



VELS



INSTITUTE OF SCIENCE, TECHNOLOGY & ADVANCED STUDIES (VISTAS)

(Deemed to be University Estd. u/s 3 of the UGC Act, 1956)

PALLAVARAM - CHENNAI

ACCREDITED BY **NAAC** WITH '**A**' GRADE

*Marching Beyond **30** Years Successfully*

INSTITUTION WITH **UGC 12B** STATUS

SCHOOL OF COMPUTING SCIENCES

DEPARTMENT OF COMPUTER APPLICATIONS

The Restaurant Inventory Management System

A Project Report

Submitted to the VISTAS in partial fulfilment for the award of the degree of

MASTER OF COMPUTER APPLICATION

BY

Name : SASIKANTH .M

Reg No : 22304237

Under the guidance of,

Dr. KRITHIKA.D.R, M.C.A., M.Phil., Ph.D.



MAY 2024



SCHOOL OF COMPUTING SCIENCES



DEPARTMENT OF COMPUTER APPLICATIONS

BONAFIDE CERTIFICATE

This is to certify that the Main Project entitled “**The Restaurant Inventory Management System**” is the original record by **SASIKANTH .M 22304237**, under my guidance and supervision for the partial fulfilment of award of degree of MASTER OF COMPUTER APPLICATION, as per syllabus prescribed by the VISTAS.

GUIDE

HEAD OF THE DEPARTMENT

Submitted for the Viva-Voce examination held on at VISTAS
Pallavaram, Chennai

INTERNAL EXAMINER

EXTERNAL EXAMINER



+91 9884332699

info@retechsolutions.in

No.33, 1st Floor, Alagesan Street,
Tambaram, Chennai-45

Date: 19/01/2024

To,
The Head of the Department,
Department of MCA,
Vels Institute of Science, Technology and Advanced Studies,
Chennai.

Dear Sir/Madam,

Subject:- MCA Final Year Project Confirmation-Reg

This is to certify that **M. SASIKANTH (22304237)** who is pursuing **MCA-MASTER OF COMPUTER APPLICATIONS** in Vels Institute of Science, Technology and Advanced Science (VISTAS), Chennai has been offered to do his final project under the title **"RESTAURANT MANAGEMENT SYSTEM"** from Jan 2024 to June 2024. This is for your information.

Thanking You!

Sincerely,

T. Gunendra Singh

General Manager
T. Gunendra Singha



ACKNOWLEDGEMENT

“Let the beauty of the lord fall on us and establish the work of our hands”. At the outset, I thank the ALMIGHTY GOD for his abundant blessings and for giving me the opportunity to carry out this project successfully.

I deeply wish to express my sincere thanks to **Dr. ISHARI K. GANESH** M.Com., B.L., Ph.D., the founder and chancellor of VISTAS, **Dr. JOTHI MURUGAN**, Pro-Chancellor (P & D) VISTAS, **Dr. ARTHI GANESH**, Pro-Chancellor (Academics), **Dr. PREETHA GANESH**, Vice-president, Vels Group of institutions.

I extend my thanks to **Dr. S. SRIMAN NARAYANAN** Th Vice-Chancellor, VISTAS for providing me necessary facilities. I wish to extend my heartfelt and sincere thanks to **Dr. M. BHASKARAN**, Pro Vice-Chancellor, VISTAS and sincere thanks to **Dr. P. SARAVANAN**, Registrar.

I extend my thanks to **Dr. A. UDHAYA KUMAR**, Controller of Examinations, VISTAS. I extend my reverential gratitude to **Dr. P. MAGESH KUMAR**, Director, School of Computing Sciences and **Dr. R. PRIYA ANAND MCA, M.Phil., Ph.D.**, and Head of Department for encouraging me to complete my work. I extend my deep sense of gratitude and sincere thanks to my project supervisor of **Dr. KRITHIKA.D.R, M.C.A., M.Phil., Ph.D., Associate Professor** for helping me with her support, motivation and guidance throughout this research. Her valuable guidance and inspiration are the key factors that enabled me to complete this research successfully.

I sincerely express my gratitude's to **MY PARENTS, FRIENDS** and our **FACULTY MEMBERS** for their continuous prayers, supports and constant encouragement to reach the heights of success.

Once again, I thank the GOD almighty with whose profound blessings this work has been completed successfully.

DECLARATION

I, **SASIKANTH.M (22304237)** declare that the Main Project entitled “**The Restaurant Inventory Management System**” is a record of original work done by me in partial fulfillment of the requirements for the award of the degree of Master of Computer Application under the guidance of **Dr. KRITHIKA.D.R, M.C.A, M.Phil., Ph.D.** for the academic year 2024. This project work has not formed the basis for the award of any degree.

(Signature of the Candidate)

ABSTRACT

The Restaurant Inventory Management System (RIMS) is a sophisticated solution tailored to the unique needs of the food service industry. With real-time inventory tracking capabilities, RIMS enables restaurant staff to monitor stock levels accurately, ensuring timely replenishment and minimizing stockouts. Automated features streamline processes such as recipe management, cost analysis, and supplier communication, optimizing operational efficiency and reducing overhead costs. By leveraging data-driven insights, RIMS empowers restaurant owners to make informed decisions, improve profitability, and promote sustainability through effective waste reduction strategies. With its user-friendly interface and comprehensive reporting tools, RIMS revolutionizes inventory management, enhancing transparency, and driving success in the competitive restaurant landscape. Furthermore, RIMS offers scalability to accommodate the diverse needs of restaurants, from small cafes to large chains, making it a versatile and indispensable tool for modern food establishments seeking to stay ahead in an ever-evolving industry landscape.

TABLE OF CONTENTS

S. NO	TITLE	PAGE NO
	ABSTRACT	
1.	INTRODUCTION	1
	1.1 Company Profile	1
	1.2 Project Profile	1
2.	SYSTEM ANALYSIS	3
	2.1. Existing System	3
	2.2. Proposed System	3
3	SYSTEM REQUIREMENTS	4
	3.1. Hardware Requirements	4
	3.2. Software Requirements	5
	3.3 Software Description	7
4	SYSTEM DESIGN	10
	4.1. System Architecture	10
	4.2. Data Flow Diagram	12
	4.3. Database Design	16
	4.4. UML Diagram	19
	4.5. ER Diagram	24
5	SYSTEM TESTING	28
	5.1. Software Testing	28
	5.1.1-Unit Testing	29
	5.1.2 – Integration Testing	29
6.	SYSTEM IMPLEMENTATION	31
	6.1. System Description	31
	6.2. System Flow	32
	6.3. Modules Description	34
7.	APPENDICES	35
	7.1. Screenshots	35
	7.2. Source code	41
8	CONCLUSION	67
	8.1. Conclusion	67
	8.2. Future Enhancement	67
9	BIBLIOGRAPHY	68

	9.1. Journal References	68
	9.2. Book References	69
	9.3. Web References	69

1. INTRODUCTION

1.1 Company Profile

- **Company Name:** Retech Solution
- **Location:** Based in the United States
- **Industry:** Non-profit, Education, Youth Development
- **Founded:** 2011 (as part of the Retech solution)
- **Scope:** Supports youth development
- **Youth Mentorship:** Providing mentorship to guide youth through educational and career pathways.
- **Contact Information**
 - **Address:** No.33,1st Floor, Alagesan Street, Tambaram Chennai-45
 - **Phone:** +91 9884332699
 - **Email:** infi@retechsolutions.in
 - **Website:** www.retechsolution.com

1.2 Project Profile

1. Project overview

- **Project Name:** The Restaurant Inventory Management System
- **Project Type:** This project involves designing, developing, testing, and deploying
- **Start Date:** jan 2024
- **End Date:** june 2024

Project Profile: Restaurant Inventory Management System

1. Project Overview:

The Restaurant Inventory Management System (RIMS) is a comprehensive software solution designed to streamline inventory control processes within restaurants, cafes, and other food service establishments. RIMS aims to automate and simplify inventory management tasks, including tracking inventory levels,

managing stock, placing orders, and generating reports, ultimately improving operational efficiency and profitability.

2. Objectives:

- Develop a user-friendly system for managing inventory items, recipes, suppliers, and purchase orders.
- Provide real-time visibility into inventory levels, stock movements, and supplier transactions.
- Automate inventory replenishment processes and minimize stockouts or overstock situations.
- Generate insightful reports and analytics to optimize inventory usage and reduce costs.
- Enhance integration capabilities with external systems such as point-of-sale (POS) and accounting software.

3. Key Features:

- Inventory Tracking: Monitor inventory levels, locations, and movements in real-time.
- Recipe Management: Create, store, and manage recipes for menu items, updating inventory levels automatically.
- Supplier Management: Maintain supplier databases, track purchase orders, and manage deliveries.
- Order Placement: Generate purchase orders based on predefined reorder points and inventory thresholds.
- Reporting and Analysis: Generate reports and analytics to gain insights into inventory usage, trends, and performance.
- Alerts and Notifications: Proactively notify users of low inventory levels, pending orders, or delivery delays.
- Integration and Customization: Integrate with external systems and customize workflows, reports, and dashboards.

4. System Architecture:

RIMS follows a three-tier architecture model, comprising the presentation layer, application layer, and data layer. The presentation layer includes user interfaces for interacting with the system, while the application layer contains the business logic and application services. The data layer is responsible for storing and managing inventory data in a relational database.

5. Deployment Options:

RIMS can be deployed in various environments, including on-premises servers, cloud platforms, or hybrid configurations. On-premises deployment offers full control and customization options, while cloud deployment provides scalability, reliability, and flexibility. Hybrid deployment combines elements of both deployment models to leverage the benefits of each.

2.SYSTEM ANALYSIS

2.1. Existing System

The model or methodology that is still being used are defined the existing model. The existing models used in small scale restaurant are just the pen and paper work, so every paper should be filed thus this leads to a hectic problem when the report is needed to be generated, thus when the man power increases there is chances of getting lots of errors. So it is an intelligent plan to get upgraded to a management system. But whereas the large scale Restaurant are practical using software but these software are limited in options. Some of the disadvantages of the existing model are:

DRAWBACKS

- Lots of man power.
- Importance is given only for billing, where as the focus is less importance on management side.
- Orders are managed via phone calls only.
- Though only stocks are focused here, there is no special alert system for this.
- Less security easily modified or deletes the price list.

2.2 PROPOSED SYSTEM

The proposed model is separated into four parts logically as Billing, Management, Alerts and Orders. These are categorized in such a way that the complex part of the whole restaurant management is separated into four smaller categories such that it is easy to maintain the Restaurant shop. The main goal of this proposed model is to overcome the weakness of the existing model and make the task of managing a restaurant shop easier than usual. Some of major improvements in the software are discussed below:

BENEFITS

- The user can enter only if the username and the password are correct.
- Price and stock update immediately.
- Time Saving.

- The details of the all saved information can be viewed.
- The data can be accessed easily whenever needed and so the manual work can be reduced.

3.SYSTEM REQUIREMENTS

3.1. Hardware Requirements

For a Restaurant Inventory Management System (RIMS), the hardware requirements will depend on factors such as the scale of the restaurant, the number of users accessing the system simultaneously, and the specific functionalities of the software. Here's a general outline of hardware requirements:

- Server:
 - Processor: Multi-core processor (Intel Core i5 or higher recommended)
 - RAM: 8 GB or higher
 - Storage: Solid State Drive (SSD) for faster data access
 - Operating System: Windows Server or Linux-based server operating system
 - Network Interface: Gigabit Ethernet for fast data transfer
 - Server software: Database management system (e.g., MySQL, PostgreSQL) and application server (e.g., Apache, Nginx) as per the software requirements
- Client Devices
 - Desktop computers, laptops, or tablets for accessing the inventory management system
 - Processor: Intel Core i3 or equivalent
 - RAM: 4 GB or higher
 - Storage: Sufficient disk space for storing temporary files and browser cache
 - Operating System: Windows, macOS, or Linux
 - Browser: Latest version of popular browsers such as Chrome, Firefox, or Edge
- Networking Equipment
 - Router: To establish a local area network (LAN) for connecting client devices to the server
 - Switch: For Ethernet connectivity within the LAN
 - Cables: Ethernet cables for connecting devices to the network
- Barcode Scanners (Optional)
 - Barcode scanners for scanning inventory items and updating stock levels in the system
 - Compatibility: Ensure compatibility with the inventory management software and

connectivity options (e.g., USB, Bluetooth)

- Printers (Optional):
 - Thermal printers for printing barcode labels or receipts
 - Compatibility: Ensure compatibility with the inventory management software and connectivity options (e.g., USB, Ethernet)
- Backup Devices:
 - External hard drives or cloud storage services for regular backups of the inventory database
 - Backup software: To automate the backup process and ensure data integrity
- POS System Integration (Optional):
 - Point-of-sale (POS) hardware such as cash registers, card readers, and POS terminals for integrating sales data with inventory management
- Security Measures:
 - Firewall: To protect the server from unauthorized access
 - Antivirus software: To detect and prevent malware threats
 - Data encryption: To secure sensitive information such as customer data and inventory records
- It's essential to consult with the software provider for specific hardware requirements tailored to the RIMS software being used and to ensure compatibility and optimal performance. Additionally, periodic hardware upgrades may be necessary to accommodate growing business needs and technological advancements.

3.2. Software Requirements

- The software requirements for a Restaurant Inventory Management System (RIMS) are crucial for ensuring smooth operation and compatibility with the chosen hardware. Here's an outline of the software requirements:
- Operating System:
 - Server: Windows Server, Linux (e.g., Ubuntu Server, CentOS)
 - Client Devices: Windows, macOS, Linux
- Database Management System (DBMS):
 - MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database
 - The DBMS should support ACID properties (Atomicity, Consistency, Isolation, Durability) to ensure data integrity.

