## INVENTORY MANAGEMENT SYSTEM

## A MINI-PROJECT REPORT

***Submitted by***

## VEERARAGHAVAN M 220701313

## UMESH SUBRAMANIAN S 220701307

***in partial fulfillment of the award of the degree***

***of***

## BACHELOR OF ENGINEERING

**IN**

**COMPUTER SCIENCE AND ENGINEERING**



**RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI**

**An Autonomous Institute**

## CHENNAI-602105 NOV-DEC

**2023**

**BONAFIDE CERTIFICATE**

Certified that this project **“INVENTORY MANAGEMENT SYSTEM”** is the bonafide work of **“VEERARAGAHAVAN M (220701313), UMESH SUBRAMANIAN S (220701307)”** who carried out the project work under mysupervision.

SIGNATURE SIGNATURE

**Dr. R.SABITHA Ms. V.JANANEE**

Professor and Academic Head, Assistant Professor(SG)

Computer Science and Engineering, Computer Science and Engineering, Rajalakshmi Engineering College Rajalakshmi Engineering College, (Autonomous), (Autonomous),

Thandalam,Chennai-602 105 Thandalam,Chennai-602 105

Submitted for the Practical examination to be Held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

INTERNAL EXAMINER EXTERNAL EXAMINER

**ABSTRACT**

This project presents an innovative solution to the problems encountered by the local transportation industry, where productivity is hampered by the lack of user-friendly software. To address this problem, our team has created an advanced database system with the goal of streamlining the management and organizing of taxi data. In addition to addressing the inefficiencies of the local market, the system places regional transportation businesses in a competitive position against large corporations such as Ola and Uber by streamlining the allocation process according to consumer demand. The main objective is to provide local agencies with the means to provide better and more effective services in the event that an easy-to-use application system is not available.

The aforementioned project represents a noteworthy advancement in augmenting the competitive advantage of local transportation services. By carefully allocating the available cabs, the project aims to improve overall service delivery standards and offer a solution that is both customer-focused and responsive. The ultimate goal is to strengthen the whole local transportation ecosystem by giving local agencies the ability to outperform larger- scale platforms.

**TABLE OF CONTENTS**

1. **INTRODUCTION**

1.1 INTRODUCTION

1.2 SCOPE OF THE WORK

1.3 PROBLEM STATEMENT

1.4 IMPLEMENTATION

1. **SYSTEM SPECIFICATIONS**

2.1 HARDWARE SPECIFICATIONS

2.2 SOFTWARE SPECIFICATIONS

2.3 ER DIAGRAM

1. **CODING**
2. **SCREENSHOTS**
3. **CONCLUSION AND FUTURE ENHANCEMENT REFERENCES**

## CHAPTER 1

**INTRODUCTION**

* 1. **INTRODUCTION :**

The Inventory Management System is a comprehensive software application designed to streamline and manage inventory processes for businesses. This system enables users to efficiently add, view, update, and delete inventory items, providing a user-friendly interface for managing stock levels, tracking product details, and maintaining records. The application is developed using Python and PyQt5 for the graphical user interface and MySQL for the database management, ensuring robust data handling and secure user authentication.

* 1. **SCOPE OF THE WORK :**

The scope of this Inventory Management System includes the following functionalities :

**User Authentication :**

Secure login and registration for users. Password visibility toggle for ease of use.

**Dashboard :**

Centralized dashboard providing access to all major functionalities. Options to add, view, and update inventory items.

**Inventory Management :**

Adding new inventory items with details such as product name, quantity, and cost. Viewing all inventory items in a tabular format with options to delete items.

Updating existing inventory items.

**User Interface :**

Aesthetic and intuitive interface with background images and customized fonts. Buttons and layouts designed for ease of navigation and interaction.

**PROBLEM STATEMENT :**

Managing inventory is a critical task for businesses to ensure optimal stock levels, reduce losses due to overstocking or stockouts, and maintain accurate records for financial and operational efficiency. Traditional manual methods of inventory management are prone to errors, time-consuming, and inefficient. There is a need for an automated, user-friendly, and secure system that can handle the complexities of inventory management, provide real-time updates, and ensure data integriy .

* 1. **IMPLEMENTATION :**

The implementation of the Inventory Management System involves several key components and steps:

1. Database Configuration

The system uses MySQL for database management. A Database class is created to establish a connection to the MySQL database with the necessary credentials. The database contains two primary tables :

authentication: Stores user credentials for login.

Inventory : Stores inventory items with details like username, item name,

quantity, and cost.

1. User Authentication

The LoginWindow class provides a secure login interface. Users can enter their credentials, and the system verifies them against the database. The class also includes functionality for new user registration via the RegistrationWindow.

1. Dashboard

The DashboardWindow serves as the central hub for users after logging in. It provides buttons to navigate to different functionalities such as adding, viewing, and updating inventory items. A text widget displays a welcome message and system information.

1. Inventory Management

Adding Items: The AddItemWindow class allows users to add new inventory items by entering the product name, quantity, and cost. The data is then inserted into the inventory table in the database.

