MOBILE APPLICATION FOR TEXTILE SHOP

PROJECT WORK I

Submitted by

KAMALESH J

21CSR075

KARAN M

21CSR077

NAHUL ATHITHYA M

21CSR122

in partial fulfillment of the requirements for

the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Estd : 1984

KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI ERODE – 638 060

MAY 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI ERODE – 638 060

MAY 2024

BONAFIDE CERTIFICATE

This is to certify that the Project report entitled **MOBILE APPLICATION FOR TEXTILE SHOP** is the bonafide record of the project work done by **KAMALESH J (Register No: 21CSR075), KARAN M (Register No: 21CSR077), NAHUL ATHITHYA M (Register No:21CSR122),** in the partial fulfillment of the requirements for the award of the Degree of Bachelor of Engineering in **Computer Science and Engineering** of Anna University Chennai during the year 2023-2024.

SUPERVISOR **HEAD OF THE DEPARTMENT**

Date: **(Signature with seal)**

ABSTRACT

This mobile application development project is for a textile shop named Best Collection Mens Wear for managing their stock products through the application. The existing application of the textile shop for managing the stocks is accessible only with the system available within the shop. So this project aims to design and develop a modern, user - friendly mobile application for product management which can be accessible from anywhere. The features of the application includes adding new stocks and managing the stocks. In addition to this, staff details management and new staff details addition is also included in the project as extra features. Upon completion, the application will be helpful for the shop owner for managing the stocks and accessing it using the mobile application.

To achieve the goals of the project, the application development team followed a structured approach to application design and development. This includes creating a wireframe and prototype to refine the application's user experience..

The application was developed using modern mobile application development technologies, such as Flutter for an intuitive UI, Node.js, and MongoDB Atlas for efficient data management. The project is optimized for performance and allows the user to easily update and manage the application.

To ensure the success of the project, the mobile application development team established a clear project plan and regularly communicated progress updates to the client. Quality assurance and testing were conducted throughout the development process to ensure that the application was fully functional and met the client's requirements.

KONGU ENGINEERING COLLEGE

(**Autonomous**)

PERUNDURAI ERODE – 638 060

MAY 2024

DECLARATION

We affirm that the Project Report titled **MOBILE APPLICATION FOR TEXTILE SHOP** being submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering is the original work carried out by us. It has not formed the part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion this or any other candidate.

Date:

KAMALESH J
21CSR075
KARAN M
21CSR077
NAHUL ATHITHYA M
21CSR122

I certify that the declaration made by the above candidates are true to the best of my knowledge.

Date:

Name and Signature of the Supervisor

ACKNOWLEDGEMENT

We express our sincere thanks and gratitude to **Thiru. A. K. ILANGO B.Com., M.B.A., LLB.,** our beloved Correspondent, and all other philanthropic trust members of Kongu Vellalar Institute of Technology Trust who have always encouraged us in academic and co-curricular activities.

We are extremely thankful with no words of a formal nature to the dynamic Principal **Dr. V. BALUSAMY, M.Tech., Ph.D**., for providing the necessary facilities to complete our work.

We would like to express our sincere gratitude to our respected Head of the Department **Dr. S. MALLIGA, M.E., Ph.D.**, for providing the necessary facilities.

We extend our thanks to **Ms. P. KALAIVANI** Assistant Professor, Computer Science and Engineering, Project Coordinator for her encouragement and valuable advice that made us carry out the project work successfully.

We extend our gratitude to our Supervisor **Dr. N. SHANTHI**, Professor of Computer Science and Engineering, for her valuable ideas and suggestions, which have been very helpful in the project. We are grateful to all the faculty members of the Computer Science and Engineering Department, for their support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Best Collection Mens Wear Textile Shop specializes in selling high-quality branded clothing for men, kids, and boys, providing a diverse selection to suit every style. With a focus on quality and fashion trends, they aim to offer customers a unique and satisfying shopping experience.

1.1 EXISTING SYSTEM

The existing application of the textile shop served as a basic tool for stock management, allowing the user to track and manage product inventory. However, it lacked modern features, user-friendly interfaces, and was used only in the shop system.

1.2 SYSTEM STUDY

I. Understanding the Business Requirements

The initial phase of the system study involves comprehending the business requirements of the project. This entails identifying the business objectives, as well as the necessary features and functionalities required in the application.

II. Analysing the Existing Systems and Processes

The next step is to analyse the existing systems and processes in place, including the current application and any related software systems. This will help identify any limitations or areas for improvement in the new application.

III. Identifying User Needs

The system study should also involve identifying the needs of the users who will be accessing the application. This can be done through surveys, focus groups, or other user research methods.

IV. Developing Use Cases

Use cases can be developed to illustrate how the application will function and how users will interact with it. This can help identify any potential issues or areas for improvement in the design and functionality.

1.3 OBJECTIVE

The primary objective of the project is to enhance the shop’s stock management by adding products, viewing products, adding staff details, viewing staff details, deleting product details and staff details accessible through the mobile application.

1.4 SCOPE

The project's scope entailed crafting a visually captivating, exceptionally functional, and intuitive mobile application to effectively convey the shop's stock management system. To guarantee project success, we committed to a collaborative approach with our client, deeply understanding their needs and aspirations. Regular progress updates were provided to ensure transparency throughout the development journey. Our overarching objective was to surpass client expectations by delivering an application that not only met but exceeded their requirements, while also serving as a compelling showcase for the company's stock management process.

CHAPTER 2

GENERAL DESCRIPTION

2.1 PROJECT PERSPECTIVE

- The application will serve as the shop's stock management and staff management.
- The application will be a marketing tool to manage and access their stocks from anywhere through the mobile application.
- The development team needed to have expertise in modern mobile application development technologies and ensure application functionality, security, and performance.