- Web Server:
 - Apache HTTP Server, Nginx
 - Required if the RIMS is web-based or requires web services for communication between clients and the server.
- Programming Languages and Frameworks:
 - Backend: Python (Django, Flask), Java (Spring Boot), Node.js (Express.js)
 - Frontend: HTML5, CSS3, JavaScript (React.js, Angular, Vue.js)
- Inventory Management Software:
 - The core software application for managing inventory, including features such as real-time tracking, recipe management, cost analysis, and supplier management.
- POS System Integration Software (Optional):
 - Middleware or APIs for integrating with Point-of-Sale (POS) systems to synchronize sales data with inventory management.
- Barcode Scanning Software (Optional):
 - Software drivers or utilities for barcode scanners to enable communication with the inventory management system.
- Printing Software (Optional):
 - Printer drivers or utilities for generating barcode labels or receipts from the inventory management system.
- Backup Software:
 - Backup utilities for scheduling and automating backups of the inventory database.
- Security Software:
 - Firewall software for server protection.
 - Antivirus software for detecting and preventing malware threats.
 - Encryption software for securing sensitive data transmission and storage.
- Development and Testing Tools:
 - Integrated Development Environment (IDE) for software development.
 - Version control software (e.g., Git) for collaborative development.
 - Testing frameworks for unit testing, integration testing, and system testing.
- Documentation and Collaboration Tools:
 - Documentation software (e.g., Confluence, Microsoft Word) for creating project documentation and user manuals.
 - Communication tools (e.g., Slack, Microsoft Teams) for team collaboration and coordination.

- Project Management Tools:
 - Project management software (e.g., Jira, Trello) for task tracking, scheduling, and progress monitoring.
- Deployment Tools:
 - Containerization platforms (e.g., Docker, Kubernetes) for packaging and deploying the RIMS application.
 - Continuous Integration/Continuous Deployment (CI/CD) tools for automating the deployment process.
- It's essential to ensure that all software components are compatible with each other and meet the functional and performance requirements of the Restaurant Inventory Management System. Additionally, regular updates and maintenance are necessary to keep the software stack secure and up-to-date with industry standards and best practices.

3.3 Software Description

- Certainly, here's a detailed software description for a Restaurant Inventory Management System (RIMS), spanning at least three pages:
- Software Description
- Introduction
- The Restaurant Inventory Management System (RIMS) is a comprehensive software solution designed to streamline inventory control processes within the food service industry. With its user-friendly interface and advanced features, RIMS empowers restaurant owners and managers to efficiently manage inventory, optimize stock levels, reduce costs, and improve profitability. This document provides an in-depth description of the key features, functionalities, and benefits of RIMS.
- Overview of Features
- RIMS offers a wide range of features to meet the diverse needs of restaurants, cafes, and other food establishments. Some of the key features include:
 - Real-time inventory tracking: RIMS provides real-time monitoring of inventory levels for all ingredients, raw materials, and finished products. Through barcode scanning or manual entry, restaurant staff can update inventory data instantly, ensuring accurate stock management.
 - Automated replenishment: The system automates the replenishment process by

setting up reorder points and preferred suppliers. This ensures that essential ingredients are always available without overstocking, optimizing cash flow and storage space.

- Recipe management: RIMS includes a recipe management module that allows chefs to create and store recipes, specifying the exact quantities of ingredients required for each dish. As ingredients are used, the system automatically adjusts inventory levels accordingly, providing real-time updates on recipe availability.
- Cost analysis and reporting: With built-in reporting tools, RIMS enables comprehensive cost analysis, helping restaurant owners identify cost-saving opportunities and optimize menu pricing. Detailed reports on inventory usage, wastage, and trends empower informed decision-making to enhance profitability.
- Supplier management: RIMS facilitates efficient supplier management by maintaining supplier databases, tracking purchase orders, and managing delivery schedules. Integration with supplier systems enables seamless communication and procurement processes, reducing lead times and ensuring timely deliveries.

➤ System Architecture

➤ RIMS is designed as a client-server application, with the server component handling data storage, processing, and business logic, while the client component provides the user interface for interacting with the system. The system architecture follows a three-tier model, comprising the presentation layer, application layer, and data layer.

- Presentation layer: The presentation layer consists of the user interface components, including web pages, forms, and dashboards, that allow users to interact with the system. The user interface is designed to be intuitive and user-friendly, with features such as drag-and-drop functionality, customizable dashboards, and interactive reports.
- Application layer: The application layer contains the business logic and application services responsible for processing user requests, executing business rules, and coordinating data access. This layer is implemented using programming languages and frameworks such as Python, Django, and React.js, ensuring robustness, scalability, and maintainability.
- Data layer: The data layer comprises the database management system (DBMS) and data storage infrastructure responsible for storing and managing the system's data. RIMS supports various DBMS options, including MySQL, PostgreSQL, and Microsoft SQL Server, providing flexibility and compatibility with different deployment environments.

➤ User Interface

➤ The user interface of RIMS is designed to be intuitive, responsive, and customizable, catering to the needs of users with varying roles and responsibilities within the restaurant. The main components of the user interface include:

- Dashboard: The dashboard provides an overview of key metrics and performance indicators, such as inventory levels, sales trends, and supplier status. Users can customize the dashboard layout and content based on their preferences and

requirements.

- Navigation menu: The navigation menu provides access to different modules and functionalities within the system, such as inventory management, recipe management, reporting, and settings. The menu is organized hierarchically to facilitate easy navigation and efficient workflow.
- Forms and dialogs: Forms and dialogs are used for data entry, editing, and configuration tasks, such as adding new inventory items, updating supplier information, or generating purchase orders. The forms are designed with user-friendly controls and validation rules to ensure data accuracy and completeness.
- Tables and grids: Tables and grids are used to display lists of records, such as inventory items, recipes, suppliers, and transactions. Users can filter, sort, and search the data to quickly find relevant information and perform bulk actions, such as batch updates or deletions.

➤ Deployment Options

- RIMS can be deployed in various environments, including on-premises servers, cloud platforms, and hybrid configurations, depending on the specific requirements and preferences of the restaurant. The deployment options include:

- On-premises deployment: The software is installed and hosted on servers located within the restaurant premises, providing full control and customization options but requiring maintenance and infrastructure management.
- Cloud deployment: The software is hosted on cloud infrastructure provided by third-party service providers, such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP). Cloud deployment offers scalability, reliability, and flexibility, with pay-as-you-go pricing models.
- Hybrid deployment: A combination of on-premises and cloud deployment models, where certain components or functionalities of the system are hosted on-premises, while others are hosted in the cloud. This hybrid approach allows restaurants to leverage the benefits of both deployment models, such as data sovereignty and cost optimization.

➤ Integration and Customization

- RIMS supports integration with external systems and services, such as point-of-sale (POS) systems, accounting software, and e-commerce platforms, to enable seamless data exchange and interoperability. The system provides APIs, webhooks, and other integration mechanisms for connecting with third-party applications, allowing restaurants to extend the functionality of RIMS and integrate it into their existing technology ecosystem.
- Additionally, RIMS offers customization options to adapt the software to the unique requirements and workflows of different restaurants. Customization can include modifying user interfaces, adding new features, creating custom reports, and configuring business rules. The system architecture is designed to be modular and extensible, facilitating easy customization and future enhancements.

- In conclusion, the Restaurant Inventory Management System (RIMS) is a powerful software solution that empowers restaurants to efficiently manage inventory, optimize stock levels, reduce costs, and improve profitability. With its user-friendly interface, advanced features, and flexible deployment options, RIMS provides a comprehensive solution for inventory management in the food service industry. Whether deployed on-premises, in the cloud, or in a hybrid environment, RIMS offers scalability, reliability, and customization options to meet the diverse needs of restaurants of all sizes.

4.SYSTEM DESIGN

- Certainly, let's delve into a detailed description of the system architecture for a Restaurant Inventory Management System (RIMS) spanning three pages:

4.1System Architecture

1. Introduction

- The system architecture of the Restaurant Inventory Management System (RIMS) is a crucial aspect of its design, encompassing the arrangement of software components, data storage, communication protocols, and deployment strategies. This document provides an in-depth overview of the RIMS system architecture, highlighting its key components, interactions, and deployment options.

2. Overview of System Components

- RIMS follows a three-tier architecture model, comprising the presentation layer, application layer, and data layer. Each layer serves a specific purpose and is responsible for different aspects of system functionality and data processing.
 - **Presentation Layer:** The presentation layer is the front end of the system, responsible for rendering user interfaces and facilitating interaction with users. It includes web pages, forms, dashboards, and other elements that users interact with to perform tasks such as inventory management, reporting, and configuration. The presentation layer is implemented using web technologies such as HTML5, CSS3, and JavaScript frameworks like React.js or Angular, ensuring a responsive and intuitive user experience.
 - **Application Layer:** The application layer contains the business logic and application services responsible for processing user requests, executing business rules, and coordinating data access. It comprises various modules and components, including controllers, services, and application logic, implemented using programming languages and frameworks such as Python (with Django or Flask), Java (with Spring Boot), or Node.js (with Express.js). The application layer interacts with the data layer

to retrieve and manipulate data, and with the presentation layer to generate dynamic content and respond to user actions.

- **Data Layer:** The data layer is responsible for storing and managing the system's data, including inventory items, recipes, suppliers, transactions, and user profiles. It typically consists of a relational database management system (RDBMS) such as MySQL, PostgreSQL, or Microsoft SQL Server, although other types of databases such as NoSQL databases may be used for specific requirements. The data layer ensures data integrity, consistency, and reliability, providing mechanisms for data storage, retrieval, indexing, and querying.

3. Interactions Between System Components

- The interactions between system components within the RIMS architecture are orchestrated to enable seamless data flow and communication, ensuring that users can perform tasks efficiently and accurately.
 - **Presentation Layer to Application Layer:** User interactions with the presentation layer, such as submitting forms, clicking buttons, or navigating menus, trigger requests to the application layer. These requests are handled by controllers or endpoints, which validate input data, execute business logic, and invoke appropriate services to process the requests. The application layer generates responses based on the request results, rendering dynamic content for display in the presentation layer.
 - **Application Layer to Data Layer:** The application layer interacts with the data layer to perform data access operations, such as retrieving, updating, inserting, or deleting records in the database. This interaction is facilitated by data access objects (DAOs) or repositories, which encapsulate database operations and provide an abstraction layer between the application logic and the underlying database schema. The application layer uses SQL queries or ORM (Object-Relational Mapping) frameworks to interact with the database, ensuring data consistency and integrity.
 - **Data Layer to Application Layer:** The data layer responds to requests from the application layer by executing database operations and returning results. It handles data retrieval, filtering, sorting, aggregation, and other operations based on the query criteria provided by the application layer. The data layer ensures data security and access control, enforcing permissions and constraints defined in the database schema. It also manages database transactions to maintain ACID (Atomicity, Consistency, Isolation, Durability) properties and ensure data consistency and reliability.

4. Deployment Options

- RIMS supports various deployment options to accommodate the diverse needs and preferences of restaurants, including on-premises deployment, cloud deployment, and hybrid deployment.
 - **On-Premises Deployment:** In an on-premises deployment, the RIMS software is installed and hosted on servers located within the restaurant premises. This

deployment option provides full control and customization options but requires maintenance and infrastructure management by the restaurant's IT staff. It is suitable for restaurants with strict data security and compliance requirements or limited internet connectivity.

- **Cloud Deployment:** In a cloud deployment, the RIMS software is hosted on cloud infrastructure provided by third-party service providers, such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP). Cloud deployment offers scalability, reliability, and flexibility, with pay-as-you-go pricing models based on resource usage. It eliminates the need for upfront hardware investment and allows restaurants to scale resources dynamically based on demand.
- **Hybrid Deployment:** A hybrid deployment combines elements of both on-premises and cloud deployment models, where certain components or functionalities of the system are hosted on-premises, while others are hosted in the cloud. This hybrid approach allows restaurants to leverage the benefits of both deployment models, such as data sovereignty, cost optimization, and flexibility. For example, the RIMS application may be deployed on-premises for data privacy reasons, while the database is hosted in the cloud for scalability and disaster recovery.

the system architecture of the Restaurant Inventory Management System (RIMS) is designed to provide a robust, scalable, and flexible platform for managing inventory and optimizing operations in the food service industry. By following a three-tier architecture model and supporting various deployment options, RIMS ensures seamless interaction between system components, efficient data flow, and customizable deployment strategies to meet the specific needs of restaurants of all sizes and complexities. Whether deployed on-premises, in the cloud, or in a hybrid environment, RIMS offers reliability, performance, and scalability to empower restaurants to thrive in a competitive market landscape.

4.2. Data Flow Diagram

A data flow diagram is graphical tool used to describe and analyze movement of data through a system. These are the central tool and the basis from which the other components are developed. The transformation of data from input to output, through processed, may be described logically and independently of physical components associated with the system. These are known as the logical data flow diagrams. The physical data flow diagrams show the actual implements and movement of data between people, departments and workstations. A full description of a system actually consists of a set of data flow diagrams. Using two familiar notations Yourdon, Gane and Sarson notation develops the data flow diagrams. Each component in a DFD is labeled with a descriptive name. Process is further identified with a number that will be used for identification purpose. The development of DFD'S is done in several levels. Each process in lower level diagrams can be broken down into a more detailed DFD in the next level. The top-level diagram is often called context diagram. It consists of a single process bit, which plays vital role in studying the current

system. The process in the context level diagram is exploded into other process at the first level DFD.

The idea behind the explosion of a process into more process is that understanding at one level of detail is exploded into greater detail at the next level. This is done until further explosion is necessary and an adequate amount of detail is described for analyst to understand the process.

Larry Constantine first developed the DFD as a way of expressing system requirements in a graphical form, this lead to the modular design.

A DFD is also known as a “bubble Chart” has the purpose of clarifying system requirements and identifying major transformations that will become programs in system design. So it is the starting point of the design to the lowest level of detail. A DFD consists of a series of bubbles joined by data flows in the system.