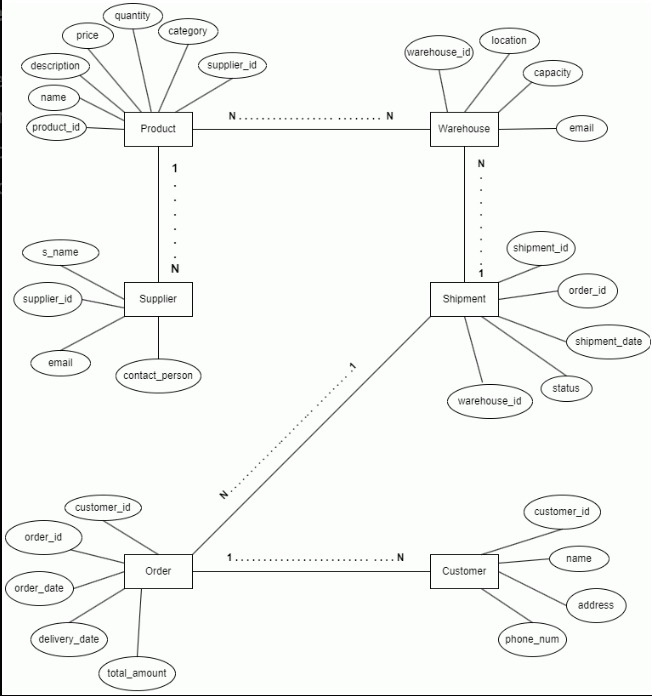
Viewing Items : The ViewItemsWindow class displays all inventory items in a tabular format. It also includes functionality to delete items directly from the table.

Updating Items : The UpdateItemsWindow class allows users to update the quantity and cost of existing inventory items. It retrieves data from the database, allows modifications, and updates the database accordingly.

1. User Interface Design

The application uses PyQt5 to create a visually appealing and user-friendly interface. Background images, custom fonts, and styles are applied to various widgets to enhance the user experience. The layout is designed to ensure easy navigation and interaction.

**ER DIAGRAM**



## CHAPTER 2

**SYSTEM SPECIFICATIONS**

* 1. **HARDWARE SPECIFICATIONS :**

PROCESSOR **:** Intel i5

RAM **:** 4GB

HARD DISK **:** 800 GB

**PRE-REQUISITIES:**

Python, Mysql

* 1. **SOFTWARE SPECIFICATIONS :**

PROGRAMMING LANGUAGES : Python

OPERATING SYSTEM : MICROSOFT WINDOWS 11

SOFTWARES :

* Python
* MySql

## CHAPTER 3

INVENTORY MANAGEMENT SYSTEM

**SOURCE CODE**

import sys

from PyQt5.QtWidgets import (QApplication, QWidget, QLabel, QLineEdit, QPushButton, QVBoxLayout, QHBoxLayout, QTextEdit, QMessageBox, QTableWidget, QTableWidgetItem)

from PyQt5.QtGui import QPalette, QBrush, QPixmap, QFont, QColor, QIcon from PyQt5.QtCore import Qt

from PyQt5.QtWidgets import QCheckBox import mysql.connector

class Database:

def init (self):

self.db = mysql.connector.connect( host="localhost",

user="root", password="123Admin", database="inventory"

)

def get\_connection(self): return self.db

class LoginWindow(QWidget): def init (self, db):

super(). init () self.setWindowTitle('Login') self.setGeometry(500, 300, 800, 500) self.set\_background\_image('bg.png')

self.setWindowIcon(QIcon('login\_icon.jpg')) # Set window icon self.db = db

self.label\_username = QLabel('Username:') self.label\_username.setStyleSheet("color: white; font-family: Arial; font-size: 14pt;") self.input\_username = QLineEdit()

self.input\_username.setPlaceholderText('Enter your username.') # Placeholder text

self.label\_password = QLabel('Password:')

self.label\_password.setStyleSheet("color: white; font-family: Arial; font-size: 14pt;") self.input\_password = QLineEdit() self.input\_password.setEchoMode(QLineEdit.Password) self.input\_password.setPlaceholderText('Enter your password.') # Placeholder text

# Checkbox to toggle password visibility self.show\_password\_checkbox = QCheckBox('Show Password')

self.show\_password\_checkbox.setStyleSheet("color: white; font-family: Arial; font-size: 14pt;")

self.show\_password\_checkbox.stateChanged.connect(self.toggle\_password\_visibility)

self.submit\_button = QPushButton('Login') self.submit\_button.setFixedSize(150, 40) self.submit\_button.clicked.connect(self.login)

self.new\_user\_button = QPushButton('New User') self.new\_user\_button.setFixedSize(150, 40) self.new\_user\_button.clicked.connect(self.register\_new\_user)

layout = QVBoxLayout() layout.addWidget(self.label\_username) layout.addWidget(self.input\_username) layout.addWidget(self.label\_password) layout.addWidget(self.input\_password)

layout.addWidget(self.show\_password\_checkbox) # Add checkbox to the layout

button\_layout1 = QHBoxLayout() button\_layout1.addStretch() button\_layout1.addWidget(self.submit\_button) button\_layout1.addStretch()

button\_layout2 = QHBoxLayout() button\_layout2.addStretch() button\_layout2.addWidget(self.new\_user\_button) button\_layout2.addStretch()

layout.addLayout(button\_layout1) layout.addLayout(button\_layout2)

self.setLayout(layout)

def toggle\_password\_visibility(self, state): if state == Qt.Checked:

self.input\_password.setEchoMode(QLineEdit.Normal) else:

self.input\_password.setEchoMode(QLineEdit.Password)

def set\_background\_image(self, image\_path): oImage = QPixmap(image\_path)

sImage = oImage.scaled(self.size()) palette = QPalette()

palette.setBrush(QPalette.Window, QBrush(sImage)) self.setPalette(palette)

def login(self):

username = self.input\_username.text() password = self.input\_password.text()

cursor = self.db.get\_connection().cursor() # Check if username and password match

sql = "SELECT \* FROM authentication WHERE username = %s AND password = %s"

val = (username, password) cursor.execute(sql, val)

user = cursor.fetchone() cursor.close()

if user:

self.dashboard\_window = DashboardWindow(self.db, username) self.dashboard\_window.show()

self.close() else:

