INVENTORY MANAGER APPLICATION

A MINI PROJECT REPORT

Submitted by

Devam [RA2011003010064]

Ashwin Prabhakar [RA2011003010036]

Kritika Sinha [RA2011003010045]

Under the guidance of

Mr. Muruganantham B

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING
With specialization in Software Engineering

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603203 APRIL 2023

COLLEGE OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
S.R.M. NAGAR, KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this project report **“Inventory Manager Application”** is the Bonafide work of **“Rishi Prashar (RA2011033010176) and Ridhima Mehrotra (RA2011033010157)”** of III Year/VI Sem B.Tech(CSE-SWE) who carried out the mini project work under my supervision for the course 18CSC303J- Database Management systems in SRM Institute of Science and Technology during the academic year 2022-2023(Even Sem).

SIGNATURE

DR S. Vimal
Assist Professor
CINTEL

ABSTRACT

The project is a rudimentary inventory app designed to store the names and quantities of products. When completed, the app will provide the ability to add, delete and search for database entries while also displaying a scrollable list of all products currently stored in the database.

This project uses SQLite-based database storage
using the Room persistence library. This product uses the tools like Android Studio and Dart as the language and the Dart library Flutter. This product list will update automatically as
database entries are added or deleted.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AES	Advanced Encryption Standard
ANN	Artificial Neural Network
CSS	Cascading Style Sheet
CV	Computer Vision
DB	Data Base
DNA	Deoxyribo Neucleic Acid
SQL	Structured Query Language
SVM	Support Vector Machine
UI	User Interface

CHAPTER1

INTRODUCTION

1.1 Introduction

Inventory management is an essential process for any business that involves tracking and managing the flow of goods and materials in and out of the organization. Proper inventory management ensures that businesses always have the right amount of stock on hand to meet customer demand while minimizing waste and costs.

An inventory management application is a software tool designed to streamline and automate inventory management processes. It allows businesses to track inventory levels, manage stock across multiple locations, and optimize purchasing and replenishment activities.
This application provides real-time visibility into inventory levels and allows businesses to make data-driven decisions about when to order new stock, how much to order, and which suppliers to use. It can also help to reduce errors and improve efficiency by automating manual processes, such as stock counting and order processing.

Overall, an inventory management application is a powerful tool that can help businesses to save time, reduce costs, and improve customer satisfaction by ensuring that the right products are available at the right time.

1.2 Problem Statement

Despite the benefits of proper inventory management, many businesses still struggle with inefficient and error-prone processes. Common challenges include inaccurate inventory tracking, stockouts, overstocking, and difficulties managing inventory across multiple locations. These issues can result in lost sales, wasted resources, and increased costs.

Additionally, many businesses still rely on manual processes for inventory management, such as paper-based record keeping and manual data entry, which can be time-consuming and error-prone. This can lead to a lack of visibility into inventory levels, delays in order processing, and difficulties in forecasting demand.

Therefore, there is a need for a reliable and efficient inventory management application that can streamline inventory management processes, provide real-time visibility into inventory levels, and optimize purchasing and replenishment activities. The application should be user-friendly and intuitive, allowing businesses to easily track inventory across multiple locations, automate manual processes, and make data-driven decisions about inventory management.

Scope and Applications

The scope of an inventory management application is broad and can be applied to a variety of industries and businesses of all sizes. It can be used in retail, manufacturing, healthcare, hospitality, and many other industries that require tracking and managing inventory.

The application can be used to manage all aspects of inventory, including tracking stock levels, managing stock across multiple locations, and optimizing purchasing and replenishment activities. It can also automate manual processes such as stock counting, order processing, and vendor management.

In addition, an inventory management application can provide real-time data and analytics to help businesses make data-driven decisions about inventory management, such as identifying slow-moving products, forecasting demand, and adjusting stock levels to match demand.
Other potential applications of an inventory management application include:

- Barcode scanning and RFID technology to automate the tracking of inventory in real-time
- Integration with point-of-sale systems to track sales data and automatically adjust inventory levels
- Automatic reorder notifications to ensure that businesses never run out of stock
- Integration with accounting software to streamline financial reporting and inventory valuation
- Mobile access for remote inventory management and monitoring

Overall, an inventory management application can help businesses to streamline processes, reduce costs, and improve efficiency by providing real-time visibility into inventory levels and automating manual processes.