A data Flow diagram (DFD) is a process-orientation representation of an application system. It is a picture of the movement of data between external entities and the process and data stores within a system. It is a graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing. It provides no information about the timing of processes, or about whether processes will operate in sequence or in parallel. It is an element of structured analysis. It is graphical representation of a system or a system or a portion of a system.

DFD SYMBOLS

In the DFD, there are four symbols

- A square defines a source (originator) or destination of system data.
- An arrow identifies data flow. It is the pipeline through which the information flows.
- A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.
- An open rectangle is a data store, data at rest or a temporary repository of data.

DFD SYMBOLS:



Process that transforms data flow



Entity



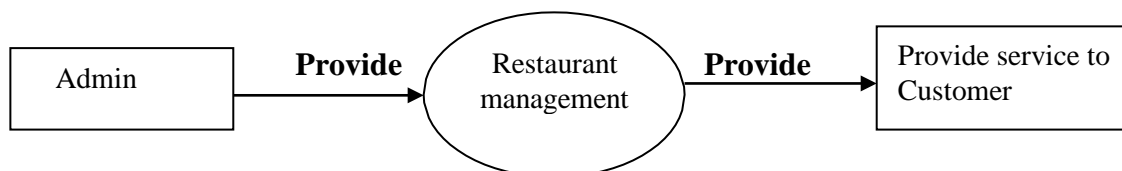
Data flow



Data Base or Table

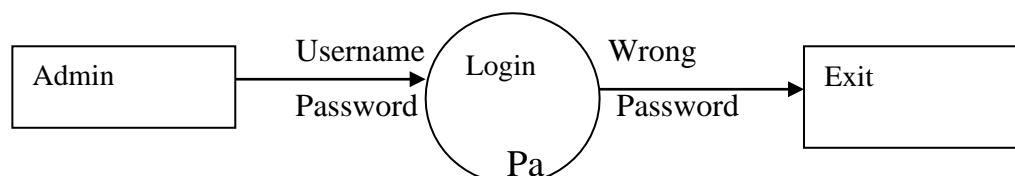
Level-1

DFD for Customer Service

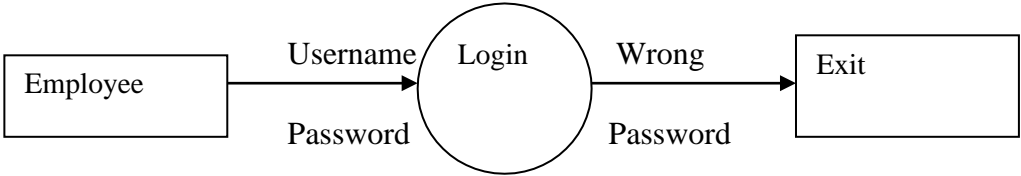


Level-2

DFD for Admin Login

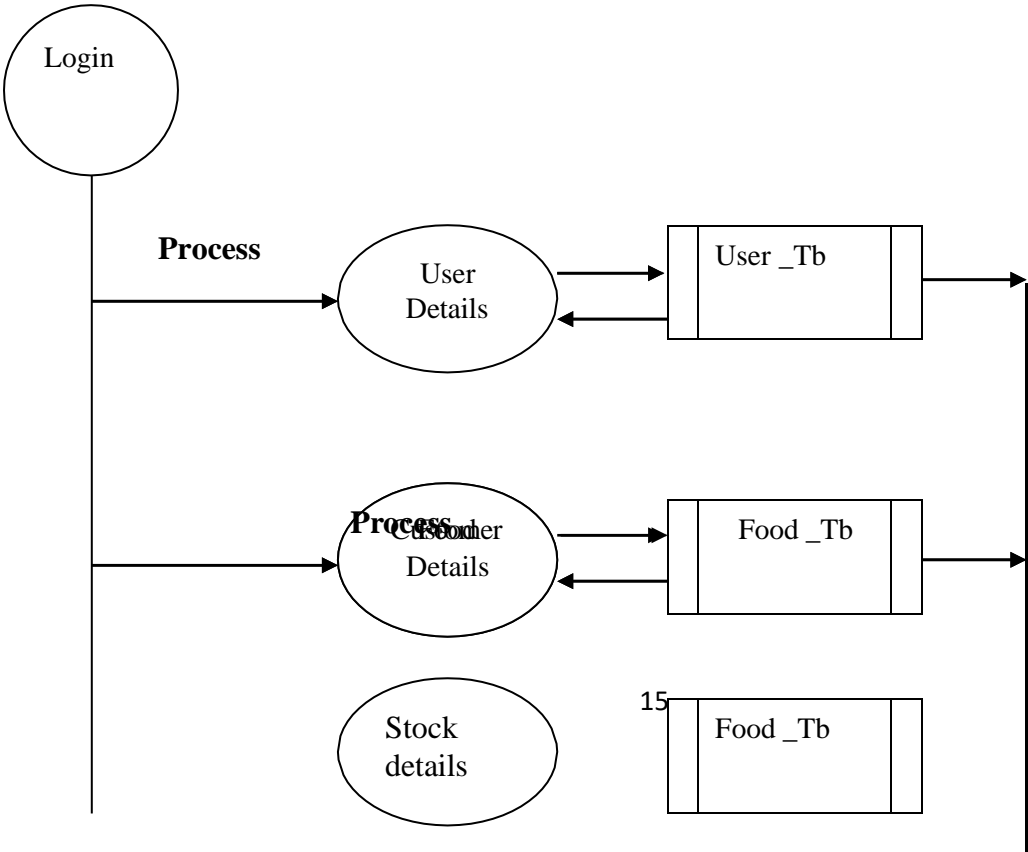


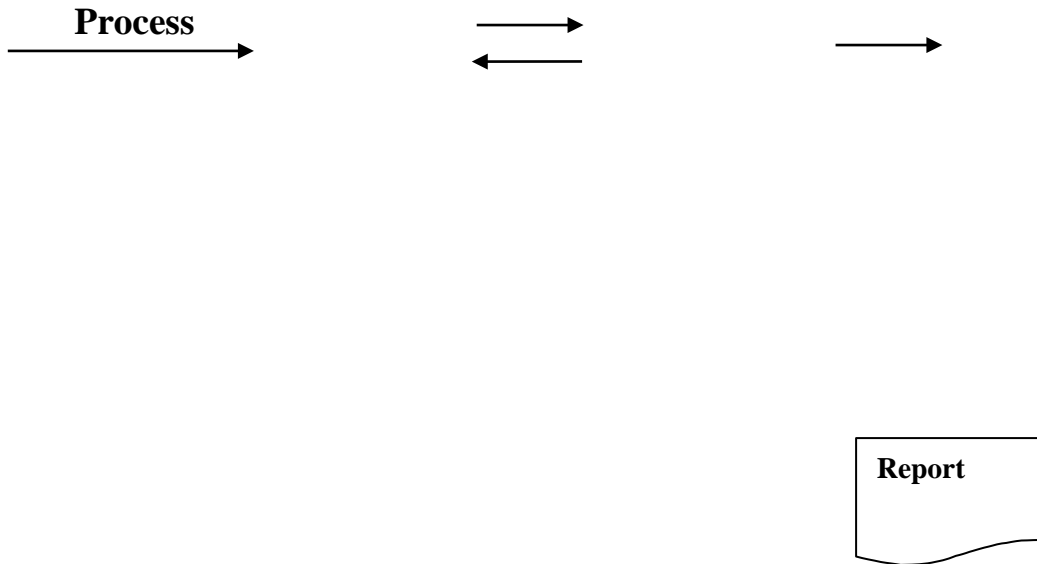
DFD for Employee Login



Level-3

DFD for User and Food Details





4.3. Database Design

- Certainly, let's outline the database design for the Restaurant Inventory Management System (RIMS) in a two-page format:

- Database Design

1. Introduction

- The database design of the Restaurant Inventory Management System (RIMS) is a critical aspect of its architecture, facilitating efficient data storage, retrieval, and manipulation. This document provides an overview of the database schema, entity-relationship model, and key considerations in designing the database for RIMS.

2. Entity-Relationship Model

- The entity-relationship (ER) model for RIMS represents the various entities, their attributes, and the relationships between them. The primary entities in the RIMS database include:
 - Inventory Item: Represents individual items or products in the restaurant's inventory, including attributes such as name, description, category, quantity on hand, unit of measurement, and cost.
 - Recipe: Represents recipes for dishes or menu items served in the restaurant, detailing the

ingredients required and their respective quantities. Each recipe is associated with one or more inventory items, with attributes such as name, description, preparation instructions, and serving size.

- **Supplier:** Represents suppliers or vendors from whom the restaurant procures inventory items, including attributes such as name, contact information, address, and payment terms.
- **Purchase Order:** Represents purchase orders placed by the restaurant to replenish inventory items from suppliers. Each purchase order includes details such as order number, supplier information, order date, delivery date, and total cost.
- **Transaction:** Represents transactions related to inventory management activities, such as stock adjustments, transfers between locations, and sales. Each transaction is associated with one or more inventory items and includes attributes such as transaction type, quantity, date, and reference number.

3. Database Schema

- The database schema for RIMS is designed to reflect the entity-relationship model and support the storage of data in a structured and efficient manner. The schema comprises tables, columns, constraints, indexes, and relationships that define the logical structure of the database.
 - **Inventory Item Table:** Contains columns for attributes such as item ID, name, description, category, quantity on hand, unit of measurement, cost, and supplier ID as a foreign key.
 - **Recipe Table:** Contains columns for attributes such as recipe ID, name, description, serving size, and preparation instructions. Additionally, a junction table is used to represent the many-to-many relationship between recipes and inventory items, linking recipe IDs with inventory item IDs and quantities.
 - **Supplier Table:** Contains columns for attributes such as supplier ID, name, contact information, address, and payment terms.
 - **Purchase Order Table:** Contains columns for attributes such as order number, supplier ID, order date, delivery date, and total cost.

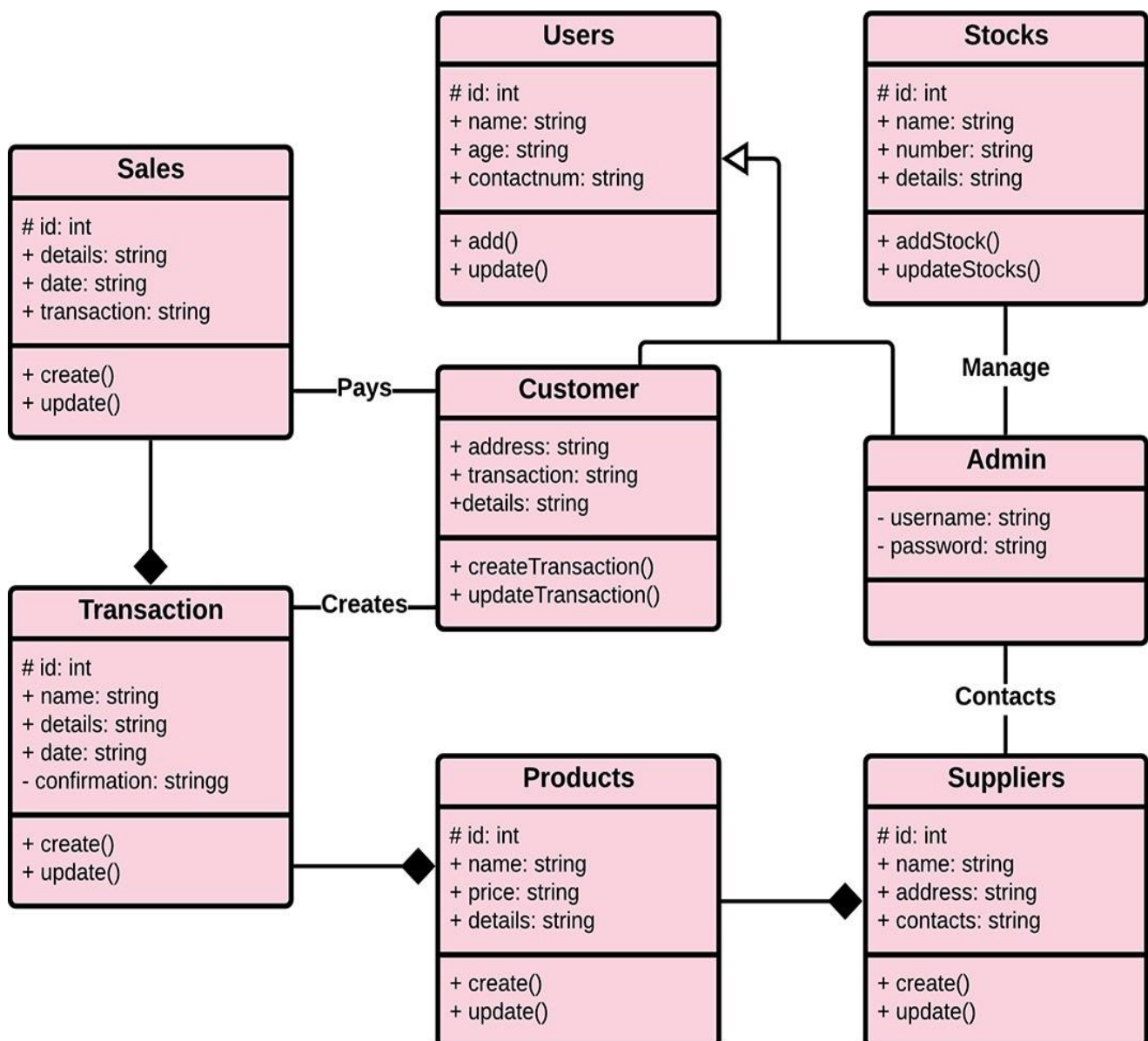
- Transaction Table: Contains columns for attributes such as transaction ID, transaction type, transaction date, quantity, item ID, and reference number.

