VOICE BASED SHOPPING IN SUPERMARKET

A Mini Project Report

Submitted in partial fulfilment of the requirements

For the award of the degree of

MASTER OF COMPUTER APPLICATIONS

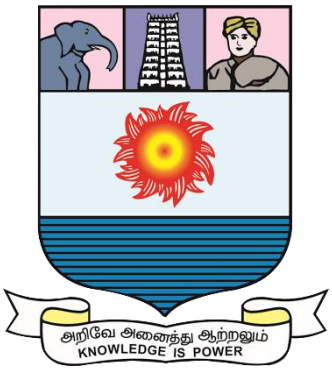
Submitted by

VELMURUGAN. M

(Reg. No: 20234012404226)

Under the Guidance of

Dr. K. ARAVIND KUMAR ME., Ph.D.,
Assistant Professor



Department of Computer Science and Engineering

Manonmaniam Sundaranar University

Tirunelveli 627012

November 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MANONMANIAM SUNDARANAR UNIVERSITY

TIRUNELVLI - 12

BONAFIDE CERTIFICATE

This is to certify that the report entitled **"VOICE BASAED SHOPPING IN SUPERMARKET"** submitted in partial fulfilment of the degree in MASTER OF COMPUTER APPLICATIONS to the Department of Computer Science & Engineering, Manonmaniam Sundaranar University, done by **VELMURUGAN.M** Register No. **20234012404226** is an authentic work carried out during the academic year 2024-25.

GUIDE **HEAD OF THE DEPARTMENT**

Submitted for the Viva – Voce Examination on: _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

I thank Lord Almighty, who blessed me with wonderful faculties, friends, and family members whose love and encouragement have given new significance to my accomplishment. I express my sincere thanks with love to my parents, who are ardent cheer leader and support throughout the project.

I whole-hearty thank **Dr.R.S.RAJESH M.E., Ph.D.,** Professor, Head of the Department of Computer Science and Engineering, Manonmaniam Sundaranar University for providing constructive criticism and suggestions for the successful completion of this project.

I thank **Dr.G.MURUGESHWARI M.Tech.,Ph.D.,** Project Coordinator, Associate Professor, Department of Computer Science and Engineering. Manonmaniam Sundaranar University, Tirunelveli.

I extend my gratitude to my guide **Dr.K.ARAVIND KUMAR M.E., Ph.D.,** Assistant Professor, Department of Computer Science and Engineering. Manonmaniam Sundaranar University for his valuable guidance and motivation throughout my project.

I thank all staff members of department for their kind advice and pleasing co-ordination throughout the course.

I also thank my friends who support with real encouragement and suggestions in each and every phase of my project

DECLARATION

I hereby declare that the project work entitles **"VOICE BASED SHOPPING IN SUPERMARKET"** submitted to Manonmaniam Sundaranar University. Tirunelveli in partial fulfillment of the requirement for the award of the degree of Master of Computer Applications is a record of original project work done during the academic year 2024-2025 under the supervision and guidance of **Dr.K.ARAVIND KUMAR M.E., Ph.D.,** Assistant Professor, Department of Computer Science and Engineering, Manonmaniam Sundarnar University, Tirunelveli.

[VELMURUGAN M]

TABLE OF CONTENT

ABSTRACT

The Voice-Based Shopping System is designed to assist visually impaired individuals in shopping independently by using voice commands for tasks such as adding products to the cart, managing the cart, and generating bills. The system also supports voice-based registration and login, ensuring a fully accessible shopping experience. For administrators, the system offers streamlined inventory management, including features to manually or bulk upload products, receive automated email notifications when product stock is low, and monitor product availability. Additionally, administrators can view statistical summaries, such as low-stock items and transaction details, through interactive charts on the dashboard. This system enhances accessibility for users while improving operational efficiency for administrators, making it a valuable tool for inclusive and effective shopping management.

CHAPTER 1

INTRODUCTION

Visually impaired individuals face significant challenges when shopping in supermarkets, including difficulties in locating products and reading product labels. Traditional shopping methods often require assistance from others, making them dependent on external support, which limits their independence. To address this issue, the **Voice-Based Shopping System** designed in ReactJs and NodeJs aims to provide a voice-interactive shopping platform specifically designed for visually impaired users.

This system empowers users to shop independently by allowing them to add products to their cart using voice commands, manage their cart, and generate a bill, all through voice interaction.. Once the products are added to the cart, the system generates a bill that the user can use to proceed with checkout and complete their purchase.

For administrators, the system simplifies inventory management by allowing them to add or update products manually or in bulk through Excel file uploads. And the dashboard contains charts for display the low quantity products and the transaction details counts. It also helps track product availability, notifies via email when stock is low, and provides tools for managing user details and viewing transaction histories. This solution not only enhances accessibility for visually impaired users but also improves administrative efficiency in store management.

CHAPTER 2

2.SYSTEM REQUIREMENTS

2.1.HARDWARE REQUIREMENTS:

Processor : intel i3

Ram : 8 GB

Hard disk drive : 500 GB

2.2.SOFTWARE REQUIREMENTS:

Operating System : 64-bit OS

Front end : Html , css and ReactJS

Middleware : NodeJS

Back end : MongoDB

2.3 SOFTWARE DESCRIPTION

ReactJS:

ReactJS is a JavaScript library used for building user interfaces, especially for web applications. It allows developers to create reusable components that make it easier to design dynamic, interactive, and responsive web pages. ReactJS is commonly used for creating the front-end part of applications, where users interact with the system.

NodeJS:

NodeJS is a runtime environment that allows JavaScript to be used for building server-side applications. This means it helps developers create the back-end of applications, like handling user requests, managing data, and communicating with databases. NodeJS is fast and efficient, making it a popular choice for developing scalable web applications.

MongoDB:

MongoDB is a NoSQL database used for storing and managing data. Unlike traditional databases, MongoDB stores data in flexible, JSON-like documents, making it easier to work with large and varied types of information. It’s widely used for applications that need to handle lots of data and change quickly, like shopping systems.

CHAPTER 3

3.SYSTEM ANALYSIS

3.1.EXISTING SYSTEM

- Visually impaired peoples face many challenges while shopping in supermarkets, including difficulty in locating products, reading product labels.
- In traditional shopping, visually challenged people need someone to guide them through the store, visit each section, and help them pick out products.
- This makes them less independent and more dependent on others for help.
- For administrators, to managing inventory with the ability to add products individually its take lots of time. And also monitors product availability for each one its too difficulty.

3.2.PROPOSED SYSTEM

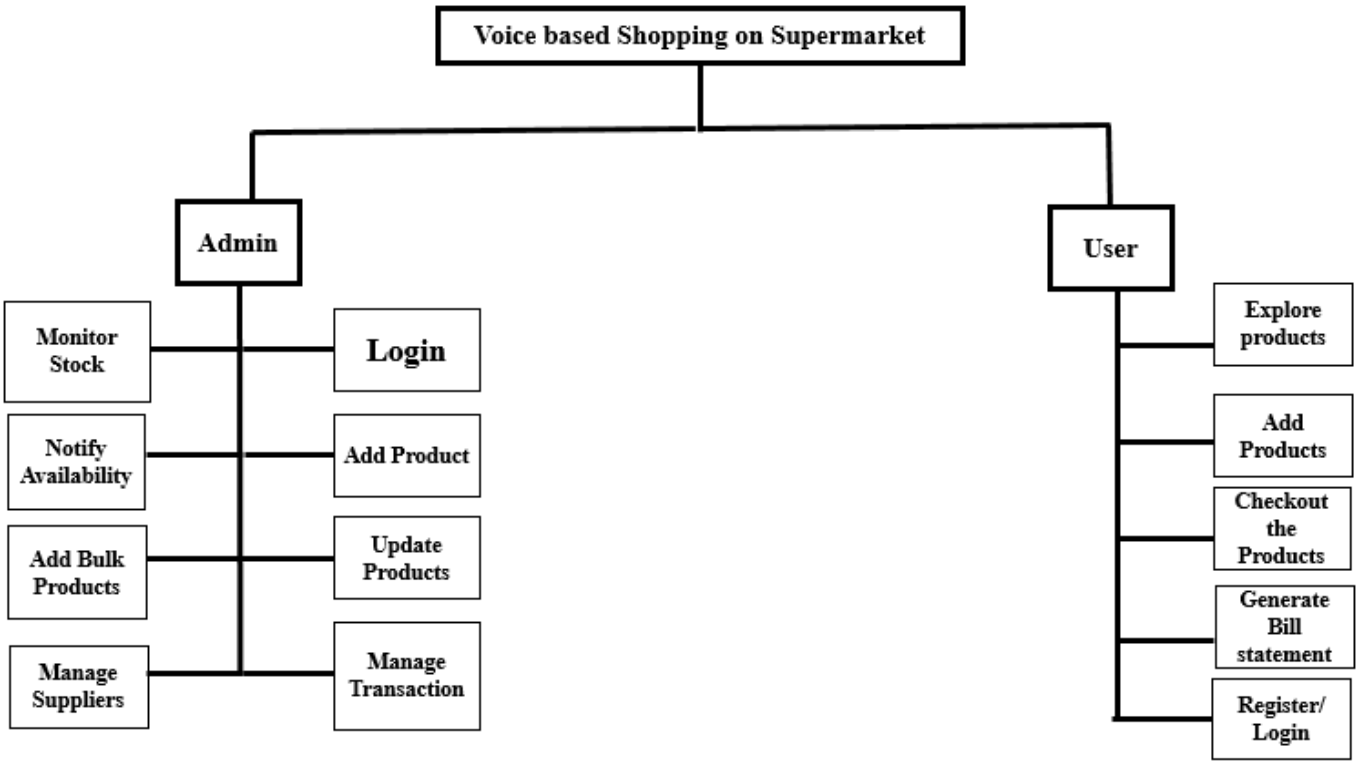
- This system allows users to add products to their cart using voice commands, making shopping easy and independent.
- After adding products to the cart, the system generates a bill that users can use to checkout and complete their purchase.
- For administrators adding products one by one or through Excel file uploads.
- And managing user details, checking product availability, and notify when low stock.

