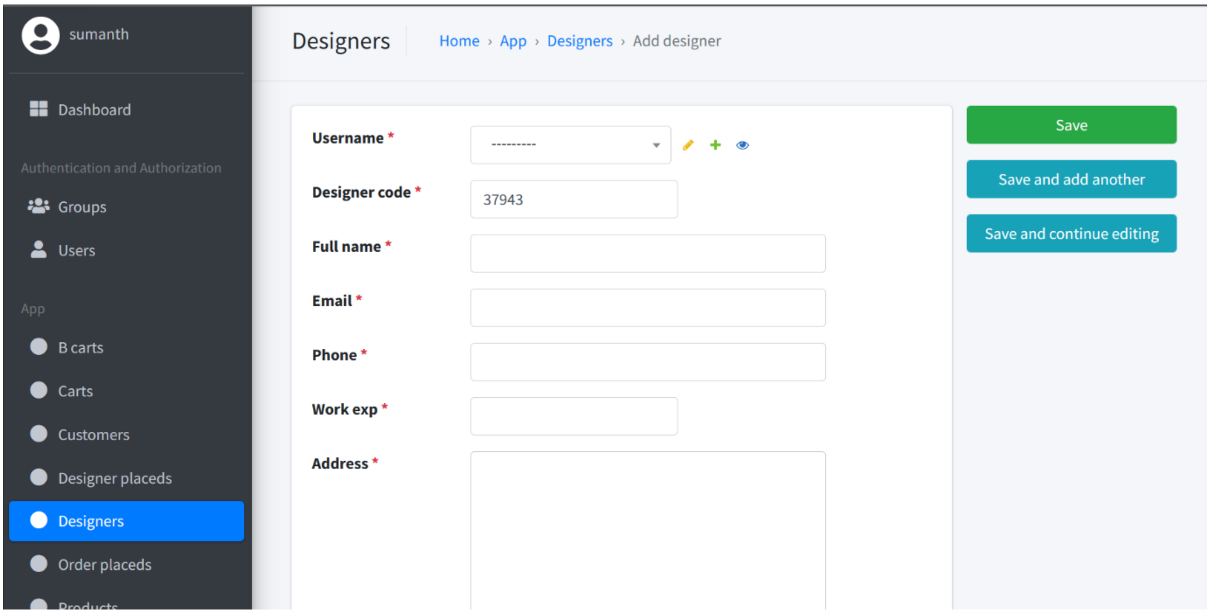
Customer * 4

Product * 4

Quantity * 1

Status * Accepted

Adding Designers



Designers | Home › App › Designers › Add designer

Username * ---------

Designer code * 37943

Full name *

Email *

Phone *

Work exp *

Address *

TESTING

8.1 Introduction

Testing is the major quantity control measure used during software development. It is used to remove errors that could have occurred during any of the phases like requirement analysis, designing, coding, etc. Testing plays a very critical role for quality assurance and for ensuring the reliability of the software.

8.2 Levels of testing:

Testing is done in different levels which includes the following

- Unit testing
- Integration testing
- System testing
- Acceptance testing

8.2.1 Unit testing

In unit testing, each module gets tested during the coding phase itself. This was done in our software development so as to produce the required output.

8.2.2 Integration testing

In integration testing, different modules are integrated into sub system. In this software too, different forms were linked to each other and the working was tested together.

8.2.3 System testing

In system testing, the whole system is put together and tested to see if it meets with all requirements specified by the user.

8.2.4 Acceptance testing

It is performed to demonstrate to the client on real life data of the client, the operation of the system.

8.3 Test cases:

A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

The test cases for this software are as follows:

Test Case id	01
Title	Login
Purpose	Allow registered users to login to the system so that the person can go through the process of ordering products
Test data	Username and Password
Steps	1. Read Username 2. Read Password 3. Click on the login button
Expected output	Valid Output: User logs in to the system and respective page will be displayed. Invalid Output: Login unsuccessful message will be displayed.

Test Case id	02
Title	Registration
Purpose	Allows the user to create a new account for further access.
Test data	Full Name, Email, Password, Confirm Password.
Steps	1. Read Full Name 2. Read Email address 3. Read password 4. Read Confirm password 5. Click on the register button
Expected output	Valid Output: All the user details will be stored to the database and user can login to the system with username and password. Invalid Output:

	User will not be registered. Respective error message will be displayed.

