**Final Year Project Report**

**QuickStock**

****

**Project Advisor:**

**Sir Noaman Saleem**

**Submitted By:**

**Amaan Majeed (F2020266286)**

**Faseeh-ud-din (F2020266256)**

**Muhammad Hashim Khurshid (F2020266272)**

**Muhammad Usaid Afzal (F2020266271)**

**Munib Ahsan Khan (F2020266270)**

**Session**

**University of Management and Technology**

**C-II Johar Town Lahore Pakistan**

**Dedication**

I would like to dedicate this project to my teacher, Sir Noaman Saleem, who has been my constant source of support and guidance throughout this journey. His encouragement and valuable feedback helped us stay motivated during challenging times. His insight and expertise were instrumental in shaping this project.

I also dedicate this work to my parents, for their endless love, sacrifices and belief in me. They have supported me unconditionally in all my pursuits. This achievement would not have been possible without their encouragement.

Finally, I dedicate this project to my teammates - Faseeh, Usaid, Hashim, and Munib. It was a pleasure working together and learning from each other over the past few months. Their dedication and problem-solving skills contributed greatly towards completing this project.

## 

**Final Approval**

* **Head of Department**  Dr. Atif Alvi

Department of Informatics & Systems

School of Systems & Technology

UMT Lahore

* **Director (Final Year Projects-CS)** Dr. Jameel Ahmed

Department of Computer Science.

School of Systems & Technology

UMT Lahore

* **Supervisor**  Sir Noaman Saleem
* Department of Informatics & Systems.
* School of Systems & Technology

UMT Lahore

**Acknowledgment**

First, I want to say thanks to my teacher Sir Noaman Saleem. He helped me a lot with this project. Whenever I had questions, he took time to explain things to me clearly until I understood. His knowledge and patience was very important for finishing this.

I also want to thank Dr. Jameel Ahmed for helping out too.

I want to thank my family and friends for supporting me during this project. Doing this project took a lot of my time, but they encouraged me to keep working hard. My friends checked on me and reminded me that I could do it when I thought it was impossible on our own.

Finally, I want to thank everyone who helped me. I learned a lot from doing this project. Sir Noaman and Dr. Jameel taught me many new skills as a computer science student. And my family and friends gave me strength and happiness during difficult times. Thank you all very much for your support and help.

**Project Title:** QuickStock

**Objective** To Simplify POS systems

**Undertaken by** Amaan Majeed

Faseeh Ud Din

Muhammad Usaid Afzal

Muhammad Hashim Khurshid

Munib Ahsan Khan

**Supervised by** Sir Noaman Saleem

**Starting Date** 03-10-23

**Completion Date** On Going

**Tools Used** Html, CSS, JavaScript, Python

**Operating System** Windows

**Plagiarism ReportAbstract**

Cloud POS and Inventory System with Smart Stock Predictions

It is difficult for small shops to keep the right number of products because sales change often and they don't have many computers. This project will create a system to help with that using the cloud. The system will learn from past sales records and predict how much of each item the shop needs next. Then it can automatically order the right amounts.

With reports and charts online, shop owners can manage inventory and sales from any device like phone or tablet. They don't need special computers. This makes work easier for small shops. By using previous sales data and modern cloud technology, the system aims to help small stores pick the correct inventory levels to make more profit. Everything is accessible anytime from anywhere on the cloud.

Revision Chart

| **Version** | **Primary Author(s)** | **Description of Version** | **Date Completed** |
| --- | --- | --- | --- |
| Draft | Amaan, Munib, Usaid | Initial draft created | 20-12-23 |
| Initial Version | Amaan | Initial version created with the initial details | 23-12-23 |
| Mid-semester Report v1 | Amaan, Munib | Second draft incorporating initial review comments, distributed for final review | 25-12-23 |
| Final | TBD | First complete draft, which is placed under change control | TBD |
| Revision 1 | TBD | Revised draft, revised according to the change control process and maintained under change control | TBD |
| Revision 2 | TBD | Revised draft, revised according to the change control process and maintained under change control | TBD |
| Etc. | TBD | TBD | TBD |

Table 1: Revision Chart

# Contents

[**Contents 1**](#_heading=h.17dp8vu)

[Definitions and Acronyms 2](#_heading=h.3rdcrjn)

[List of Figures 3](#_heading=h.2nusc19)

[List of Tables 4](#_heading=h.35nkun2)

[**1. Introduction 6**](#_heading=h.1ksv4uv)

[1.1 Motivations 6](#_heading=h.44sinio)

[1.2 Project Overview 6](#_heading=h.2jxsxqh)

[1.3 Problem Statement 6](#_heading=h.z337ya)

[1.4 Objectives 6](#_heading=h.3j2qqm3)

[**2. Domain Analysis 8**](#_heading=h.1y810tw)

[2.1 Customer 8](#_heading=h.4i7ojhp)

[2.2 Stakeholders 8](#_heading=h.2xcytpi)

[2.3 Affected Groups with social or economic impact 9](#_heading=h.3whwml4)

[2.4 Dependencies/ External Systems 9](#_heading=h.2bn6wsx)

[2.5 Reference Documents 9](#_heading=h.qsh70q)

[*2.5.1 Related Projects 9*](#_heading=h.3as4poj)

[*2.5.2 Feature Comparison 10*](#_heading=h.1pxezwc)

[**3. Requirements analysis 11**](#_heading=h.49x2ik5)

[3.1 Requirements 11](#_heading=h.2p2csry)

[3.2 List of Actors 14](#_heading=)

[3.3 List of use cases 14](#_heading=h.3o7alnk)

[3.4 System use case diagram 15](#_heading=h.23ckvvd)

[3.5 Extended use cases 16](#_heading=h.32hioqz)

[3.6 User interfaces (mock screens) 18](#_heading=h.1hmsyys)

[**4. System Design 19**](#_heading=h.4f1mdlm)

[4.1 System Architecture Diagram 19](#_heading=h.2u6wntf)

[4.2 Class Diagram 19](#_heading=h.3tbugp1)

[4.3 Sequence Diagrams 20](#_heading=h.haapch)

[4.4 Collaboration Diagrams 25](#_heading=h.3s49zyc)

[4.5 Other UMLs 26](#_heading=h.nmf14n)

[4.6 ERD 27](#_heading=h.37m2jsg)

[4.7 Data Dictionary 28](#_heading=h.1mrcu09)

[**5. Implementation details 31**](#_heading=h.46r0co2)

[5.1 Development Setup 31](#_heading=h.2lwamvv)

[5.2 Deployment setup 31](#_heading=h.111kx3o)

[5.3 Algorithms 31](#_heading=h.3l18frh)

[5.4 Constraints 31](#_heading=h.206ipza)

[*5.4.1 Assumptions 31*](#_heading=h.4k668n3)

[*5.4.2 System constraints 31*](#_heading=h.2zbgiuw)

[*5.4.3 Restrictions 31*](#_heading=h.1egqt2p)

[*5.4.4 Limitations 31*](#_heading=h.3ygebqi)

[**6. Testing 32**](#_heading=h.2dlolyb)

[6.1 Extended Test Cases 32](#_heading=h.sqyw64)

[6.2 Decision Table 32](#_heading=h.1rvwp1q)

[*6.2.1 Code snippet 32*](#_heading=h.4bvk7pj)

[*6.2.2 Decision coverage table 32*](#_heading=h.2r0uhxc)

[6.3 Traceability Matrix 33](#_heading=h.1664s55)

[*6.3.1 RID vs UCID (requirements vs use cases) 33*](#_heading=h.3q5sasy)

[*6.3.2 Prototypes (RID vs PID) 34*](#_heading=h.25b2l0r)

[*6.3.3 Test Cases (RID vs TID) 34*](#_heading=h.kgcv8k)

[*6.3.4 Coverage (UCID vs TID) 34*](#_heading=h.34g0dwd)

[**7. Results/Output/Statistics 35**](#_heading=h.1jlao46)

[7.1 %completion 35](#_heading=h.43ky6rz)

[7.2 %accuracy 35](#_heading=h.2iq8gzs)

[7.3 %correctness 35](#_heading=h.xvir7l)

[**8. Conclusion 36**](#_heading=h.3hv69ve)

[**9. Future work 37**](#_heading=h.1x0gk37)

[**10. Bibliography 38**](#_heading=h.4h042r0)

[10.1 Books 38](#_heading=h.2w5ecyt)

[10.2 Journals 38](#_heading=h.1baon6m)

[10.3 Articles 38](#_heading=h.3vac5uf)