CHAPTER 4

4.SYSTEM DESIGN

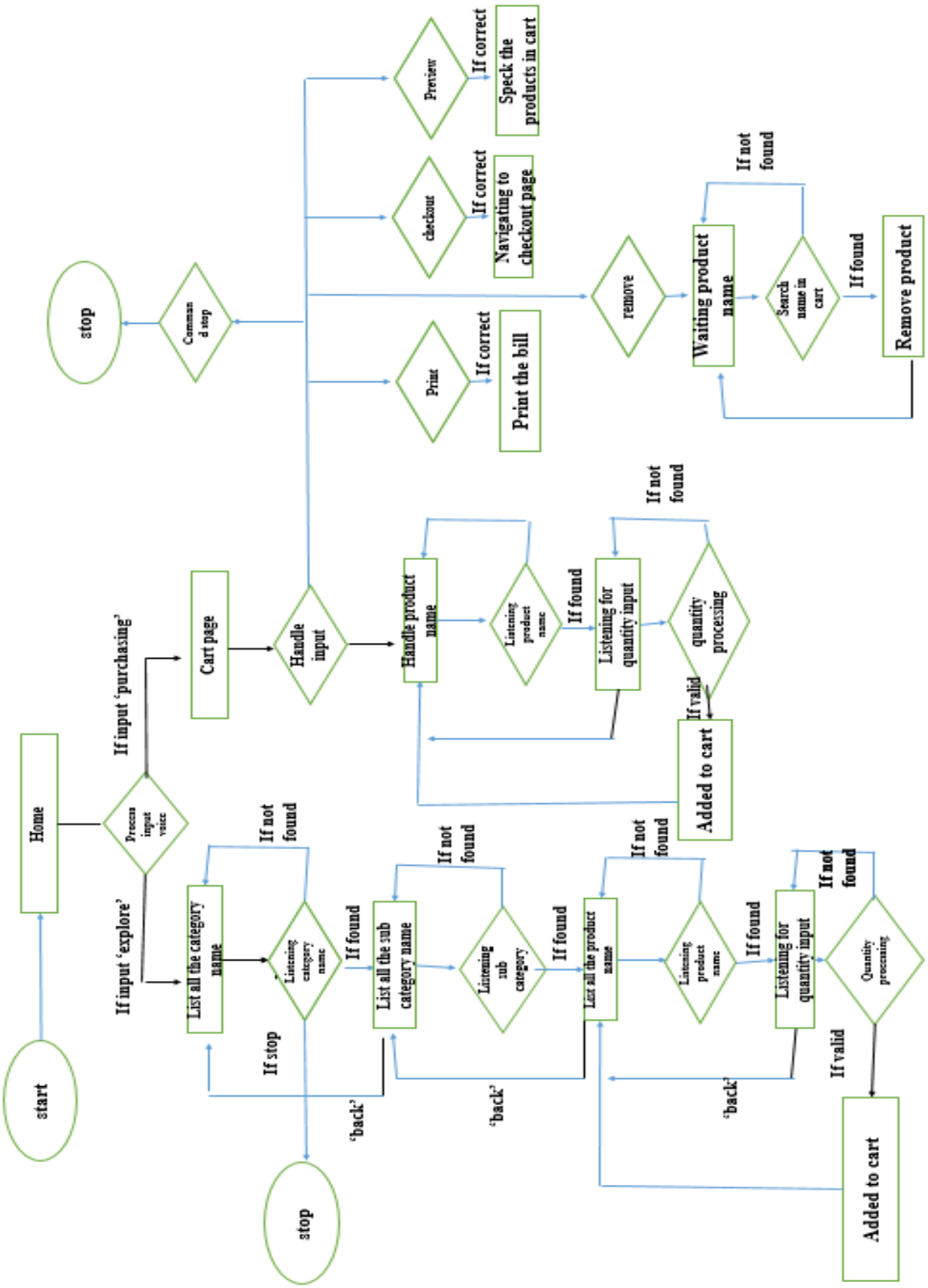
System Design is the process of designing a system as per the user specification. Design is mainly concerned with providing the model for implementation phase. In this project, Architectural design, Data flow diagrams are used and system is designed. The database design is also given.

4.1 ARCHITECTURE DESIGN:



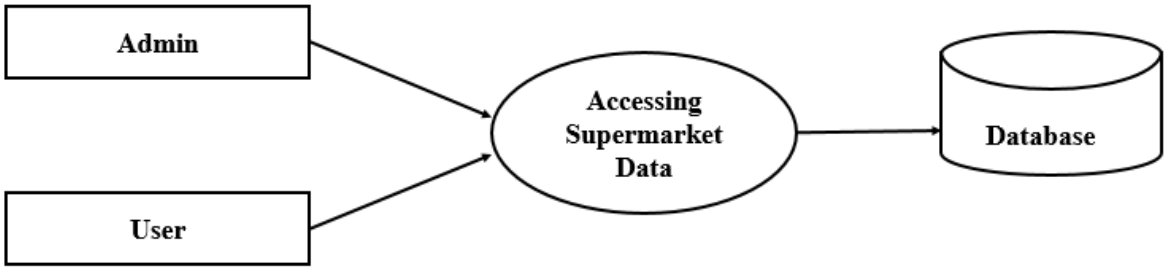
4.2 PROCESS FLOW DIAGRAM

In this diagram narrate the process of user interaction with the website. For how to purchase.



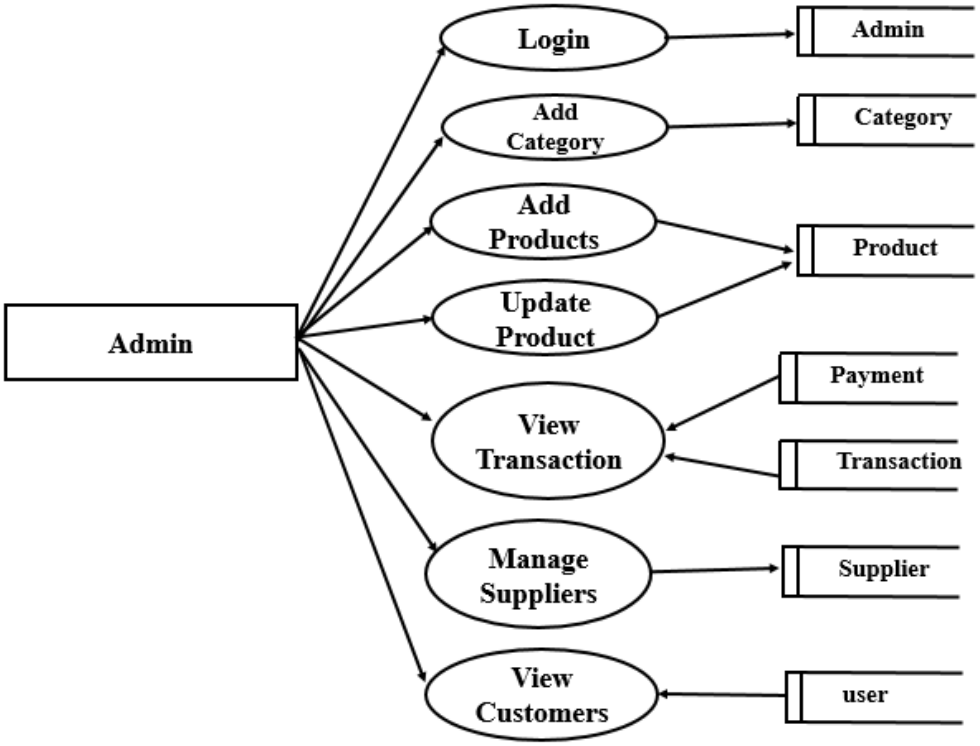
4.3 DATA FLOW DIAGRAM

Level 0



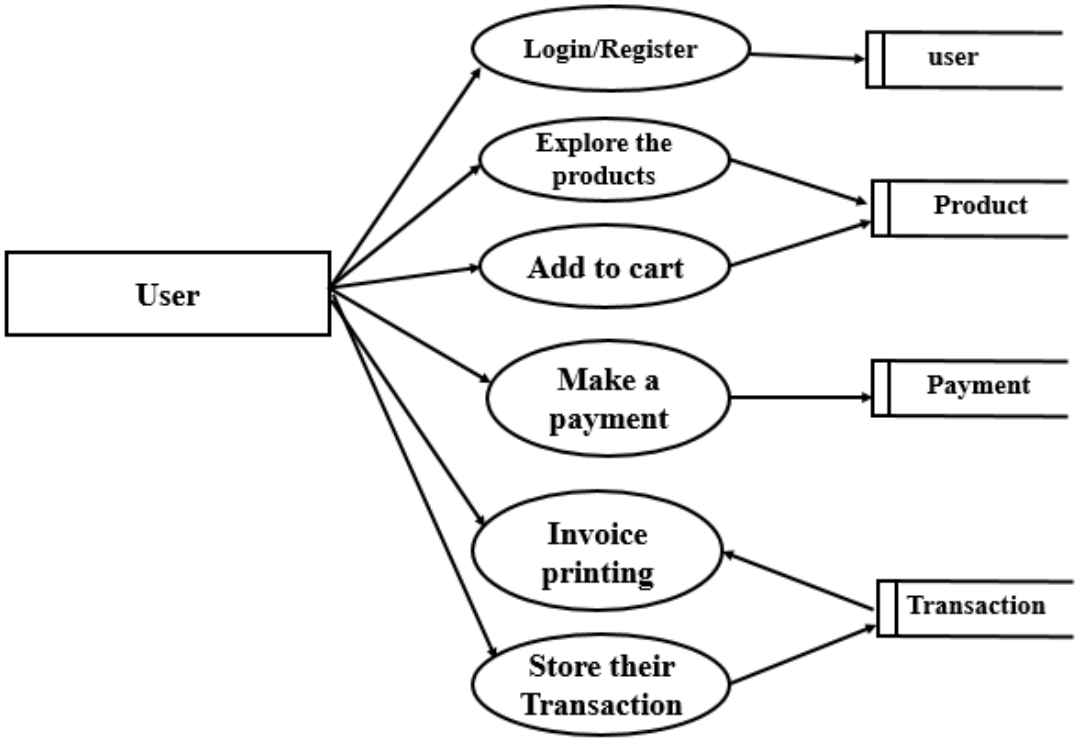
Level 1

Admin



Level 1

User


User
Login/Register
user
Explore the products
Product
Add to cart
Make a payment
Payment
Invoice printing
Transaction
Store their Transaction

4.4 DATABASE DESIGN

Database Name: Supermarket

1.Collection Name : user.
Purpose : To Store customer information

S.No.	Attribute Name	Data Type	Description
1.	Cid	String	Customer Id
2.	Name	String	Customer Information
3.	Cpassword	String	Password
4.	Email	String	Email Address

2.Collection Name: Admin
Purpose: To store admin details.

S.No.	Attribute Name	Data Type	Description
1.	Aid	ObjectId	Admin Id
2.	Aname	String	Admin Name
3.	Apass	String	Password
4.	Aemail	String	Email Address

3.Collection Name: Category
Purpose: To store new Category.

S.No	Attribute Name	Data Type	Description
1.	CategoryId	ObjectId	Category Id
2.	Category Name	String	Category Name

4.Collection Name: Product
Purpose: To store Product details.

S.No.	Attribute Name	Data Type	Description
1.	Pro_id	ObjectID	Product Id
2.	Proname	String	Product Name
3.	Procate	String	Category Name
4.	SubCate	String	Sub Category
5.	Quantity	Int	No of Stock
6.	Price	Float	Product Price
7.	Discount	Float	Discount percentage
8.	Supplier_id	Int	Suppliers Id

5.Collection Name: Cart
Purpose: To store Cart details.