QMessageBox.warning(self, 'Warning', 'Invalid username or password.')

def register\_new\_user(self):

self.registration\_window = RegistrationWindow(self.db, self) self.registration\_window.show()

class DashboardWindow(QWidget): def init (self, db, username):

super(). init () self.setWindowTitle('Dashboard') self.setGeometry(500, 300, 900, 500) self.set\_background\_image('bg.png')

self.setWindowIcon(QIcon('dashboard\_icon.jpg')) # Set window icon

self.db = db self.username = username

self.add\_item\_button = QPushButton('Add Items') self.add\_item\_button.setFixedSize(200, 100) self.add\_item\_button.clicked.connect(self.add\_items)

self.view\_items\_button = QPushButton('View Items') self.view\_items\_button.setFixedSize(200, 100) self.view\_items\_button.clicked.connect(self.view\_items)

self.update\_items\_button = QPushButton('Update Items') self.update\_items\_button.setFixedSize(200, 100) self.update\_items\_button.clicked.connect(self.update\_items)

self.info\_text = QTextEdit() self.info\_text.setReadOnly(True)

self.info\_text.setText("Welcome to the Inventory Management System. Use the buttons above to add items and view items.")

# Set font style for QTextEdit

font = QFont("Brush Script MT", 16) # Change the font family and size as desired self.info\_text.setFont(font)

self.info\_text.setStyleSheet("background-color: rgba(255, 255, 255, 0); color: white; color: white;") # 0 alpha for transparency

self.info\_text.setAlignment self.info\_text.setAlignment(Qt.AlignBottom | Qt.AlignHCenter)

self.logout\_button = QPushButton('Logout') self.logout\_button.setFixedSize(150, 40) self.logout\_button.clicked.connect(self.logout)

# Horizontal layout for the Add Items, View Items, and Update Items buttons button\_layout = QHBoxLayout()

button\_layout.addStretch() button\_layout.addWidget(self.add\_item\_button) button\_layout.addWidget(self.view\_items\_button) button\_layout.addWidget(self.update\_items\_button) button\_layout.addStretch()

# Horizontal layout for centering the logout button logout\_button\_layout = QHBoxLayout() logout\_button\_layout.addStretch() logout\_button\_layout.addWidget(self.logout\_button) logout\_button\_layout.addStretch()

# Main layout to center the buttons vertically layout = QVBoxLayout() layout.addStretch() layout.addLayout(button\_layout) layout.addWidget(self.info\_text) layout.addStretch() layout.addLayout(logout\_button\_layout) layout.addStretch()

self.setLayout(layout)

def set\_background\_image(self, image\_path): oImage = QPixmap(image\_path)

sImage = oImage.scaled(self.size()) palette = QPalette()

palette.setBrush(QPalette.Window, QBrush(sImage)) self.setPalette(palette)

def add\_items(self):

self.add\_item\_window = AddItemWindow(self.db, self.username) self.add\_item\_window.show()

self.close()

def view\_items(self):

self.view\_items\_window = ViewItemsWindow(self.db, self.username) self.view\_items\_window.show()

self.close()

def update\_items(self):

self.update\_items\_window = UpdateItemsWindow(self.db, self.username) self.update\_items\_window.show()

self.close()

def logout(self):

self.login\_window = LoginWindow(self.db) self.login\_window.show()

self.close()

class AddItemWindow(QWidget): def init (self, db, username):

super(). init () self.setWindowTitle('Add Items')

self.setGeometry(500, 300, 800, 500) self.set\_background\_image('bg.png') self.setWindowIcon(QIcon('add\_item\_icon.jpg')) # Set window icon

self.db = db self.username = username

self.label\_name = QLabel('Product Name:') self.label\_name.setStyleSheet("color: white; font-family: Arial; font-size: 14pt;") self.input\_name = QLineEdit()

self.input\_name.setPlaceholderText('Enter Product name.') # Placeholder text

self.label\_quantity = QLabel('Quantity:')

self.label\_quantity.setStyleSheet("color: white; font-family: Arial; font-size: 14pt;") self.input\_quantity = QLineEdit()

self.input\_quantity.setPlaceholderText('Enter the Quantity.') # Placeholder text

self.label\_cost = QLabel('Cost:')

self.label\_cost.setStyleSheet("color: white; font-family: Arial; font-size: 14pt;") self.input\_cost = QLineEdit()

self.input\_cost.setPlaceholderText('Enter product cost.') # Placeholder text

self.submit\_button = QPushButton('Add Item') self.submit\_button.setFixedSize(150, 40) self.submit\_button.clicked.connect(self.add\_to\_inventory)