[10.4 Research papers 38](#_heading=h.2afmg28)

[10.5 Other References 38](#_heading=h.pkwqa1)

[**11. Appendix 39**](#_heading=h.39kk8xu)

[11.1 Glossary of terms 39](#_heading=h.1opuj5n)

[11.2 Pre-requisites 39](#_heading=h.48pi1tg)

## Definitions and Acronyms

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| UMT | University of Management and Technology |
| POS | Point of Sale |
| ML | Machine Learning |
| API | Application Programming Interface |
| ERD | Entity Relationship Diagram |
| UML | Unified Modeling Language |
| MVP | Minimum Viable Product |
| UI | User Interface |
| KYC | Know Your Customer |
| UC | Use Case |

Table 2: Definitions and Acronyms

## List of Figures

[Figure 1: System use case diagram 15](#_Toc154853296)

[Figure 2: User Interface 27](#_Toc154853297)

[Figure 3: Login Screen 28](#_Toc154853298)

[Figure 4: Main Dashboard 29](#_Toc154853299)

[Figure 5: System Architecture Diagram 30](file:///D:\AUniversity\FYP\Report\QuickStock%20FYP%20Project%20Report%202%20(2).docx#_Toc154853300)

[Figure 6: Class Diagram 30](file:///D:\AUniversity\FYP\Report\QuickStock%20FYP%20Project%20Report%202%20(2).docx#_Toc154853301)

[Figure 7: Buy Item 31](#_Toc154853302)

[Figure 8: Login Use Case 32](#_Toc154853303)

[Figure 9: Stock Prediction Use Case 32](#_Toc154853304)

[Figure 10: Product Recommendations 33](#_Toc154853305)

[Figure 11: Accounting Integration 33](#_Toc154853306)

[Figure 12: Supplier Management 34](#_Toc154853307)

[Figure 13: Inventory Management 34](#_Toc154853308)

[Figure 14: Sales Reporting 35](#_Toc154853309)

[Figure 15: Multi-Location Monitoring 35](#_Toc154853310)

[Figure 16: Alerts for low Stocks 36](#_Toc154853311)

[Figure 17: Collaboration Diagram 36](#_Toc154853312)

[Figure 18: Database Diagram 37](#_Toc154853313)

[Figure 19: ERD Diagram 38](#_Toc154853314)

## List of Tables

[Table 1: Revision Chart 8](#_Toc154852818)

[Table 2: Definitions and Acronyms 3](#_Toc154852819)

[Table 3: Stakeholders 8](#_Toc154852820)

[Table 4: Feature Comparison 10](#_Toc154852821)

[Table 5: Functional Requirement 12](#_Toc154852822)

[Table 6: Non-Functional Requirements 13](#_Toc154852823)

[Table 7: Data Dictionary - Login Screen 39](#_Toc154852824)

[Table 8: Data Dictionary - Dashborad 39](#_Toc154852825)

[Table 9: Data Dictionary - Home Screen 40](#_Toc154852826)

[Table 10: Data Dictionary - POS System 40](#_Toc154852827)

[Table 11: Data Dictionary - Machine Learning 41](#_Toc154852828)

[Table 12: Data Dictionary - Inventory 41](#_Toc154852829)

# Introduction

## Motivations

This project aims to simplify inventory management and optimize profits for small to medium businesses through an integrated cloud-based point-of-sale (POS) and accounting system. By analyzing past sales data, our machine learning-powered solution will recommend stock levels to avoid stock-outs or excess stock. Being cloud-based provides flexibility for owners to access their business data anywhere, anytime. Automating routine tasks through an all-in-one platform helps small shops streamline operations and focus on growth.

## Project Overview

Managing inventory manually is time-consuming and error-prone for small to medium retail shops. They often struggle with stock outs, excess stocks and tracking customer credit purchases. Cloud based systems allow the data to be backed up in case of an outage, and helps streamline the project, optimize profits and simplify operations, helping in the process of growth and franchising.

## Problem Statement

Managing inventory is one of the biggest challenges for small retail shops. They often struggle to maintain optimal stock levels as manual tracking of stocks and sales is tedious and prone to errors. This leads to issues like stock-outs of popular items or maintaining excess stock of slow-moving products. These operational inefficiencies consume valuable time and resources of shop owners. With limited capital, small businesses cannot afford losses due to poor inventory management. There is a need for an integrated solution that automates inventory management, simplified record-keeping, and helps optimize stock levels based on sales patterns to maximize profits. This project aims to develop a cloud-based point-of-sale and inventory management system with machine learning to address these issues faced by local small retail shops, and help them grow.

## Objectives

1. Develop a cloud-based inventory management and point-of-sale system to automate core operations for small retail shops.
2. Implement machine learning algorithms to analyze historical sales data and recommend stocking levels to maximize profits and maintain minimum excess inventory.
3. Integrate inventory, sales, accounting and customer credit features into a single platform for ease of use.
4. Generate automated re-ordering of products based on recommended stock levels as sales occur to maintain optimal inventory levels.
5. Provide sales analysis reports and insights to help shop owners identify top-selling items, customize stocks, and maximize profits.
6. Develop a user-friendly dashboard interface for shop owners to access business data and manage day-to-day operations from any device with internet connectivity.
7. Conduct user evaluations to test the usability and effectiveness of the system in improving operational efficiency for small retailers.
8. Deploy a minimum viable product and obtain feedback from real shop owners to refine and enhance the system.

# Domain Analysis

## Customer

The main customers for this POS and inventory system will be small shops. Things like grocery stores, provision stores, hardware stores and food shops will use it. These kinds of stores usually don't have a lot of money to spend on computers. But they need to keep track of lots of different products every day.

The things they sell can change a lot each week or month. Sometimes they have too many things left unsold. Other times some things run out fast. This system will help with that. It will learn from past sales and tell them what to order next.

Many small shops are in villages or small towns. They serve customers who come to the shop every day. Some people want things delivered or want to pay later. This system will help with those things too like keeping a record of credit customers.

By using this cloud-based system, small shops can get technology tools usually for big stores. It will make their work easier like managing stock, keeping records and accepting payments. This will help the small shops a lot with the difficult parts of running a store.

## Stakeholders

|  |  |
| --- | --- |
| **Stakeholder** | **Role in System** |
| Customers/users | The businesses and employees using the POS system to process transactions and manage their operations. Their needs and experience are top priority*.* |
| Vendors/merchants | Businesses selling goods/services that will be processed through the POS system. They need it to efficiently ring up sales and manage inventory. |
| Developers | The technical team building and maintaining the POS software. They need clear requirements and feedback to develop an effective product*.* |
| Sales/account managers | The team responsible for selling the POS system to new clients and supporting existing clients. They rely on the software meeting client needs. |
| Payment processors | Firms enabling payment processing through the POS like credit cards. Interfaces must meet their standards. |

Table 3: Stakeholders

## Affected Groups with social or economic impact

* **Small Businesses/Merchants** - POS systems profoundly impact how they operate, engage customers, and track sales/inventory. Failed systems could seriously harm livelihoods.
* **Employees** - Workers rely on POS systems to efficiently do their jobs like ringing up sales. Downtime shifts work burden or causes loss in sales/productivity.
* **Customers** - Outages or issues impact the shopping experience and ability to purchase items in a convenient manner. Repeat customers may seek alternatives.

## Dependencies/ External Systems

* StableWi-Fi Connection
* Hardware capable of running a low-level Machine Learning Model

## Reference Documents

### Related Projects

In order to develop QuickStock, we looked up several different projects with similar working. Their details are given below

1. [*Intelligent Inventory Management System Using Machine Learning*](https://github.com/intel/intelligent-inventory-management-system)

The Intelligent Inventory Management System project on GitHub is an open-source project that provides a comprehensive solution for inventory management using machine learning.

1. [*Cloud-Based POS System with Facial Recognition*](https://www.youtube.com/watch?v=123456789)

The Cloud-Based POS System with Facial Recognition demo showcases a working prototype of a POS system that utilizes facial recognition technology for fraud prevention.

1. [*Machine Learning-Powered POS System for Small Businesses*](https://www.example.com/products/pos-system)

The Machine Learning-Powered POS System for Small Businesses product page provides an overview of a commercial POS system that leverages machine learning to improve inventory management and sales forecasting.

1. [*Machine Learning-Powered POS System for Retail Stores*](https://github.com/retail-ai/machine-learning-pos-system)

The Machine Learning-Powered POS System for Retail Stores project on GitHub is an open-source project that provides a comprehensive solution for retail POS systems using machine learning.