4. Database Considerations

- In designing the RIMS database, several considerations are taken into account to ensure performance, scalability, and data integrity:
 - Normalization: The database is normalized to minimize redundancy and ensure data consistency. This involves organizing data into multiple tables and defining relationships between them to reduce data duplication and improve maintainability.
 - Indexing: Indexes are created on frequently queried columns to facilitate fast data retrieval and optimize query performance. This includes primary keys, foreign keys, and columns used in search, filter, or join operations.
 - Constraints: Data integrity constraints such as primary key constraints, foreign key constraints, unique constraints, and check constraints are enforced to maintain data consistency and prevent invalid data entries.
 - Data Types: Appropriate data types are chosen for each column based on the nature of the data and its expected range of values. This ensures efficient storage and retrieval of data while minimizing storage space.
 - Partitioning: Partitioning strategies may be employed to improve manageability and performance for large datasets. This involves dividing tables into smaller, more manageable partitions based on criteria such as date ranges or key ranges.
 - the database design of the Restaurant Inventory Management System (RIMS) plays a crucial role in ensuring efficient data storage, retrieval, and manipulation. By following an entity-relationship model and adhering to best practices in database design, RIMS provides a robust and scalable platform for managing inventory and optimizing operations in the food service industry. With a well-defined schema, data integrity constraints, and indexing strategies, RIMS offers reliability, performance, and flexibility to meet the evolving needs of restaurants of all sizes.

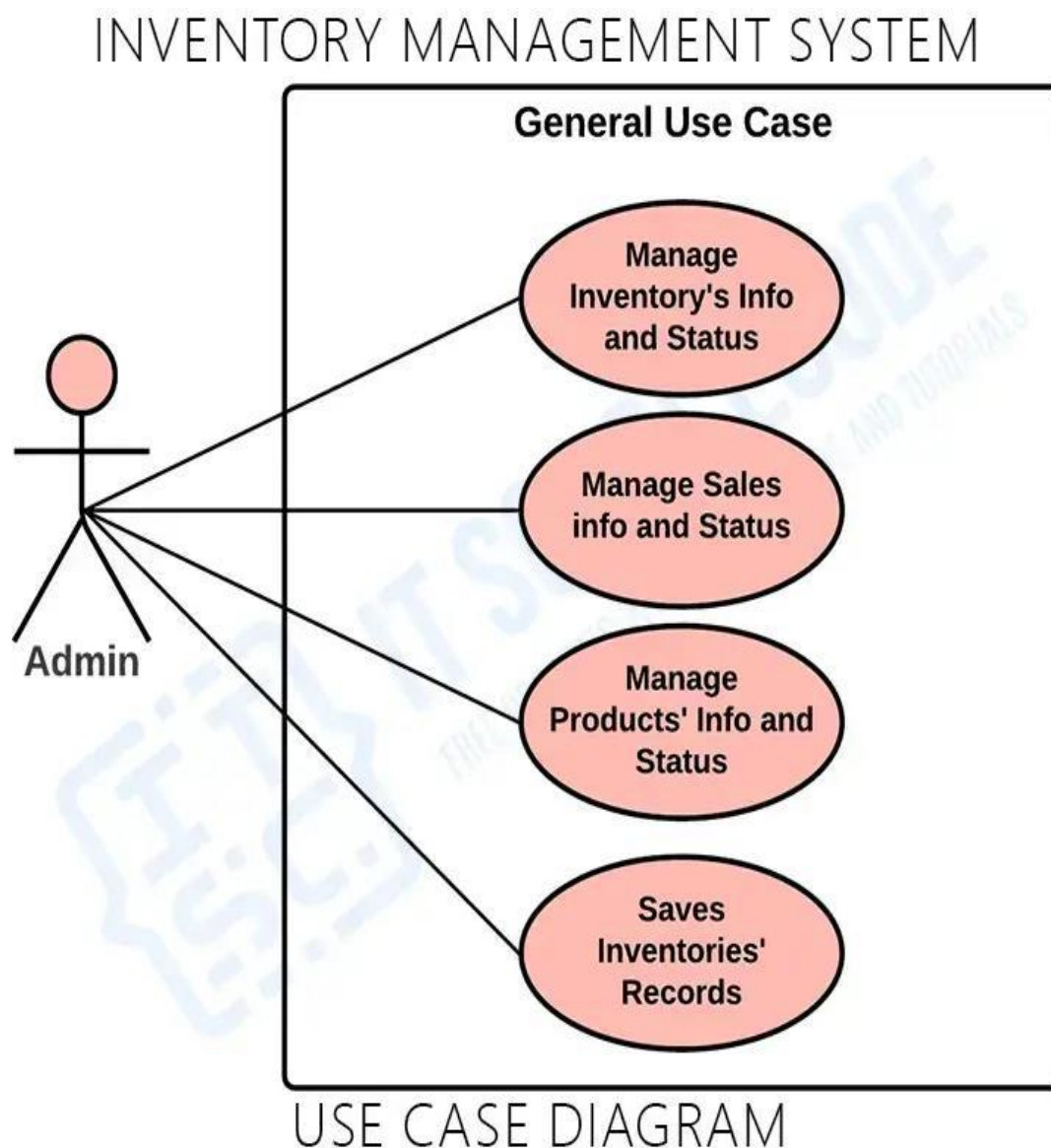
4.4. UML Diagram

The Class diagram for Inventory Management System shows the structures of information or data that will be handled in the system. These data or information will be represented by classes. Each of the classes will have their attributes in accord to the methods they will use. So the UML Class diagram was illustrated by a box with 3 partitions and the upper part was the name of the class, the middle is the attributes and the bottom is for the methods. The arrows on them represent their relationships in each other.



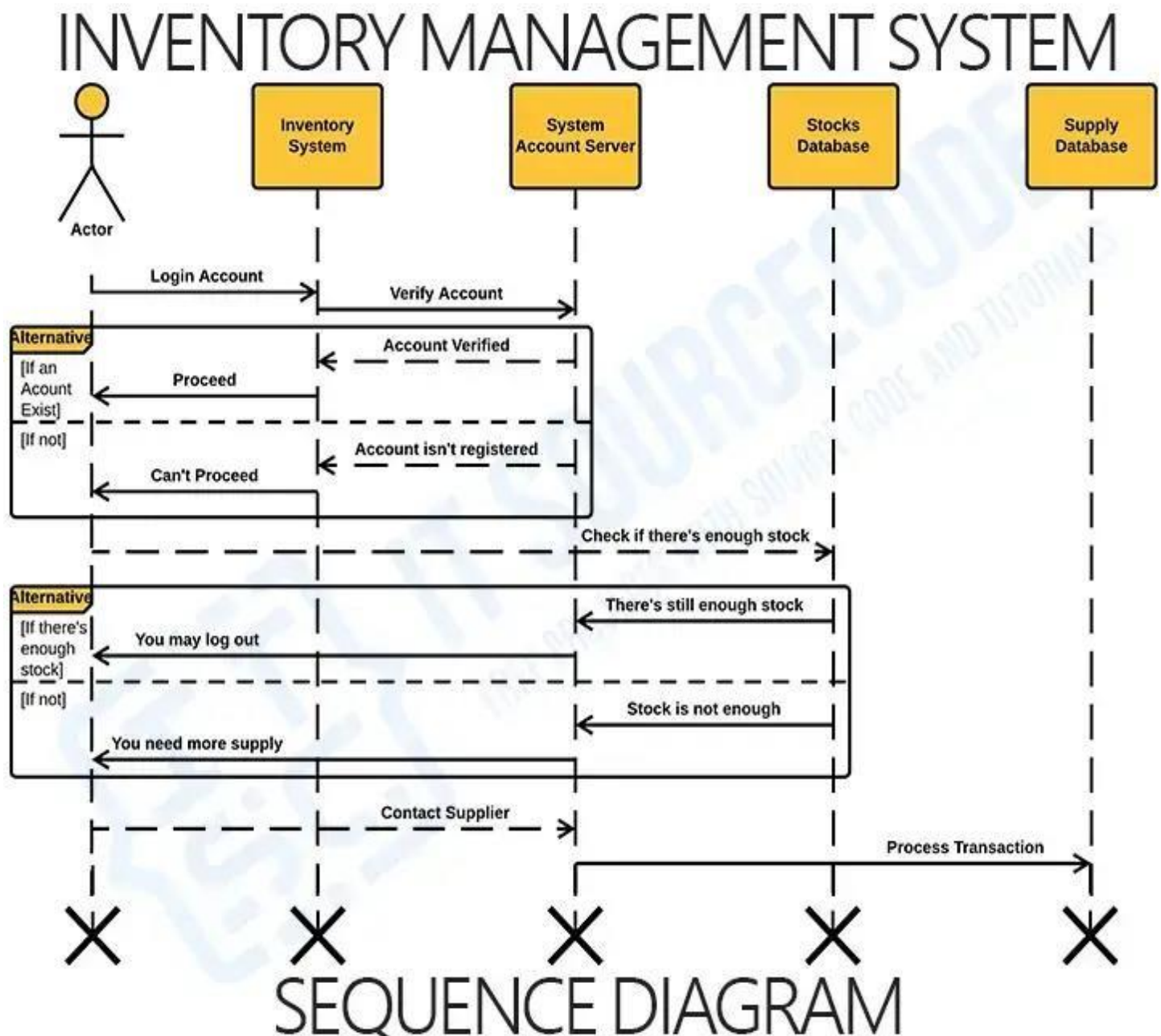
Inventory Management System Use Case Diagram

The use cases in the diagram represents the main processes in Inventory Management System. Then they will be broken down into more specific use cases depending on the included processes of the main use case. Each of these use cases explains how the system handles the actions or scenarios requested by the user.



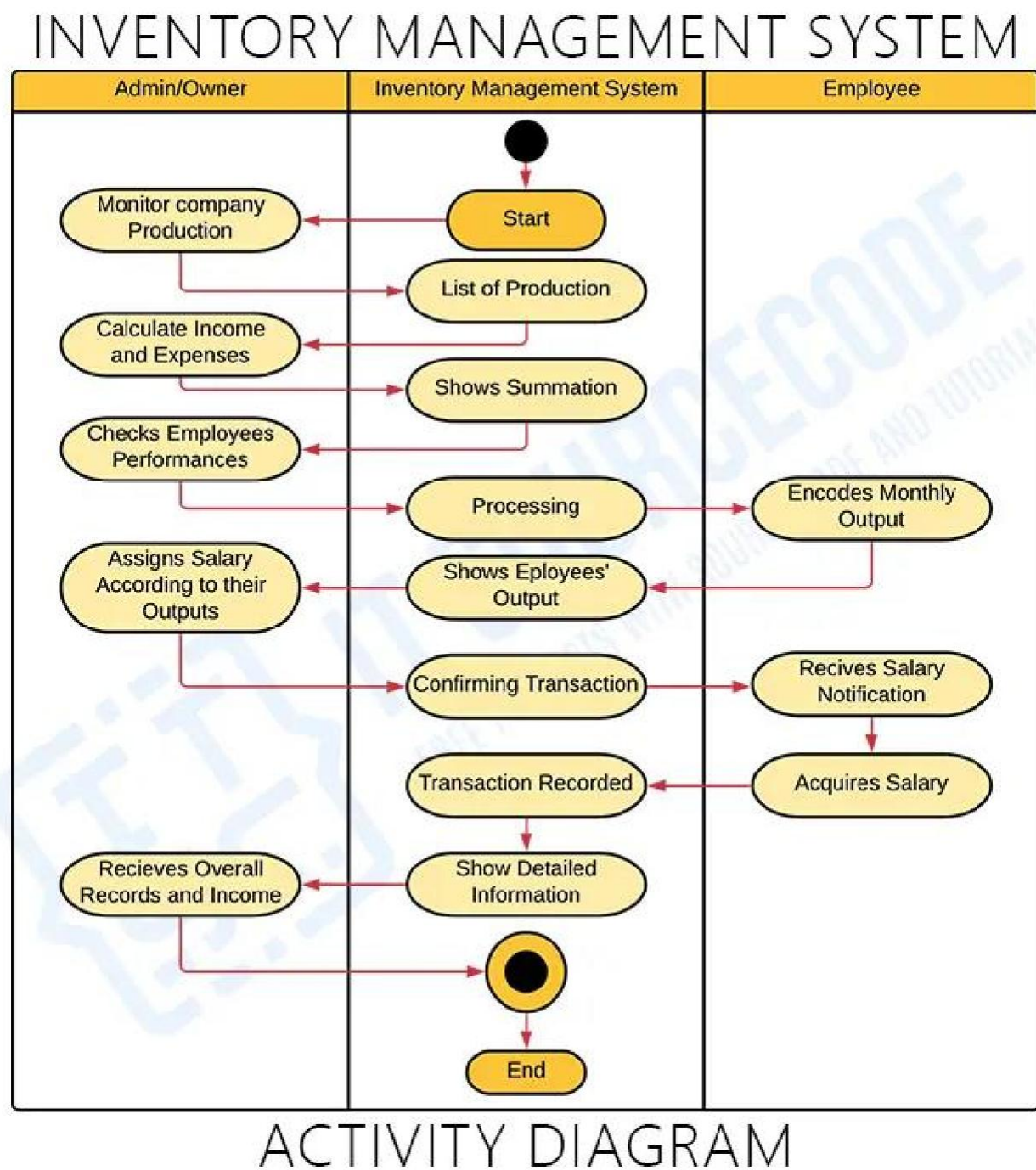
Inventory Management System Sequence Diagram

The designed sequence diagram illustrates the series of events that occurs in Inventory Management System. In this illustration, the actors are represented by a stick man and the transactions or classes are represented by objects. It will give you clear explanation about the behavior of an Inventory Management System in terms of processing the flow of instructions