S.No.	Attribute Name	Data type	Description
1.	Proid	ObjectId	Product id from product collection
2.	proQuantity	Float	Product quantity
3.	Cid	String	Customer id

6.Collection Name: Supplier
Purpose: To store Supplier details.

S.No.	Attribute Name	Data type	Description
1.	Supplier_id	Int	Suppliers Id
2.	S_Name	String	Supplier Name
3.	Contact	int	Contact Phone number

7.Collection Name: Payment
Purpose: To store order details.

S.No.	Attribute Name	Data type	Description
1.	Tran_Id	ObjectId	Transaction Id
2.	Cid	String	Customer Id
3.	Mode_of_payment	String	Payment Method
4.	Day	Date	Date of Purchase
5.	Amount	Float	Total Amount

8.Collection Name: Transaction
Purpose: To store per transaction information.

S.No.	Attribute Name	Data type	Description
1.	Transaction_Id	ObjectId	Transaction Id from payment collection
2.	Proid	ObjectId	Product ID from product collection
3.	Proname	String	Product Name
4.	ProQua	Float	Product Quantity
5.	Proprice	Float	Product Price

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 MODULES DESCRIPTION

MODULES:

- Admin
- user

Admin Module

The Admin Module is designed to streamline the management of product inventory, user accounts, and order transactions for store administrators.

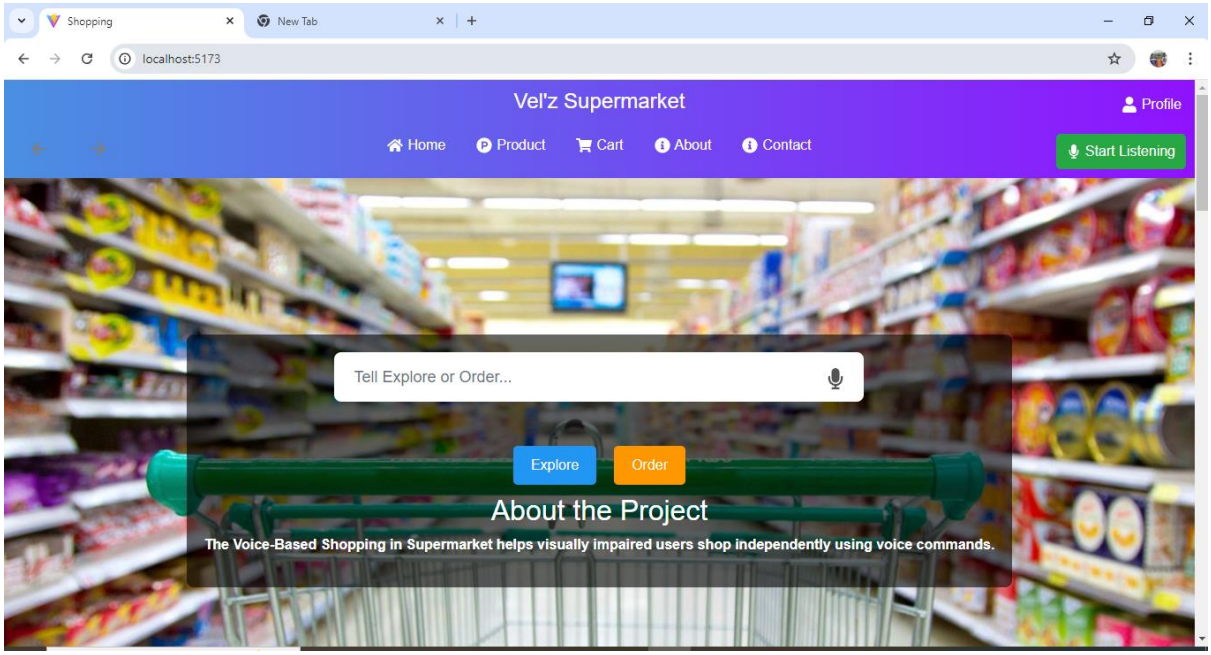
- Add or update products individually or in bulk using Excel uploads.
- Manage product categories (add, edit, delete) for better product organization.
- Easily upload large product inventories with an Excel file, updating multiple products at once.
- Monitor stock levels in real time.
- Automatically send email notifications to admins when product quantities drop below a set threshold.
- View and manage supplier details for restocking.
- System sends automated email alerts for low-quantity products, ensuring timely restocking and preventing shortages.
- Track all orders placed by users with detailed information, including product details and payment status.
- Search and filter orders by date, user, or product for easier management.
- Visual charts (bar, line, pie) for tracking low-stock products, daily/weekly/monthly transactions, and total revenue.
- Visualize customer purchasing patterns with charts, showing frequently purchased items and trends in user behavior.
- Analyze daily and monthly sales through visual charts, helping admins make data-driven decisions.
- View, edit, or deactivate user accounts.
- Access full user transaction histories for monitoring individual customer activity.
- Manage current orders, update order statuses, and track payments to ensure smooth order fulfillment.

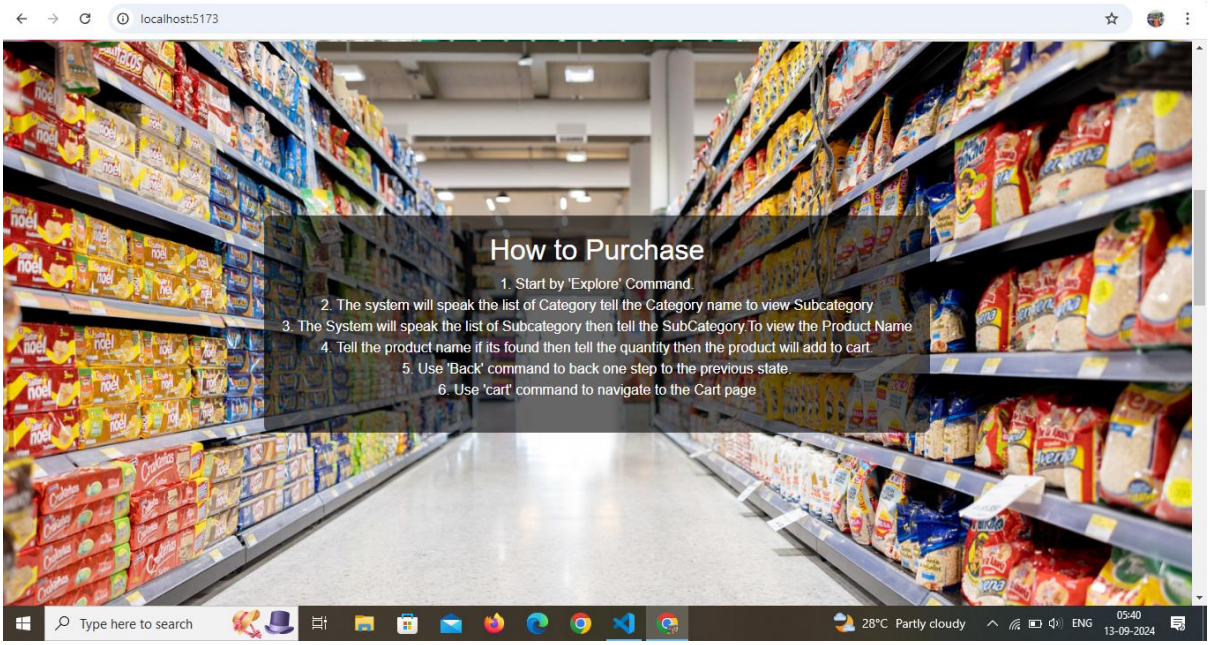
User Module

The User Module is the core feature of the Voice-Based Shopping System, specifically designed to provide visually impaired users with a fully voice-interactive shopping experience.

- Users interact with the system using voice commands for searching products, adding items to the cart, and checking out.
- Voice recognition allows users to add items by name, category, or specific details.
- Users can manage their shopping cart via voice, including adding, removing, and adjusting product quantities.
- The system confirms actions with voice feedback, ensuring users know the status of their cart.
- Billing is handled through voice commands, with the system generating and reading out the total bill.
- Users can complete the checkout process via voice, ensuring independence throughout the shopping experience.
- User authentication allows login or signup using voice input, securely storing session data for future logins.
- Users receive voice prompts for order confirmation and summary before finalizing the purchase.
- Product search is voice-driven, allowing users to explore products by speaking the product name or browsing categories.
- Cart updates, including quantity changes or item removals, are done entirely through voice commands.
- Users receive audio feedback for low stock or unavailable items, guiding them to alternatives.
- The system is designed with accessibility in mind, providing step-by-step voice instructions for all actions.
- Error handling ensures that any invalid commands or issues are addressed with clear audio feedback, allowing users to retry actions easily.
- Users can access and review their order history through voice commands, receiving detailed audio summaries of past purchases.
- The system is built to offer full autonomy for visually impaired users, eliminating the need for assistance while shopping.

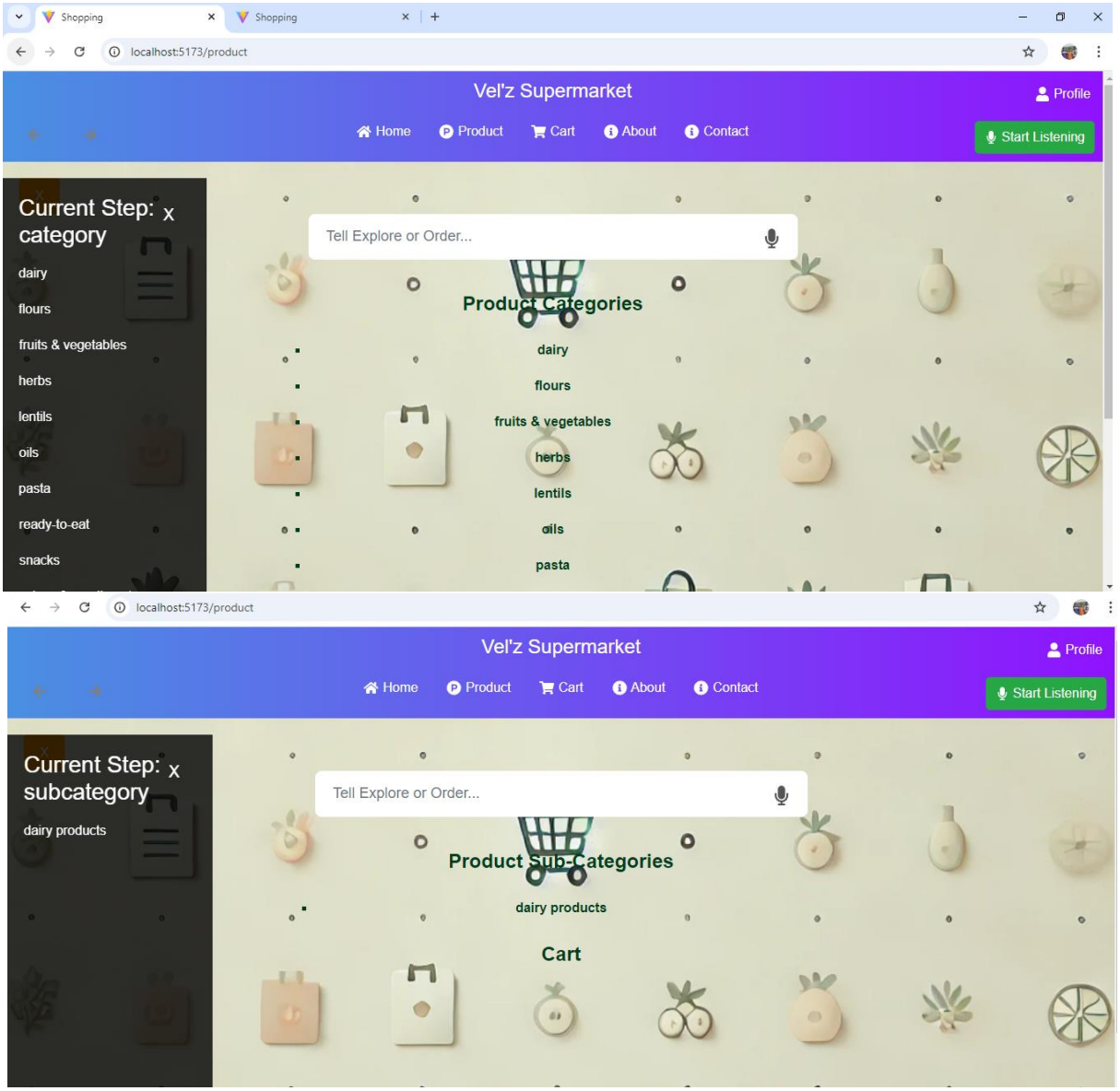
Fig 5.1 Home page:





In is page contain navigation instruction and navigation buttons

Fig 5.2 Product display page:



In this product page user tell the product name through voice command and add to cart or else by click the product name then add to cart.

Fig 5.3 Product view page

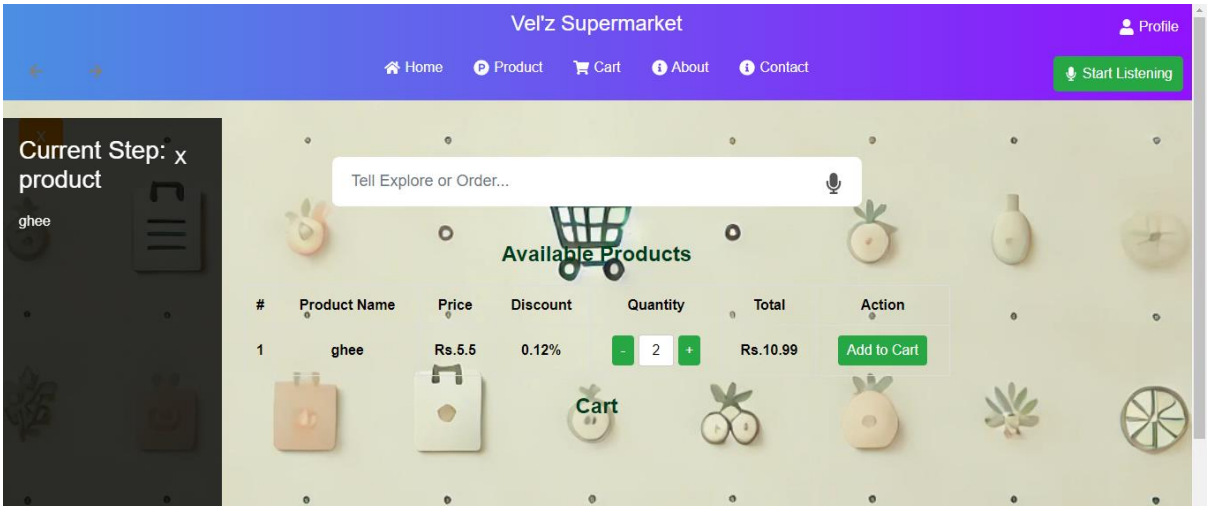
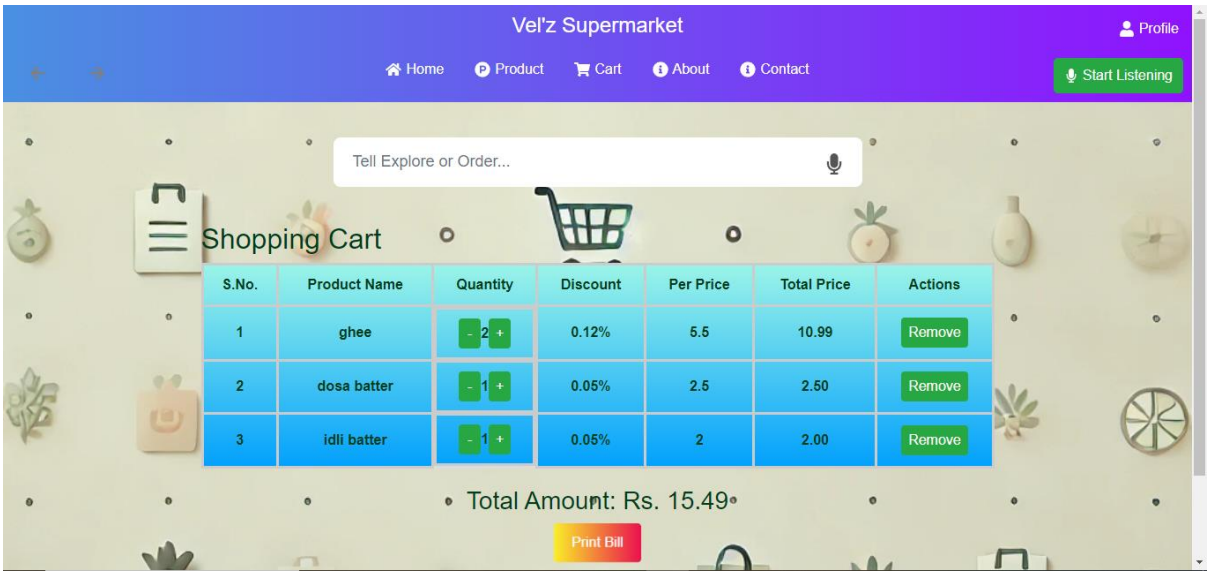
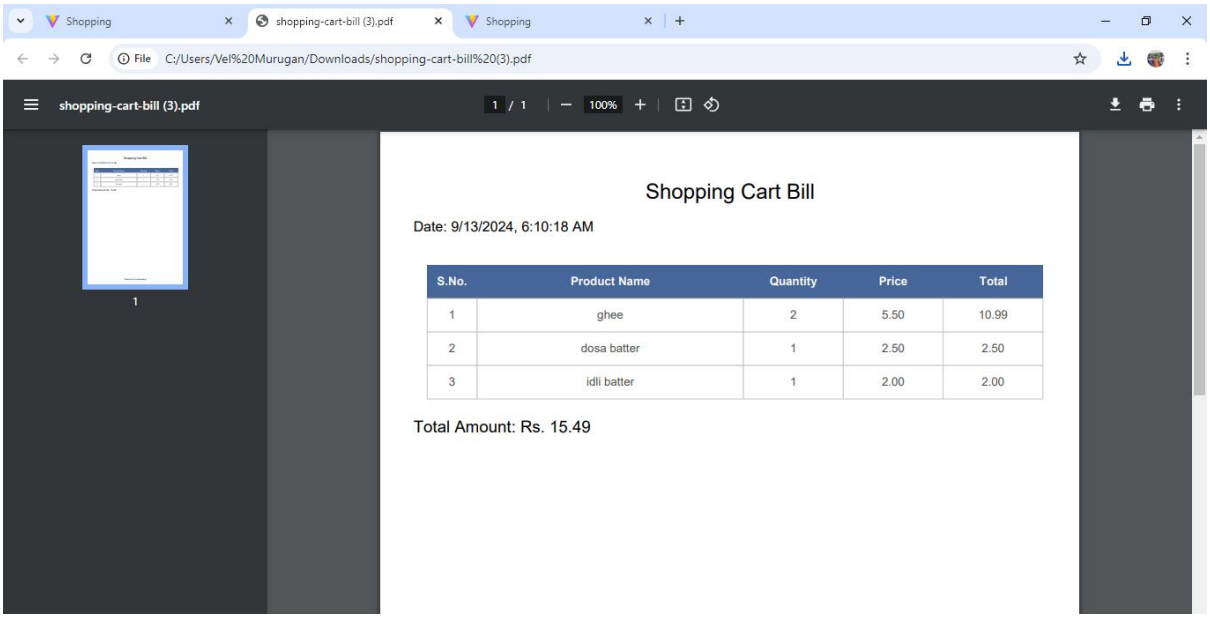


Fig 5.4 Cart page



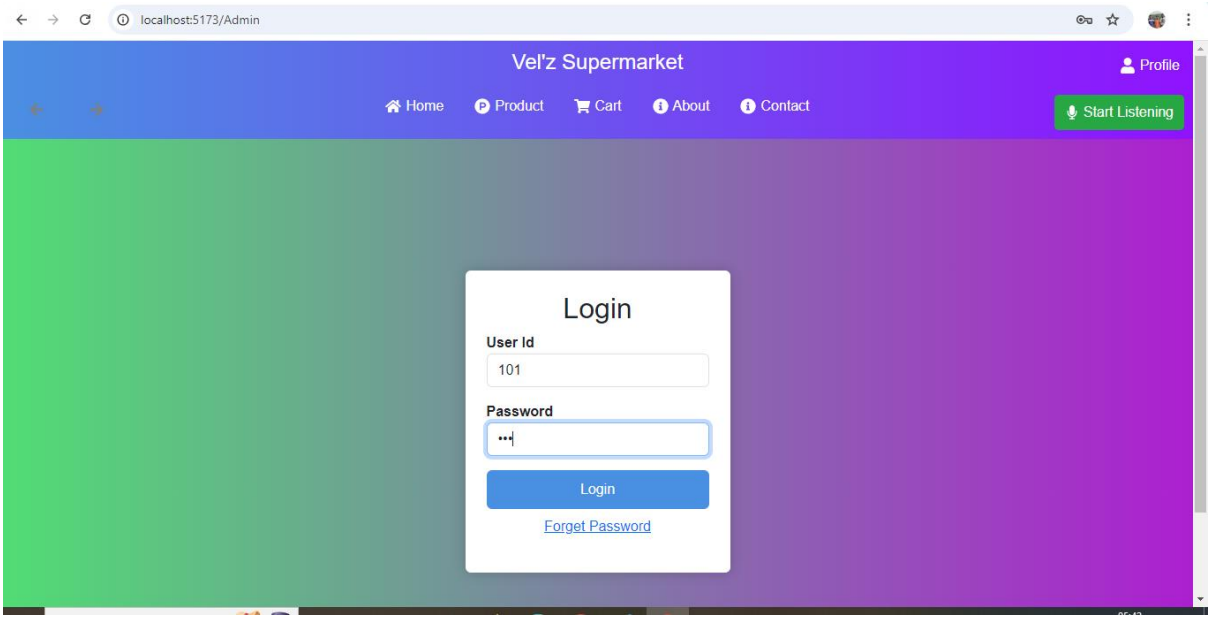
In this cart page user tell the product name then add to cart likewise remove the product by tell the product name.