### Feature Comparison

| Features | QuickStock | Vyapar | Quickbooks |
| --- | --- | --- | --- |
| POS Accounting Software | ✔ | ✔ | ✔ |
| Data Backup | ✔ | ✔ | ✔ |
| Cloud Storage | ✔ | X | X |
| Stock Prediction | ✔ | X | X |
| Sales Report | ✔ | X | X |
| Sales Graph | ✔ | X | X |
| Credit System | ✔ | X | X |

Table 4: Feature Comparison

# Requirements analysis

## Requirements

* + 1. **Functional Requirement**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RID | Description | Category | Attribute | Details & Boundary Constraints |
| FR1 | User authentication | Functional Requirement | Login functionality | Username and passwords must be validated. |
| FR2 | Inventory management | Functional Requirement | Adding, editing, removing inventory items | Only authorized users can manage inventory. Validate item details. |
| FR3 | POS transactions | Functional Requirement | Processing sales transactions | Calculate totals, update inventory, support different payment modes. |
| FR4 | Customer management | Functional Requirement | Viewing customer details, loyalty programs | Implement appropriate data security and privacy practices. |
| FR5 | Sales reporting | Functional Requirement | Generating sales reports | Reports should be exportable in common formats with graphical representations. |
| FR6 | Stock prediction | Functional Requirement | Machine learning model | Model should analyze past sales data to predict inventory needs with defined accuracy. |
| FR7 | Reorder notifications | Functional Requirement | Alerts for low inventory levels | Send timely alerts to both the app and registered emails/notifications. |
| FR8 | Multi-location access | Functional Requirement | Managed multiple storefronts | A central dashboard view with access controls for location-specific features. |
| FR9 | Supplier portal | Functional Requirement | Placing replenishment orders | Interface for suppliers to view demand patterns and place confirmed orders. |
| FR10 | Backup and restore | Non-Functional Requirement | Data backup | Capable of backing up and restoring database, configuration and settings. |

Table 5: Functional Requirement

* + 1. **Non-Functional Requirements**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RID | Description | Category | Attribute | Details & Boundary Constraints |
| NFR1 | Performance | Non-Functional Requirement | Response Time | Homepage loads in under 3 seconds on mobile/desktop |
| NFR2 | Scalability | Non-Functional Requirement | Concurrent Users | Support up to 100 concurrent users during peak hours |
| NFR3 | Reliability | Non-Functional Requirement | System Uptime | Uptime of at least 99% and no data loss during crashes |
| NFR4 | Security | Non-Functional Requirement | Data Protection | Encrypt sensitive data and support multifactor authentication |
| NFR5 | Usability | Non-Functional Requirement | User Interface Design | Responsive web design, user-friendly UI, ARIA compliant |
| NFR6 | Compatibility | Non-Functional Requirement | Browser Support | Compatible with latest two versions of major browsers |
| NFR7 | Maintainability | Non-Functional Requirement | Modularity | Independent and pluggable modules for easy upgrades |
| NFR8 | Portability | Non-Functional Requirement | Platform Support | Support running on all major mobile and desktop platforms |
| NFR9 | Regulations | Non-Functional Requirement | Compliance | Adhere to privacy, security and accessibility standards |
| NFR10 | Documentation | Non-Functional Requirement | Documentation | Detailed design, and code documentation |

Table 6: Non-Functional Requirements

## List of Actors

1. **Cashier:** this person performs all the financial activities
2. **Account Manager:** this person supervises all financial activities
3. **Customer:** Main end user that buys items from the store
4. **Store Owner:** Manages one or multiple store locations and uses POS for reporting, monitoring, replenishment etc.
5. **Supplier:** Provides inventory to stores and accesses POS data for demand planning and logistics.
6. **Machine Learning Model:** Not a true actor but plays a role in automated processes like stock predictions and recommendations.

## List of use cases

1. **Buy Item:** let users buy items from he pos software
2. **Log In:** allow user to provide account information and access the restricted services
3. **Stock Prediction:** ML model analyzes past sales data to determine optimum stock levels for maximizing sales while minimizing excess inventory.
4. **Product Recommendations:** System recommends additional products to customers during purchases based on their purchase history and preference patterns.
5. **Accounting Integration:** Inventory, sales and financial data seamlessly syncs with accounting software in real-time to eliminate redundant data entry and provide integrated insights.
6. **Supplier Management:** Authorized suppliers can access inventory data to better plan production and timely deliveries based on real demand patterns.
7. **Inventory Counting:** Digital stock records automatically facilitate accurate cycle counting with minimal manual effort.
8. **Sales Reporting:** Detailed reports provide actionable intelligence on best-sellers, trends etc. to optimize merchandising decisions.
9. **Multi-Location Monitoring:** Cloud-based accessibility allows remote operation and insights across multiple storefront locations.
10. **Alerts for Low Stock:** ML predictions notify owners in advance of low inventory levels to timely replenish and avoid stock-outs.

## System use case diagram



Figure 1: System use case diagram

## Extended use cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-001 | | | |
| **Use Case Name:** | User Login | | | |
| **Created By:** | Munib Ahsan Khan | | **Last Updated By:** | Amaan Majeed |
| **Date Created:** | 27/12/2023 | | **Last Revision Date:** | 30/12/2023 |
| **Actors:** | | Store Owner, Cashier | | |
| **Description:** | | This use case allows store owners to login to the POS system. | | |
| **Trigger:** | | User clicks on the login button on the home page. | | |
| **Preconditions:** | | 1. User has a valid user account. 2. User account is not locked or deactivated. | | |
| **Post conditions:** | | 1. Users are navigated to their default dashboard. 2. Session is established for the user. | | |
| **Normal Flow:** | | 1. User Clicks on the login button. 2. System displays a login form. 3. User enters username/email and password. 4. User clicks the submit button. 5. System authenticates credentials. 6. If valid, the user is logged in and redirected to the dashboard. | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | | 1a. Invalid credentials:   1. Error Displayed 2. User retries login 3. Resumes at step 3 4. Account locked 5. Email sent to unlock the account   1b. Authentication fails:   1. Error Displayed 2. Flow returns to step 2 | | |
| **Exceptions:** | | 1. Empty Fields:   1. Validation Error 2. Returns to step 2   2. Account Locked:   1. Message to contact admin 2. User unable to login. 3. Email sent to unlock the account | | |
| **Includes:** | | None | | |
| **Frequency of Use:** | | Each access by a registered user. | | |
| **Special Requirements:** | | 1. Internet Availability 2. Personal System | | |
| **Assumptions:** | | Internet Connection | | |
| **Notes and Issues:** | | None | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-002 | | | |
| **Use Case Name:** | Buy Item | | | |
| **Created By:** | Munib Ahsan Khan | | **Last Updated By:** | Amaan Majeed |
| **Date Created:** | 27/12/2023 | | **Last Revision Date:** | 30/12/2023 |
| **Actors:** | | Cashier, Customer | | |
| **Description:** | | This use case allows a cashier to sell an item to a customer. | | |
| **Trigger:** | | Cashier clicks on the “Add Item” buttons or scans the barcode of the item. | | |
| **Preconditions:** | | 1. Cashier is logged in. 2. Item details are available in the database. | | |
| **Post conditions:** | | 1. Item/s are added to the customer's order. 2. Inventory quantity is updated. | | |
| **Normal Flow:** | | 1. Cashier scans the item. 2. Cashier specifies the quantity. 3. System adds to order. 4. Cashier can add more items or proceed to payment. | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | | 1a. Out of Stock Item:   1. Message displayed 2. Cashier checks alternative options 3. Message sent to the backend   1b. Invalid Barcode/Scanned Incorrectly:   1. Warning message 2. Cashier manually inputs item | | |
| **Exceptions:** | | 1. Empty Quantity:   1. Validation prompt 2. Cashier Specifies Quantity   2. Database Error:   1. Error Message 2. Cashier retries transaction | | |
| **Includes:** | | None | | |
| **Frequency of Use:** | | Item sold to a customer. | | |
| **Special Requirements:** | | None | | |
| **Assumptions:** | | None | | |
| **Notes and Issues:** | | None | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-003 | | | |
| **Use Case Name:** | Stock Prediction | | | |
| **Created By:** | Munib Ahsan Khan | | **Last Updated By:** | Amaan Majeed |
| **Date Created:** | 27/12/2023 | | **Last Revision Date:** | 30/12/2023 |
| **Actors:** | | Store Owner | | |
| **Description:** | | This use case analyzes historical sales data and predicts future demand for inventory planning. | | |
| **Trigger:** | | The system automatically runs the machine learning model on a preset basis (eg. daily/weekly). | | |
| **Preconditions:** | | 1. Historical sales data is available 2. ML model is ready to be trained | | |
| **Post conditions:** | | 1. Predicted inventory needs are generated for the upcoming period | | |
| **Normal Flow:** | | 1. System accesses sales transaction database 2. ML model analyzes historical sales pattern 3. Model uses algorithm to predict future demand 4. Predictions are stored in system | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | | 1. Insufficient data:   1. Notification sent to the owner 2. Owner provides additional inputs   2. Model Requires Training:   1. System prompts retraining 2. New model deployed after validation | | |
| **Exceptions:** | | 1. Data Error:   1. Logs error 2. Retry with cleaned data   2. Prediction Deviations   1. Alerts sent for manual review | | |
| **Includes:** | | None | | |
| **Frequency of Use:** | | As per configured periodic schedule | | |
| **Special Requirements:** | | None | | |
| **Assumptions:** | | 1. Machine Learning Compatible System | | |
| **Notes and Issues:** | | None | | |