2.2 USER CHARACTERISTICS

ADMINISTRATOR :

- The key responsibilities and permissions of the administrator with the app include the ability to view and delete details of existing stock in the database.
- Add new stock products to the inventory, and manage staff details of the employees currently working in the shop.

2.3 DESIGN AND IMPLEMENTATION CONSTRAINTS

I. **Time**

There may be a deadline or time constraint for the development of the application which can impact the design and implementation process.

II. Security

The application may need to meet certain security requirements, secure login features, which can impact the design and implementation process.

III. Scalability

The application may need to be designed and developed with scalability in mind, to accommodate future growth or changes in the company's needs.

IV. User Experience

The application may need to meet certain user experience standards, such as accessibility or ease of use, which can impact the design and implementation process.

CHAPTER 3
REQUIREMENTS

3.1 FUNCTIONAL REQUIREMENTS

I. Dashboard

The application should have a dashboard for the easy access of the features that are available in the application.

II. Product List

The application should have a product list page to fetch the products from the database and show them to the user dynamically.

III. Add Product

The application should have the add product page with the necessary fields for storing the product details in the database.

IV. Staff Details:

The application should have the staff list page to fetch the staff details from the database, show to the user dynamically for knowing the staff details.

V. Add Staff

The application should have the add staff page with the necessary fields for storing the staff details in the database.

3.2 NON-FUNCTIONAL REQUIREMENTS

I. Performance

The application should be designed to load quickly and respond to user interactions in a timely manner, to ensure a positive user experience.

II. Usability

The application should be easy to use and navigate, with a clear and intuitive user interface, to ensure a positive user experience.

III. Reliability

The application should be reliable and available for use at all times, with minimal downtime or interruptions.

IV. Maintenance

The application should be easy to maintain and update, with a well-organized codebase and documentation, to minimize maintenance and support costs.

3.3 USER INTERFACES

DASHBOARD:

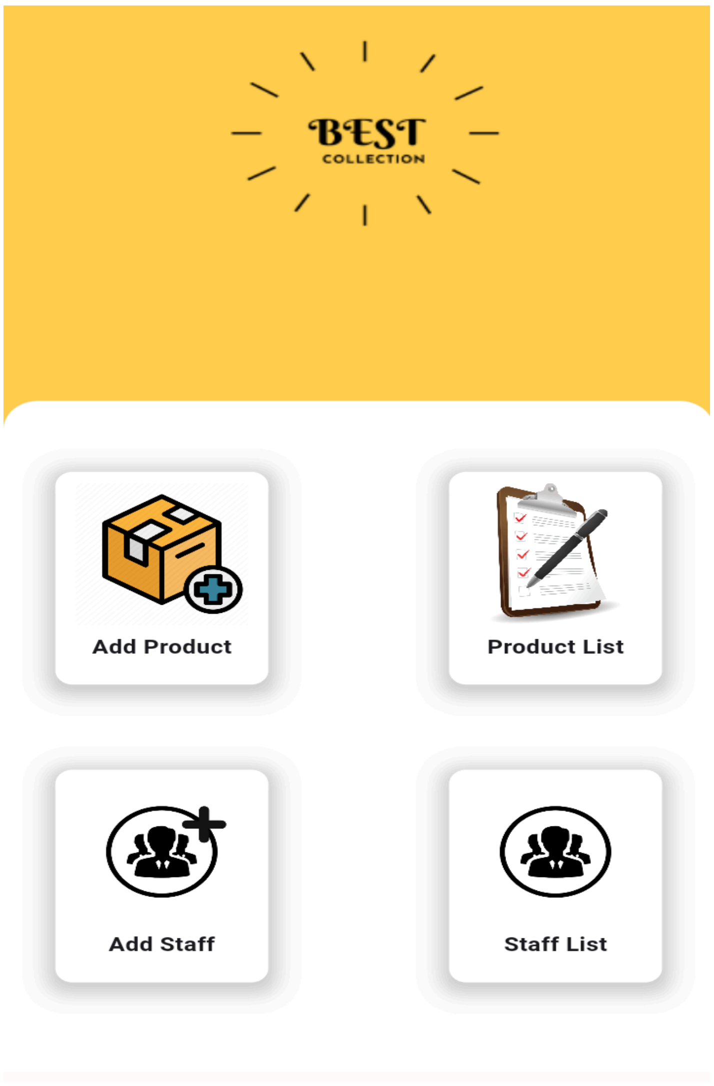


FIGURE 3.1 DASHBOARD

ADD PRODUCT :

The application should have an add product page with the necessary fields for storing the product details in the database.