General and unique services in the database application

General Services in Database Application:

- Database application provides the fundamental service of storing and retrieving data efficiently. It allows users to create, store, update, and delete data in a structured manner, making it easily accessible for various operations and applications.
- Database applications offer services for organizing and managing data in a structured manner, including defining data types, creating tables, establishing relationships between tables, and enforcing data integrity constraints. This ensures that data is organized, consistent, and accurate.
- Database applications provide services for securing data and controlling access to it. This includes features such as authentication, authorization, and encryption to ensure that data is protected from unauthorized access and tampering.

Unique Services in Database Application:

- Some database applications may offer the ability to define custom data models or schemas, allowing users to create unique data structures tailored to their specific requirements. This provides flexibility in designing the database according to the unique needs of the application or business
- Some database applications may provide advanced analytics and data mining capabilities, allowing users to analyze data trends, patterns, and relationships, and gain insights from the data stored in the database. This can be particularly useful for businesses that require data-driven decision making
- Certain database applications may offer integration capabilities with other systems, such as APIs (Application Programming Interfaces) or data connectors, allowing users to easily exchange data with other applications or systems. This can streamline data flow and enable seamless integration with other business processes.

1.3 Software requirement specification

Introduction: The purpose of this software requirement specification (SRS) is to define the requirements for an inventory management application. The application is designed to help businesses manage inventory levels, track stock across multiple locations, and optimize purchasing and replenishment activities.

Functional Requirements:

2 Inventory Tracking: The system shall allow users to track inventory levels for each product in real-time.

3 Multiple Location Management: The system shall allow users to manage inventory across multiple locations.

4 Purchasing and Replenishment: The system shall optimize purchasing and replenishment activities based on real-time inventory levels.

5 Barcode Scanning: The system shall support barcode scanning and RFID technology to automate inventory tracking.

6 Point of Sale Integration: The system shall integrate with point-of-sale systems to track sales data and automatically adjust inventory levels.

7 Automatic Reorder Notifications: The system shall send automatic reorder notifications to ensure that businesses never run out of stock.

8 Vendor Management: The system shall allow users to manage vendor information and automate the ordering process.

9 Analytics and Reporting: The system shall provide real-time data and analytics to help businesses make data-driven decisions about inventory management.

Non-Functional Requirements:

10 Performance: The system shall be able to handle high volumes of data and transactions without significant performance degradation.

11 Security: The system shall ensure that inventory data is stored securely and is only accessible to authorized users.

12 Scalability: The system shall be scalable to accommodate the growth of the business and additional inventory items.

13 User Interface: The system shall provide a user-friendly interface that is

intuitive and easy to use.

14 Availability: The system shall be available 24/7 with minimal downtime for maintenance.

Constraints:

15 The system shall be designed to run on a variety of platforms and operating systems.

16 The system shall be designed to work with a variety of hardware and software configurations.

17 The system shall comply with applicable industry standards and regulations.

Assumptions:

18 The users of the system have basic computer literacy and are familiar with inventory management processes.

19 The system will be deployed in a stable and secure environment.

Conclusion: The inventory management application shall be designed to provide businesses with real-time visibility into inventory levels, automate manual processes, and optimize purchasing and replenishment activities. The system shall be scalable, secure, and user-friendly, and shall comply with industry standards and regulations.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing Systems

There are several existing inventory management systems available in the market.

Some of the popular ones include:

QuickBooks: QuickBooks is a popular accounting software that includes inventory management features. It allows businesses to track inventory levels, set reorder points, and create purchase orders. QuickBooks also integrates with point-of-sale systems and provides basic reporting capabilities.

Zoho Inventory: Zoho Inventory is a cloud-based inventory management system that allows businesses to manage stock across multiple locations, track inventory levels in real-time, and automate purchasing and replenishment. It also includes barcode scanning, vendor management, and reporting features.

Fishbowl: Fishbowl is an inventory management system that integrates with QuickBooks and other accounting software. It includes features such as inventory tracking, barcode scanning, order management, and reporting.

Trade Gecko: Trade Gecko is a cloud-based inventory management system that allows businesses to manage inventory across multiple channels and locations. It includes features such as order management, shipping and fulfillment, and reporting.

Odoo: Odoo is an open-source inventory management system that allows businesses to manage inventory, sales, and purchasing in one platform. It includes features such as barcode scanning, vendor management, and reporting.