self.back\_button = QPushButton('Back') self.back\_button.setFixedSize(150, 40) self.back\_button.clicked.connect(self.back\_to\_dashboard)

layout = QVBoxLayout() layout.addWidget(self.label\_name) layout.addWidget(self.input\_name) layout.addWidget(self.label\_quantity) layout.addWidget(self.input\_quantity) layout.addWidget(self.label\_cost) layout.addWidget(self.input\_cost)

button\_layout1 = QHBoxLayout() button\_layout1.addStretch() button\_layout1.addWidget(self.submit\_button) button\_layout1.addStretch()

button\_layout2 = QHBoxLayout() button\_layout2.addStretch() button\_layout2.addWidget(self.back\_button) button\_layout2.addStretch()

layout.addLayout(button\_layout1) layout.addLayout(button\_layout2)

self.setLayout(layout)

def set\_background\_image(self, image\_path): oImage = QPixmap(image\_path)

sImage = oImage.scaled(self.size()) palette = QPalette()

palette.setBrush(QPalette.Window, QBrush(sImage)) self.setPalette(palette)

def add\_to\_inventory(self): item\_name = self.input\_name.text()

quantity = int(self.input\_quantity.text()) cost = float(self.input\_cost.text())

cursor = self.db.get\_connection().cursor()

sql = "INSERT INTO inventory (username, item\_name, quantity, cost) VALUES (%s, %s, %s, %s)" val = (self.username, item\_name, quantity, cost)

cursor.execute(sql, val) self.db.get\_connection().commit() cursor.close()

QMessageBox.information(self, 'Success', 'Item added successfully.') self.input\_name.clear()

self.input\_quantity.clear() self.input\_cost.clear()

def back\_to\_dashboard(self):

self.dashboard\_window = DashboardWindow(self.db, self.username) self.dashboard\_window.show()

self.close()

class ViewItemsWindow(QWidget): def init (self, db, username):

super(). init () self.setWindowTitle('View Items') self.setGeometry(500, 300, 800, 500) self.set\_background\_image('bg.png')

self.setWindowIcon(QIcon('view\_item\_icon.jpg')) # Set window icon

self.db = db self.username = username

self.table = QTableWidget()

self.table.setColumnCount(4) # Add one more column for delete buttons self.table.setHorizontalHeaderLabels(['Item Name', 'Quantity', 'Cost', 'Action'])

# Customize table appearance

self.table.setStyleSheet("background-color: transparent; gridline-color: white;") header = self.table.horizontalHeader()

header.setStyleSheet("::section{background-color: gray; color: white; font-weight: bold;}") self.load\_data()

self.back\_button = QPushButton('Back') self.back\_button.setFixedSize(150, 40) self.back\_button.clicked.connect(self.back\_to\_dashboard)

# Create a horizontal layout for the back button and add it back\_button\_layout = QHBoxLayout()

back\_button\_layout.addStretch() back\_button\_layout.addWidget(self.back\_button) back\_button\_layout.addStretch()

# Main layout

layout = QVBoxLayout() layout.addWidget(self.table) layout.addLayout(back\_button\_layout)

self.setLayout(layout)

def set\_background\_image(self, image\_path): oImage = QPixmap(image\_path)

sImage = oImage.scaled(self.size()) palette = QPalette()

palette.setBrush(QPalette.Window, QBrush(sImage)) self.setPalette(palette)

def load\_data(self):

cursor = self.db.get\_connection().cursor()

sql = "SELECT item\_name, quantity, cost FROM inventory WHERE username = %s" val = (self.username,)

cursor.execute(sql, val) items = cursor.fetchall()

self.table.setRowCount(len(items))

for row\_index, item in enumerate(items): item\_name = QTableWidgetItem(item[0])

item\_name.setForeground(QBrush(QColor(255, 255, 255))) # Set text color to white self.table.setItem(row\_index, 0, item\_name)

item\_quantity = QTableWidgetItem(str(item[1])) item\_quantity.setForeground(QBrush(QColor(255, 255, 255))) # Set text color to white self.table.setItem(row\_index, 1, item\_quantity)

item\_cost = QTableWidgetItem(str(item[2])) item\_cost.setForeground(QBrush(QColor(255, 255, 255))) # Set text color to white self.table.setItem(row\_index, 2, item\_cost)

# Create delete button

delete\_button = QPushButton('Delete') delete\_button.setStyleSheet("color: white;") # Set text color to white

delete\_button.clicked.connect(lambda state, row=row\_index: self.delete\_item(row)) # Pass row index to delete\_item function

# Create layout for the delete button layout = QHBoxLayout() layout.addWidget(delete\_button)

layout.setAlignment(Qt.AlignCenter) # Align button to center horizontally layout.setContentsMargins(0, 0, 0, 0) # Set margins to zero