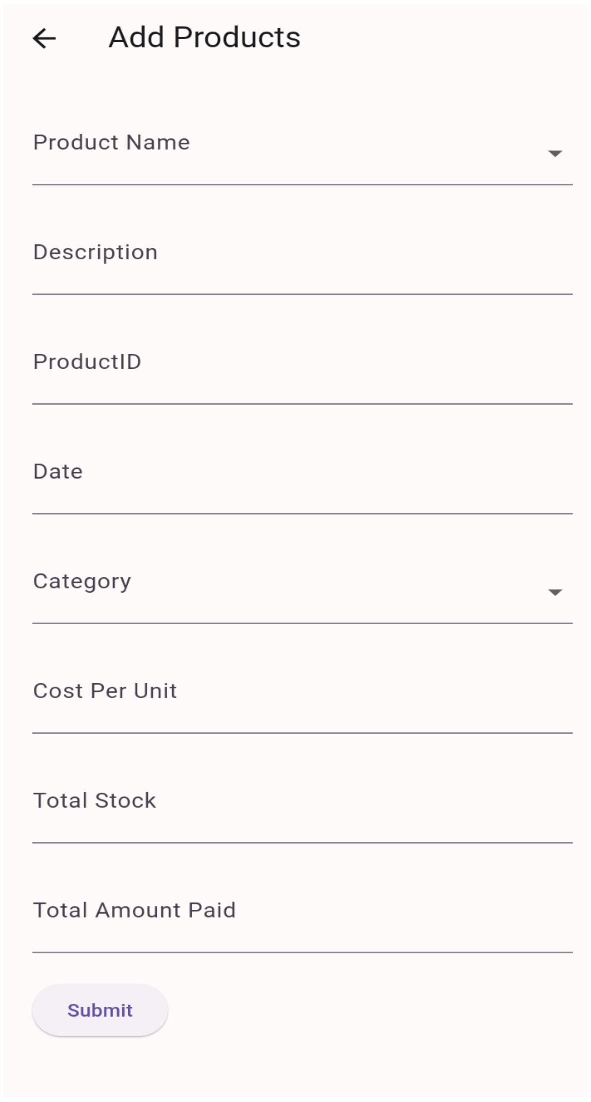


FIGURE 3.2 ADD-PRODUCT

ADD STAFF DETAILS:

The add staff details page should contain the necessary fields for adding details to the database.

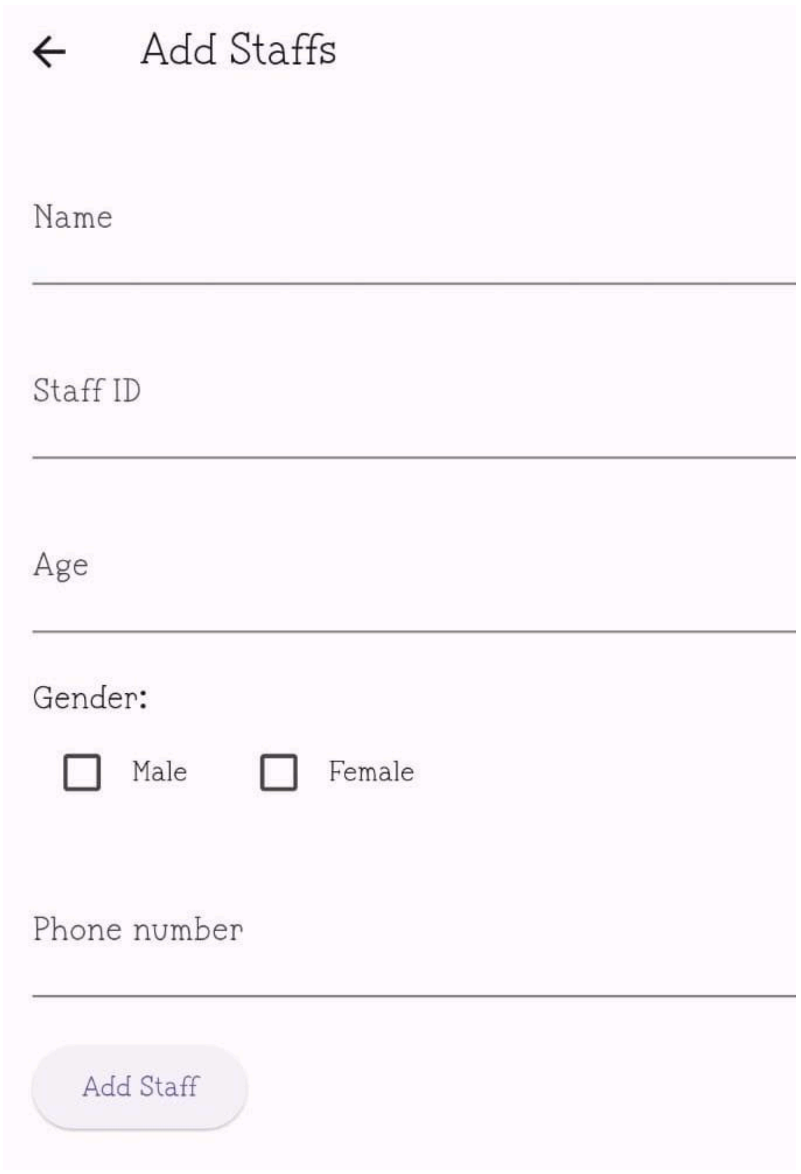


FIGURE 3.3 ADD STAFF

PRODUCT LIST

The product list page retrieves the stock details from the database and show to the user based on the category.

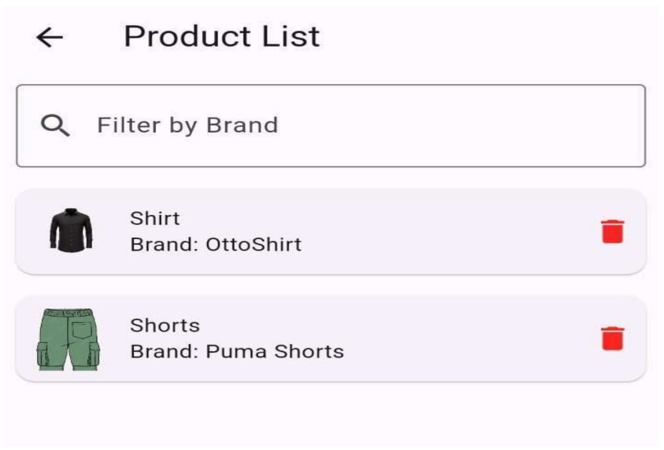
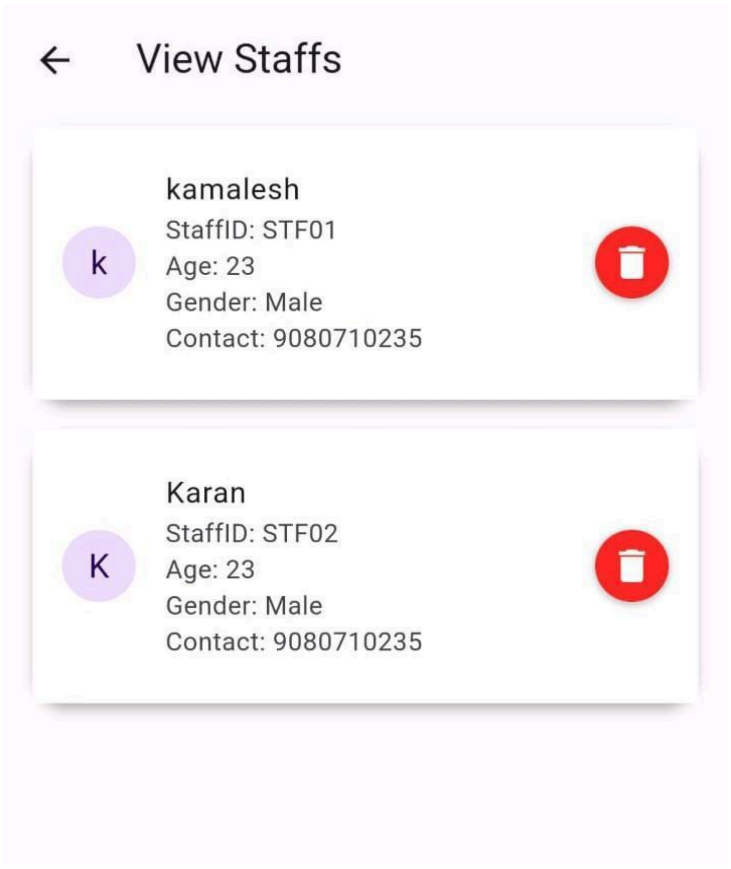