Fig 5.5 Downloaded Bill



Here when user click the print bill button it will generate the bill based on the cart products.

Fig 5.6 Admin Login



It is a login page for admin panel. Admin can login using their email address and password

Fig 5.7 Admin DashBoard

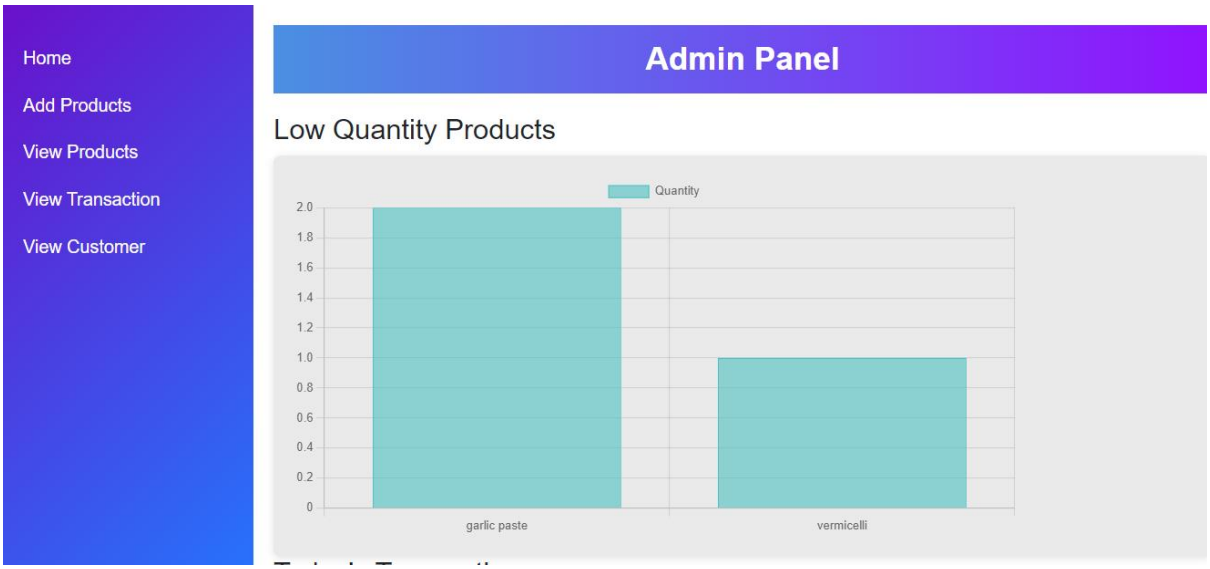
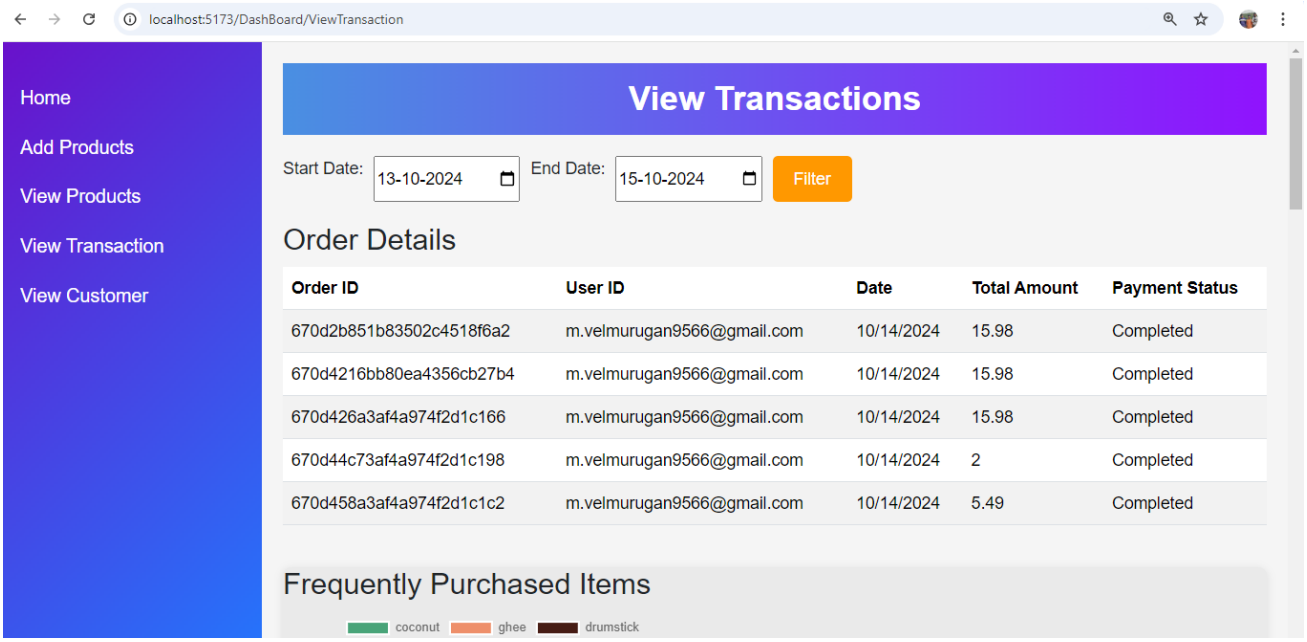
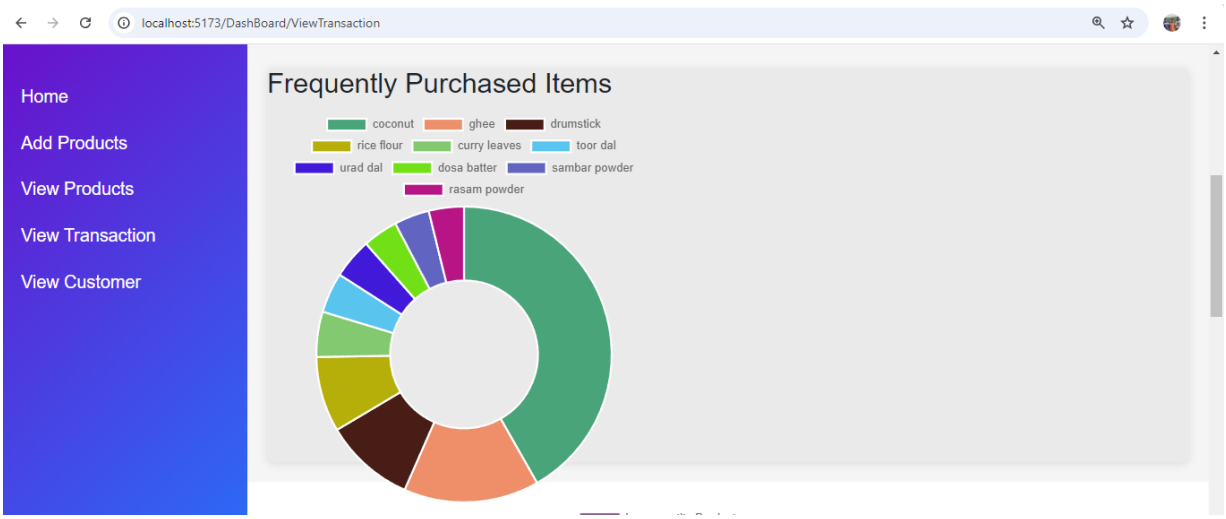
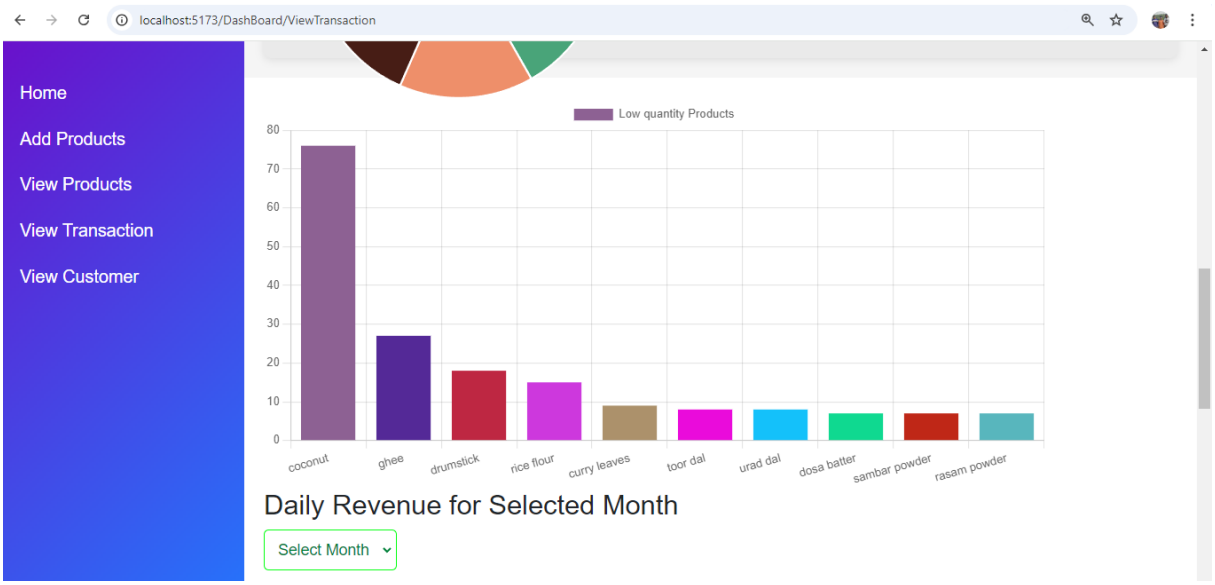