Each of these systems has its own strengths and weaknesses, and the choice of an inventory management system will depend on the specific needs and requirements of the business.

2.2 Comparison of existing vs proposed system

Existing inventory management systems and the proposed inventory management application have similarities and differences in terms of their features, functionalities, and capabilities. Here is a comparison of the two:

Similarities:

Inventory Tracking: Both existing systems and the proposed application allow businesses to track inventory levels for each product in real-time.

Multiple Location Management: Both systems allow businesses to manage inventory across multiple locations.

Purchasing and Replenishment: Both systems optimize purchasing and replenishment activities based on real-time inventory levels.

Barcode Scanning: Both systems support barcode scanning and RFID technology to automate inventory tracking.

Reporting: Both systems provide reporting capabilities to help businesses make data-driven decisions about inventory management.

Differences:

Customization: The proposed inventory management application may be more customizable than existing systems, allowing businesses to tailor the application to their specific needs and requirements.

Point of Sale Integration: The proposed inventory management application may provide more robust integration with point-of-sale systems, allowing businesses to track sales data and automatically adjust inventory levels in real-time.

Automatic Reorder Notifications: The proposed inventory management application may provide more advanced automatic reorder notifications, allowing businesses to set specific rules for when and how to reorder inventory.

Mobile Access: The proposed inventory management application may provide mobile access for remote inventory management and monitoring, while some existing systems may not have this feature.

Scalability: The proposed inventory management application may be more scalable than existing systems, allowing businesses to accommodate the growth of their business and additional inventory items.

Overall, the proposed inventory management application may provide more flexibility, customization, and scalability than existing systems, while also providing advanced features such as point of sale integration, mobile access, and automatic reorder notifications. However, the specific choice of system will depend on the needs and requirements of the business.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 Architecture Diagram