FIGURE 3.4 PRODUCT LIST

STAFF LIST

The staff list page retrieves the staff details from the database and shows them to the user.



View Staffs

kamalesh
StaffID: STF01
Age: 23
Gender: Male
Contact: 9080710235

Karan
StaffID: STF02
Age: 23
Gender: Male
Contact: 9080710235

FIGURE 3.5 STAFF LIST

CHAPTER 4

DETAILED DESIGN

4.1 ARCHITECTURAL DESIGN

The architectural design includes various diagrams such as use case diagram,sequence diagram and activity diagram for admin modules.

4.1.1 MODULE CHARACTERISTICS

The project contains admin modules as follows and their description is given below

- Dashboard
- Add Product Details
- Product List
- Staff Details
- Add Staff Details

4.1.1.1 ADMIN MODULE

The admin module contains a product list, add product, Staff details, Addstaff. The admin can view all the pages as mentioned in the admin module flow.

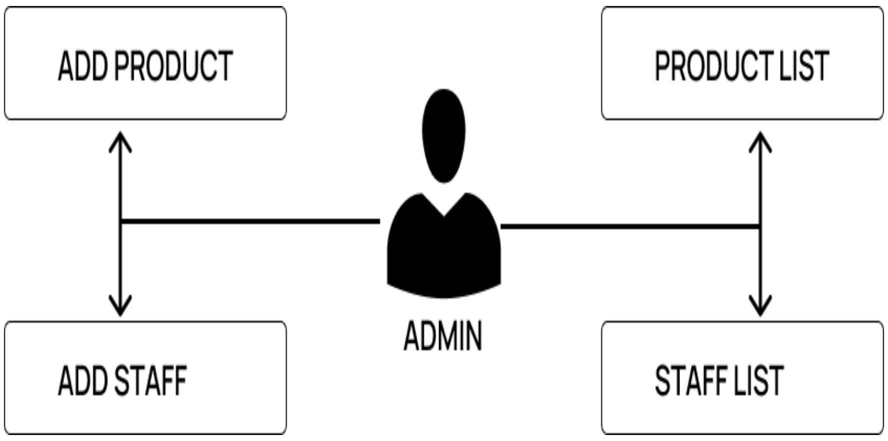


FIGURE 4.1 ADMIN MODULE DIAGRAM

4.1.2 USE CASE DIAGRAM

A use case diagram is a dynamic or behaviour diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. The use cases are represented by either circles or ellipses. Due to their simplistic nature, use case diagrams can be a good communication tool for stakeholders.

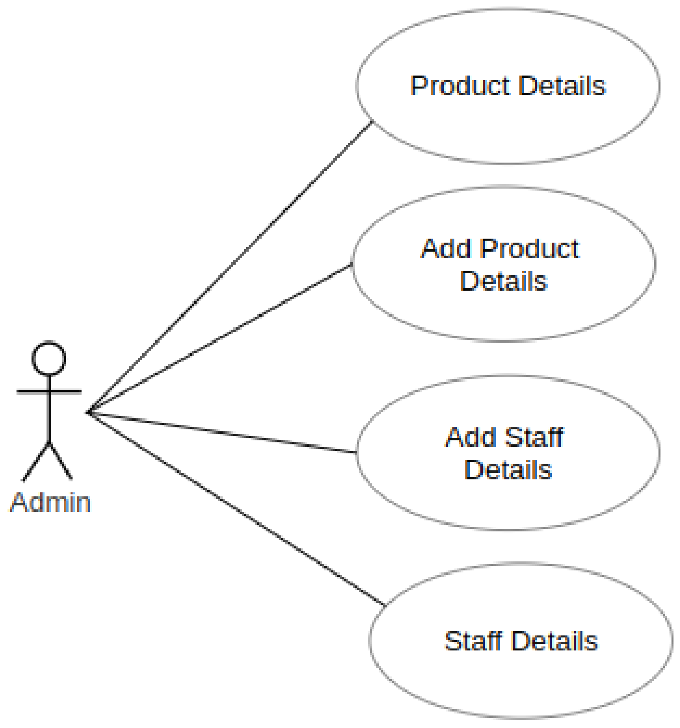


FIGURE 4.2 USE CASE DIAGRAM

4.1.3 SEQUENCE DIAGRAM

Sequence chart diagram is one of the five UML diagrams used to model the dynamic nature of a system. The admin defines the sequence of an object during its lifetime.