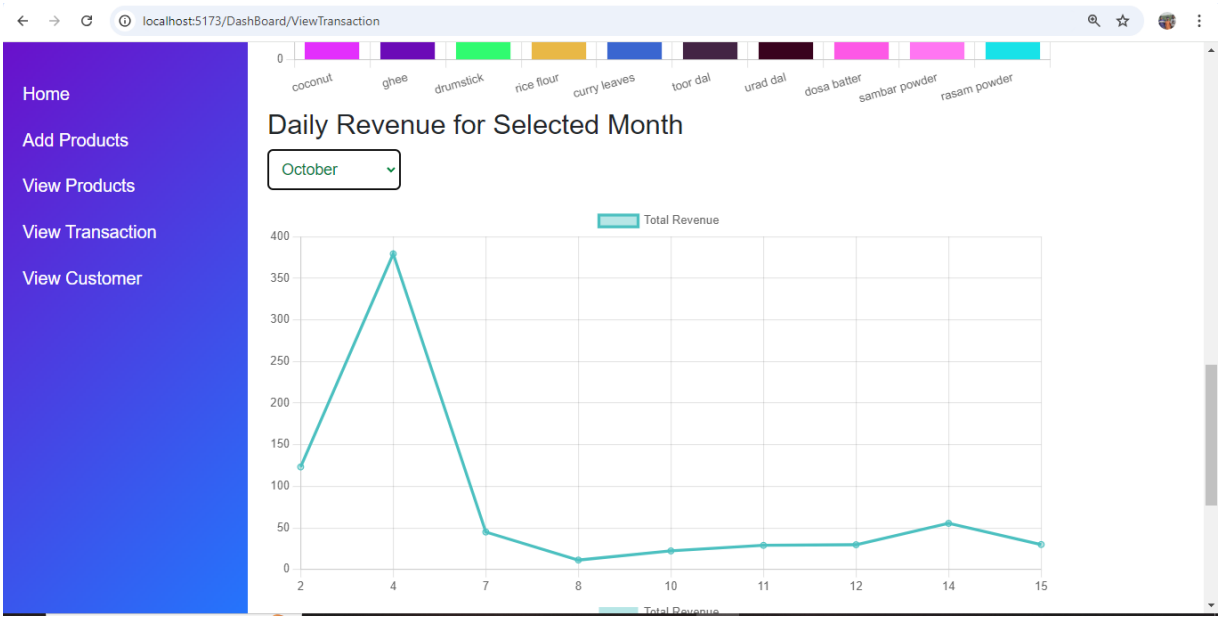
Fig 5.8 View Transaction page



Here display the transaction details between the dates based on the selected values

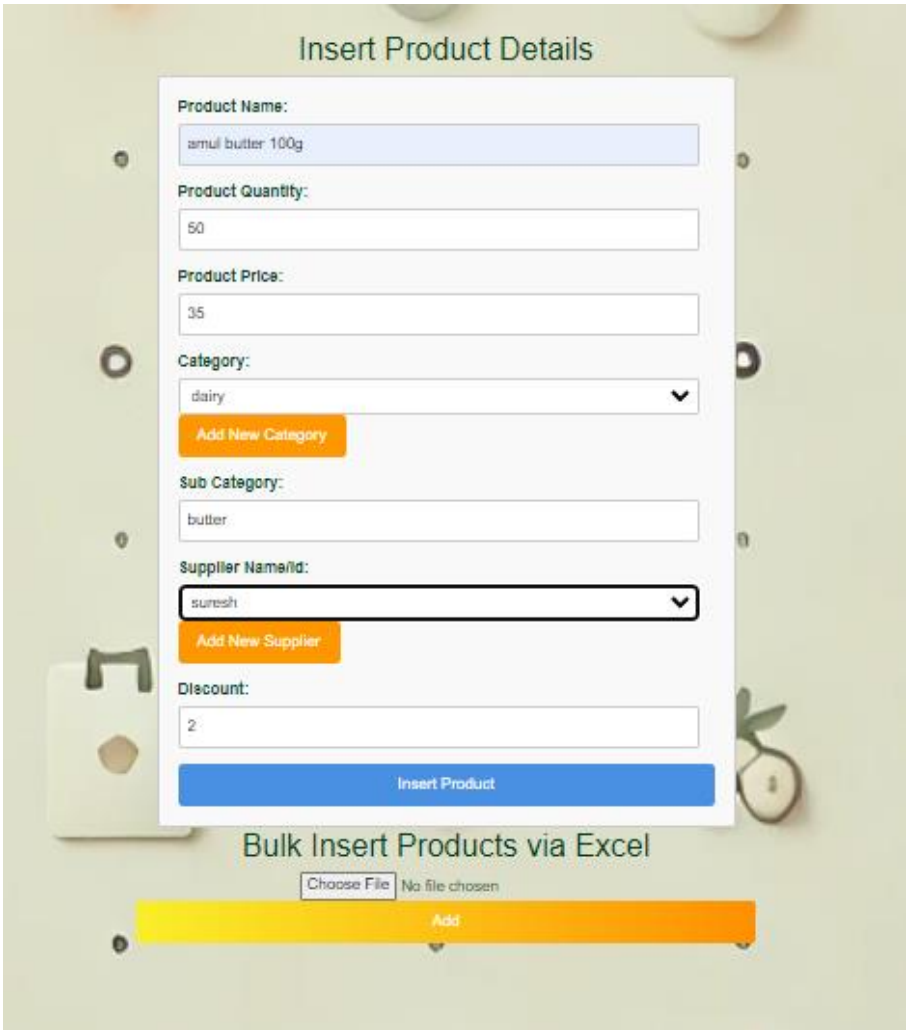






Here in these pages display the frequently purchased products in bar chart and daily revenue based on the month to be display in the line chart.

Fig 5.9 Insert Product



Here the insert product page contain the necessary fields to be insert for a product. here the option to add product category and suppliers details are there.

Fig 5.10 Database View

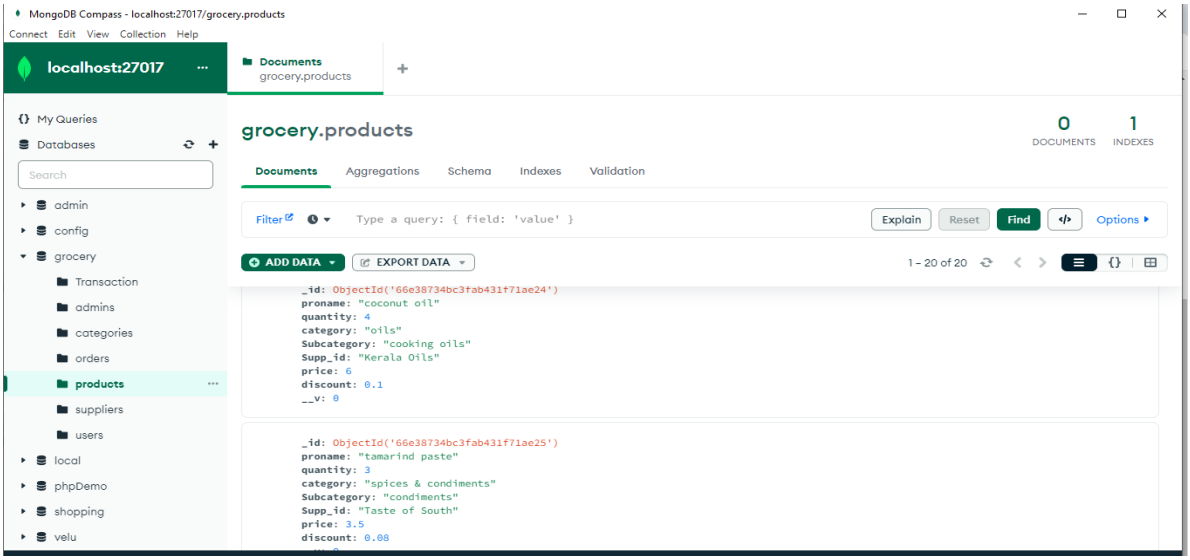


Fig 5.11 View Product

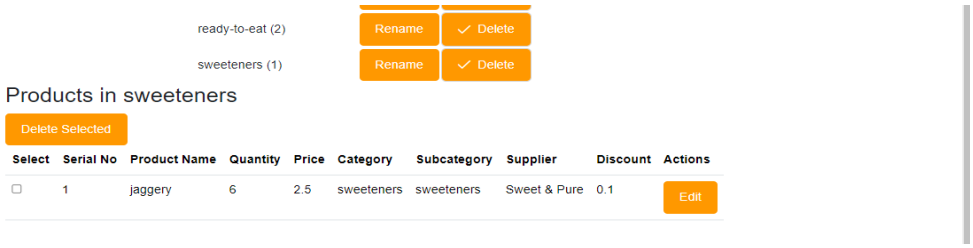
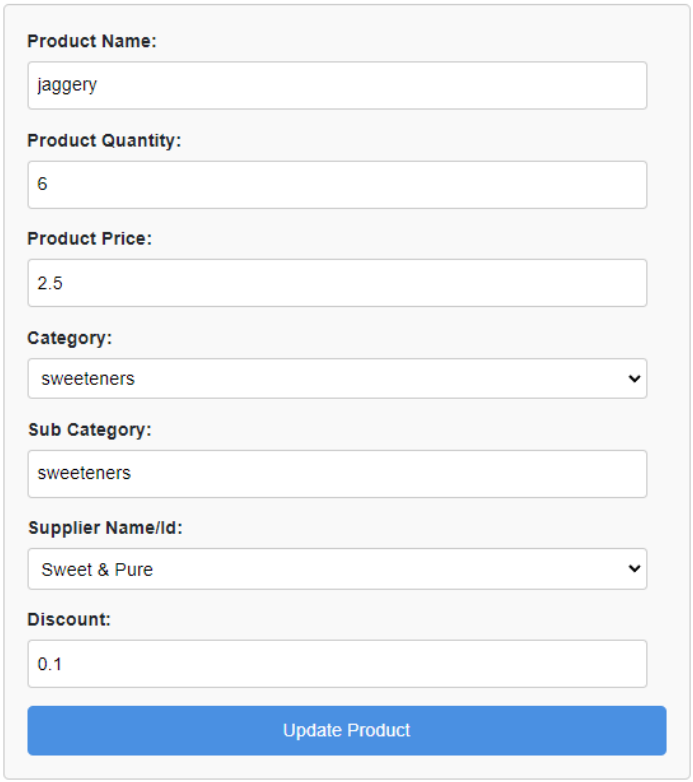


Fig 5.12 Edit Product into page



Here the edit product page change the fields value as select the specified product name