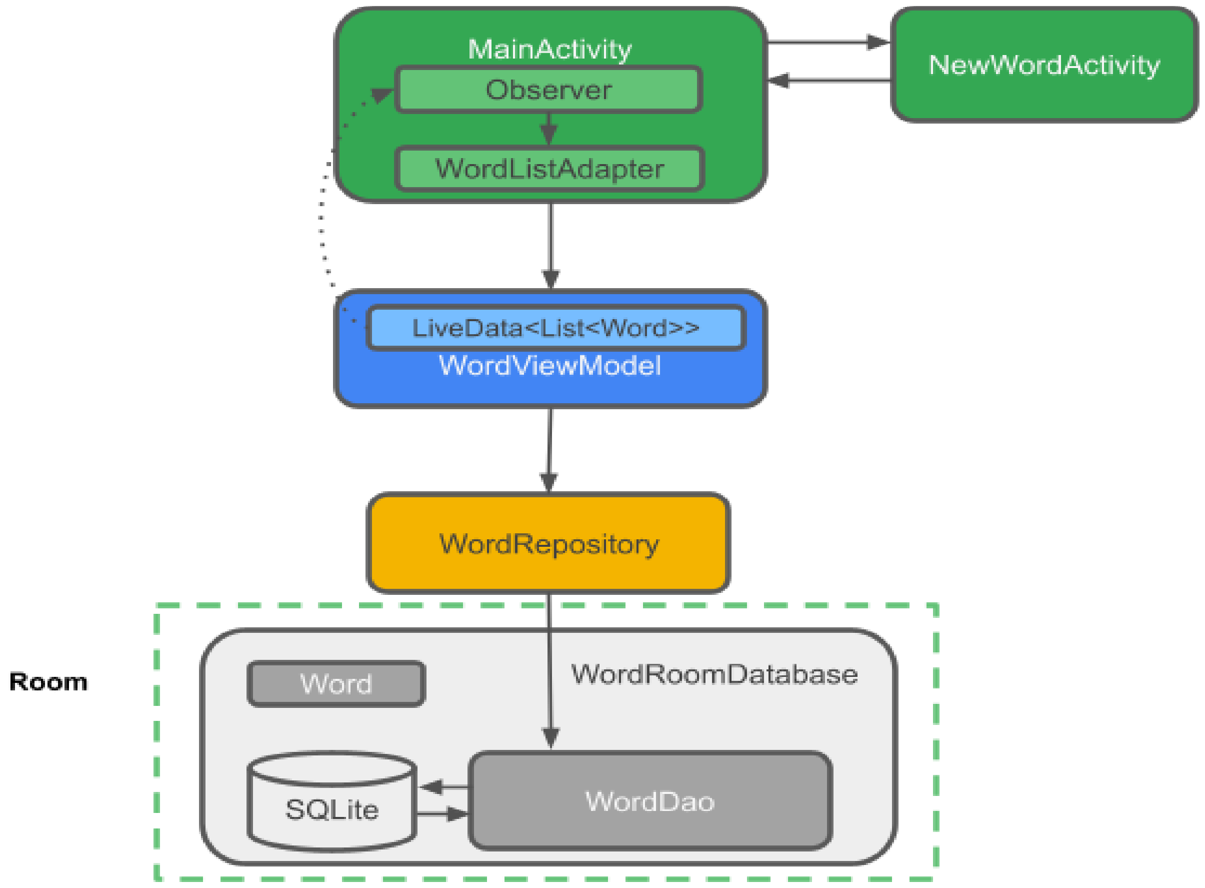
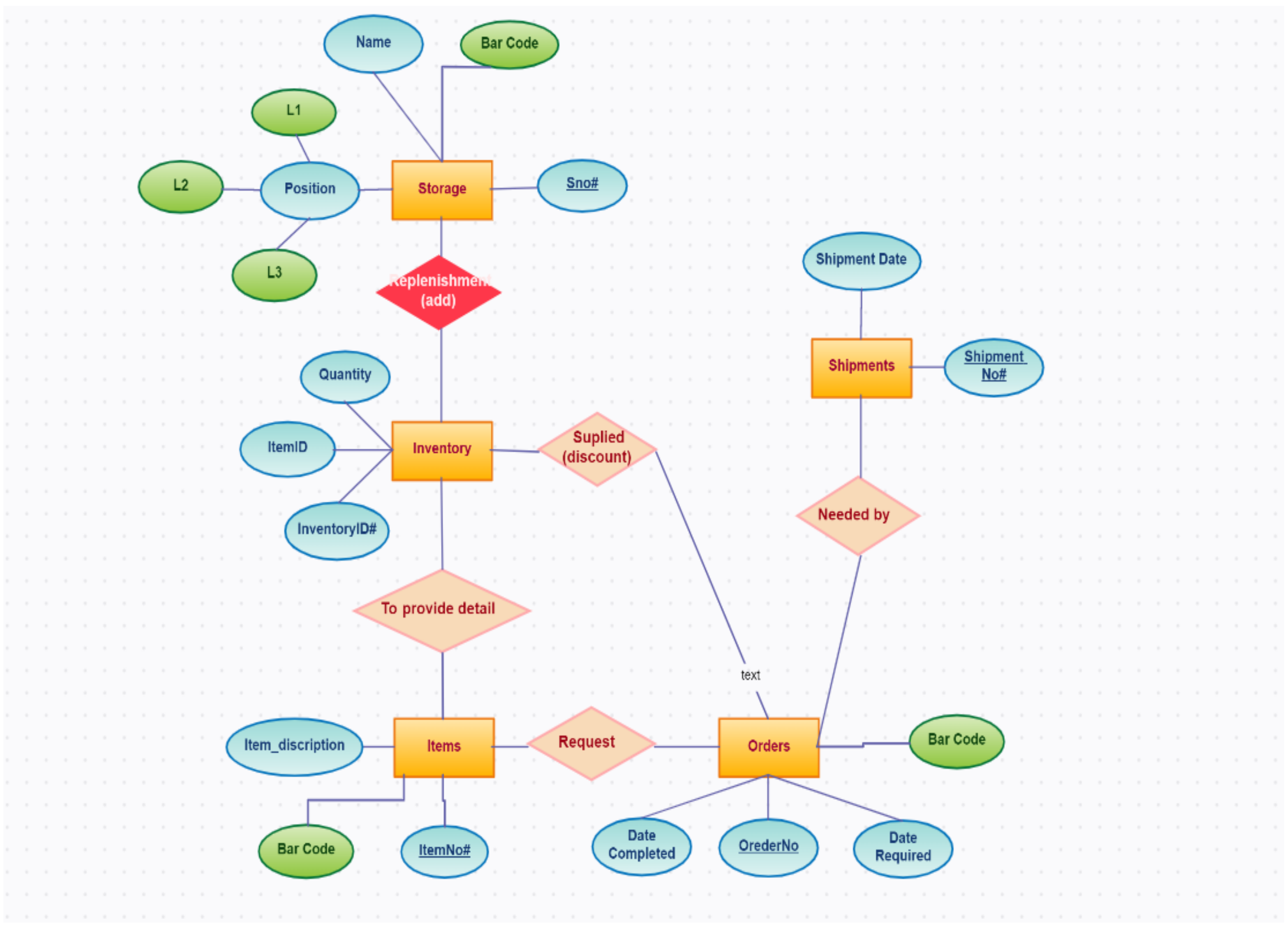