# Set layout as cell widget in the last column cell\_widget = QWidget() cell\_widget.setLayout(layout) self.table.setCellWidget(row\_index, 3, cell\_widget)

cursor.close()

def delete\_item(self, row):

item\_name = self.table.item(row, 0).text() # Get item name from the first column of the selected row

cursor = self.db.get\_connection().cursor()

sql = "DELETE FROM inventory WHERE item\_name = %s AND username = %s" val = (item\_name, self.username)

cursor.execute(sql, val) self.db.get\_connection().commit() cursor.close()

self.load\_data() # Reload data after deletion def back\_to\_dashboard(self):

self.dashboard\_window = DashboardWindow(self.db, self.username) self.dashboard\_window.show()

self.close()

class UpdateItemsWindow(QWidget): def init (self, db, username):

super(). init () self.setWindowTitle('Update Items') self.setGeometry(500, 300, 800, 500) self.set\_background\_image('bg.png')

self.setWindowIcon(QIcon('update\_icon.PNG')) # Set window icon

self.db = db self.username = username

self.table = QTableWidget()

self.table.setColumnCount(4) # Add columns for item name, quantity, cost, and action self.table.setHorizontalHeaderLabels(['Item Name', 'Quantity', 'Cost', 'Action'])

# Customize table appearance

self.table.setStyleSheet("background-color: transparent; gridline-color: white;") header = self.table.horizontalHeader()

header.setStyleSheet("::section{background-color: gray; color: white; font-weight: bold;}") self.load\_data()

self.back\_button = QPushButton('Back') self.back\_button.setFixedSize(150, 40) self.back\_button.clicked.connect(self.back\_to\_dashboard)

# Create a horizontal layout for the back button and add it back\_button\_layout = QHBoxLayout() back\_button\_layout.addStretch() back\_button\_layout.addWidget(self.back\_button) back\_button\_layout.addStretch()

# Main layout

layout = QVBoxLayout()

layout.addWidget(self.table) layout.addLayout(back\_button\_layout)

self.setLayout(layout)

def set\_background\_image(self, image\_path): oImage = QPixmap(image\_path)

sImage = oImage.scaled(self.size()) palette = QPalette()

palette.setBrush(QPalette.Window, QBrush(sImage)) self.setPalette(palette)

def load\_data(self):

cursor = self.db.get\_connection().cursor()

sql = "SELECT item\_name, quantity, cost FROM inventory WHERE username = %s" val = (self.username,)

cursor.execute(sql, val) items = cursor.fetchall()

self.table.setRowCount(len(items))

for row\_index, item in enumerate(items): item\_name = QTableWidgetItem(item[0])

item\_name.setForeground(QBrush(QColor(255, 255, 255))) # Set text color to white self.table.setItem(row\_index, 0, item\_name)

item\_quantity = QTableWidgetItem(str(item[1])) item\_quantity.setForeground(QBrush(QColor(255, 255, 255))) # Set text color to white self.table.setItem(row\_index, 1, item\_quantity)

item\_cost = QTableWidgetItem(str(item[2])) item\_cost.setForeground(QBrush(QColor(255, 255, 255))) # Set text color to white self.table.setItem(row\_index, 2, item\_cost)

# Create update button

update\_button = QPushButton('Update') update\_button.setStyleSheet("color: white;") # Set text color to white

update\_button.clicked.connect(lambda state, row=row\_index: self.update\_item(row)) # Pass row index to update\_item function

# Create layout for the update button layout = QHBoxLayout() layout.addWidget(update\_button)

layout.setAlignment(Qt.AlignCenter) # Align button to center horizontally layout.setContentsMargins(0, 0, 0, 0) # Set margins to zero

# Set layout as cell widget in the last column cell\_widget = QWidget() cell\_widget.setLayout(layout) self.table.setCellWidget(row\_index, 3, cell\_widget)

cursor.close()

def update\_item(self, row):

item\_name = self.table.item(row, 0).text() new\_quantity = int(self.table.item(row, 1).text())

new\_cost = float(self.table.item(row, 2).text())

cursor = self.db.get\_connection().cursor()

sql = "UPDATE inventory SET quantity = %s, cost = %s WHERE item\_name = %s AND username = %s" val = (new\_quantity, new\_cost, item\_name, self.username)

cursor.execute(sql, val) self.db.get\_connection().commit() cursor.close()

QMessageBox.information(self, 'Success', 'Item updated successfully.') self.load\_data() # Reload data after updating

def back\_to\_dashboard(self):

self.dashboard\_window = DashboardWindow(self.db, self.username) self.dashboard\_window.show()

self.close()

class RegistrationWindow(QWidget): def init (self, db, login\_window):

super(). init () self.setWindowTitle('Registration') self.setGeometry(500, 300, 800, 500) self.set\_background\_image('bg.png')

self.setWindowIcon(QIcon('registration\_icon.jpg')) # Set window icon

self.db = db

self.login\_window = login\_window

self.label\_username = QLabel('Username:') self.label\_username.setStyleSheet("color: white; font-family: Arial; font-size: 14pt;") self.input\_username = QLineEdit()

self.input\_username.setPlaceholderText('Enter your username.') # Placeholder text

self.label\_password = QLabel('Password:')

self.label\_password.setStyleSheet("color: white; font-family: Arial; font-size: 14pt;") self.input\_password = QLineEdit() self.input\_password.setEchoMode(QLineEdit.Password) self.input\_password.setPlaceholderText('Enter your password.') # Placeholder text

self.submit\_button = QPushButton('Register') self.submit\_button.setFixedSize(150, 40) self.submit\_button.clicked.connect(self.register)

self.back\_button = QPushButton('Back') self.back\_button.setFixedSize(150, 40) self.back\_button.clicked.connect(self.back\_to\_login)