CHAPTER 6

6.TESTING

6.1 UNIT TESTING:

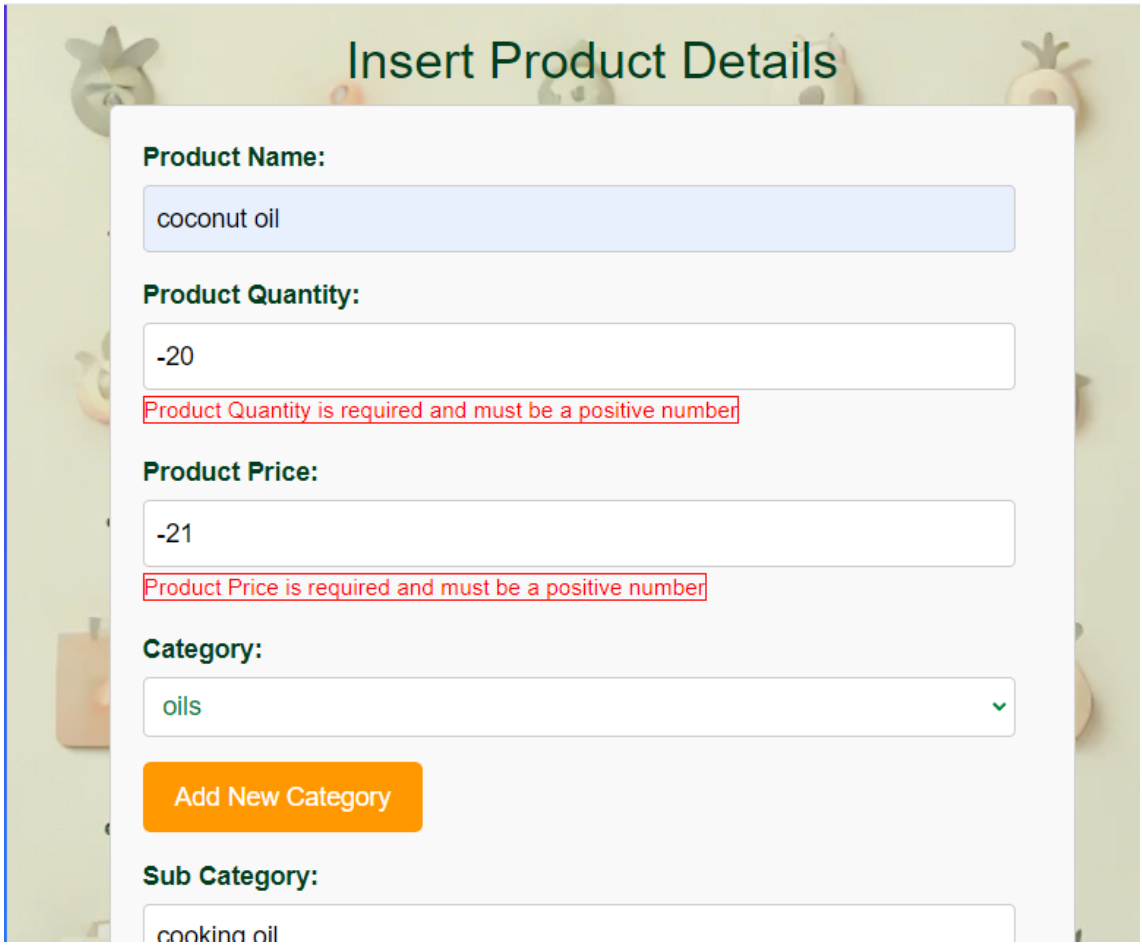
Unit testing was performed to validate that individual features, such as voice commands for adding products to the cart, managing the cart, and generating the bill, worked as expected. For example, I tested the accuracy of product recognition and cart updates through voice commands.

6.2 INTEGRATION TESTING

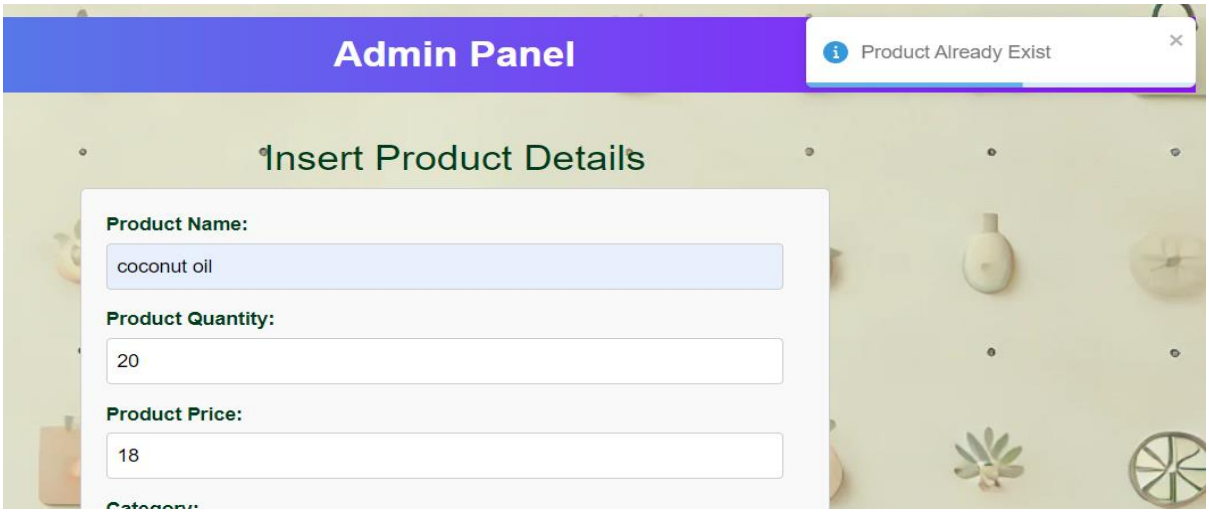
Integration testing was conducted to ensure that different components of the system worked together without any issues. This included verifying that the voice-based shopping functions properly interacted with the product database, cart management, and billing systems. I also tested the administrator features, such as product updates via Excel uploads, to confirm that they integrated smoothly with the inventory management and notification system.

6.3 SYSTEM TESTING

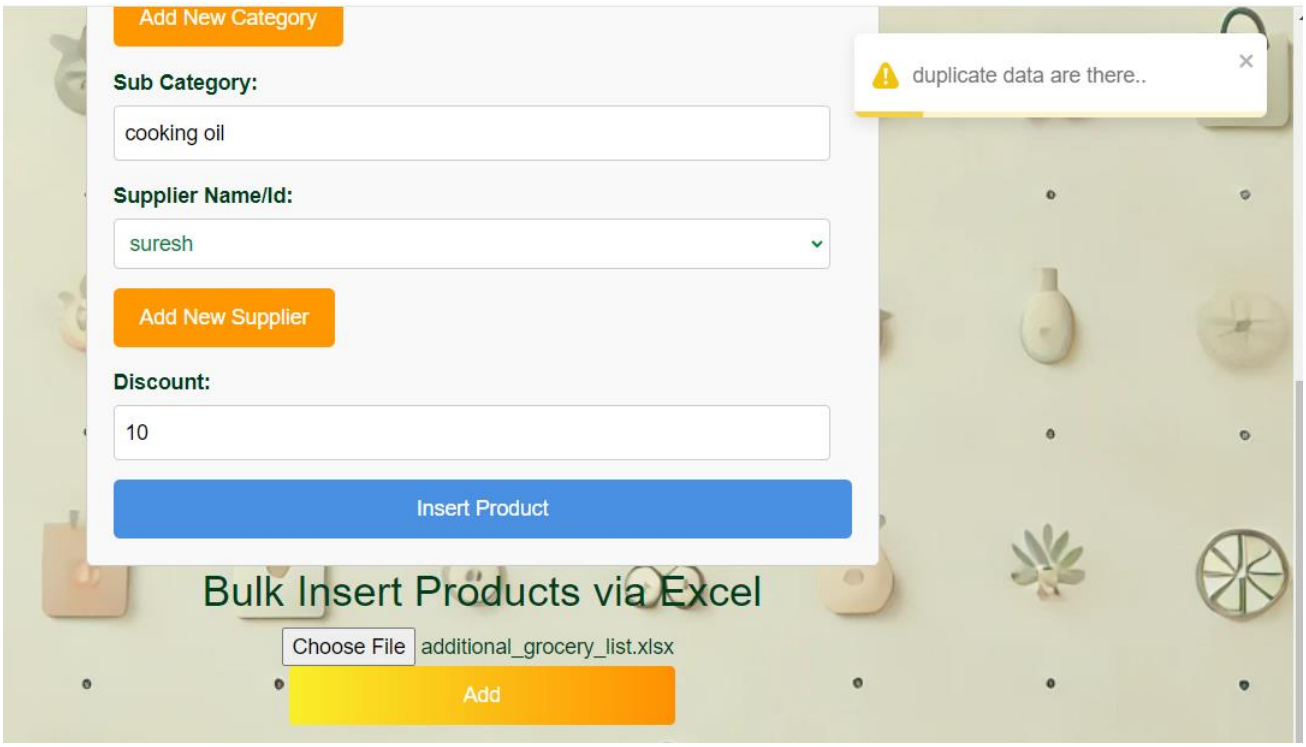
system testing was carried out to validate the entire workflow from the user's perspective, ensuring that visually impaired users could independently add products to their cart, manage them, and complete their purchase using voice interaction. I also tested the administrative dashboard, ensuring the correct display of low-quantity products and transaction data in charts, as well as email notifications for low stock. Through this comprehensive testing approach, I ensured the system functioned reliably for both users and administrators.



Here in this page product insertion not accept the negative values.



Here in this page product name has already exists error shown for product name validation



Here the insertion via excel sheet to check the each product name if the product name already exists in the database it will not permit to add the products.

CHAPTER 7

7. CONCLUSION

The Voice-Based Shopping System is a helpful tool that makes it easier for visually impaired people to shop on their own, without needing help from others. By using voice commands to search for products, manage the cart, and generate bills, the system provides a smooth and accessible shopping experience. For store administrators, it simplifies tasks like managing inventory and tracking transactions. Overall, this system improves shopping accessibility for users and makes store management more efficient.

CHAPTER 8

8. FUTURE SCOPE

- **Advanced Voice Features:**
  Add support for more detailed voice commands, such as asking for product details, reviews, or nutritional information, to enhance the shopping experience.

- **Integration with Navigation Systems:**
  Introduce voice-guided in-store navigation to help users locate products within a supermarket using GPS or indoor navigation technologies.

- **Multi-language Support:**
  Expand the system to support multiple languages, making it accessible to users from different regions.

- **Personalized Shopping Experience:**
  Implement machine learning to recommend products based on users' previous shopping habits and preferences.