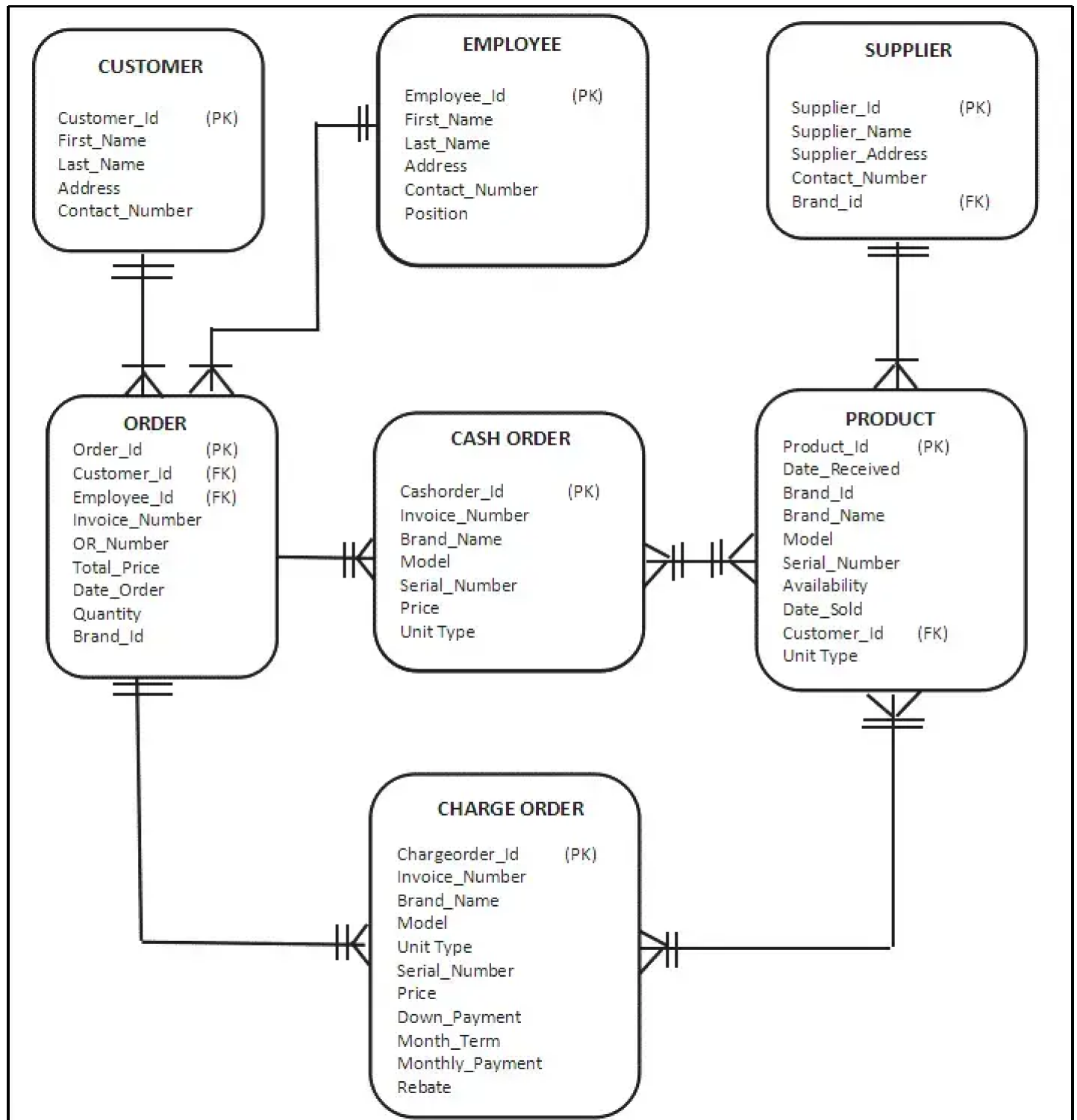
Inventory Management System Activity Diagram

Here's the UML activity diagram design of Inventory Management System that you can use for your own Final year Project. The UML activity Diagram is used to show the interaction of the user and the system. By creating it, you'll be able to see the flaws of the system and you may avoid it once you apply it to the project development. So it is important to have your diagrams designed first before jumping into its development.



Inventory Management System Activity Diagram

The Activity Diagram for the Inventory Management System (IMS) provides a visual representation of the workflow involved in managing inventory within the system. It outlines the various activities performed by system actors and how they interact with the IMS to accomplish tasks such as adding inventory items, updating quantities, placing orders, and generating reports.



4.5. ER Diagram

ENTITY RELATIONSHIP DIAGRAM

The Entity-Relationship (ER) model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram, which is used to visually represent data objects. The model has been extended and today it is commonly used for database design. The entity relationship diagram is based on a perception of real world that consists of a collection of basic objects, called entities and of relationship among the objects. Entities are described in a database by a set of attributes. The set of all entities of the same type, and the set of all relationships of the same type, are termed as an entity set, and relationship set respectively. The overall logical structure of a database can be expressed graphically by an entity relationship diagram, which is built up using the notations.

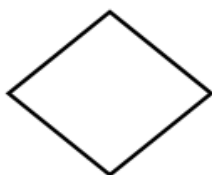
FEATURES OF ER MODEL ARE

- It maps well to relational model.
- It is simple and easy to understand with minimum of training. Therefore, database designer to communicate with the end user can use the model.
- In addition, the model can be used as a design plan by the database to implement a data model in specific database management software.

ER-DIAGRM SYMBOL



Entity

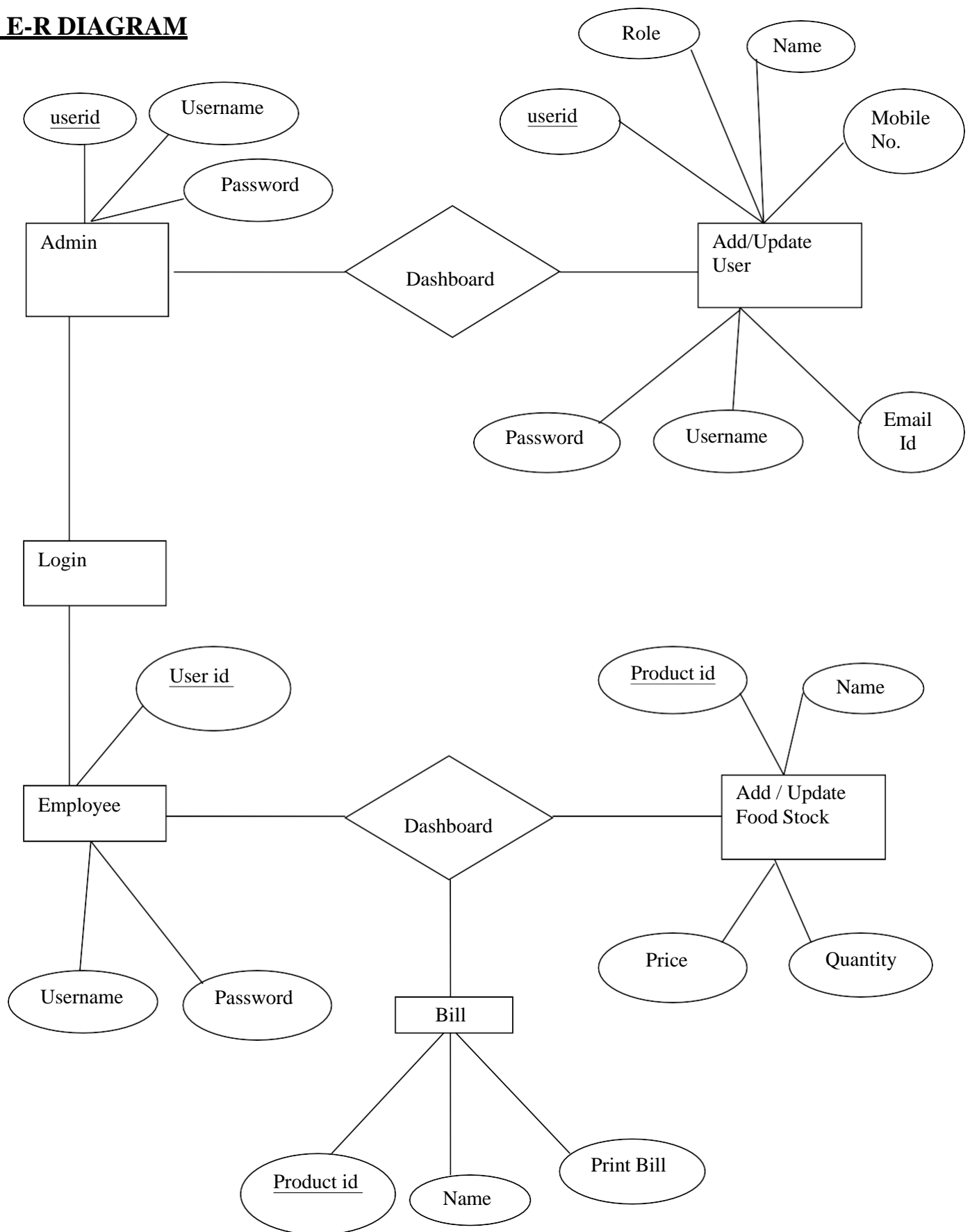


Relationship



Attributes

E-R DIAGRAM



DATABASE TABLES

The Management system of Zesty Café Restaurant is fully connected with the database SQL. The main of choosing this database is the flexibility and the data security that can provide to the entire program. The general theme behind database is to handle information as a whole. There is no artificiality that is normally embedded in separate files or applications. A database is a collection of interrelated data stored with minimum redundancy to serve many users quickly and efficiently. The general objective is to make information access easy, quick, inexpensive and flexible for the user.

Database design is the most critical part of the Design phase. An elegantly designed, well defined Database is a strong foundation for the whole system. The tables are classified and fully normalized by inputting with Primary Keys and Foreign Keys for each table.

TABLE DESIGN

Table Name: tbluser

Primary Key: uid

Field Name	Data type	Constraint	Description
uid	int	Not Null	User Id
urole	varchar(50)	Not Null	User Role
uname	varchar(50)	Not Null	User Name
DOB	varchar(50)	Not Null	DOB
cno	bigint	Not Null	Contact No
eid	varchar(50)	Not Null	Email Id
ulname	varchar(50)	Not Null	User Login Id

ulpass	varchar(50)	Not Null	User Login Password
--------	-------------	----------	---------------------

Description: This table is used for storing user details.

Table Name: tblfood

Primary Key: mid

Field Name	Data type	Constraint	Description
mid	int	Not Null	Product Id
mname	varchar(50)	Not Null	Product Name
mnumber	varchar(50)	Not Null	Product Number
mqty	bigint	Not Null	Product Quantity
perunit	bigunit	Not Null	Product Rate

Description: This table is used for storing product details

process. Testing phase is the development phase that validates the code against the functional specifications. Testing is a vital to the achievement of the system goals. The objective of testing is to discover errors. To fulfill this objective a series of test step such as the unit test, integration test, validation and system test where planned and executed.

5.SYSTEM TESTING

5.1. Software Testing

System testing is the state of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It certifies that the whole set of programs hang together. System testing requires a test plan that consists of several key activities and steps for run program, string, system and user acceptance testing. The implementation of newly design package is important in adopting a successful new system. Testing is important stage in software development. System test is implementation should be a confirmation that all is correct and an opportunity to show the users that the system works as they expected. It accounts the largest percentage of technical effort in software development process. Testing phase is the development phase that validates the code against the functional specifications. Testing is a vital to the achievement of the system goals. The objective of testing is to discover errors. To fulfill this objective a series of test step such as the unit test, integration test, validation and system test where planned and executed.

UNIT TESTING

Here each program is tested individually so any error apply unit is debugged. The sample data are given for the unit testing. The unit test results are recorded for further references. During unit testing the functions of the program unit validation and the limitations are tested. Unit testing is testing changes made in a existing or new program this test is carried out during the programming and each module is found to be working satisfactorily. For example in the registration form after entering all the fields we click the submit button. When submit button is clicked, all the data in form are validated. Only after validation entries will be added to the database. Unit testing comprises the set of tests performed by an individual prior to integration of the unit into large system. The situation is illustrated in as follows Coding-> Debugging ->Unit Testing -> Integration testing-> Output Testing. The four categories of test that a programmer will typically perform on a program unit

- Functional test
- Performance test
- Stress Test
- Structure test

Functional test involve exercising the code with nominal input values for which the expected results are known as well as boundary values and special values. Performance testing determines the amount of execution time spent in various parts of unit program through put and response time and device utilization by

5.1.1-Unit Testing

the program. A variation of stress testing called sensitivity testing in some situations a very small range of data contained in the bounds of valid data may cause extreme and even erroneous processing or profound performance degradation. Structured testing is concerned with exercising the internal logic of a program and traversing paths. Functional testing, stress testing performance testing are referred to as “black box” testing and structure testing is referred to as “white box” testing. Field testing will be performed manually and functional tests will be written in detail.

Test Objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format.
- No duplicate entries should be allowed.
- All links should take the user to the correct page.
-

5.1.2 INTEGRATED TESTING

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

TEST RESULTS

All the test cases mentioned above passed successfully. No defects encountered.

FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. **Functional testing is centered on the following items:**

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

WHITE BOX TESTING

White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing.

BLACK BOX TESTING

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied virtually to every level of software testing: unit, integration, system and acceptance. It is a testing in which the software under test is treated, as a black box .you cannot —see into it. The test provides inputs and responds to outputs without considering how the software works.

ACCEPTANCE TESTING

Acceptance testing involves planning an execution of a functional test, performance test and stress test to verify that the implemented system satisfies the requirement. The acceptance testing is the final stage of the user the various possibilities of the data are entered and the results are tested.

VALIDATION TESTING

Software validation is achieved through a series of test that demonstrates the conformity and requirements. Thus the proposed system under consideration has to be tested by validation and found to be working satisfactorily. For example in customer enters phone number field should contain number otherwise it

6. SYSTEM IMPLEMENTATION

6.1. System Description

- Certainly! Here's a system description for an Inventory Management System (IMS) for a restaurant:
- System Description: Restaurant Inventory Management System (RIMS)

1. Introduction

- The Restaurant Inventory Management System (RIMS) is a comprehensive software solution designed to streamline inventory control processes within restaurants, cafes, and other food service establishments. RIMS automates and simplifies inventory management tasks, including tracking inventory levels, managing stock, placing orders, and generating reports. By providing real-time visibility into inventory data and streamlining operations, RIMS empowers restaurant owners and managers to optimize inventory usage, reduce costs, minimize wastage, and enhance profitability.

