BLUESHEET

POS SYSTEM

Presented By: CHEW YONG LE
JERICK ANG
LIANG QI XUN

01

Studio Shodwe Project. 2024

INTRODUCTION

01

Project. 2024

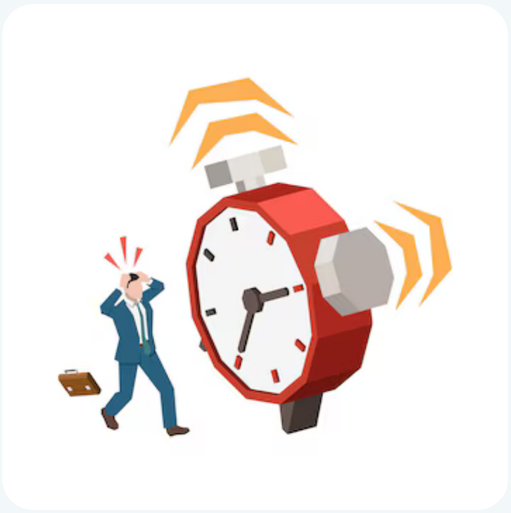
Studio Shodwe

BLUESHEET

A SMART POS SYSTEM THAT SIMPLIFIES SALES AND INVENTORY TASKS WITH **REAL-TIME TRACKING** AND **ROLE-BASED ACCESS** TO BOOST RETAIL EFFICIENCY

PROBLEM STATEMENT

HUMAN ERROR



TIME-CONSUMING

LACK REAL-TIME DATA

OBJECTIVE

01 TO PROVIDE A SECURE AND USER-FRIENDLY INTERFACE FOR DIFFERENT USER ROLES

02 TO AUTOMATE SALES AND INVENTORY TASKS & REDUCE MANUAL EFFORT AND ERRORS

03 TO SUPPORT BUSINESS DECISIONS THROUGH CLEAR AND CENTRALIZED SALES DATA

RESEARCH QUESTIONS

01 HOW CAN ROLE-BASED ACCESS IMPROVE SYSTEM SECURITY AND TASK EFFICIENCY?

02 HOW DOES A SIMPLE AND CLEAR INTERFACE HELP STAFF IN DAILY STORE TASKS?

03 HOW CAN SALES DATA HELP MANAGERS MAKE BETTER BUSINESS DECISIONS?

02

Studio Shodwe Project. 2024

LITERATURE REVIEW

EXISTING SYSTEMS ANALYSIS

EXISTING SYSTEMS

Zoho Inventory

Square POS

	Zoho Inventory	Square POS	Bluesheet
System Architecture	Cloud-based	Cloud-based	Web-based
UI/UX Design	RIch-complex	Clean, intuitive	Simple and beginner-friendly
Data Handling	Cloud storage	Cloud storage	Local database with web access
Limitations	Costly, steep learning curve	Requires stable internet	No mobile app
Relevance	Inspires feature depth	UI and flow inspiration	Tailored for small retail efficiency