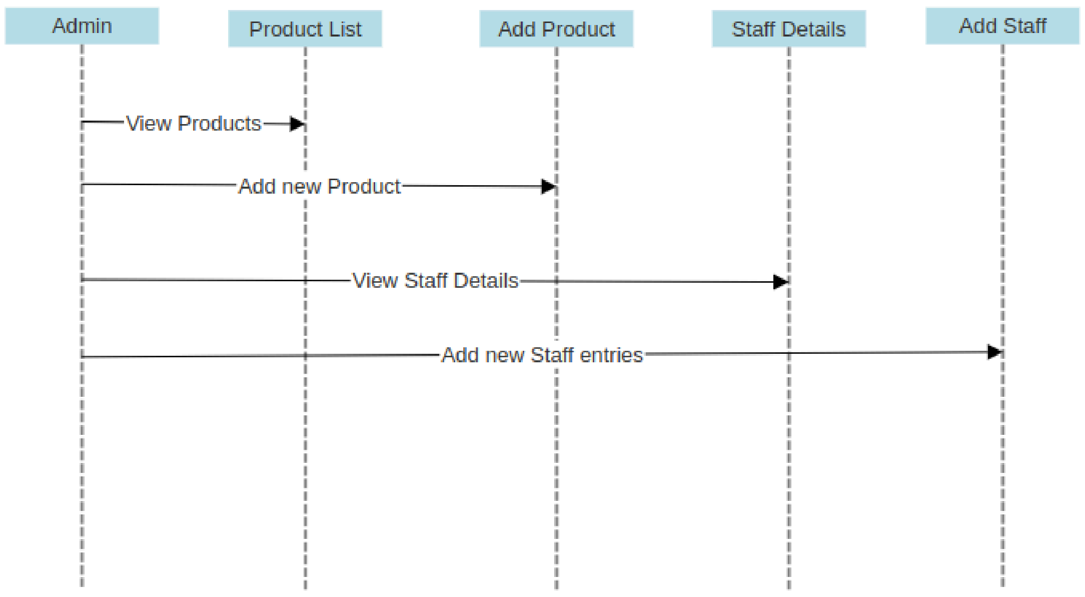


FIGURE 4.3 SEQUENCE DIAGRAM

4.2 INTERFACE DESIGN

The application provides various interfaces to the user that make them more convenient with the application. The user interface contains Dashboard, Product list, Add product, Staff Details, Add Staffs. This is simple to use and clear to understand. The user interface is more attractive and consistent on all interface screens. The interface is completely responsive.

4.3 DATABASE DESIGN

The Database used in this system is MongoDBAtlas. In Mongo, the data is stored in the form of collections and documents. There are several collections like services to store, update, delete, retrieve as well as delete the stock details.Product details like product name, description, date, category, cost per unit, total stocks,total amount paid and Staff details like name, age, gender, mobile number stored in the MongoDBAtlas.

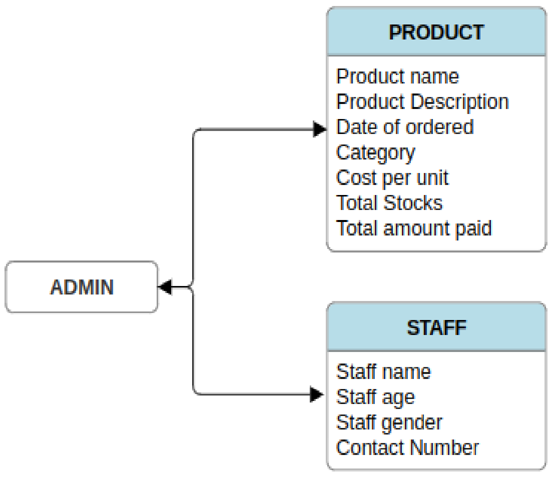


FIGURE 4.4 DATABASE DESIGN

4.4 OUTPUT DESIGN

Output design generally refers to the results and information that are generated by the system for many end-users; it should be understandable with the enhanced format. Previewing the output reports by the user is extremely important because the user is the ultimate judge of the quality of the output and, in turn, the success of the system. The output is designed in such a way that it is attractive, convenient and informative.

CHAPTER 5
TESTING

5.1 UNIT TESTING

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. Unit testing can be done manually, but is often automated. In this process, each module is considered as individual units and are tested for proper operation. If each module meets up with the user's requirement, then it is subjected to integration testing where more than one module is integrated and tested.

5.2 REGRESSION TESTING

Regression testing helps in identifying any unintended side effects or issues that may arise due to code changes,updates to dependencies, or modifications in the infrastructure By rerunning test cases, the development team can ensure that previously implemented features and functionality have not been negatively impacted by recent updates.

The primary objective of regression testing is to ensure that the application remains fully functional, reliable, and user-friendly, even after modifications or enhancements have been made.

Overall, regression testing serves as an essential quality assurance measure to maintain the integrity and performance of the application throughout its lifecycle. It helps in minimizing the risk of functional regressions, ensuring that the website continues to meet the expectations and requirements of the end users and stakeholders.

5.3 VALIDATION TESTING

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements. Validation Testing ensures that the service actually meets the client's needs. It can also be defined as to demonstrate that the service fulfills its intended use when deployed in an appropriate environment.

5.4 VERIFICATION TESTING

Verification is the process of evaluating work-service of a development phase to determine whether they meet the specified requirements. When developing each module, the individuality of the service is checked at its development stage. Thus, the modules must be verified at the development stage.

5.5 INTEGRATION TESTING

Integration tests are designed to test the integrated software components to determine if they actually run as a program. It is specifically aimed at exposing problems that arise from the combination of components. Integration testing is done after integration of the model with the core service. The integration testing can be done for our project by integrating the user module.