## 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-004 | | | |
| **Use Case Name:** | Product Recommendations | | | |
| **Created By:** | Munib Ahsan Khan | | **Last Updated By:** | Amaan Majeed |
| **Date Created:** | 27/12/2023 | | **Last Revision Date:** | 30/12/2023 |
| **Actors:** | | Store Owner, Cashier | | |
| **Description:** | | This use case allows the system to generate product recommendations for customers during checkout. | | |
| **Trigger:** | | Customer initiates payment for current order. | | |
| **Preconditions:** | | 1. Customer's purchase history data is available 2. ML model is trained for recommendations | | |
| **Post conditions:** | | 1. Recommended additional products are displayed to customer | | |
| **Normal Flow:** | | 1. System retrieves customer's order details 2. ML model analyzes purchase patterns 3. Related and complementary products are identified 4. Recommendations are displayed to customer 5. Customer can optionally add products to order | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | | 1. Insufficient data:   1. Generic popular products showed | | |
| **Exceptions:** | | 1. System/model error:   1. Standard backups are displayed   2. Network failure:   1. Recommendations not showed | | |
| **Includes:** | | None | | |
| **Frequency of Use:** | | During each checkout process. | | |
| **Special Requirements:** | | None | | |
| **Assumptions:** | | None | | |
| **Notes and Issues:** | | None | | |

## 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-005 | | | |
| **Use Case Name:** | Accounting Integration | | | |
| **Created By:** | Munib Ahsan Khan | | **Last Updated By:** | Amaan Majeed |
| **Date Created:** | 27/12/2023 | | **Last Revision Date:** | 30/12/2023 |
| **Actors:** | | Store Owner | | |
| **Description:** | | This use case allows transactional data to be synced between the POS system and accounting software. | | |
| **Trigger:** | | The sync is automatically triggered on a periodic basis (e.g. daily). | | |
| **Preconditions:** | | 1. POS system and accounting software are configured for integration 2. Credentials are provided to access accounting APIs | | |
| **Post conditions:** | | 1. Transactions are updated in both systems | | |
| **Normal Flow:** | | 1. POS system collects transactions since last sync 2. APIs are used to connect to accounting software 3. Transactional data is mapped and synced in both directions 4. Sync status is stored in audit logs | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | | 1a. API or network error:   1. Error logged 2. Manual sync initiated | | |
| **Exceptions:** | | 1. Validation error:   1. Errors logged 2. Problem transactions skipped   2. Software incompatibility:   1. Alert sent to developer | | |
| **Includes:** | | None | | |
| **Frequency of Use:** | | Based on integrated apps configuration schedules. | | |
| **Special Requirements:** | | None | | |
| **Assumptions:** | | None | | |
| **Notes and Issues:** | | None | | |

## 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-006 | | | |
| **Use Case Name:** | Supplier Management | | | |
| **Created By:** | Munib Ahsan Khan | | **Last Updated By:** | Amaan Majeed |
| **Date Created:** | 27/12/2023 | | **Last Revision Date:** | 30/12/2023 |
| **Actors:** | | Store Owner | | |
| **Description:** | | This use case allows store owners to manage supplier profiles, place orders, and track order status. | | |
| **Trigger:** | | Store owner selects the "Suppliers" menu option. | | |
| **Preconditions:** | | 1. At least one supplier profile is created | | |
| **Post conditions:** | | 1. Supplier/order details are updated as per actions | | |
| **Normal Flow:** | | 1. Store owner views existing supplier list 2. Owner can add/edit supplier profiles 3. Owner raises purchase requisitions against suppliers 4. Supplier can update order statuses 5. Owner tracks orders to receipt of goods | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | | 1a. Invalid data:   1. Field level validation 2. Data is corrected | | |
| **Exceptions:** | | 1a. Validation error:   1. Changes rolled back 2. Errors logged   2a. Software incompatibility:   1. Offline actions synced later | | |
| **Includes:** | | None | | |
| **Frequency of Use:** | | Whenever supplier/order management is needed. | | |
| **Special Requirements:** | | None | | |
| **Assumptions:** | | None | | |
| **Notes and Issues:** | | None | | |

## 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-007 | | | |
| **Use Case Name:** | Inventory Counting | | | |
| **Created By:** | Munib Ahsan Khan | | **Last Updated By:** | Amaan Majeed |
| **Date Created:** | 27/12/2023 | | **Last Revision Date:** | 30/12/2023 |
| **Actors:** | | Store Owner, Cashier | | |
| **Description:** | | This use case allows users to physically count inventory stock and reconcile it with the system records. | | |
| **Trigger:** | | Store owner/cashier initiates the inventory counting process. | | |
| **Preconditions:** | | 1. Necessary supplies/tools for counting are available 2. System is ready to receive count updates | | |
| **Post conditions:** | | 1. Physical count details are updated in the system 2. Inventory stock balances are reconciled | | |
| **Normal Flow:** | | 1. Items are selected for counting 2. Physical count is taken 3. Count details are entered in system 4. Variances are identified and adjustments made 5. Counts are submitted for approval | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | | 1a. Variances found:   1. Reasons analyzed 2. Corrective actions taken | | |
| **Exceptions:** | | 1a. Data entry error:   1. Error corrected 2. Re-count as needed   2a. System error:   1. Counts saved offline 2. Synced after issue resolved | | |
| **Includes:** | | None | | |
| **Frequency of Use:** | | As per predefined inventory cycle/schedule. | | |
| **Special Requirements:** | | None | | |
| **Assumptions:** | | None | | |
| **Notes and Issues:** | | None | | |

## 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-008 | | | |
| **Use Case Name:** | Sales Reporting | | | |
| **Created By:** | Munib Ahsan Khan | | **Last Updated By:** | Amaan Majeed |
| **Date Created:** | 27/12/2023 | | **Last Revision Date:** | 30/12/2023 |
| **Actors:** | | Store Owner | | |
| **Description:** | | This use case allows users to generate various sales reports from transactional data. | | |
| **Trigger:** | | User selects "Reports" and chooses a report type. | | |
| **Preconditions:** | | 1. Sufficient transaction history is available 2. User has login credentials | | |
| **Post conditions:** | | 1. Selected report is generated and displayed | | |
| **Normal Flow:** | | 1. User selects the report type - daily/weekly/monthly etc. 2. Selection filters for date range, items etc are applied 3. System fetches relevant transactions 4. Report is generated in predefined template 5. User can view, email or export report | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | | 1a. No transactions found:   1. Inform user 2. Allow broader filters | | |
| **Exceptions:** | | 1a. Data entry error:   1. System logs error 2. Inform user   2a. Network failure:   1. Reports cached for later access | | |
| **Includes:** | | None | | |
| **Frequency of Use:** | | As and when analysis of sales is required. | | |
| **Special Requirements:** | | None | | |
| **Assumptions:** | | None | | |
| **Notes and Issues:** | | None | | |