2. Key Features

- Inventory Tracking: RIMS allows restaurant staff to track inventory levels for all ingredients, raw materials, and finished products in real-time. It provides visibility into stock quantities, locations, and movement within the restaurant.
- Recipe Management: The system includes a recipe management module that enables chefs to create, store, and manage recipes for menu items. Chefs can specify the ingredients, quantities, and preparation instructions for each recipe, and the system automatically updates inventory levels as items are used.
- Supplier Management: RIMS facilitates efficient supplier management by maintaining supplier databases, tracking supplier information, and managing purchase orders. Restaurant staff can create and send purchase orders to suppliers, track order status, and manage deliveries.
- Order Management: The system streamlines order management processes, allowing restaurant staff to place orders for inventory items directly within the system. Automated reorder points and notifications help ensure that essential items are restocked in a timely manner.
- Reporting and Analysis: RIMS provides robust reporting and analysis capabilities, allowing restaurant owners and managers to gain insights into inventory usage, trends, and performance. Standard reports include inventory levels, usage history, purchase history, and cost analysis.

3. System Architecture

- RIMS follows a three-tier architecture model, comprising the presentation layer, application layer, and data layer. The presentation layer includes user interfaces for interacting with the system, while the application layer contains the business logic and application services. The data layer is responsible for storing and managing inventory data in a relational database.

4. Deployment Options

- RIMS can be deployed in various environments, including on-premises servers, cloud platforms, or hybrid configurations. On-premises deployment offers full control and customization options, while cloud deployment provides scalability, reliability, and flexibility. Hybrid deployment combines elements of both deployment models to leverage the benefits of each.

5. Benefits

- **Efficiency:** RIMS streamlines inventory management processes, saving time and effort for restaurant staff. Automation reduces manual errors and improves accuracy in inventory tracking and ordering.
 - **Cost Reduction:** By optimizing inventory levels, reducing wastage, and leveraging data-driven insights, RIMS helps restaurants minimize costs and improve profitability.
 - **Visibility and Control:** The system provides real-time visibility into inventory data, allowing restaurant owners and managers to make informed decisions and maintain control over inventory operations.
 - **Scalability:** RIMS is scalable and adaptable to the needs of restaurants of all sizes, from small cafes to large restaurant chains. It can grow with the business and accommodate changing requirements over time.
- The Restaurant Inventory Management System (RIMS) is a powerful tool for streamlining inventory control processes and optimizing operations within restaurants. With its user-friendly interface, advanced features, and flexible deployment options, RIMS empowers restaurant owners and managers to efficiently manage inventory, reduce costs, and improve profitability. By leveraging automation, data-driven insights, and scalable technology, RIMS helps restaurants stay competitive in a dynamic and challenging market environment.

6.2. System Flow

- Below is a description of the system flow for the Restaurant Inventory Management System (RIMS) in a two-page format:
- System Flow for Restaurant Inventory Management System (RIMS)
- Overview
- The system flow for the Restaurant Inventory Management System (RIMS) outlines the sequence of steps and interactions involved in managing inventory within a restaurant. It provides a comprehensive overview of how users interact with the system to perform tasks such as adding inventory items, updating quantities, placing orders, receiving deliveries, and generating reports.
- High-Level System Flow
 - **Step 1: User Authentication**
 - The system begins with user authentication, where restaurant staff log in using their credentials

(username and password) to access the RIMS dashboard.

➤ Step 2: Dashboard Navigation

- -Upon successful login, users are directed to the RIMS dashboard, where they can view key metrics, alerts, and navigation options. The dashboard provides an overview of inventory levels, pending orders, and recent transactions.

➤ Step 3: Inventory Management

- Users can navigate to the Inventory Management module to perform various inventory-related tasks, including adding new inventory items, updating quantities, and viewing inventory details. They can search for specific items, filter by category or supplier, and perform bulk actions.

➤ Step 4: Recipe Management

- The Recipe Management module allows chefs and kitchen staff to create, edit, and manage recipes for menu items. They can specify ingredients, quantities, preparation instructions, and serving sizes. The system automatically updates inventory levels as items are used in recipes

➤ Step 5: Supplier Management

- Users can access the Supplier Management module to manage supplier information, track purchase orders, and communicate with suppliers. They can create new suppliers, view supplier details, and send purchase orders for replenishing inventory stock.

➤ Step 6: Order Placement

- When inventory levels are low, users can initiate the order placement process by creating purchase orders within the system. They specify the items, quantities, delivery dates, and supplier details. Automated reorder points and notifications help streamline the ordering process.

➤ Step 7: Receiving Deliveries

- Upon receiving inventory deliveries from suppliers, users update the system to reflect the received quantities. They verify the items against the purchase orders, reconcile any discrepancies, and update inventory levels accordingly.

➤ Step 8: Reporting and Analysis

- Users can generate various reports and analytics to gain insights into inventory usage, trends, and performance. Standard reports include inventory levels, usage history, purchase history, and cost analysis. Customizable dashboards and visualizations provide a graphical representation of data.

- The system flow for the Restaurant Inventory Management System (RIMS) provides a structured overview of how users interact with the system to manage inventory operations effectively. By following a predefined sequence of steps, users can navigate through different modules, perform tasks, and access relevant information to make informed decisions and optimize inventory management processes within the restaurant. This systematic approach ensures efficiency, accuracy, and control over inventory operations, ultimately contributing to the success and profitability of the restaurant.

6.3. Modules Description

The Restaurant Inventory Management System (RIMS) comprises several interconnected modules designed to streamline inventory control processes and optimize restaurant operations. The Inventory Management module serves as the core component, allowing users to add, update, and track inventory items in real-time. Chefs and kitchen staff utilize the Recipe Management module to create, store, and manage recipes for menu items, specifying ingredients, quantities, and preparation instructions. Supplier Management facilitates efficient supplier communication and purchase order management, enabling users to maintain supplier databases, track orders, and manage deliveries seamlessly. The Order Placement module automates the ordering process by generating purchase orders based on predefined reorder points and inventory thresholds, ensuring timely replenishment of stock. Upon receiving deliveries, users update inventory levels and reconcile received quantities using the Receiving Deliveries module. Finally, the Reporting and Analysis module provides comprehensive reporting and analytics capabilities, enabling users to generate insights into inventory usage, trends, and performance metrics, empowering informed decision-making and strategic planning. Together, these modules form a cohesive system that empowers restaurant owners and managers to optimize inventory operations, reduce costs, and improve profitability.

Moreover, the Employee Management module facilitates the administration of user accounts, roles, and permissions within the system, ensuring secure access and accountability. Restaurant managers can assign roles to employees based on their responsibilities, granting appropriate permissions to view, edit, or approve inventory-related tasks. Additionally, the Forecasting and Planning module utilizes historical data and predictive analytics to forecast demand, optimize inventory levels, and plan future inventory requirements. By analyzing factors such as seasonality, trends, and customer preferences, restaurant owners can make informed decisions regarding inventory stocking levels, promotions, and menu offerings. Furthermore, the Integration and Customization module enables seamless integration with external systems such as point-of-sale (POS) systems, accounting software, and supply chain management systems, enhancing interoperability and data exchange. Customization options allow restaurants to tailor RIMS to their specific requirements, configuring workflows, reports, and dashboards to align with their business processes. Overall, these modules work synergistically to provide a comprehensive and adaptable solution for inventory management in the food service industry, empowering restaurants to optimize efficiency, reduce costs, and deliver exceptional dining experiences.

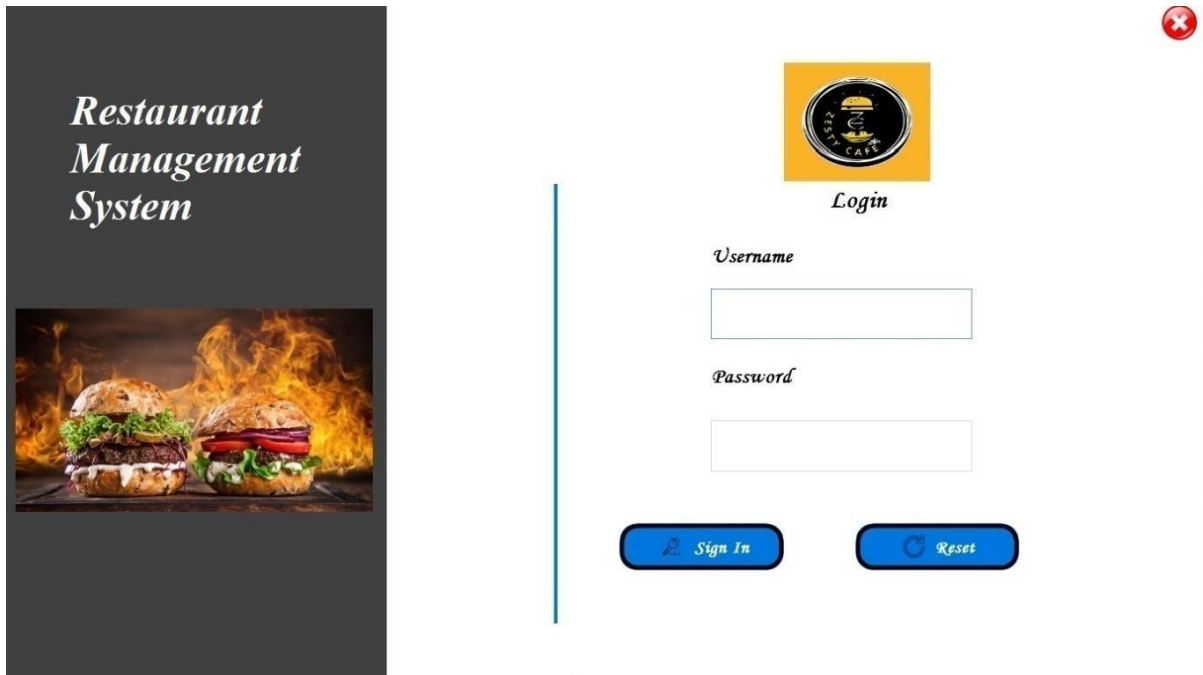
Furthermore, the Alerts and Notifications module provides proactive alerts and notifications to users based on predefined triggers and thresholds. This module monitors inventory levels, order statuses, and critical events, sending notifications to relevant stakeholders via email, SMS, or in-app notifications. Alerts can notify users of low inventory levels, pending orders, delivery delays, or discrepancies in received quantities, enabling timely action and preventing stockouts or overstock situations. By keeping users informed of important events and tasks, the Alerts and Notifications module enhances operational efficiency, minimizes risks, and ensures smooth inventory management processes within the restaurant.

7.


APPENDICES

7.1. Screenshots

LOGIN





Restaurant Management System



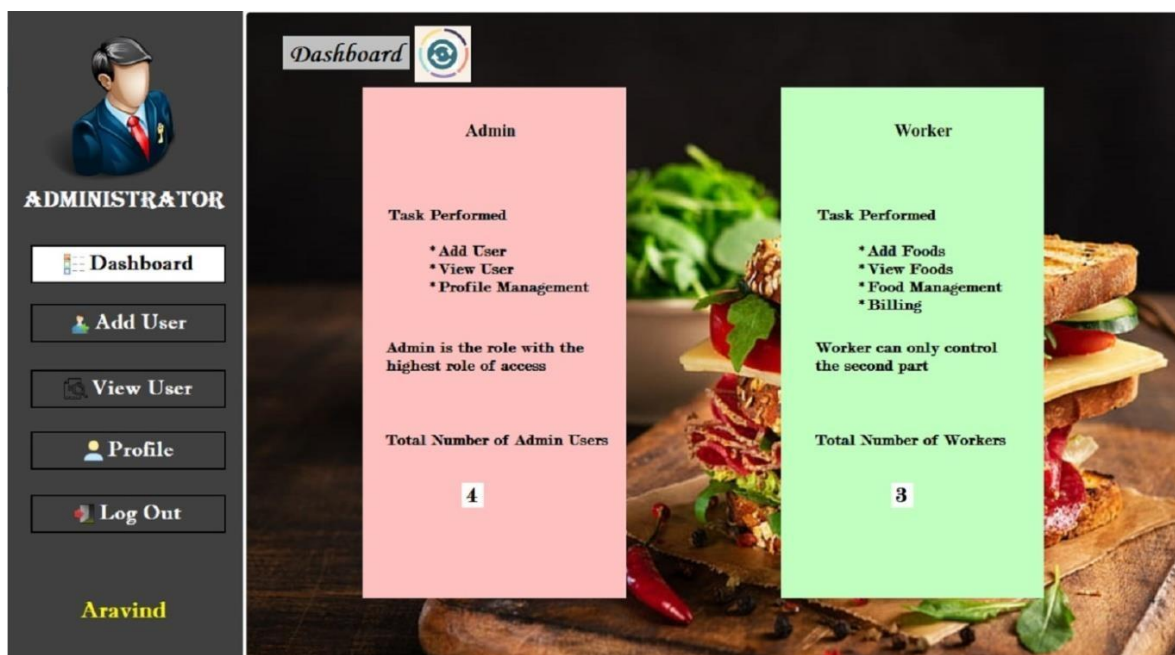
Login


Username


Password


 *Sign In*  *Reset*


ADMIN DASHBBOARD





 **ADMINISTRATOR**

 **Dashboard**

 **Add User**

 **View User**

 **Profile**

 **Log Out**

Aravind

Dashboard

Admin

Task Performed

- * Add User
- * View User
- * Profile Management

Admin is the role with the highest role of access

Total Number of Admin Users

4

Worker

Task Performed


- * Add Foods
- * View Foods
- * Food Management
- * Billing

Worker can only control the second part

Total Number of Workers

3

ADD USER



ADMINISTRATOR

Dashboard

Add User

View User

Profile

Log Out

Aravind

Add User

User Role

Name

DOB (Date of Birth)

Sunday, April 18, 2021

Mobile No.