CHAPTER 6
CONCLUSION AND FUTURE WORK

Best Collection Mens Wear specializes in selling high-quality branded clothing for men, kids, and boys, providing a diverse selection to suit every style. With a focus on quality and fashion trends, they aim to offer customers a unique and satisfying shopping experience.The project involves managing stocks of the products and staff details of Best Collection Mens Wear shop through mobile application. The application has been tested well, and end users' satisfaction is high. In the future, the project can be enhanced with additional features based on user requests, such as enabling users to buy products through the application.

APPENDIX 1

CODING:

main.dart:

```
import 'package:flutter/material.dart';
import 'productlist.dart';
import 'addproduct.dart';
import 'staff.dart';
import 'addstaff.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const Home(),    );  }
class Home extends StatefulWidget {
  const Home({super.key});
  @override
  State<Home> createState() => _HomeState();
}
class _HomeState extends State<Home> {
```

```
@override
 Widget build(BuildContext context) {
   return Scaffold(
     body: SingleChildScrollView(
       child: Container(
         color: Color(0xFFFFCD4E),
         width: MediaQuery.of(context).size.width,
         child: Column(
           children: [
             Image.asset("images/desgin.png",height: 360,width: 300,),
             Padding(
               padding: const EdgeInsets.only(top: 15),
               child: Container(
                 width: MediaQuery.of(context).size.width,
                   height: 473,
                 child: Column(
                   children: [
                     Padding(
                       padding: const EdgeInsets.only(top:20),
                       child: Row(
                         children: [
                       Padding(
                             padding: const EdgeInsets.all(30),
                             child: GestureDetector(
                               onTap: ()=>{
                                 Navigator.push(
           context,MaterialPageRoute(builder: (context) => const AddProductListPage()),)
                               },
                               child: Container
```

```
      width: MediaQuery.of(context).size.width*0.3,
     height: 150,
    decoration: BoxDecoration(
    color: Colors.white,
    borderRadius: BorderRadius.circular(10),
    boxShadow: [
      BoxShadow(
        color: Colors.black.withOpacity(0.2),
        spreadRadius: 5,
        blurRadius: 10,
        offset: Offset(0, 3), // changes position of shadow
        ),
      ],
    ),
     child: Padding(
       padding: const EdgeInsets.all(8.0),
       child: Column(children: [
        Image.asset("images/addproduct.png",height: 100,width:150,),
        Padding(
          padding: const EdgeInsets.only(top: 5),
                    child: Text("Add Product",style: TextStyle(fontWeight:
FontWeight.bold),),
        )
       ],),
     )
 Padding(
padding: const EdgeInsets.only(right:30),
child: GestureDetector(
  onTap: () => {
```

```
                        Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => ProductListPage()),
        )
                      },
                      child: Container(
                        width: MediaQuery.of(context).size.width*0.3,
                        height: 150,

                       decoration: BoxDecoration(
                       color: Colors.white,
                       borderRadius: BorderRadius.circular(10),
                       boxShadow: [
                         BoxShadow(
                           color: Colors.black.withOpacity(0.2),
                           spreadRadius: 5,
                           blurRadius: 10,
                           offset: Offset(0, 3), // changes position of shadow
                           ),
                         ],
                       ),
                        child: Padding(
                          padding: const EdgeInsets.all(8.0),
                          child: Column(children: [
                           Image.asset("images/productlist.png",height: 100,width:100,),
                           Padding(
                             padding: const EdgeInsets.only(top: 5),
                                         child: Text("Product List",style: TextStyle(fontWeight:
FontWeight.bold  mainAxisAlignment: MainAxisAlignment.spaceBetween,
        ),
       ),
       Row(
        children: [


         Padding(
          padding: const EdgeInsets.all(30),
          child : GestureDetector(
           onTap: ()=>{
              Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => const AddStaff()),
)
           },
          child: Container(
            width: MediaQuery.of(context).size.width*0.3,
            height: 150,
                    decoration: BoxDecoration(
           color: Colors.white,
           borderRadius: BorderRadius.circular(10),
           boxShadow: [
            BoxShadow(
             color: Colors.black.withOpacity(0.2),
             spreadRadius: 5,
             blurRadius: 10,
             offset: Offset(0, 3), // changes position of shadow
             ),
            ],
           ),
                    child: Padding(
                        padding: const EdgeInsets.all(8.0),
                        child: Column(children: [
                         Image.asset("images/addstaff.png",height: 100,width:100,),
                         Padding(
                           padding: const EdgeInsets.only(top: 5),
                                            child: Text("Add Staff",style: TextStyle(fontWeight:
FontWeight.bold),),
                          )
                   Padding(
                 padding: const EdgeInsets.only(right:30),
                 child : GestureDetector(onTap: ()=>{
                     Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => const ViewStaff()),
  )
                 },
                 child: Container(
                    width: MediaQuery.of(context).size.width*0.3,
                    height: 150,


                   decoration: BoxDecoration(
                   color: Colors.white,
                   borderRadius: BorderRadius.circular(10),
                   boxShadow: [
                     BoxShadow(
                       color: Colors.black.withOpacity(0.2),
                       spreadRadius: 5,
                       blurRadius: 10,
                          offset: Offset(0, 3), // changes position of shadow
                          ),
                         ],
                        ),
                        child: Padding(
                          padding: const EdgeInsets.all(8.0),
                          child: Column(children: [
                           Image.asset("images/viewstaff.png",height: 100,width:100,),
                           Padding(
                             padding: const EdgeInsets.only(top: 5),
                                        child: Text("Staff List",style: TextStyle(fontWeight:
FontWeight.bold),),
                   mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  )
                 ]
                ),
                decoration: BoxDecoration(
                     borderRadius: BorderRadius.only(topLeft: Radius.circular(20),topRight:
Radius.circular(20)),
                 color: Colors.white
```