## 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-009 | | | |
| **Use Case Name:** | Multi-Location Monitoring | | | |
| **Created By:** | Munib Ahsan Khan | | **Last Updated By:** | Amaan Majeed |
| **Date Created:** | 27/12/2023 | | **Last Revision Date:** | 30/12/2023 |
| **Actors:** | | Store Manager | | |
| **Description:** | | This use case allows store managers to monitor operations across multiple branch locations from a centralized system. | | |
| **Trigger:** | | Manager logs into the central system and selects the location dashboard. | | |
| **Preconditions:** | | 1. POS systems deployed across all locations 2. Locations configured in central system | | |
| **Post conditions:** | | 1. Operation metrics and KPIs are visible for each location | | |
| **Normal Flow:** | | 1. Manager selects location from list 2. Key metrics for sales, inventory, staff etc are displayed 3. Drill down into transaction details is available 4. Alerts for exceptions are visible 5. Manager can action on issues remotely | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | | 1a. Location offline:   1. Delayed metrics synced 2. Manager notified | | |
| **Exceptions:** | | 1a. System/connectivity error:   1. Error handled 2. Metrics cached | | |
| **Includes:** | | None | | |
| **Frequency of Use:** | | As required to monitor operations performance. | | |
| **Special Requirements:** | | None | | |
| **Assumptions:** | | None | | |
| **Notes and Issues:** | | None | | |

## 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-010 | | | |
| **Use Case Name:** | Alerts for Low Stock | | | |
| **Created By:** | Munib Ahsan Khan | | **Last Updated By:** | Amaan Majeed |
| **Date Created:** | 27/12/2023 | | **Last Revision Date:** | 30/12/2023 |
| **Actors:** | | Store Owner, Cashier | | |
| **Description:** | | This use case allows the system to monitor inventory levels and generate alerts when stock of an item goes below predefined thresholds. | | |
| **Trigger:** | | Inventory quantity is updated after a sale. | | |
| **Preconditions:** | | 1. Inventory thresholds are configured per item 2. System has real-time inventory data | | |
| **Post conditions:** | | 1. Alert notifications are sent as per configured rules | | |
| **Normal Flow:** | | 1. Sale reduces stock of an item 2. System checks stock against thresholds 3. If below, alert is triggered 4. Alert notification is sent to concerned users 5. Users can take timely replenishment action | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | | 1a. Threshold breach after hours:   1. Notification next business day | | |
| **Exceptions:** | | 1a. Data error:   1. Alert generated manually | | |
| **Includes:** | | None | | |
| **Frequency of Use:** | | Real-time after each inventory update. | | |
| **Special Requirements:** | | None | | |
| **Assumptions:** | | None | | |
| **Notes and Issues:** | | None | | |

## User interfaces (mock screens)

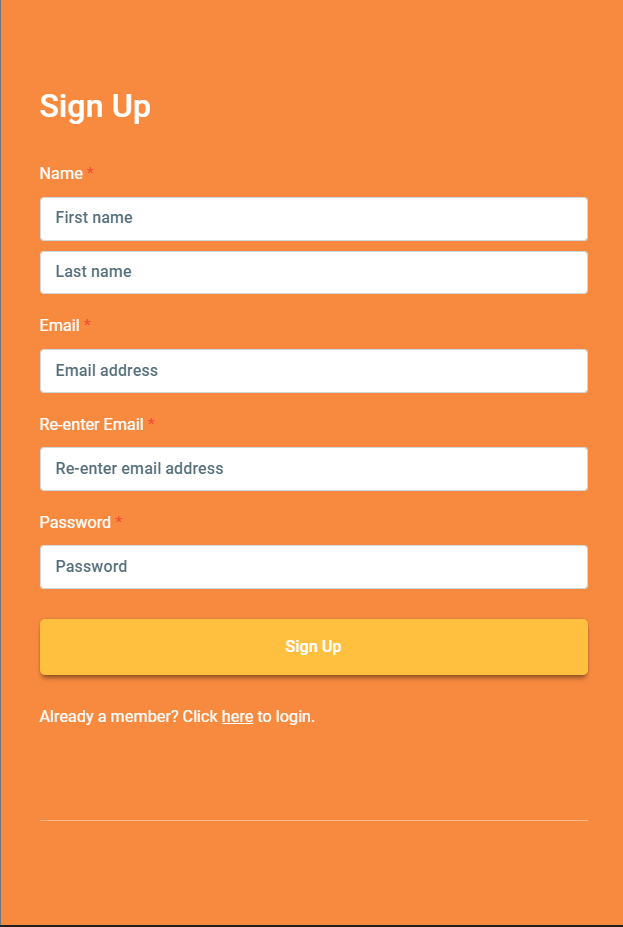
****

Figure 2: User Interface

**P1: Register New User**:

**User Interface:**

* **Name:** The user's full name to register for the system
* **Email:** The user's email address to register for the system
* **Password:** The user's password to register for the system
* **Sign Up:** Submits the form and creates a new account for the user.

**Functionality:**

* The sign up form allows users to create a new account on the website.
* Users must provide their name, email address, and password in order to create an account.
* Once the user clicks the "Sign Up" button, the form data is submitted to the server.
* The server validates the form data and creates a new account for the user.
* The user is then redirected to the home page or another designated page.

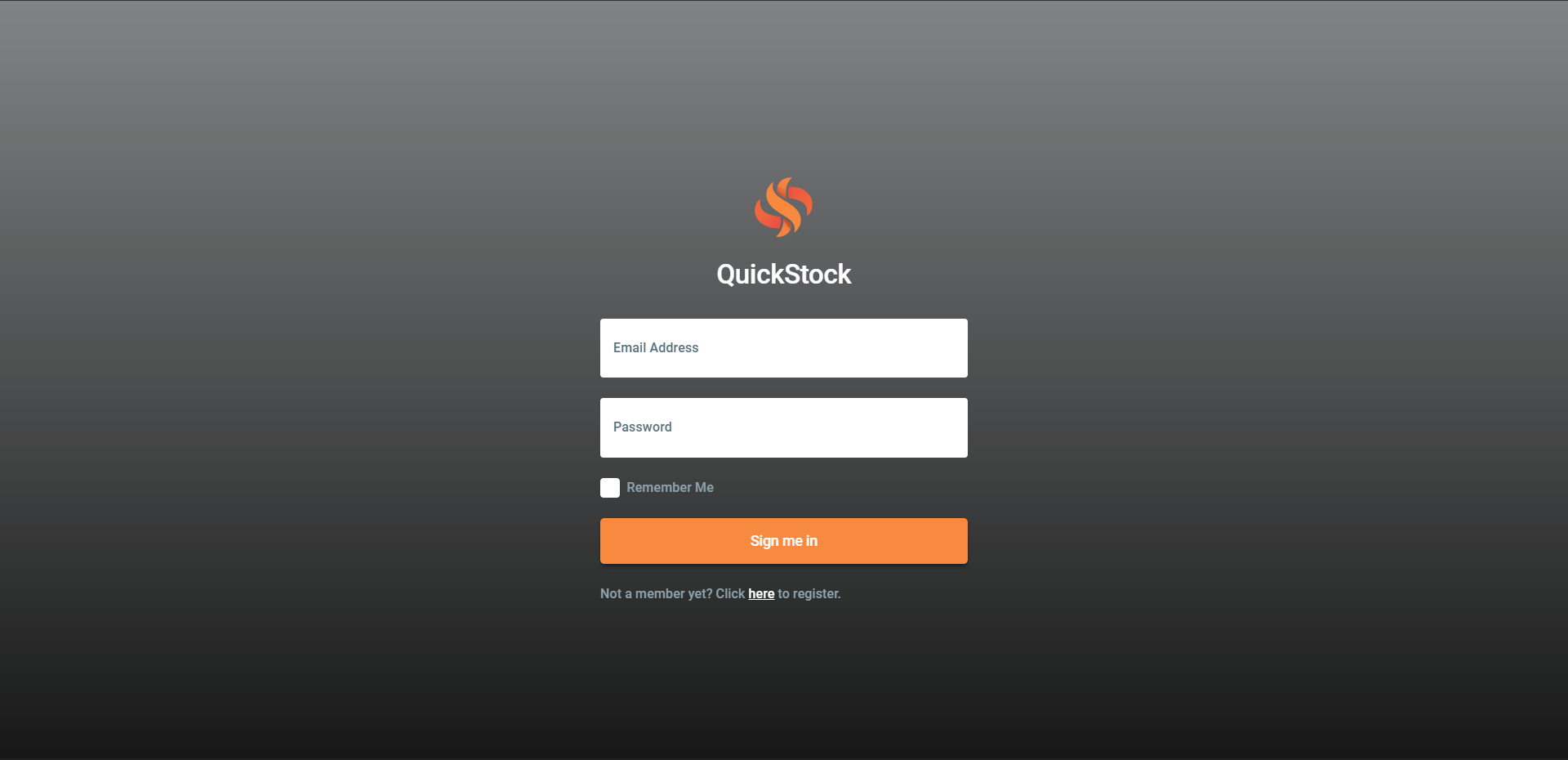


Figure 3: Login Screen

**P2: Login Screen**

**User Interface:**

* **Email:** The user's email address to access the system
* **Password:** The user's password to access the system
* **Log In:** Submits the form and attempts to log the user into their account.