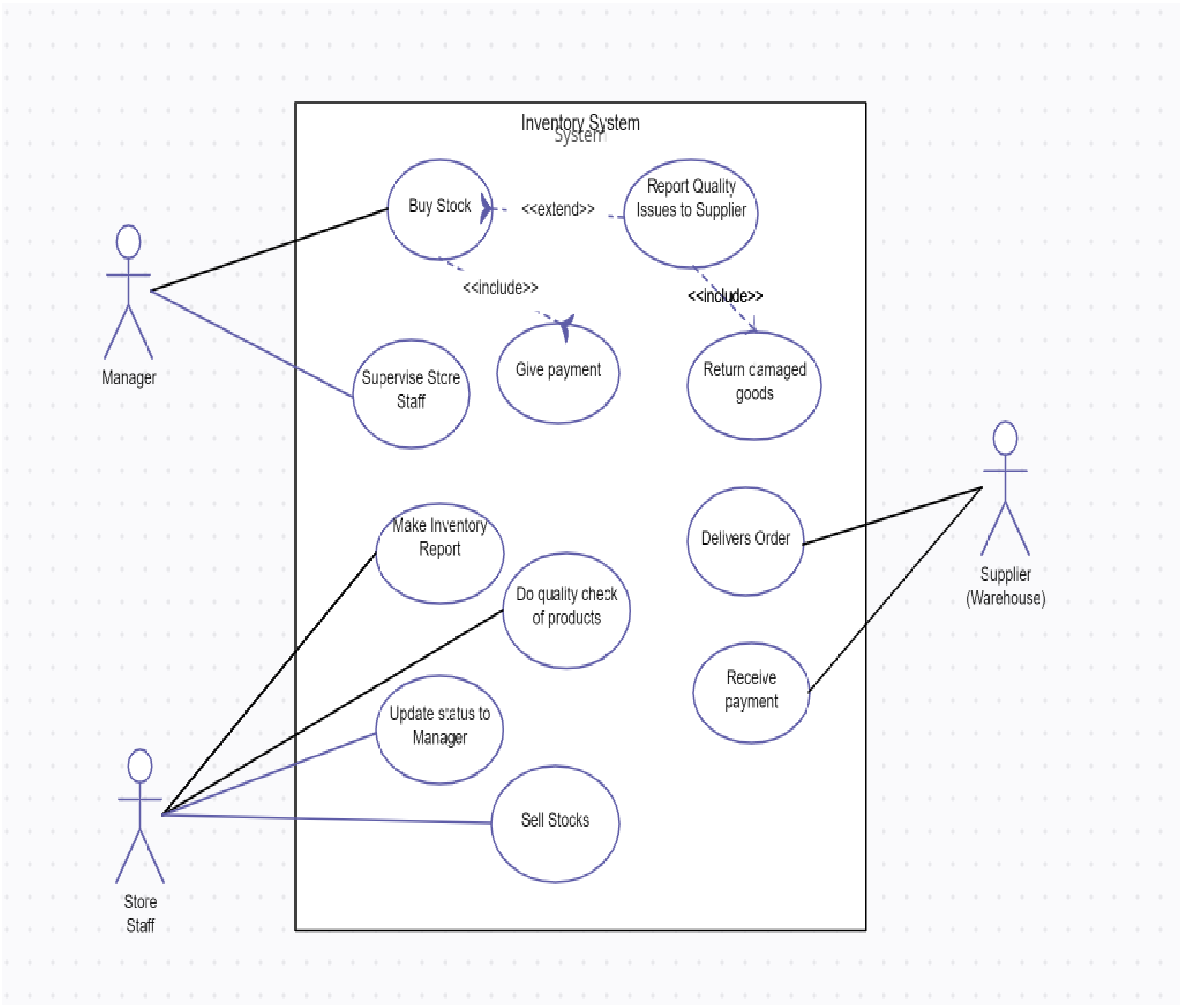