productlist.dart:

```
import 'package:flutter/material.dart';
import 'package:dio/dio.dart;
class ProductListPage extends StatefulWidget {
 const ProductListPage({Key? key}) : super(key: key);
  @override
  _ProductListPageState createState() => _ProductListPageState();
}
class _ProductListPageState extends State<ProductListPage> {
  List<Map<String, dynamic>> productList = []
```

```
@override
void initState() {
  super.initState();
  _fetchProducts();
}
Future<void> _fetchProducts() async {
  final dio = Dio();
  try {
    final response = await dio.get('https://consultancy-server.onrender.com/getproduct');
    if (response.statusCode == 200) {
      final List<dynamic> data = response.data;
      setState(() {
        productList = List<Map<String, dynamic>>.from(data);
      });
    } else {
      throw Exception('Failed to load products: ${response.statusCode}');
    }
  } catch (e) {
    print('Error fetching products: $e');
  }
}
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Product List'),
    ),
    body: ListView.builder(
      itemCount: productList.length,
      itemBuilder: (context, index) {
        final product = productList[index];
        return ListTile(
          title: Text(product['productname'] ?? 'No Name'),
          subtitle: Text('Price: ${product['costperunit'] ?? '0.0'}'),
          onTap: () {
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => ProductDetailPage(product: product
class ProductDetailPage extends StatelessWidget {
  final Map<String, dynamic> product;

  const ProductDetailPage({Key? key, required this.product}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(product['productname'] ?? 'No Product Name'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text('Name: ${product['productname'] ?? 'No Name'}'),
            Text('Description: ${product['description'] ?? 'No Description'}'),
            Text('Category: ${product['category'] ?? 'No Category'}'),
            Text('OrderedDate: ${product['date'] ?? 'No Date'}'),
            Text('Price: ${product['costperunit'] ?? '0.0'}'),
            Text('Stock: ${product['totalstock'] ?? '0'}'
void main() {
  runApp(MaterialApp(
    home: ProductListPage(),
  ));
}
```

addproduct.dart:

```
import 'package:flutter/material.dart';
import 'package:dio/dio.dart';

class AddProductListPage extends StatefulWidget {
  const AddProductListPage({Key? key}) : super(key: key);

  @override
  _AddProductListPageState createState() => _AddProductListPageState();
}

class _AddProductListPageState extends State<AddProductListPage> {
  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
  DateTime? _selectedDate;
  TextEditingController _descriptionController = TextEditingController();
  TextEditingController _costPerUnitController = TextEditingController();
  TextEditingController _totalStockController = TextEditingController();
  TextEditingController _totalAmountPaidController = TextEditingController();
  String? _selectedCategory;
  String? _selectedProductName;
```

```
 void _addProduct(productName, description, date, costPerUnit, totalStock,
     totalAmountPaid, selectedCategory) async {
   final dio = Dio();
   try {
     var bodyValues = {
       "productname": productName,
       "description": description,
       "date": date,
       "category": selectedCategory,
       "costperunit": costPerUnit,
       "totalstock": totalStock,
       "totalamount": totalAmountPaid
     };

       final response = await dio.post('https://consultancy-server.onrender.com/addproduct',
data: bodyValues);
     if (response.statusCode == 200) {
       // Show success message
       ScaffoldMessenger.of(context).showSnackBar(
         const SnackBar(
           content: Text('Product added successfully!'),
           duration: Duration(seconds: 3),
         ),
       );
        _formKey.currentState?.reset();
     }
   } catch (e) {
     print('Error connecting to the server: $e');
   }
```

```
}

@override
Widget build(BuildContext context) {
 return Scaffold(
   appBar: AppBar(
     title: Text("Add Products"),
   ),
   body: SingleChildScrollView(
     child: Padding(
       padding: const EdgeInsets.all(20.0),
       child: Form(
         key: _formKey,
         child: Column(
           crossAxisAlignment: CrossAxisAlignment.start,
           children: <Widget>[
             DropdownButtonFormField<String>(
               value: _selectedProductName,
               decoration: InputDecoration(
                 labelText: 'Product Name',
               ),
               onChanged: (value) {
                 setState(() {
                   _selectedProductName = value;
                 });
               },
               items: ['Shirt', 'T-shirt', 'Trousers', 'Shorts', 'Innerwear']
                   .map<DropdownMenuItem<String>>((String value) {
                 return DropdownMenuItem<String>(
        value: value,
        child: Text(value),
      );
    }).toList(),
  ),
  SizedBox(height: 20),
  TextFormField(
    controller: _descriptionController,
    decoration: InputDecoration(
      labelText: 'Description',
    ),
  ),
  SizedBox(height: 20),
  TextFormField(
    readOnly: true,
    onTap: () => _selectDate(context),
    decoration: InputDecoration(
      labelText: 'Date',
    ),
    controller: _selectedDate != null
        ? TextEditingController(
                                                              text:
'${_selectedDate!.year}-${_selectedDate!.month}-${_selectedDate!.day}')
        : null,
  ),
  SizedBox(height: 20),
  DropdownButtonFormField<String>(
    value: _selectedCategory,
    decoration: InputDecoration(
    labelText: 'Category',
  ),
  onChanged: (value) {
    setState(() {
      _selectedCategory = value;
    });
  },
  items: ['Kids', 'Mens']
      .map<DropdownMenuItem<String>>((String value) {
    return DropdownMenuItem<String>(
      value: value,
      child: Text(value),
    );
  }).toList(),
),
SizedBox(height: 20),
TextFormField(
  controller: _costPerUnitController,
  keyboardType: TextInputType.number,
  decoration: InputDecoration(
    labelText: 'Cost Per Unit',
  ),
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter cost per unit';
    }
    return null;
  },
),
SizedBox(height: 20),
TextFormField(
  controller: _totalStockController,
  keyboardType: TextInputType.number,
  decoration: InputDecoration(
    labelText: 'Total Stock',
  ),
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter total stock';
    }
    return null;
  },
),
SizedBox(height: 20),
TextFormField(
  controller: _totalAmountPaidController,
  keyboardType: TextInputType.number,
  decoration: InputDecoration(
    labelText: 'Total Amount Paid',
  ),
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter total amount paid';
    }
    return null;
  },
),
SizedBox(height: 20),
ElevatedButton(
  onPressed: () {
    if (_formKey.currentState!.validate()) {
      String productName = _selectedProductName!;
      String description = _descriptionController.text;
      String? date = _selectedDate != null
                                                                              ?
'${_selectedDate!.year}-${_selectedDate!.month}-${_selectedDate!.day}'
          : null;
      String costPerUnit = _costPerUnitController.text;
      String totalStock = _totalStockController.text;
      String totalAmountPaid = _totalAmountPaidController.text;

      print('Product Name: $productName');
      print('Description: $description');
      print('Date: $date');
      print('Cost Per Unit: $costPerUnit');
      print('Total Stock: $totalStock');
      print('Total Amount Paid: $totalAmountPaid');
      print('Category: $_selectedCategory');
      _addProduct(productName, description, date, costPerUnit,
          totalStock, totalAmountPaid, _selectedCategory);
      setState(() {
        _selectedProductName = null;
        _selectedCategory = null;
      });
    }
  },
  child: Text('Submit'),
              ),
            ],
          ),
        ),
      ),
    ),
  );
}

Future<void> _selectDate(BuildContext context) async {
  final DateTime? picked = await showDatePicker(
    context: context,
    initialDate: _selectedDate ?? DateTime.now(),
    firstDate: DateTime(2000), // Adjust the start date as needed
    lastDate: DateTime.now(),
  );
  if (picked != null && picked != _selectedDate) {
    setState(() {
      _selectedDate = picked;
    });
  }
}

@override
void dispose() {
  _descriptionController.dispose();
  _costPerUnitController.dispose();
  _totalStockController.dispose();
  _totalAmountPaidController.dispose();
```

```
    super.dispose();
  }
 }
```

