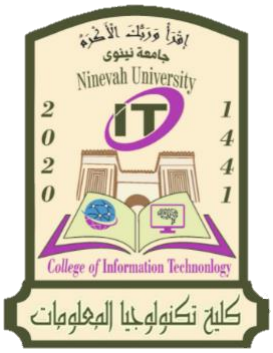

وزارة التعليم العالي والبحث العلمي
جامعة نينوى
كلية تكنلوجيا المعلومات

Project Management

inventory management system for a grocery store using a linear approach

رهيد فادي بطرس

Collage of information technology

Software Department

AGENDA

- Introduction
- Requirements Gathering
- Design
- Implementation
- Testing
- Deployment
- Maintenance
- Conclusion
- Future Enhancements
- The Final Code

Introduction

The purpose of this report is to outline the linear approach used in the development of a Grocery Store Inventory Management System. The system aims to assist store owners or managers in efficiently managing their inventory, tracking stock levels, and generating reports to aid in decision-making processes.

Requirements Gathering

- Conducted meetings with the store owner or manager to understand their specific needs and challenges related to inventory management.
- Identified key features required, including inventory tracking, stock management, and report generation capabilities.
- Documented the requirements to serve as a guideline throughout the development process.

Design

- Developed a database schema to organize and store product information efficiently, considering factors such as product name, category, quantity, and price.
- Designed user interfaces with input from stakeholders to ensure usability and functionality.
- Planned the system architecture, selecting appropriate technologies and frameworks based on the project requirements and constraints.

Implementation

- Implemented the database using SQL (Structured Query Language) to create tables for storing product data and managing relationships between entities.
- Developed the frontend using Python with the Flask framework, providing a user-friendly interface for interacting with the system.
- Connected the frontend and backend components to enable seamless data exchange and functionality.
- Implemented core features such as adding new products, updating inventory levels, and generating reports based on user inputs.

Testing

- Conducted unit testing to validate individual components and ensure they function correctly according to specifications.
- Performed integration testing to verify the interaction between different parts of the system and identify any potential issues.
- Engaged in user acceptance testing with stakeholders to gather feedback and ensure the system meets their expectations and requirements.

Deployment

- Deployed the system to a server or cloud platform, making it accessible to authorized users within the grocery store.
- Configured security measures to protect sensitive data and ensure secure access to the system.
- Provided training sessions for store employees to familiarize them with the system and its features.

Maintenance

- Established procedures for monitoring the system's performance and addressing any issues or bugs that arise post-deployment.
- Gathered feedback from users to identify areas for improvement and implement necessary updates and enhancements.
- Ensured the system remains up to date with technological advancements and evolving business requirements.

Conclusion

The Grocery Store Inventory Management System provides an efficient solution for grocery stores to manage their inventory effectively. By following a structured approach encompassing requirement gathering, design, implementation, testing, deployment, and maintenance, we have delivered a robust and user-friendly system that meets the needs of our client.

Future Enhancements

- Integration with barcode scanners for faster inventory updates.
- Implementation of predictive analytics to forecast demand and optimize stock levels.
- Integration with online ordering platforms for seamless inventory management across multiple channels.
-

The Final Code

```
class Item:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

    def update_quantity(self, change):
        self.quantity += change

    def __str__(self):
        return f"{self.name}: ${self.price:.2f}
({self.quantity})"

class Inventory:
    def __init__(self):
        self.items = {}

    def add_item(self, item):
        if item.name in self.items:

self.items[item.name].update_quantity(item.quantity)
        else:
            self.items[item.name] = item
```

```
    def remove_item(self, name, quantit)
        if name in self.items:
            item = self.items[name]
            if item.quantity >= quantity:
                item.update_quantity(-quantity)
            else:
                print(f"Insufficient quantity of {name}.
Only {item.quantity} available.")
        else:
            print(f"Item {name} not found in inventory.")

    def sell_item(self, name, quantity):
        self.remove_item(name, quantity)

    def list_inventory(self):
        if not self.items:
            print("Inventory is empty.")
        else:
            for item in self.items.values():
                print(item)

# Create some sample items
apple = Item("Apple", 0.5, 10)
banana = Item("Banana", 0.75, 5)
milk = Item("Milk", 2.5, 3)
# Create an inventory instance
inventory = Inventory()
# Add items to the inventory
inventory.add_item(apple)
inventory.add_item(banana)
inventory.add_item(milk)
# Print the initial inventory
print("Initial inventory:")
inventory.list_inventory()
# Sell some items
inventory.sell_item("Apple", 2)
inventory.sell_item("Banana", 1)
# Print the updated inventory
print("\nInventory after selling:")
inventory.list_inventory()
# Add more items
inventory.add_item(Item("Orange", 1.0, 7))
inventory.add_item(Item("Bread", 2.0, 2))
# Print the final inventory
print("\nFinal inventory:")
inventory.list_inventory()
```