Fig 3.1 Architecture diagram

3.2 ER Diagram and Use case Diagram


Name
Bar Code
L1
L2
Position
Storage
Sno#
L3
Replenishment (add)
Shipment Date
Shipments
Shipment No#
Quantity
ItemID
Inventory
Suplied (discount)
InventoryID#
Needed by
To provide detail
text
Item_discription
Items
Request
Orders
Bar Code
Bar Code
ItemNo#
Date Completed
OrederNo
Date Required



Inventory System
System
Buy Stock
<<extend>>
Report Quality Issues to Supplier
<<include>>
<<include>>
Manager
Supervise Store Staff
Give payment
Return damaged goods
Make Inventory Report
Delivers Order
Supplier (Warehouse)
Do quality check of products
Receive payment
Update status to Manager
Sell Stocks
Store Staff

CHAPTER 4
MODULES AND FUNCTIONALITIES

4.1 Modules

Here are some modules that can be included in an Inventory Management Application:

Product Management Module: This module allows the user to add, edit, and delete products, along with their attributes such as name, SKU, description, price, and quantity. It also provides a view of the product inventory level.

Order Management Module: This module tracks customer orders and allows the user to create, edit, and fulfill orders. It includes order history and status updates for the user.

Purchase Management Module: This module tracks purchase orders and allows the user to create, edit, and receive purchase orders. It includes purchase order history and status updates for the user.

Stock Management Module: This module manages stock levels and tracks stock movements such as sales, purchases, and returns. It includes real-time inventory tracking and allows the user to set reorder points and receive low stock alerts.

Warehouse Management Module: This module manages multiple warehouses, locations, and stock transfers. It allows the user to track inventory levels across multiple locations and manage stock transfers between them.

Reporting and Analytics Module: This module provides the user with reports and analytics such as inventory levels, stock movement, sales, and purchases. It also includes business intelligence tools to help the user make data-driven decisions.

Barcode Scanning and RFID Module: This module enables the user to scan barcodes or use RFID technology to track inventory levels, movements, and locations.

User Management Module: This module manages user roles, permissions, and access to specific features and data within the application.

Integration Module: This module allows the Inventory Management Application to integrate with other applications such as accounting software, point of sale systems, and ecommerce platforms.

These modules can be customized and adapted to suit the specific needs and requirements of the business.

Here are some of the key functionalities that can be included in an Inventory Management Application:

Real-time Inventory Tracking: The application should provide real-time tracking of inventory levels for each product, across multiple warehouses or locations. This allows businesses to make informed decisions about restocking, pricing, and other inventory-related activities.

Purchase Order Management: The application should allow businesses to create, edit, and manage purchase orders with suppliers. This includes tracking order status, receiving items, and managing payments.

Sales Order Management: The application should allow businesses to manage customer orders, including tracking order status, shipping, and returns.

Barcode Scanning and RFID Technology: The application should support barcode scanning and RFID technology to automate inventory tracking and improve accuracy.

Stock Movement Management: The application should provide tools for managing stock movements such as transfers between warehouses or locations, returns, and adjustments.

Inventory Optimization: The application should provide features for optimizing inventory levels, including setting reorder points, automatic reorder notifications, and managing safety stock levels.

Reporting and Analytics: The application should provide reports and analytics on inventory levels, sales, purchases, and other inventory-related activities. This can help businesses make data-driven decisions about inventory management.

User Management: The application should provide user management features, including role-based access control, permissions management, and auditing capabilities.

Integration with Other Systems: The application should be able to integrate with other systems, such as accounting software, point of sale systems, and ecommerce platforms.

Mobile Access: The application should provide mobile access for remote inventory management and monitoring, allowing users to access the system from anywhere, at any time.

These are just some of the key functionalities that can be included in an Inventory Management Application. The specific features and functionalities will depend on the needs and requirements of the business.

3.3 Connectivity used for database access

The choice of connectivity for database access in an Inventory Management Application will depend on the specific requirements of the application, as well as the type and location of the database being used. Here are some common connectivity options that can be used for database access in an Inventory Management Application:

JDBC (Java Database Connectivity): JDBC is a Java-based API that allows Java applications to access various types of relational databases. It provides a standard way for Java applications to connect to databases, execute SQL statements, and retrieve data.