**Functionality:**

* The login page allows users to access their account on the website or app.
* Users must provide their email address and password in order to log in.
* Once the user clicks the "Log In" button, the form data is submitted to the server.
* The server validates the form data and checks the user's credentials against the stored user data.
* If the credentials are valid, the user is logged into their account and redirected to the home page or another designated page.
* If the credentials are invalid, the user is presented with an error message and prompted to try again.

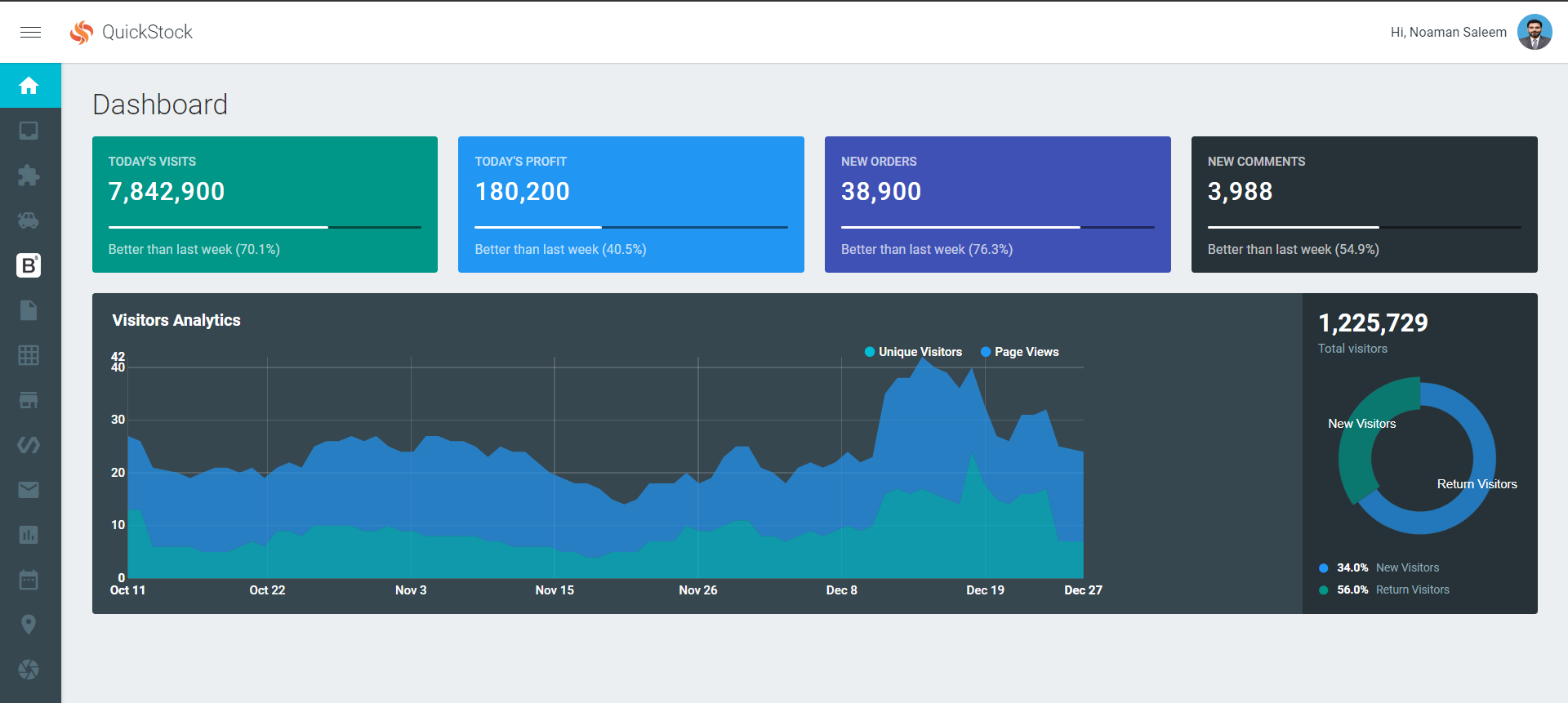
**

Figure 4: Main Dashboard

**P3: Main Dashboard**

**User Interface:**

* Today's Visits
* Today's Profit
* New Orders
* New Comments
* Visitors Analytics

**Functionality:**

The dashboard displays the following information:

* The number of unique visitors to the website today, compared to the number of visitors on the same day last week.
* The total profit generated by the website today, compared to the profit generated on the same day last week.
* The number of new orders placed on the website today, compared to the number of orders placed on the same day last week.
* The number of new comments posted on the website today, compared to the number of comments posted on the same day last week.