Test Case id	03
Title	Profile
Purpose	Allows the registered user to add or delete his/her profile and address.
Test data	Full Name, Phone No., Locality, City, State, Zip code
Steps	1. Read Full Name 2. Read Phone No. 3. Read Locality 4. Read City 5. Read State 6. Read Zip code 7. Click on Submit button
Expected output	Valid Output: If the user entered profile details is correct then Updated details will be stored in the database, success message will be displayed. Invalid Output: If the profile details is incorrect then details cannot be stored in database, error message will be displayed.

Test Case id	04
Title	Order
Purpose	Allows the user to order a product.

Test data	Address, Quantity, Payment mode.
Steps	1. Read Address 2. Read Payment Mode 3. Click on the Continue button.
Expected output	Valid Output: If Order details are stored in the database, then order successful message will be displayed. Invalid Output: If the user not select the payment Mode then ordering details are not stored to database and error message will displayed.

Test Case id	05
Title	Add Product
Purpose	Allows the admin to add products.
Test data	Product Title, Selling price, Discounted price, Description, Brand, Category, product image.
Steps	1. Read Product Title, 2. Read Selling price, 3. Read Discounted price, 4. Read Product Description, 5. Read Product Brand, 6. Read Category, 7. Read Product image. 8. Click on save button.

Expected output	Valid Output: Product added to the database and Product added successfully message will be displayed. Invalid Output: If image path is not available then Something wrong on server message will be displayed.

Test Case id	06
Title	Add Designer
Purpose	Allows the admin to add Designer.
Test data	Username, Designer code, Full name, Email, Phone number, Work experience, Address, Zip code, State, Designer image.
Steps	1. Read Username, 2. Read Designer code, 3. Read Full name, 4. Read Email, 5. Read phone number, 6. Read Work experience, 7. Read Address, 8. Read Zip code, 9. Read State, 10. Read Designer image 11. Click on Save button.
Expected output	Valid Output: Designer added to the database and Designer added successfully message will be displayed. Invalid Output:

	If Designer is not available then something wrong on server error message will be displayed.

Test Case id	07
Title	Logout
Purpose	Allow registered users to logout from the system.
Steps	1. Click on the logout button
Expected output	Valid Output: User really wants to logout from site then logout successfully message will be displayed. Invalid Output: In case user don't want to logout then come back by pressing close button.

Test Case id	08
Title	Forgot Password
Purpose	Allows the user to update password when they forgot.
Test data	Email id, New password, Confirm password
Steps	1. Read Email Id. 2. Read new password 3. Read confirm password 4. Click on submit button
Expected output	Valid Output: Password will be updated. User can login with the new password. Invalid Output:

	Email Id does not match with existed Email Id and the error message will be displayed.

Test Case id	09
Title	Change Password
Purpose	Allows the user to change password when they needed.
Test data	Old password, New password, Confirm new password
Steps	1. Read Old password. 2. Read new password 3. Read confirm new password 4. Click on save button
Expected output	Valid Output: Password will be updated. User can login with the new password. Invalid Output: When password is incorrect then the error message will be displayed.

Test Case id	08
Title	Add to Cart
Purpose	Allows the user to Add the product to the cart.
Steps	1. Click on Add to cart button
Expected output	Valid Output:

	Password will be updated. User can login with the new password. Invalid Output: Email Id does not match with existed Email Id and the error message will be displayed.

FUTURE SCOPE

- We can create android application for this project.

- **Augmented reality**: With the help of augmented reality, customers can try on clothes virtually and get a better sense of how they will look before making a purchase. This technology could also be used to show customers how clothes will fit on different body types.

- **Social media integration**: Social media has become an important part of the fashion industry, and e-commerce websites can take advantage of this by integrating social media platforms into their websites. This could include allowing customers to share their purchases on social media, or using social media influencers to promote products.

- **Drone Delivery:** Another popular online shopping trends 2023 is the use of drones for delivery. This is a popular trend that can be popular while businesses try to make contactless deliveries.

BIBLIOGRAPHY

References

➢ RamezElmasri and ShamkanthB.Navate, Fundamentals of

Database Systems, 7th Edition, Pearson Education .

➢ Pankaj Jalote, An Integrated Approach to Software

Engineering, 3 rd Edition, Narosa Publishing House.

➢ Stack overflow.com

➢ docs.djangoproject.com

➢ www.w3 schools.com