ODBC (Open Database Connectivity): ODBC is a platform-independent API that provides a standard way for applications to access databases. It allows applications to connect to various types of databases, execute SQL statements, and retrieve data.

ADO.NET (ActiveX Data Objects .NET): ADO.NET is a set of libraries provided by Microsoft for accessing databases from .NET applications. It allows applications to connect to various types of databases, execute SQL statements, and retrieve data.

ORM (Object-Relational Mapping) frameworks: ORM frameworks such as Hibernate, Entity Framework, and Rails Active Record provide a higher-level abstraction for database access. They allow developers to work with objects and classes rather than directly manipulating the database.

Web services: Web services such as RESTful APIs can be used to provide database access over the web. This allows applications to access the database from anywhere, using standard web protocols.

The choice of connectivity will depend on factors such as the type and location of the database being used, the programming language and framework being used, and the specific requirements of the application. It's important to choose a connectivity option that is reliable, efficient, and secure.

CHAPTER 5

CODING AND TESTING

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


import 'sql_helper.dart';


void main() {
  runApp(const MyApp());
}


class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);


  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        // Remove the debug banner
        debugShowCheckedModeBanner: false,
        title: 'Inventory Manager',
        theme: ThemeData(
          primarySwatch: Colors.orange,        ),
        home: const HomePage());
```

```
  }
}

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

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  // All journals
  List<Map<String, dynamic>> _journals = [];

  bool _isLoading = true;
  // This function is used to fetch all data from the database
  void _refreshJournals() async {
    final data = await SQLHelper.getItems();
    setState(() {
      _journals = data;
      _isLoading = false;
    });
  }

  @override
  void initState() {
    super.initState();
    _refreshJournals(); // Loading the diary when the app starts
```

```
final TextEditingController _titleController = TextEditingController();
final TextEditingController _descriptionController = TextEditingController();


// This function will be triggered when the floating button is pressed
// It will also be triggered when you want to update an item
void _showForm(int? id) async {
  if (id != null) {
    // id == null -> create new item
    // id != null -> update an existing item
    final existingJournal =
        _journals.firstWhere((element)  => element['id'] == id);
    _titleController.text = existingJournal['title'];
    _descriptionController.text = existingJournal['description'];
  }


  showModalBottomSheet(
      context: context,
      elevation: 5,
      isScrollControlled: true,
      builder: (_) => Container(
            padding: EdgeInsets.only(
              top: 15,
              left: 15,
              right: 15,
              // this will prevent the soft keyboard from covering the text fields
              bottom: MediaQuery.of(context).viewInsets.bottom + 120,
            ),
```

```
              mainAxisSize: MainAxisSize.min,
              crossAxisAlignment: CrossAxisAlignment.end,
children: [
  TextField(
    controller: _titleController,
    decoration: const InputDecoration(hintText: 'Title'),
  ),
  const SizedBox(
    height: 10,
  ),
  TextField(
    controller: _descriptionController,
    decoration: const InputDecoration(hintText: 'Description'),
  ),
  const SizedBox(
    height: 20,
  ),
  ElevatedButton(
    onPressed: () async {
      // Save new journal
      if (id == null) {
        await _addItem();
      }

      if (id != null) {
        await _updateItem(id);
      }
```

```
      _titleController.text = '';
      _descriptionController.text = '';

      // Close the bottom sheet
```

```
                    Navigator.of(context).pop();
                  },
                  child: Text(id == null ? 'Create New' : 'Update'),
                )
              ],
            ),
          ));
  }

// Insert a new journal to the database
  Future<void> _addItem() async {
    await SQLHelper.createItem(
        _titleController.text, _descriptionController.text);
    _refreshJournals();
  }

  // Update an existing journal
  Future<void> _updateItem(int id) async {
    await SQLHelper.updateItem(
        id, _titleController.text, _descriptionController.text);
    _refreshJournals();
  }

  void _deleteItem(int id) async {
```

```
   await SQLHelper.deleteItem(id);
    ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
        content: Text(
          'Successfully deleted a journal!',
        ),
        behavior: SnackBarBehavior.floating,
        backgroundColor: Colors.orange));
    _refreshJournals();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        toolbarHeight: 80,
        leading: IconButton(
          onPressed: () {
            ScaffoldMessenger.of(context).showSnackBar(SnackBar(
                content: Text("Use + button to add items"),
                behavior: SnackBarBehavior.floating,
                backgroundColor: Colors.orange));
          },
          icon: Image.asset(
            "assets/inventory-management.png",