layout = QVBoxLayout() layout.addWidget(self.label\_username) layout.addWidget(self.input\_username) layout.addWidget(self.label\_password) layout.addWidget(self.input\_password)

button\_layout1 = QHBoxLayout() button\_layout1.addStretch()

button\_layout1.addWidget(self.submit\_button) button\_layout1.addStretch()

button\_layout2 = QHBoxLayout() button\_layout2.addStretch() button\_layout2.addWidget(self.back\_button) button\_layout2.addStretch()

layout.addLayout(button\_layout1) layout.addLayout(button\_layout2)

self.setLayout(layout)

def set\_background\_image(self, image\_path): oImage = QPixmap(image\_path)

sImage = oImage.scaled(self.size()) palette = QPalette()

palette.setBrush(QPalette.Window, QBrush(sImage)) self.setPalette(palette)

def register(self):

username = self.input\_username.text() password = self.input\_password.text()

cursor = self.db.get\_connection().cursor()

sql = "INSERT INTO authentication (username, password) VALUES (%s, %s)" val = (username, password)

cursor.execute(sql, val) self.db.get\_connection().commit() cursor.close()

QMessageBox.information(self, 'Success', 'User registered successfully.') self.login\_window.show()

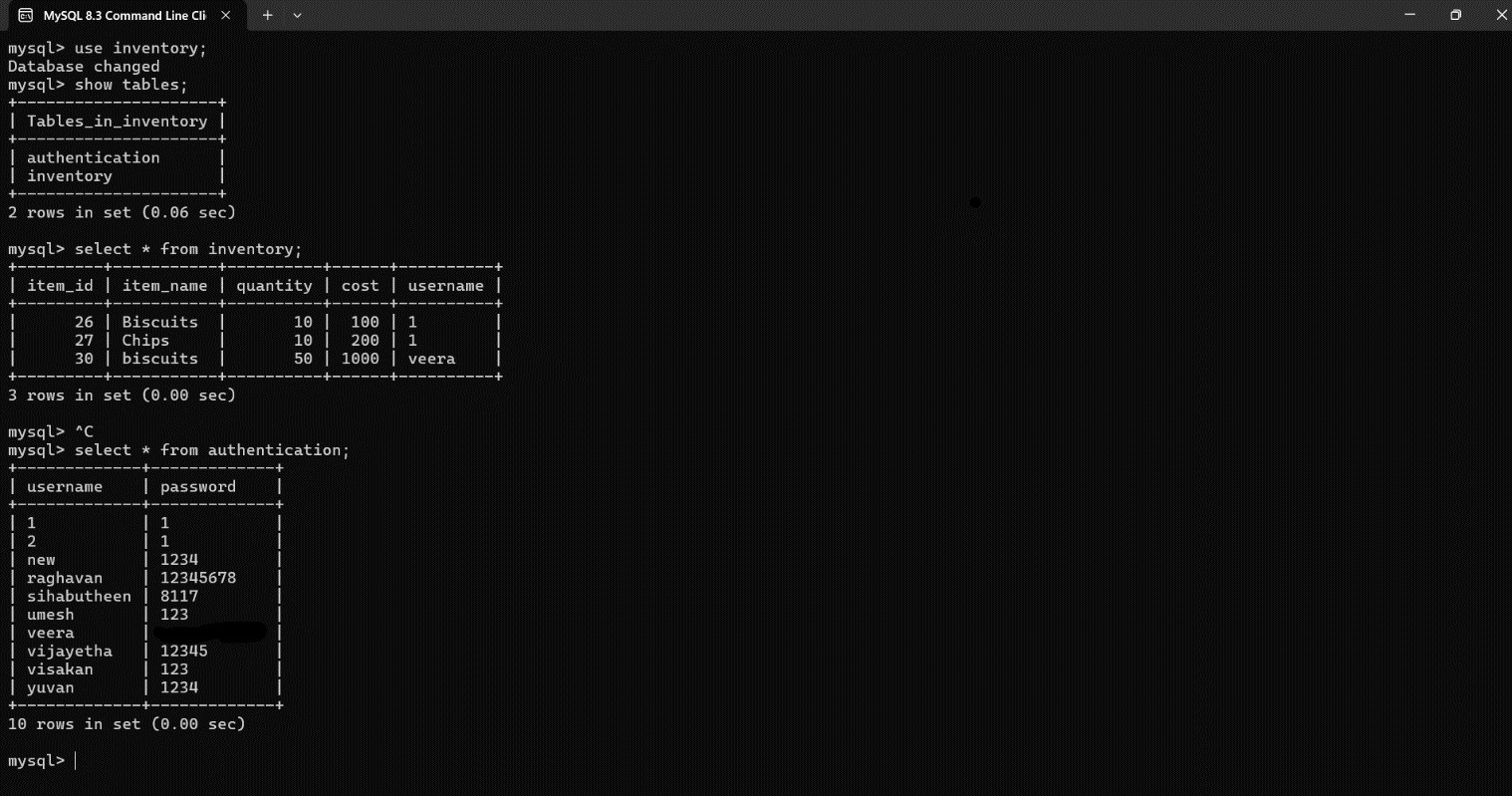
self.close()

def back\_to\_login(self): self.login\_window.show() self.close()

if name == ' main ': app = QApplication(sys.argv) db = Database()

login\_window = LoginWindow(db) login\_window.show() sys.exit(app.exec\_())