RESEARCH GAP

01 LACK OF AFFORDABLE, ROLE-BASED POS SYSTEMS FOR SMALL RETAILERS

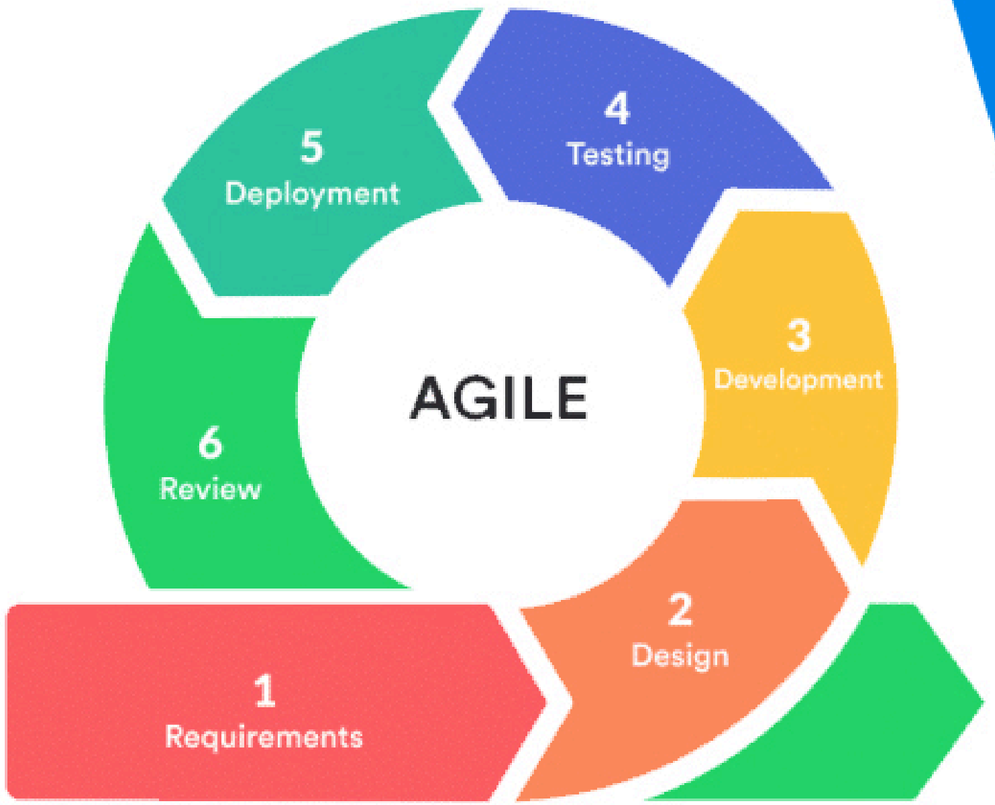
02 LIMITED INTEGRATION OF REAL-TIME INVENTORY ALERTS WITHOUT CLOUD DEPENDENCY

03 INSUFFICIENT FOCUS ON BEGINNER-FRIENDLY UI FOR NON-TECH-SAVVY STAFF

03

Studio Shodwe Project. 2024

METHODOLOGY



Agile Project Management

AGILE

BLUESHEET

1 ITERATION

Split system into 3 modules for different roles:
Admin, Cashier, Manager

2 DESIGN

ERD Diagram & Use Case Diagram

3 DEVELOPMENT

Coding- Python
Framework- Flask
IDE- VS Code
Database- SQLite
Frontend- HTML, Javascript, CSS