            //scale: 100,
          ),
        centerTitle: true,
```

```
Title: const Text(
          'Inventory Manager',
          style: TextStyle(
            fontSize: 26,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
body: _isLoading
    ? const Center(
        child: CircularProgressIndicator(),
      )
    : ListView.builder(
        itemCount: _journals.length,
        itemBuilder: (context, index) => Card(
          color: Colors.orange[200],
          margin: const EdgeInsets.all(15),
          child: ListTile(
              title: Text(_journals[index]['title']),
              subtitle: Text(_journals[index]['description']),
              trailing: SizedBox(
                width: 100,
                child: Row(
                  children: [
                    IconButton(
                      icon: const Icon(Icons.edit),
                      onPressed: () => _showForm(_journals[index]['id']),
                    ),
```

```
                      icon: const Icon(Icons.delete),
                      onPressed: () =>
                          _deleteItem(_journals[index]['id']),
                    ),
                  ],
                ),
              )),
        ),
      ),
floatingActionButton: FloatingActionButton(
        child: const Icon(Icons.add),
        onPressed: () => _showForm(null),
      ),
    );
  }
}
```

Room Database:

```
import 'package:flutter/foundation.dart';
import 'package:sqflite/sqflite.dart' as sql;

class SQLHelper {
  static Future<void> createTables(sql.Database database) async {
    await database.execute("""CREATE TABLE items(
        id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        title TEXT,
        description TEXT,
        createdAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
```

```
      """);
  }
// id: the id of a item
// title, description: name and description of your activity
// created_at: the time that the item was created. It will be automatically handled by SQLite

  static Future<sql.Database> db() async {
    return sql.openDatabase(
      'inventorymanager.db',
      version: 1,
      onCreate: (sql.Database database, int version) async {
        await createTables(database);
    },
  );
}

// Create new item (journal)
static Future<int> createItem(String title, String? descrption) async {
  final db = await SQLHelper.db();

  final data = {'title': title, 'description': descrption};
  final id = await db.insert('items', data,
      conflictAlgorithm: sql.ConflictAlgorithm.replace);
  return id;
}

// Read all items (journals)
static Future<List<Map<String, dynamic>>> getItems() async {
```

```
  return db.query('items', orderBy: "id");
}

// Read a single item by id
// The app doesn't use this method but I put here in case you want to see it
static Future<List<Map<String, dynamic>>> getItem(int id) async {
  final db = await SQLHelper.db();
  return db.query('items', where: "id = ?", whereArgs: [id], limit: 1);
}

// Update an item by id
static Future<int> updateItem(
    int id, String title, String? descrption) async {
  final db = await SQLHelper.db();
```

```
    final data = {
      'title': title,
      'description': descrption,
      'createdAt': DateTime.now().toString()
    };

    final result =
        await db.update('items', data, where: "id = ?", whereArgs: [id]);
    return result;
  }

  // Delete
  static Future<void> deleteItem(int id) async {
```

```
    try {
      await db.delete("items", where: "id = ?", whereArgs: [id]);
    } catch (err) {
      debugPrint("Something went wrong when deleting an item: $err");
    }
  }
}
```

CHAPTER 6
CONCLUSION AND FUTURE ENHANCEMENT

In conclusion, an Inventory Management Application can be a valuable tool for businesses of all sizes to manage their inventory more efficiently and effectively. With the ability to track inventory levels in real-time, manage purchase and sales orders, and optimize inventory levels, businesses can make informed decisions about inventory management and improve their bottom line.

In terms of future enhancements, there are several areas that could be explored to improve the functionality and usefulness of the application. For example:

Integration with machine learning algorithms to predict inventory demand and optimize inventory levels automatically.

Integration with blockchain technology to provide greater security and transparency in the supply chain.

Implementation of a mobile app for inventory management on-the-go.

Integration with IoT devices to track inventory in real-time and automate inventory management tasks.

Integration with virtual reality technology for more immersive and interactive inventory management.

Overall, an Inventory Management Application can be a powerful tool for businesses to improve their inventory management processes and increase their profitability. With the right features and functionalities, businesses can gain greater visibility and control over their inventory, reduce waste, and improve their customer satisfaction. Future enhancements will continue to improve the functionality and usefulness of these applications, allowing businesses to stay ahead of the curve in the rapidly evolving world of inventory management.