Email Address

Username


Password

Sign Up

Reset

**Check User Role once before Sign Up*

VIEW USER (REPORT)



ADMINISTRATOR

Dashboard

Add User

View User

Profile

Log Out

Aravind

View User


Username

Search


19	Administrator	Aravind K	Monday, June 25, ...	9003229567	aravind@gmail.com	a	a
20	Worker	Aravind K	Monday, June 25, ...	9003229566	aravind@gmail.com	aw	aw
21	Administrator	Sam	Tuesday, Decembe...	9500533628	victorsam26@gma...	s	s
22	Worker	Sam	Tuesday, Decembe...	9500533628	victorsam26@gma...	sw	sw
24	Administrator	Mani Prasad	Thursday, May 24,...	7845056086	maniprasad@gmai...	m	m
25	Worker	Mani Prasad	Thursday, May 24,...	7845056086	maniprasad@gmai...	mw	mw
26	Administrator	Aravind	Monday, June 25, ...	9003229566	aravind@gmail.com	Aravind	a

Delete


PROFILE





ADMINISTRATOR

 Dashboard

 Add User


 View User

 **Profile**

 Log Out

Aravind

Profile



Aravind

User Role

Administrator

Name

Aravind

DOB (Date of Birth)

Monday, June 25, 2001

Mobile No.


9003229566

Email Address

aravind@gmail.com

Password

a

*Check the details before UPDATION

EMPLOYEE LOGIN

Restaurant Management System








Login

Username


aw

Password



EMPLOYEE DASHBOARD



WORKER

Dashboard

+ Add Food

View Food

Modify Food

Seller

Log Out

Dashboard

Add and View Food

Task Performed

- * Add Food
- * View Food
- * Food Management

In this, we can add food products and view them .

Update Foods

Task Performed

- * Add Foods
- * View Foods
- * Food Management
- * Billing

Here, we can update the Name of the product, Price, and also the quantity of them.


Billing

Task Performed

- * Search Food .
- * Add the product to the Billing Section .
- * Total Amount
- * Save and Print .

We will bill the items that the customer has ordered.

ADD FOOD



WORKER

Dashboard

+ Add Food

View Food

Modify Food

Seller

Log Out

Add Food

Food ID

Name

Food Number

Quantity

Price Per Unit

+ Add

Reset

VIEW FOOD (REPORT)



WORKER

[Dashboard](#)

[Add Food](#)

[View Food](#)

[Modify Food](#)

[Seller](#)

[Log Out](#)


View Food

Food Name

10	1	Veg Burger	1	50	60
11	2	Chicken Burger	2	50	75
12	3	Tower Chicken Burger	3	30	120
13	4	Veg Pizza	4	25	50
14	5	Chicken Pizza	5	15	100
15	6	Classic Chicken Pizza	6	10	150
16	7	Veg Wrap	7	28	60
17	8	Chicken Wrap	8	8	70
18	9	Chocolate Milkshake	9	12	50
19	10	StrawBerry Milkshake	10	12	50

[Delete](#)

UPDATE FOOD STOCK



WORKER

[Dashboard](#)

[Add Food](#)

[View Food](#)

[Modify Food](#)

[Seller](#)

[Log Out](#)

Update Food

Food ID

[Search](#)

Food Name

Food Number

Available Quantity


Add Quantity

Price Per Unit


[Update](#) [Reset](#)


*Check the details before UPDATION


BILLING





WORKER


 Dashboard


 Add Food

 View Food

 Modify Food

 Seller

 Log Out

Billing 

Search

Veg Burger

Chicken Burger

Tower Chicken Burge

Veg Pizza

Chicken Pizza

Classic Chicken Pizza

Veg Wrap

Chicken Wrap

Chocolate Milkshake

StrawBerry Milkshak


Food ID

Food Name


Price Per Unit

No. of Items


Total Price

 Add to Cart

Food ID	Food Name	Price Per Unit	No. of Units	Total Price
---------	-----------	----------------	--------------	-------------

 Remove

Rs. 00

 Purchase & Print

7.2. Source code

LOGIN FORM:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Restaurant_Management_System
{
    public partial class Login : Form
    {
        function fn = new function();
        String query;
        DataSet ds;
        public Login()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void btnExit_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

```

    }

    private void btnReset_Click(object sender, EventArgs e)
    {
        txtUsername.Clear();
        txtPassword.Clear();
    }

    private void btnSignIn_Click(object sender, EventArgs e)
    {
        query = "select * from users";
        ds = fn.getData(query);
        if (ds.Tables[0].Rows.Count == 0)
        {
            if (txtUsername.Text=="root" && txtPassword.Text ==
"root")
            {
                Administrator admin = new Administrator();
                admin.Show();
                this.Hide();
            }
        }
        else
        {
            query = "select * from users where username
='"+txtUsername.Text+"' and pass='"+txtPassword.Text+"'";
            ds = fn.getData(query);
            if (ds.Tables[0].Rows.Count != 0)
            {
                String role = ds.Tables[0].Rows[0][1].ToString();
                if (role == "Administrator")
                {
                    Administrator admin = new
Administrator(txtUsername.Text);
                    admin.Show();
                }
            }
        }
    }
}

```

```

        this.Hide();
    }
    else if (role == "Employee")
    {
        Worker work = new Worker();
        work.Show();
        this.Hide();
    }
}
else
{
    MessageBox.Show("Wrong Username or
Password", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error );
}
}

```

```

/*txtUsername.Text != "admin" || txtPassword.Text != "admin")
{
    MessageBox.Show("Wrong Username or Password", "Error",
    MessageBoxButtons.OK, MessageBoxIcon.Error);
    txtUsername.Text = "";
    txtPassword.Text = "";
}

else
{
    MessageBox.Show("Hey Admin - Welcome to Zesty Cafe",
"Login Success");
    Administrator am = new Administrator();
    am.Show();
    this.Hide();
}*/
}

```

```

        private void label3_Click(object sender, EventArgs e)
        {

        }
    }
}

```

ADD USER

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Restaurant_Management_System.AdministratorUC
{
    public partial class UC_AddUser : UserControl
    {

        function fn = new function();
        String query;

        public UC_AddUser()
        {
            InitializeComponent();
        }
    }
}

```

```

private void btnSignUp_Click(object sender, EventArgs e)
{
    String role = txtUserRole.Text;
    String name = txtName.Text;
    String dob = txtDob.Text;
    Int64 mobile = Int64.Parse(txtMobileNo.Text);
    String email = txtEmail.Text;
    String username = txtUsername.Text;
    String pass = txtPassword.Text;

    try
    {
        query = "insert into
users(userRole,name,dob,mobile,email,username,pass) values ('"+ role +
"', '" + name + "', '" + dob + "', '" + mobile + "', '" + email + "', '" +
username + "', '" + pass + "')";
        fn.setData(query, "Sign Up Successful.");
    }
    catch (Exception)
    {
        MessageBox.Show("Username already
exist.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void btnReset_Click(object sender, EventArgs e)
{
    clearall();
}

public void clearall()
{
    txtName.Clear();
    txtDob.ResetText();
    txtMobileNo.Clear();
}

```

```

        txtEmail.Clear();
        txtUsername.Clear();
        txtPassword.Clear();
        txtUserRole.SelectedIndex = -1;

    }

    private void txtUsername_TextChanged(object sender, EventArgs
e)
    {
        query = "select * from users where username='" +
txtUsername.Text + "'";
        DataSet ds = fn.getData(query);

        if (ds.Tables[0].Rows.Count == 0)
        {
            pictureBox1.ImageLocation = @"C:\Users\HOME\Desktop\Zesty
Cafe\C# Pharmacy Management System\Pharmacy Management System in
C#\yes.png";
        }
        else
        {
            pictureBox1.ImageLocation =
@"C:\Users\HOME\Desktop\Zesty Cafe\C# Pharmacy Management
System\Pharmacy Management System in C#\no.png";
        }
    }

    private void UC_AddUser_Load(object sender, EventArgs e)
    {

    }

}
}

```

USER REPORT

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Restaurant_Management_System.AdministratorUC
{
    public partial class UC_ViewUser : UserControl
    {
        function fn = new function();
        String query;
        String currentUser = "";

        public UC_ViewUser()
        {
            InitializeComponent();
        }

        public String ID
        {
            set { currentUser = value; }
        }

        private void UC_ViewUser_Load(object sender, EventArgs e)
        {
            query = "select * from users";
            DataSet ds = fn.getData(query);
            guna2DataGridView1.DataSource = ds.Tables[0];
        }
    }
}
```

```

    }

    private void btnSync_Click(object sender, EventArgs e)
    {
        UC_ViewUser_Load(this, null);
    }

    private void guna2Button1_Click(object sender, EventArgs e)
    {
        if (MessageBox.Show("Are you sure?", "Delete Confirmation
!", MessageBoxButtons.YesNo, MessageBoxIcon.Warning) ==
DialogResult.Yes)
        {

            if (currentUser != userName)
            {
                query = "delete from users where username='" +
userName + "'";

                fn.setData(query, "User Record Deleted.");
                UC_ViewUser_Load(this, null);
            }
            else
            {
                MessageBox.Show("You are trying to delete \n your own
Profile.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            }
        }
    }

    private void txtUserName_TextChanged(object sender, EventArgs
e)
    {
        query = "select * from users where username like
'" + txtSearch.Text + "%'"; //a%

```



```

        DataSet ds = fn.getData(query);
        guna2DataGridView1.DataSource = ds.Tables[0];
    }

    String userName;
    private void guna2DataGridView1_CellClick(object sender,
DataGridViewCellEventArgs e)
    {
        try
        {
            userName =
guna2DataGridView1.Rows[e.RowIndex].Cells[6].Value.ToString();
        }
        catch
        {
        }
    }
}

```

EMPLOYEE LOGIN FORM

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Restaurant_Management_System

```

```

{
    public partial class Login : Form
    {
        function fn = new function();
        String query;
        DataSet ds;
        public Login()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void btnExit_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void btnReset_Click(object sender, EventArgs e)
        {
            txtUsername.Clear();
            txtPassword.Clear();
        }

        private void btnSignIn_Click(object sender, EventArgs e)
        {
            query = "select * from users";
            ds = fn.getData(query);
            if (ds.Tables[0].Rows.Count == 0)
            {
                if(txtUsername.Text=="root" && txtPassword.Text ==
"root")

```

```

        {
            Administrator admin = new Administrator();
            admin.Show();
            this.Hide();
        }
    }
else
{
    query = "select * from users where username
='" + txtUsername.Text + "' and pass='" + txtPassword.Text + "'";
    ds = fn.getData(query);
    if (ds.Tables[0].Rows.Count != 0)
    {
        String role = ds.Tables[0].Rows[0][1].ToString();
        if (role == "Administrator")
        {
            Administrator admin = new
Administrator(txtUsername.Text);
            admin.Show();
            this.Hide();
        }
        else if (role == "Employee")
        {
            Worker work = new Worker();
            work.Show();
            this.Hide();
        }
    }
else
{
    MessageBox.Show("Wrong Username or
Password", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
}

```

```

/*txtUsername.Text != "admin" || txtPassword.Text != "admin")
    {
        MessageBox.Show("Wrong Username or Password", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        txtUsername.Text = "";
        txtPassword.Text = "";
    }

    else
    {
        MessageBox.Show("Hey Admin - Welcome to Zesty Cafe",
"Login Success");
        Administrator am = new Administrator();
        am.Show();
        this.Hide();
    }*/
}

private void label3_Click(object sender, EventArgs e)
{
    }
}
}

```

ADD FOOD

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;

```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Restaurant_Management_System.WorkerUC
{
    public partial class UC_W_AddFood : UserControl
    {
        function fn = new function();
        String query;
        public UC_W_AddFood()
        {
            InitializeComponent();
        }

        private void label4_Click(object sender, EventArgs e)
        {

        }

        private void btnAdd_Click(object sender, EventArgs e)
        {
            if (txtMediId.Text != "" && txtMediName.Text != "" &&
txtMediNumber.Text != "" && txtQuantity.Text != "" &&
txtPricePerUnit.Text != "")
            {
                String mid = txtMediId.Text;
                String mname = txtMediName.Text;
                String mnumber = txtMediNumber.Text;
                Int64 quantity = Int64.Parse(txtQuantity.Text);
                Int64 perunit = Int64.Parse(txtPricePerUnit.Text);
            }
        }
    }
}

```

```

        query = "insert into food
(mid,mname,mnumber,quantity,perUnit) values ('"+mid+ "','"+ mname + "','"+
+ mnumber + "','"+ quantity + "','"+ perunit + "')";
        fn.setData(query, "Food Items Added to Database.!");
    }
    else
    {
        MessageBox.Show("Enter all Data.", "Information",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}

private void btnReset_Click(object sender, EventArgs e)
{
    clearAll();
}

public void clearAll()
{
    txtMediId.Clear();
    txtMediName.Clear();
    txtMediNumber.Clear();
    txtQuantity.Clear();
    txtPricePerUnit.Clear();
}
}
}

```

UPDATE FOOD

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;

```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Restaurant_Management_System.WorkerUC
{
    public partial class UC_W_UpdateFood : UserControl
    {

        function fn = new function();
        String query;

        public UC_W_UpdateFood()
        {
            InitializeComponent();
        }

        private void btnSearch_Click(object sender, EventArgs e)
        {
            if (txtFoodId.Text != "")
            {
                query = "select * from food where mid = '" +
txtFoodId.Text + "'";
                DataSet ds = fn.getData(query);
                if (ds.Tables[0].Rows.Count != 0)
                {
                    txtFoodName.Text = ds.Tables[0].Rows[0][2].ToString();
                    txtFoodNumber.Text =
ds.Tables[0].Rows[0][3].ToString();
                    txtAvailableQuantity.Text =
ds.Tables[0].Rows[0][4].ToString();
                    txtPricePerUnit.Text =
ds.Tables[0].Rows[0][5].ToString();
                }
            }
        }
    }
}

```