CHAPTER 9

9. REFERENCES

Website Reference

https://www.npmjs.com/package/react-speech-recognition

https://blog.logrocket.com/using-react-toastify-style-toast-messages/

https://habtesoft.medium.com/node-js-email-with-nodemailer-a-comprehensive-tutorial-2866fcfc2336

https://react-charts.tanstack.com/

Book Reference

RESTful Web API Design with Node.js 10, Third Edition: Learn to Create Robust RESTful Web Services with Node.js, MongoDB, and Express.js, 3rd Edition

Vasan Subramanian-Pro MERN Stack_ Full Stack Web App Development with Mongo, Express, React, and Node-Apress (2017).pdf

CHAPTER 10

APPENDIX

SOURCE CODE

```
import React, { useState, useEffect } from 'react';
import 'slick-carousel/slick/slick.css';
import 'slick-carousel/slick/slick-theme.css';
import $ from 'jquery';
import 'slick-carousel';
import { Link, useNavigate } from 'react-router-dom';
import 'regenerator-runtime/runtime';
import { FaUser, FaShoppingCart,FaMicrophone, FaMicrophoneSlash } from 'react-icons/fa';
import Header from './UserHeader.jsx'
import '../style/homee.css';
import SpeechRecognition, { useSpeechRecognition } from "react-speech-recognition";

function App() {
  const navigate = useNavigate();
  const [mode, setMode] = useState(null);
  const [isListening, setIsListening] = useState(false);

  const startListening = () => {
    resetTranscript();
    SpeechRecognition.startListening({ continuous: true, language: "en-IN" });
    speak("Select the Choice Exploring or Purchasing");
    setIsListening(true)
  }

  const stopListening = () => {SpeechRecognition.stopListening(); setIsListening(false)};

  const {
    transcript,
    resetTranscript,
```

```
  browserSupportsSpeechRecognition
} = useSpeechRecognition();
useEffect(() => {
  if (!transcript) return;
  const timer = setTimeout(() => {
    handleModeSelection(transcript.trim().toLowerCase());
    resetTranscript();
  }, 5000);
  return () => clearTimeout(timer);
}, [transcript]);
useEffect(() => {
  // Stop speaking when the component unmounts (e.g., navigating to a new page)
  return () => {
    speechSynthesis.cancel();
  };
}, []);

if (!browserSupportsSpeechRecognition) {
  return <span>Browser doesn't support speech recognition.</span>;
}
const speak = (text) => {
  // Stop any ongoing speech before starting a new one
  speechSynthesis.cancel();
  const utterance = new SpeechSynthesisUtterance(text);
  speechSynthesis.speak(utterance);
};
const handleModeSelection = (input) => {
  if (input.includes("explore") || input.includes("exploring")) {
    setMode("exploring");
    speak("Navigating to Exploring Page");
    navigate("/product");
  } else if (input.includes("purchase") || input.includes("purchasing")) {
    setMode("purchasing");
    speak("Navigating to Purchasing Page");
```

```
    navigate("/cart");
  } else {
    speak("Command not recognized. Please say 'Explore' or 'Purchase'.");
    resetTranscript();
  }
};

const nav = (e) => {
  if (e === 'explore') {
    navigate("/product");
  }
  if (e === 'order') {
    navigate("/cart");
  }
};
useEffect(() => {
  const sections = document.querySelectorAll('.section');
  const handleScroll = () => {
    sections.forEach(section => {
      const sectionTop = section.getBoundingClientRect().top;
      const windowHeight = window.innerHeight;
      // Reveal section when it's in the viewport
      if (sectionTop < windowHeight - 100) {
        section.classList.add('active');
      } else {
        section.classList.remove('active');
      }
    });
  };

  window.addEventListener('scroll', handleScroll);

  return () => {
    window.removeEventListener('scroll', handleScroll);
```

```
  };
 }, []);
 return (
  <div>
   <div className="App">
    <Header
     isListening={isListening}
     startListening={startListening}
     stopListening={stopListening}
    />
   </div>
   <section className="section section1">
    <div className="content">
    <div className="hp-textbox"> <input type='text' className='textp' placeholder='Tell Explore or Order...' value={transcript} readOnly></input><div className='svg'><FaMicrophone /></div></div>
    <br/>
    <button className="hp-btn hp-explore-btn" onClick={() => nav("explore")}>
     Explore
    </button>
    {/* <button onClick={startListening} className="hp-btn hp-btn-success">
     Start
    </button>
    <button onClick={stopListening} className="hp-btn hp-btn-success">
     Stop
    </button> */}
    <button className="hp-btn hp-order-btn" onClick={() => nav("order")}>
     Order
    </button>
    <h2>About the Project</h2>
    <p><b>
     The Voice-Based Shopping in Supermarket helps visually impaired users shop independently using voice commands.
    </b></p>
```

```
      </div>
    </section>
    <section className="section section3">
    </section>
    <section className="section section2>
    </section>
    <section className="section section3">
      <div className="content">
      <h2>list of Voice Commands in Cart page</h2>
      <p>
        Example voice commands:<br />
        - "Add milk to cart"<br />
        - "Checkout now"<br />
        - "Show my cart"
      </p>
      </div>
    </section>
      <footer className="footer">
      <h2>Footer</h2>
    </footer>
  </div>
 );
};
export default App;
```

Product page

```
useEffect(() => {
  if (!transcript) return;
  const processTranscript = () => {
    const lowerTranscript = transcript.trim().toLowerCase();
    if (awaitingSelection && similarMatches.length > 0) {
      const foundMatch = similarMatches.find(match =>
lowerTranscript.includes(match.toLowerCase()));
```

```
      if (foundMatch) {
        if (currentStep === "category") {
          setSelectedCategory(foundMatch);
          fetchSubcategories(foundMatch);
          setCurrentStep("subcategory");
        } else if (currentStep === "subcategory") {
          setSelectedSubcategory(foundMatch);
          fetchProducts(foundMatch);
          setCurrentStep("product");
        } else if (currentStep === "product") {
          const foundProduct = products.find(product => product.proname.toLowerCase() ===
foundMatch.toLowerCase());
          speak(`How many ${foundProduct.proname} would you like to add to your cart?`);
          setCurrentStep("quantity");
        }
        setSimilarMatches([]);
        setAwaitingSelection(false);
        resetTranscript();
        return;
      } else {
        speak("No exact match found among the options. Please try again.");
        resetTranscript();
        speakOptions(similarMatches);
        return;
      }
    }

    if (currentStep === "category") {
      const matchedCategories = categories.filter(category => new
RegExp(category.toLowerCase()).test(lowerTranscript));
      if (matchedCategories.length === 1) {
        setSelectedCategory(matchedCategories[0]);
        fetchSubcategories(matchedCategories[0]);
        setCurrentStep("subcategory");
```

```
        resetTranscript();
       } else if (matchedCategories.length > 1) {
        setSimilarMatches(matchedCategories);
        setAwaitingSelection(true);
        speakOptions(matchedCategories);
        resetTranscript();
       } else {
        speak("Category not matched. Please try again.");
        resetTranscript();
        speakCategories(categories);
       }
      } else if (currentStep === "subcategory") {
       const matchedSubcategories = subcategories.filter(subcategory => new
RegExp(subcategory.toLowerCase()).test(lowerTranscript));
       if (matchedSubcategories.length === 1) {
        setSelectedSubcategory(matchedSubcategories[0]);
        fetchProducts(matchedSubcategories[0]);
        setCurrentStep("product");
        resetTranscript();
       } else if (matchedSubcategories.length > 1) {
        setSimilarMatches(matchedSubcategories);
        setAwaitingSelection(true);
        speakOptions(matchedSubcategories);
        resetTranscript();
       } else {
        speak("Subcategory not matched. Please try again.");
        resetTranscript();
        speakSubcategories(subcategories);
       }
      } else if (currentStep === "product") {
       const matchedProducts = products.filter(product => new
RegExp(product.proname.toLowerCase()).test(lowerTranscript));
       if (matchedProducts.length === 1) {
        console.log(matchedProducts + " match")
```

```
        const foundProduct = matchedProducts[0];
        setCurrentProduct(foundProduct)
        speak(`How many ${foundProduct.proname} would you like to add to your cart?`);
        setCurrentStep("quantity");
        resetTranscript();
       } else if (matchedProducts.length > 1) {
        setSimilarMatches(matchedProducts.map(product => product.proname));
        setAwaitingSelection(true);
        speakOptions(matchedProducts.map(product => product.proname));
        resetTranscript();
       } else {
        speak("Product not matched. Please try again.");
        resetTranscript();
        speakProducts(products);
       }
      } else if (currentStep === "quantity") {
       const quantityMatch = lowerTranscript.match(/\d+/);
       console.log(quantityMatch +"quantitymatch")
       if (quantityMatch) {
        const quantity = parseInt(quantityMatch[0], 10);
         console.log("quantity"+quantity)
        if (quantity > 0) {
         //const foundProduct = products.find(product =>
lowerTranscript.includes(product.proname.toLowerCase()));
          const foundProduct = currentProduct
          if (foundProduct) {
           if (quantity <= foundProduct.quantity) {
            addToCart(foundProduct, quantity);
            setCurrentStep("product");
            toast.success("Product added to cart..")
            speak("Product added to the cart. You can add another product or say 'back' to choose a
different category.");
           } else {
```

```
            speak(`Only ${foundProduct.quantity} units available. Please specify a quantity up to ${foundProduct.quantity}.`);
          }
          resetTranscript();
        }
      }
      else{
        speak(`Please tell a valid quantity..`);
        setCurrentStep("quantity");
        resetTranscript();
      }
    }else{
      speak(`Please repeat the quantity value.`);
      setCurrentStep("quantity");
      resetTranscript();
    }
  }
  if (lowerTranscript.includes("repeat products")) {
    speakProducts(products);
  }

  if (lowerTranscript.includes("back")) {
    if (currentStep === "product") {
      setSelectedSubcategory("");
      setCurrentStep("subcategory");
      speakSubcategories(subcategories);
    } else if (currentStep === "subcategory") {
      setSelectedCategory("");
      setCurrentStep("category");
      speakCategories(categories);
    }else if (currentStep === "quantity") {
      setCurrentStep("product");
      speakProducts(products);
    }
```

```
      resetTranscript();
    }

    if (lowerTranscript.includes("stop")) {
      stopListening();
    }
    if (lowerTranscript.includes("repeat")) {
      if (currentStep === "category") {
        speakCategories(categories);
      } else if (currentStep === "subcategory") {
        speakSubcategories(subcategories);
      } else if (currentStep === "product") {
        speakProducts(products);
      } else if (currentStep === "quantity") {
        speak("Please specify the quantity for the selected product.");
      }
      resetTranscript();
    }
    if (lowerTranscript.includes("cart")) {

      resetTranscript();
      speak("Navigating to Cart page...")
      navigate("/cart");
    }
  };
  const timeoutId = setTimeout(processTranscript, 3000); // 3 seconds delay
  return () => clearTimeout(timeoutId);
}, [transcript, categories, subcategories, products, currentStep, similarMatches, awaitingSelection]);
```

Index.js

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
```

```
require('dotenv').config();
const app = express();
const Product = require('./models/Users');
const Adminlog= require('./models/Admin');
app.use(cors());
app.use(express.json());
mongoose.connect(process.env.MONGO_URI,{
//mongoose.connect('mongodb+srv://velmca24:vel9566@cluster0.i4qp0rb.mongodb.net/grocery?retr
yWrites=true&w=majority&appName=Cluster0', {
}).then(() => {
   console.log('Connected to MongoDB');
}).catch(err => {
   console.error('Error connecting to MongoDB', err);
});


app.get('/categories', async (req, res) => {
   try {
      const categories = await Product.distinct('category');
      res.json(categories);
   } catch (err) {
      console.error(err);
      res.status(500).json({ error: 'Internal server error' });
   }
});
app.get('/subcategories/:category', async (req, res) => {
  try {
   const category = req.params.category;
   // Fetch distinct subcategories for the given category
   const subcategories = await Product.distinct('Subcategory', { category: category });
   if (subcategories.length === 0) {
     return res.status(404).json({});
   }
   res.status(200).json(subcategories);
  } catch (error) {
   console.error('Error fetching subcategories:', error);
   res.status(500).json({ message: 'Server error while fetching subcategories.' });
  }
});
```