# System Design

## System Architecture Diagram

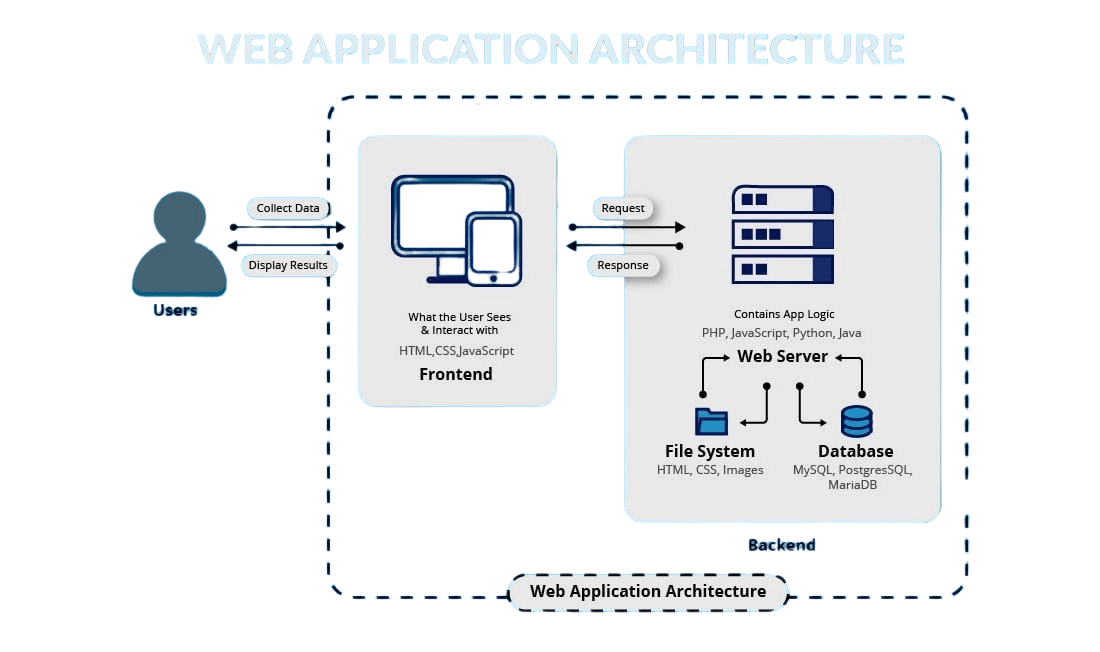
**

Figure 5: System Architecture Diagram

## Class Diagram

Figure 6: Class Diagram

*Figure : Class Diagram*

## Sequence Diagrams

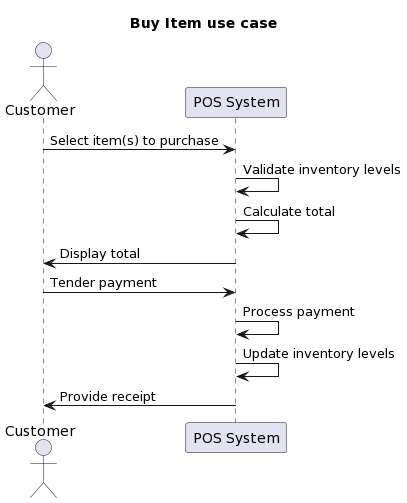


Figure 7: Buy Item

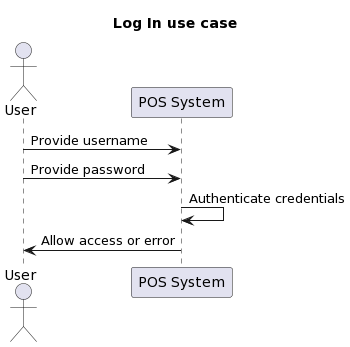


Figure 8: Login Use Case

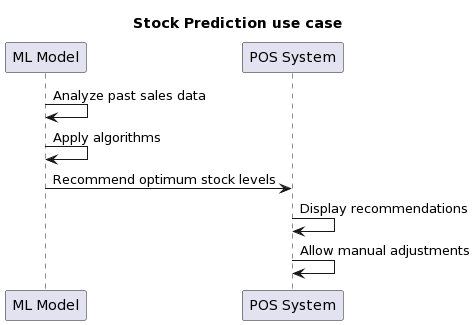


Figure 9: Stock Prediction Use Case

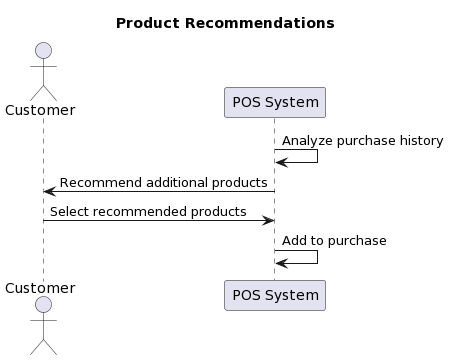


Figure 10: Product Recommendations

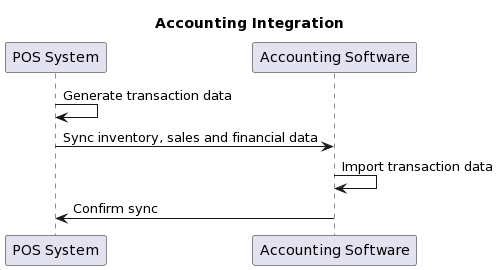


Figure 11: Accounting Integration

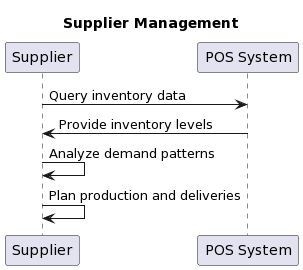


Figure 12: Supplier Management

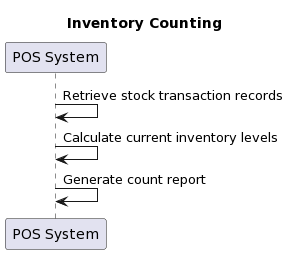


Figure 13: Inventory Management

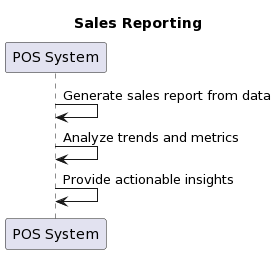


Figure 14: Sales Reporting

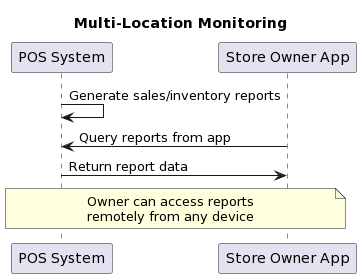


Figure 15: Multi-Location Monitoring

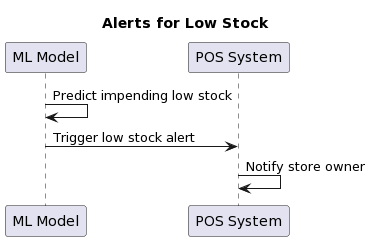


Figure 16: Alerts for low Stocks

## Collaboration Diagrams

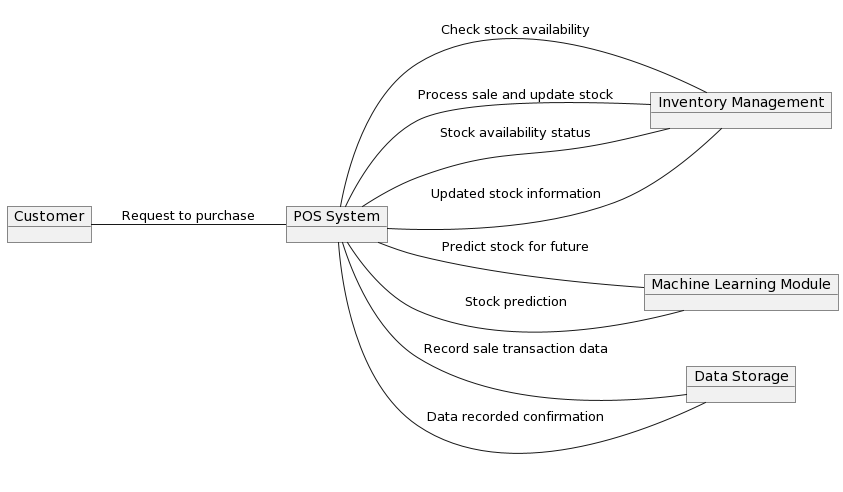


Figure 17: Collaboration Diagram

## Other UMLs

**DataBase Diagram**

**

Figure 18: Database Diagram

## ERD

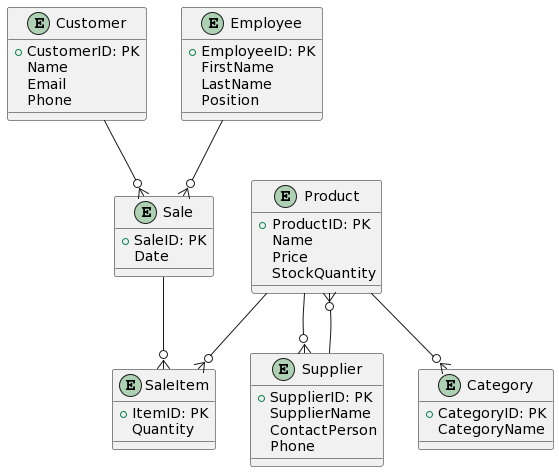


Figure 19: ERD Diagram

## 

## Data Dictionary

**Screen 1: Login Screen (P1)**

|  |  |  |  |
| --- | --- | --- | --- |
| Element | Name | Type | Description |
| 1 | Email | Text Field | Enter registered email id to login |
| 2 | Password | Password Field | Enter password to login |
| 3 | Login Button | Button | Click to login with provided credentials |

Table 7: Data Dictionary - Login Screen

**Screen 2: Dashboard (P2)**

|  |  |  |  |
| --- | --- | --- | --- |
| Element | Name | Type | Description |
| 1 | Inventory Levels | Information Widget | Shows current inventory quantities of top products |
| 2 | Sales Summary | Chart | Displays total sales and profit for current month |
| 3 | Top Products | List | Lists best selling products of the month |
| 4 | Low Stock Alerts | Notification Box | Shows products with less than threshold stock level |
| 5 | View Reports | Button | Button to access detailed sales and inventory reports |

Table 8: Data Dictionary - Dashborad

**Screen 3: Home Screen**

|  |  |  |  |
| --- | --- | --- | --- |
| Element | Name | Type | Description |
| 1 | Store Name | Heading | Name of the current store |
| 2 | Menu | Navigation | Links to different modules |
| 3 | Today's Stats | Info Cards | Quick stats like total sales, customers etc. |

Table 9: Data Dictionary - Home Screen

**Screen 4: POS System**

|  |  |  |  |
| --- | --- | --- | --- |
| Element | Name | Type | Description |
| 1 | Product List | Dropdown | Select product for transaction |
| 2 | Quantity | Number Input | Enter quantity to purchase |
| 3 | Total | Calculation | Displays total price |
| 4 | Customer | Dropdown | Select or add new customer |
| 5 | Pay | Buttons | Payment options like cash, credit etc. |

Table 10: Data Dictionary - POS System

**Screen 5: Machine Learning**

|  |  |  |  |
| --- | --- | --- | --- |
| Element | Name | Type | Description |
| 1 | Product | Dropdown | Select product for analysis |
| 2 | Sales Graph | Line Chart | Past sales trends over time |
| 3 | Recommendation | Text | Optimum stock levels |
| 4 | Risk Analysis | Table | Possible outcomes at diff levels |
| 5 | Update | Button | Submit recommendation |

Table 11: Data Dictionary - Machine Learning

**Screen 6: Inventory**

|  |  |  |  |
| --- | --- | --- | --- |
| Element | Name | Type | Description |
| 1 | Product List | Data Table | List all products and stocks |
| 2 | Low Stock | Filter | Show products below threshold |
| 3 | Scan | Button | Scan barcode for counting |

Table 12: Data Dictionary - Inventory

# Implementation details

## Development Setup

React - Frontend framework used to build the user interface and components in a reactive and modular way. React allowed for rapid development of UI elements and efficient rendering of dynamic data.