```

        else
        {
            MessageBox.Show("No Food with ID: " + txtFoodId.Text +
" exists.", "Info", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
    else
    {
        clearAll();
    }
}

```

```

private void clearAll()
{
    txtFoodName.Clear();
    txtFoodNumber.Clear();
    txtAvailableQuantity.Clear();
    txtPricePerUnit.Clear();
    if(txtAddQuantity.Text != "0")
    {
        txtAddQuantity.Text = "0";
    }
    else
    {
        txtAddQuantity.Text = "0";
    }
}

```

```

private void btnReset_Click(object sender, EventArgs e)
{
    clearAll();
}

```

```

Int64 totalQuantity;

```



```

private void btnUpdate_Click(object sender, EventArgs e)
{
    String mname = txtFoodName.Text;
    String mnumber = txtFoodNumber.Text;
    Int64 quantity = Int64.Parse(txtAvailableQuantity.Text);
    Int64 addQuantity = Int64.Parse(txtAddQuantity.Text);
    Int64 unitprice = Int64.Parse(txtPricePerUnit.Text);

    totalQuantity = quantity + addQuantity;

    query = "update food set mname = '" + mname + "',mnumber = '"
+ mnumber + "',quantity =" + totalQuantity + ",perunit =" + unitprice + "
where mid = '" + txtFoodId.Text + "'";
    fn.setData(query, "Food Details Updated.!");
}
}
}

```

FOOD REPORT

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Restaurant_Management_System.WorkerUC
{

```

```

public partial class UC_W_ViewFood : UserControl
{
    function fn = new function();
    String query;

    public UC_W_ViewFood()
    {
        InitializeComponent();
    }

    private void UC_W_ViewFood_Load(object sender, EventArgs e)
    {
        query = "select * from food";
        setDataGridView(query);
    }

    private void txtFoodName_TextChanged(object sender, EventArgs e)
    {
        query = "select * from food where mname like
'" + txtFoodName.Text + "%'";
        setDataGridView(query);
    }

    private void setDataGridView(String query)
    {
        DataSet ds = fn.getData(query);
        guna2DataGridview1.DataSource = ds.Tables[0];
    }

    String foodId;
    private void guna2DataGridview1_CellClick(object sender,
DataGridViewCellEventArgs e)
    {
        try
        {

```

```

        foodId =
guna2DataGridView1.Rows[e.RowIndex].Cells[1].Value.ToString();
    }

    catch
    {

    }

}

private void btnDelete_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Are you sure?", "Delete Confirmation
!", MessageBoxButtons.YesNo, MessageBoxIcon.Warning) == DialogResult.Yes)
    {
        query = "delete from food where mid = '"+foodId+"'";
        fn.setData(query, "Food Record Deleted.");
        UC_W_ViewFood_Load(this, null);
    }
}

private void btnSync_Click(object sender, EventArgs e)
{
    UC_W_ViewFood_Load(this, null);
}

private void pictureBox1_Click(object sender, EventArgs e)
{
}

}
}

```

BILLING

```

using DGVPrinterHelper;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Restaurant_Management_System.WorkerUC
{
    public partial class UC_W_Billing : UserControl
    {
        function fn = new function();
        String query;
        DataSet ds;

        public UC_W_Billing()
        {
            InitializeComponent();
        }

        private void UC_W_Billing_Load(object sender, EventArgs e)
        {
            listBoxFoods.Items.Clear();
            query = "select mname from food where quantity >'0'";
            ds = fn.getData(query);

            for (int i = 0; i < ds.Tables[0].Rows.Count; i++)
            {

listBoxFoods.Items.Add(ds.Tables[0].Rows[i][0].ToString());

            }
        }
    }
}

```

```

    }

    private void txtSearch_TextChanged(object sender, EventArgs e)
    {
        listBoxFoods.Items.Clear();
        query = "select mname from food where mname like '" +
txtSearch.Text + "%' and quantity >'0' ";
        ds = fn.getData(query);

        for (int i = 0; i < ds.Tables[0].Rows.Count; i++)
        {

listBoxFoods.Items.Add(ds.Tables[0].Rows[i][0].ToString());
        }
    }

    private void listBoxFoods_SelectedIndexChanged(object sender,
EventArgs e)
    {
        txtNumberOfItems.Clear();

        String name =
listBoxFoods.GetItemText(listBoxFoods.SelectedItem);

        txtFoodName.Text = name;
        query = "select mid,perUnit from food where mname ='" + name
+ "'";
        ds = fn.getData(query);
        txtFoodId.Text = ds.Tables[0].Rows[0][0].ToString();
        txtPricePerUnit.Text = ds.Tables[0].Rows[0][1].ToString();
    }

    private void txtNumberOfItems_TextChanged(object sender,
EventArgs e)
    {

```

```

        if (txtNumberofItems.Text != "")
        {
            Int64 unitPrice = Int64.Parse(txtPricePerUnit.Text);
            Int64 noOfUnit = Int64.Parse(txtNumberofItems.Text);
            Int64 totalAmount = unitPrice * noOfUnit;
            txtTotalPrice.Text = totalAmount.ToString();
        }
        else
        {
            txtTotalPrice.Clear();
        }
    }

    private void btnSync_Click(object sender, EventArgs e)
    {
        UC_W_Billing_Load(this, null);
    }

    protected int n, totalamount = 0;
    protected Int64 quantity, newQuantity;

    private void btnAddtoCart_Click(object sender, EventArgs e)
    {
        if (txtFoodId.Text != "")
        {
            query = "select quantity from food where mid='" +
txtFoodId.Text + "'";
            ds = fn.getData(query);

            quantity =
Int64.Parse(ds.Tables[0].Rows[0][0].ToString());
            newQuantity = quantity -
Int64.Parse(txtNumberofItems.Text);

```

```

        if (newQuantity >= 0)
        {
            n = guna2DataGridView1.Rows.Add();
            guna2DataGridView1.Rows[n].Cells[0].Value =
txtFoodId.Text;
            guna2DataGridView1.Rows[n].Cells[1].Value =
txtFoodName.Text;
            guna2DataGridView1.Rows[n].Cells[2].Value =
txtPricePerUnit.Text;
            guna2DataGridView1.Rows[n].Cells[3].Value =
txtNumberofItems.Text;
            guna2DataGridView1.Rows[n].Cells[4].Value =
txtTotalPrice.Text;

            totalamount = totalamount +
int.Parse(txtTotalPrice.Text);
            totalLabel.Text = "Rs. " +      totalamount.ToString();

            query = "update food set quantity='" + newQuantity +
"' where mid = '" + txtFoodId.Text + "'";
            fn.setData(query, "Food Added.");
        }
        else
        {
            MessageBox.Show("Food is Out of Stock.\n Only " +
quantity + " left", "Warning !!", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }
        clearAll();
        UC_W_Billing_Load(this, null);
    }
    else
    {
        MessageBox.Show("Select Food First.", "Information !!",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}

```

```

        }
    }

    int valueAmount;
    String valueId;
    protected Int64 noOfItems;

    private void guna2DataGridView1_CellClick(object sender,
DataGridViewCellEventArgs e)
    {
        try
        {
            valueAmount =
int.Parse(guna2DataGridView1.Rows[e.RowIndex].Cells[4].Value.ToString())
;

            valueId =
guna2DataGridView1.Rows[e.RowIndex].Cells[0].Value.ToString();
            noOfItems =
Int64.Parse(guna2DataGridView1.Rows[e.RowIndex].Cells[3].Value.ToString(
));

        }
        catch (Exception)
        {

        }
    }

    private void btnRemove_Click(object sender, EventArgs e)
    {
        if (valueId != null)
        {
            try
            {

```



```

guna2DataGridView1.Rows.RemoveAt(this.guna2DataGridView1.SelectedRows[0]
.Index);

        }
        catch
        {

        }
        finally
        {
            query = "select quantity from food where mid      = '"
+ valueId + "'";

            ds = fn.getData(query);
            quantity =
Int64.Parse(ds.Tables[0].Rows[0][0].ToString());
            newQuantity = quantity + noOfItems;

            query = "update food set quantity = '" +
newQuantity + "' where mid = '" + valueId + "'";
            fn.setData(query, "Food Removed from      Cart.");
            totalamount = totalamount - valueAmount;
            totalLabel.Text = "Rs. " +      totalamount.ToString();
        }
        UC_W_Billing_Load(this, null);
    }
}

private void btnPurchasePrint_Click(object sender, EventArgs e)
{
    DGVPrinter print = new DGVPrinter();
    print.Title = "ZESTY CAFE";
    print.SubTitle = String.Format("Date:- {0}",
DateTime.Now.Date);

```

```

        print.SubTitleFormatFlags = StringFormatFlags.LineLimit |
StringFormatFlags.NoClip;
        print.PageNumbers = true;
        print.PageNumberInHeader = false;
        print.PorportionalColumns = true;
        print.HeaderCellAlignment = StringAlignment.Near;
        print.Footer = "Total Payable Amount : " +
totalLabel.Text;
        print.FooterSpacing = 15;
        print.PrintDataGridview(guna2DataGridview1);

        totalamount = 0;
        totalLabel.Text = "Rs. 00";
        guna2DataGridview1.DataSource = 0;
    }

    private void clearAll()
    {
        txtFoodId.Clear();
        txtFoodName.Clear();
        txtPricePerUnit.Clear();
        txtNumberofItems.Clear();
    }
}
}

```

8.CONCLUSION

8.1. Conclusion

The main objective of the project is to bring a full-fledged computerized organization, and to enable the transaction details to maintain records, which makes of the employees easier. Thus, the proposed system has been developed with good amount of flexibility without compromising on the response time. Computerization of the entire system will enhance more accuracy and reduces major part of clerical works. Fast, clear and legible reports can be generated without any ambiguity. Integrated database design and ease of maintenance is a major advantage of the system. User friendliness is a unique feather of the system. Hence by developing a system that is user-friendly in nature, many users are able to work on the system with little of computer knowledge and training.

8.2. Future Enhancement

The project has been developed and the objectives are achieved successfully. The project has been developed with front end as VB.Net and backend as SQL Server. The frontend can also be changed. ASP.Net can replace the front-end tool such as HTML and CSS for more speed. The system is currently developed and ready for implementation to include the system is highly feasible and user friendly. To provide better facility regarding security, it uses security provider software and we access any place and any where use that. It can have an enhancement on proper in the future according to the user's requirements.

- Can add more modules.
- Can change the tools used according to the user requirements.
- Can add more type of graph design.
- Can add these modules to the present ongoing project.
- Can add, design and change various types of reports.

9.BIBLIOGRAPHY

9.1. Journal References

1. Title:"A Review of Inventory Management Research in Major Logistics Journals: Themes and Future Directions"
 - Authors: Kevin B. Hendricks, Vinod R. Singhal
 - Journal: European Journal of Operational Research
 - Year:2005
 - Link: [DOI: 10.1016/j.ejor.2005.01.026](https://doi.org/10.1016/j.ejor.2005.01.026)
2. Title : "Restaurant Inventory Management: A Literature Review"
 - Authors: Elnaz S. Iranmanesh, Michelle R. Worosz
 - Journal: International Journal of Hospitality Management
 - Year: 2018
 - Link: [DOI: 10.1016/j.ijhm.2017.09.011](https://doi.org/10.1016/j.ijhm.2017.09.011)
3. Title: "Inventory Management in a Restaurant: A Case Study of a Successful Restaurant in Quebec City"
 - Authors:Dina M. El-Khoury, Michel Doumpos, Marc Ménard
 - Journal: International Journal of Production Economics
 - Year:2016
 - Link: [DOI: 10.1016/j.ijpe.2015.12.010](https://doi.org/10.1016/j.ijpe.2015.12.010)
4. Title: "Supply Chain Management in the Fast Food Industry: A McDonald's Perspective"
 - Authors: A. Michael Knemeyer, Kevin J. Linderman, Douglas M. Murphy
 - Journal: Journal of Business Logistics
 - Year:2003
 - Link: [DOI: 10.1002/j.2158-1592.2003.tb00021.x](https://doi.org/10.1002/j.2158-1592.2003.tb00021.x)
5. Title: "Inventory Control in Restaurants: A Research Framework"
 - Authors: Evangelos D. Diamantopoulos, Andreas S. Andreou
 - Journal: British Food Journal
 - Year: 1999
 - Link: [DOI: 10.1108/00070709910278993](https://doi.org/10.1108/00070709910278993)

These journal references provide valuable insights into inventory management practices, challenges, and strategies within the restaurant industry, offering a solid foundation for research and development in the field.

9.2. Book References

1. to Program (2nd Edition) by Harvey M. Deitel, Paul J. Deitel, Tem R. Nieto. Elias Awath, “**SYSTEM ANALYSIS AND DESIGN**”, Tata Mc Graw Hill Publication, Sixth Edition, 2003.
2. S.Ramachandran, ”**COMPUTER AIDED DESIGN**”, Air Walk Publication, Third Edition,2003.
3. Richard Fairley, ”**SOFTWARE ENGINEERING CONCEPTS**”, Tata Mc Graw Hill Publication, Second Edition, 1997.
4. Professional VB.NET, 2nd Edition by Fred Barwell.
5. The .NET Languages: A Quick Translation Guide by Brian Bisch.
6. Programming VB.NET: A Guide for Experienced Programmers by Gary Cornell, Jonathan Morrison.
7. Learning Visual Basic.NET Through Applications by Clayton Crooks II
8. Visual Basic .NET How

9.3. Web References

1. www.tutorialspoint.com/vb.net
2. www.geeksforgeeks.org/sqlserver
3. www.w3schools.com/tutorials/vb.net
4. www.stackoverflow.com