4 TESTING

Unit testing
UAT testing
System testing

5 RELEASE

deploy system and receive continous constructive feedback

6 REVIEW

Feedback are used for improvements. Final version is used for submission

04

Studio Shodwe Project. 2024

SYSTEM DEMO

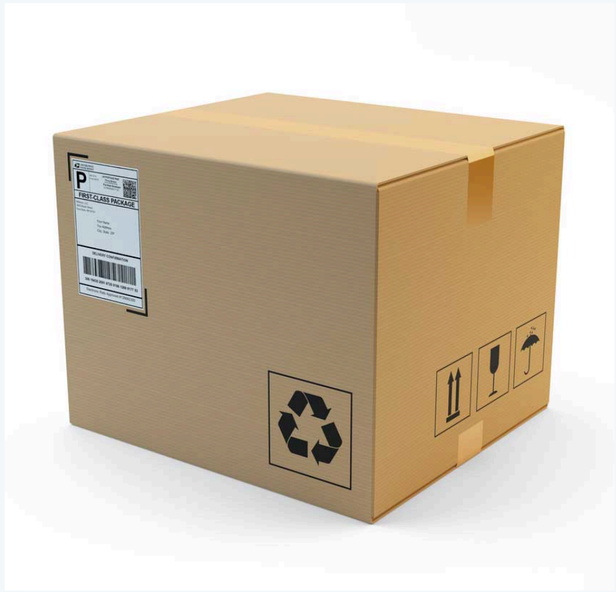
05

Studio Shodwe Project. 2024

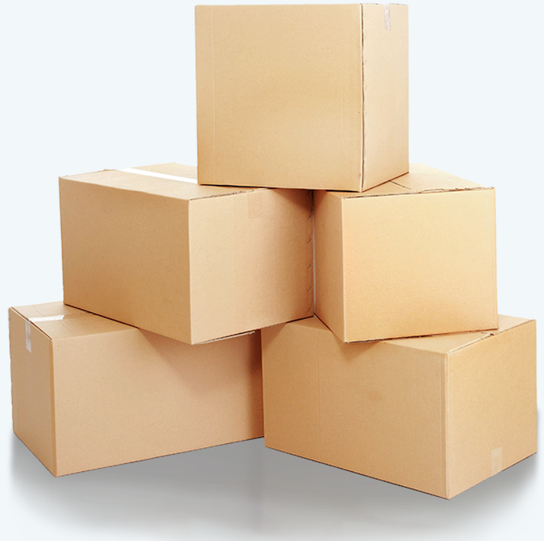
TESTING

TESTING STRATEGIES

Unit Testing



System Testing



User Acceptance Testing

TEST LOGIN

testlogin.py

```python
import unittest
from unittest.mock import patch, MagicMock
from werkzeug.security import generate_password_hash
from app import app


class TestLogin(unittest.TestCase):
    def setUp(self):
        app.config['TESTING'] = True
        app.config['WTF_CSRF_ENABLED'] = False
        self.client = app.test_client()

        self.test_user = MagicMock()
        self.test_user.Email = "test@example.com"
        self.test_user.Password = generate_password_hash("correctpass")
        self.test_user.IsActive = True
        self.test_user.Role = "cashier"
        self.test_user.UserID = 1

    @patch('routes.login.User')
    def test_successful_login(self, mock_user):
        mock_user.query.filter_by.return_value.first.return_value = self.test_user

        response = self.client.post('/login', data={
            'email': 'test@example.com',
            'password': 'correctpass'
        }, follow_redirects=True)

        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Cashier Dashboard', response.data)

    @patch('routes.login.User')
    def test_invalid_password(self, mock_user):
        mock_user.query.filter_by.return_value.first.return_value = self.test_user

        response = self.client.post('/login', data={
            'email': 'test@example.com',
            'password': 'wrongpass'
        }, follow_redirects=True)

        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Invalid email or password', response.data)
```

TEST REGISTERING PRODUCT

testregproduct.py

```python
import unittest
from unittest.mock import patch
from io import BytesIO
from app import app


class TestProductRegistration(unittest.TestCase):
    def setUp(self):
        app.config['TESTING'] = True
        app.config['WTF_CSRF_ENABLED'] = False
        self.client = app.test_client()

        # Simulate admin login session
        with self.client.session_transaction() as session:
            session['user_id'] = 1
            session['role'] = 'admin'
            session['_fresh'] = True

    @patch('sqlite3.connect')
    def test_product_registration(self, mock_sqlite):
        mock_conn = mock_sqlite.return_value
        mock_cursor = mock_conn.cursor.return_value

        # simulate insert returning last row id
        mock_cursor.lastrowid = 123

        test_data = {
            'category': 'Food',
            'brand': 'Test Brand',
            'product': 'Test Product',
            'price': '10.99',
            'quantity': '100',
            'image': (BytesIO(b"dummy image data"), 'test.jpg')
        }

        response = self.client.post('/register-container',
                                    data=test_data,
                                    content_type='multipart/form-data',
                                    follow_redirects=True)

        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Product registered successfully', response.data)
        mock_conn.commit.assert_called_once()
```

TEST TRANSACTION

testtransaction.py

```python
import unittest
from unittest.mock import patch, MagicMock
from app import app


class TestTransactions(unittest.TestCase):
    def setUp(self):
        app.config['TESTING'] = True
        app.config['WTF_CSRF_ENABLED'] = False
        self.client = app.test_client()

        with self.client.session_transaction() as session:
            session['user_id'] = 1
            session['role'] = 'cashier'
            session['_fresh'] = True

    @patch('db.models.Product')
    def test_get_product(self, mock_product):
        mock_product_obj = MagicMock()
        mock_product_obj.ProductID = 1
        mock_product_obj.ProductName = "Chocolate Milk 1L"
        mock_product_obj.Price = 9.99
        mock_product_obj.StockQuantity = 10
        mock_product_obj.serialize.return_value = {
            'id': 1,
            'name': "Chocolate Milk 1L",
            'price': 9.99,
            'stock': 10
        }

        mock_product.query.get.return_value = mock_product_obj

        response = self.client.get('/cashier/get-product/1')
        self.assertEqual(response.status_code, 200)
        data = response.get_json()
        self.assertEqual(data['id'], 1)
        self.assertEqual(data['name'], "Chocolate Milk 1L")

    @patch('db.models.Product')
    def test_get_nonexistent_product(self, mock_product):
        mock_product.query.get.return_value = None

        response = self.client.get('/cashier/get-product/999')
        self.assertEqual(response.status_code, 404)
        self.assertIn('error', response.get_json())
```

05

Studio Shodwe Project. 2024

ACHIEMENTS

ACHIEVEMENTS

01 DEMAND FORECASTING FOR ENHANCED BUSINESS DECISIONS

02 PROEFFICIENT & STREAMLINED, EASY TO USE, INVENTORY MANAGEMENT SYSTEM

03 QR CODE GENERATION FOR PRODUCTS

05

Studio Shodwe Project. 2024

FUTURE IMPROVEMENTS

FUTURE IMPROVEMENTS

01 QR CODE SCANNING FEATURE WHEN MAKING TRANSACTIONS

02 REAL-TIME AUTOMATED ALERTS FOR UNUSUAL AND HARMFUL ACTIVITIES FOR SYSTEM SECURITY

03 INTEGRATION OF AI ANALYTICS TO PREDICT BUSINESS TRENDS

05

Studio Shodwe Project. 2024

CONCLUSION

THANK YOU