## Database-Mysql

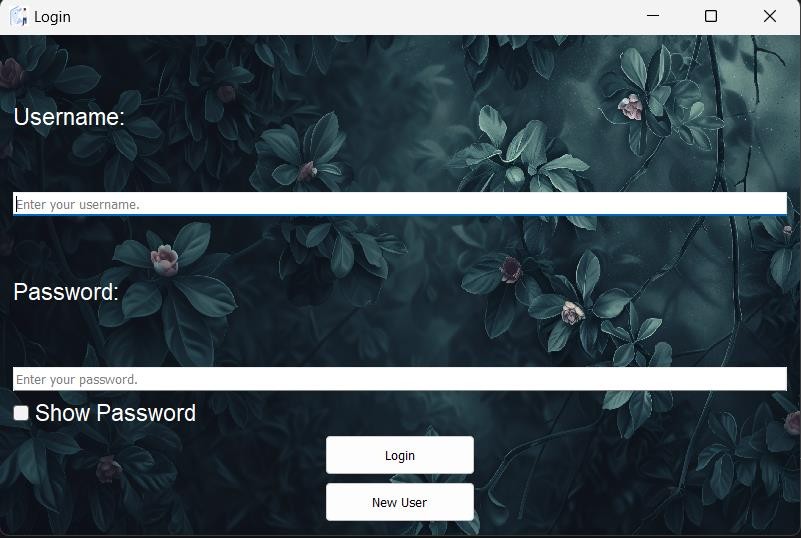


Similarly, the database was created for the items.