DATABASE MODEL FOR PRODUCT:

```
const mongoose=require('mongoose')

const ProductSchema=new mongoose.Schema({
  productname:{
    type:String,
    required:true,
    trim:true,
    maxLength:50
  },
  description:{
    type:String,
    required:true,
    maxLength:30,
    trim:true
  },
  date:{
    type:String,
    required:true
  },
  category:{
    type:String,
    required:true,
  },
  costperunit:{
    type:String,
    required:true,
  },
  totalstock:{
    type:String,
    required:true,

  },
  totalamount:{
    type:String,
    required:true,

  }

},{timestamps:true})
module.exports=mongoose.model('products',ProductSchema);
```

DATABASE MODEL FOR STAFF:

```
const mongoose=require('mongoose')

const StaffSchema=new mongoose.Schema({
  staffname:{
    type:String,
    required:true,
    trim:true,
    maxLength:50
  },
  age:{
    type:String,
    required:true,
    maxLength:20,
    trim:true
  },
  gender:{
    type:String,
    required:true
  },
  contact:{
    type:String,
    required:true,
    trim:true
  }
})
module.exports=mongoose.model('staffs',StaffSchema);
```

route.js:

```
const express=require('express');
const router= express.Router();
const {addproduct,getproduct,addstaff,getstaff} = require('../controllers/controller');
router.post('/addproduct',addproduct)
router.get('/getproduct',getproduct)
router.post('/addstaff',addstaff)
router.get('/staffdetails',getstaff)
module.exports=router;
```

controller.js

```
const StaffSchema= require('../model/staff')
const ProductSchema = require('../model/product')


//add new product entry in the product table
```

```
const addproduct= async(req,res)=>{
                                                                                    const
{productname,description,date,category,costperunit,totalstock,totalamount}=req.body;
    const product= ProductSchema({
      productname,
      description,
      date,
      category,
      costperunit,
      totalstock,
      totalamount
    })
    try {
       await product.save()
       res.status(200).json({ message: 'success'});
    } catch (error) {
      return res.status(500).json({ error: 'Internal server error' });
    }


}
//get the product entries from the product table
const getproduct= async(req,res)=>{
  try {
    const products = await ProductSchema.find().sort({createdAT:-1});
        if (products.length > 0) {
            res.status(200).json(products);
        } else {
            res.status(401).json({ error: 'Empty products' }}}
  } catch (error) {
    return res.status(500).json({ error: 'Internal server error' });
  }}
//add new staff entry in the staff table
const addstaff= async(req,res)=>{
  const {staffname,age,gender,contact}=req.body;
  const staff=StaffSchema({staffname,age,gender,contact})
  try {
    await staff.save()
    res.status(200).json({ message: 'success'});
  } catch (error) {
    return res.status(500).json({ error: 'Internal server error' });
  }
}
//get the staff entries from the product table
const getstaff= async(req,res)=>{
  try {
    const products = await StaffSchema.find();
        if (products.length > 0) {
            res.status(200).json(products);
        } else {
            res.status(401).json({ error: 'Empty staffs' });
        }
  } catch (error) {
    return res.status(500).json({ error: 'Internal server error' });
  }
}
module.exports={addproduct,getproduct,addstaff,getstaff};
```

CONNECT DATABASE

```
const mongoose = require('mongoose');
const db = async () => {
   try {
      mongoose.set('strictQuery', false)
      await mongoose.connect(process.env.MONGO_URL)
      console.log('Db Connected')
   } catch (error) {
      console.log('DB Connection Error');
   }
}
module.exports = {db}
```

MAIN SERVER CODE

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

const { db } = require('./db/db');
const app = express()
require('dotenv').config()
const port = 3000;
const router = require('./routes/route')
app.use(express.json());
app.use(cors())
app.use(router)
const server = () => {
   db()
   app.listen(port, () => {
      console.log('listening to port:',port)
   })}
server()
```

GITHUB LINK : https://github.com/kamalj57/Consultancy