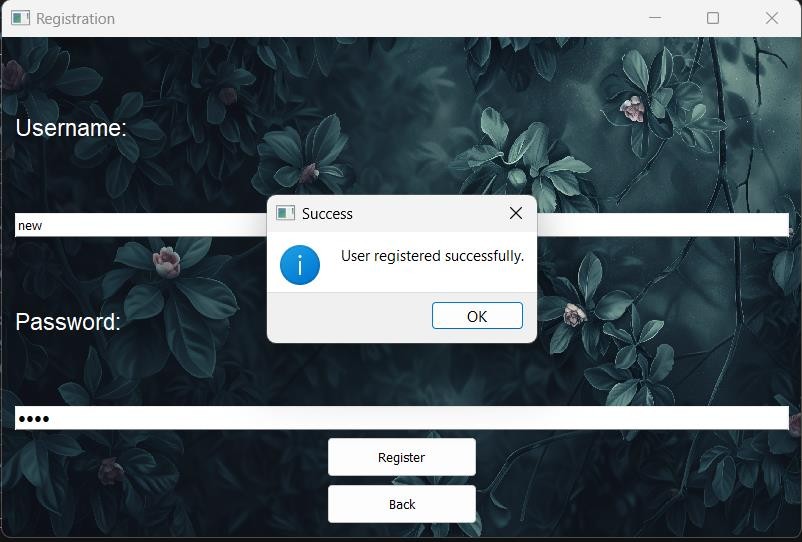
**CHAPTER 4 SNAP SHOTS**

* 1. **OUTPUT:**

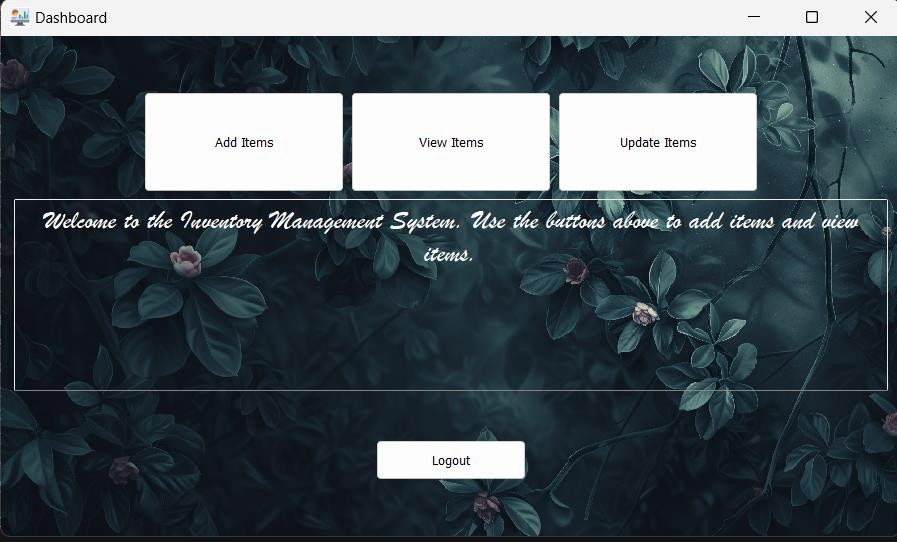
**Login page**



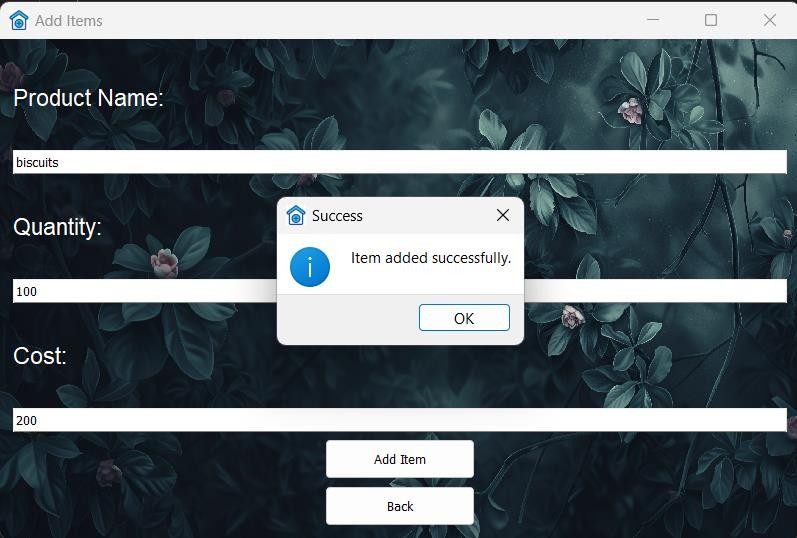
# Registration:



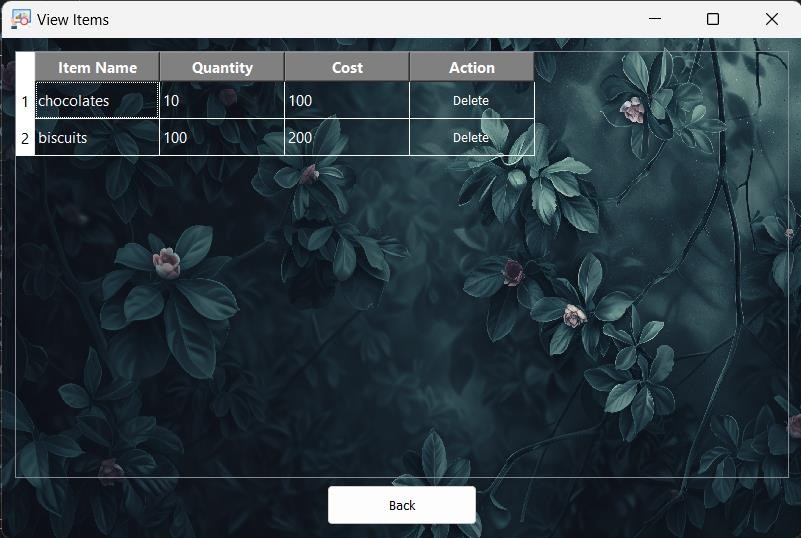
**Dashboard:**



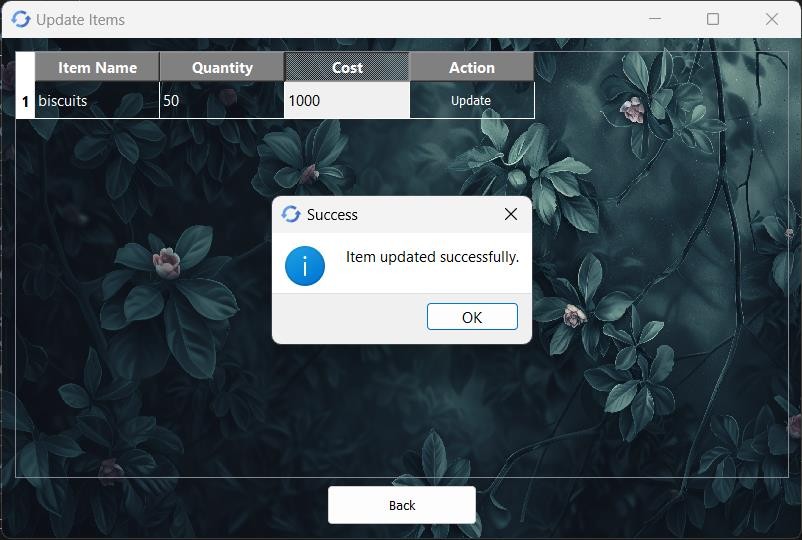
# Add items:



**View items:**



# Update items:



## CHAPTER 5

**CONCLUSION AND FUTURE ENHANCEMENT**

The Inventory Management System developed in this project addresses the critical need for efficient and reliable inventory control within organizations. The system provides a comprehensive solution for managing inventory, allowing users to add, view, and update items seamlessly. The user-friendly interface ensures that even those with minimal technical expertise can navigate and utilize the system effectively.

The implementation leverages PyQt5 for the graphical user interface, ensuring a visually appealing and responsive design. The integration with a MySQL database guarantees secure and reliable data storage, enabling users to manage their inventory with confidence. The system's features, including user authentication, password visibility toggle, and dynamic item management, highlight the attention to detail and user-centric approach adopted in this project.

Overall, this Inventory Management System offers a robust platform for organizations to maintain their inventory accurately, reduce operational inefficiencies, and improve overall productivity. Future enhancements could include adding more advanced features such as inventory forecasting, generating detailed reports, and incorporating barcode scanning to further streamline inventory management processes.

**BIBILOGROPHY**

The following websites helped us in gaining more knowledge on the subject and in completing the project.

1. https://[www.w3schools.com/sql/](http://www.w3schools.com/sql/)
2. [Python | Introduction to PyQt5 - GeeksforGeeks](https://www.geeksforgeeks.org/python-introduction-to-pyqt5/)