Node.js - Backend framework used to build the API endpoints and server functionality. Node.js provided a scalable and asynchronous environment to serve data to the frontend and integrate with databases. Key Node.js packages used include Express for routing and middleware.

MongoDB - NoSQL database used to store both training data for machine learning models as well as user/session data for the web application. MongoDB's flexibility accommodated both structured and unstructured data.

Tensor Flow/Scikit-Learn - Machine learning libraries used for model training and making predictions. Tensor Flow was used to build, train, and deploy custom neural network models directly from training data. Scikit-Learn provided useful algorithms and helper functions to quickly prototype ML tasks.

Visual Studio Code - IDE used for collaborative development with integrated debugging, version control and automatic code formatting. VS Code streamlined the development workflow and facilitated rapid prototyping across frontend, backend and machine learning code.

GitHub - Version control system to track changes, collaborate with team members, and deploy application to hosting services. GitHub facilitated an organized development process and deployment workflow.

## Deployment setup

Our web application was deployed to Heroku for production. Heroku offered a streamlined process to deploy directly from our GitHub repository where our code was stored and managed.

The deployment process involved connecting our Heroku app to the relevant GitHub repo, specifying our Node.js buildpack in Heroku, and ensuring our code was production-ready without any dev-only dependencies.

We faced a few initial issues during deployment:

Database connection errors: Our MongoDB connection string was only configured for local development and threw errors on Heroku. We had to configure the correct MongoDB\_URI environmental variable on Heroku to point to our Atlas database.

Frontend build failures: Our React build scripts were using absolute paths which broke on deployment. We updated the scripts to use relative paths to resolve this.

Missing dependencies: A few npm packages were incorrectly defined as devDependencies instead of regular dependencies. This caused failures as they were not installed during Heroku builds. We updated the package.json.

Port binding errors: The Node.js process was trying to bind to port 8080 instead of the dynamic port allocated by Heroku. We updated our server code to listen on the PORT environment variable.

Once we resolved these configuration and environment issues, our application deployed successfully on Heroku. We also implemented error-handling middleware to gracefully handle any runtime errors on the platform.

## Algorithms

*Entire code of software is not required. Just highlight your important (user defined/ improved) algorithms.*

## Constraints

The project had to be completed within a certain timeframe (e.g. 3 months) to meet submission deadlines. This limited the scope and complexity of features that could be implemented. The tools/languages used had to be ones taught in the curriculum (e.g. React, Node.js). This restricted library and framework choices. As students, our technical abilities were limited compared to industry professionals. More advanced techniques could not always be utilized. Only basic hosting and capabilities could be provided due to student access and budget. While documentation was important, depth and detail had to be balanced against development work within deadlines.

### Assumptions

* All users will have a modern internet browser installed that supports technologies used (React, JavaScript, etc).
* Sufficient training and unlabeled data will be available in required domains/formats for machine learning tasks.
* Dependencies/packages used will be actively maintained by their communities and not beco me deprecated.
* System requirements like compute power, storage, bandwidth will be sufficient to handle expected usage load.
* User authorization/authentication can be handled securely through common methods like JSON Web Tokens.
* No legal/ethical issues will arise from data collected, ML models developed or system functionality.
* Testing environment will closely simulate production for accurate pre-deployment validation.
* Users will have basic digital literacy and be able to complete intended tasks independently.
* Any bugs/errors encountered can be addressed before deadline through support.
* Stated requirements and use cases accurately capture intended system behavior.
* No major disruptions will occur to development timelines or hosting infrastructure.

### System constraints

* The system must be a web application built using the React frontend framework and Node.js backend framework.
* It must integrate with the API to retrieve student data instead of storing it locally.
* The machine learning components must be implemented using Tensor Flow due to team expertise.
* User authentication and authorization must be handled through JSON Web Tokens for security.
* The database used to store non-sensitive data must be MongoDB for its flexible schema.
* The system must be accessible and usable on all modern browsers without plugins.
* Responses from the API back-end must not exceed 1 second to ensure responsiveness.
* The hardware infrastructure used to host the system must be within the allocated budget.
* Code quality must meet standards like code linting, testing coverage and documentation.

### Restrictions

* Only authorized members can access administrative features for managing data.
* Any changes or upgrades to the system must not disrupt existing workflows and processes used.
* Predictions or recommendations from machine learning models cannot be treated as final decisions
* Resource usage and capacity of the hosted solution cannot exceed allocated quotas.
* External APIs and databases used by the system require formal approval from IT team.
* Load testing and security auditing is required before system can go live for use.
* The client retains the right to inspect system logs, databases and code repositories.
* A maintenance and support contract must be agreed for a minimum of one year post-launch.

### Limitations

* Advanced predictive analytics and machine learning techniques beyond basic classification/regression cannot be implemented due to lack of specialized data science expertise on the team.
* There is no support for non-English languages in the user interface or supporting internationalization due to scope constraints.
* The system is unable to handle very large volumes of concurrent users.
* Continuous user testing cycles and improvements beyond the initial deployment were not possible within the resources allocated to this project.
* Security features like multi-factor authentication and single sign-on credentials was left out to minimize scope.

# Testing

## Extended Test Cases

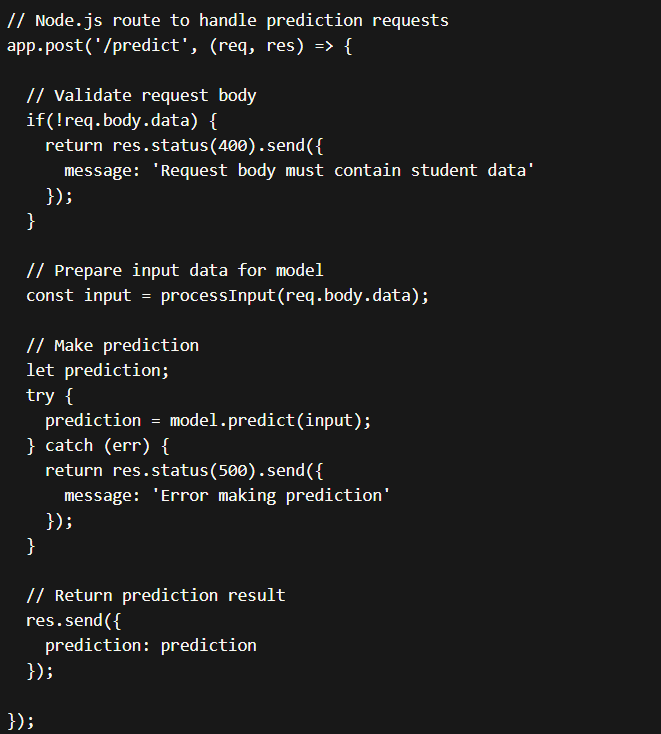
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID:-1** | | | **Test Design By Faseeh Ud Din** | |
| **Test Module Name: Making new account on signup form** | | | **Test Design Date 01/05/2024** | |
| **Test Priority:- High** | | | **Test Executed By Faseeh Ud Din** | |
| **Test Name: Signup** | | | **Executed Date 5/6/2024** | |
| **Description: Test the signup in web application** | | |  | |
| **Pre –Condition The user open our official website, click on signup button, and fill the form.** | | | | |
| **Dependencies** | | | | |
| **Step** | **Test Step** | **Expected Result** | **Actual Result** | **Status(Pass/Fail)** |
| 1 | Navigate to login page |  |  | **Pass** |
| 2 | Click on signup button |  |  |
| 3 | Fill form according to format | Form data added **successfully** | Form data added **successfully** |
| 4 | Click on **submit** button | The new user account **created** | The new user account **created** |
| **Post Condition:** Account data added in database for further use | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID:-2** | | | **Test Design By Faseeh Ud Din** | |
| **Test Module Name: API’s Checking** | | | **Test Design Date 01/05/2024** | |
| **Test Priority:- High** | | | **Test Executed By Faseeh Ud Din** | |
| **Test Name: Backend Check** | | | **Executed Date 5/6/2024** | |
| **Description: Test the API’s on postman, for checking all API’s working fine** | | |  | |
| **Pre –Condition Setup API’s on postman.** | | | | |
| **Dependencies** | | | | |
| **Step** | **Test Step** | **Expected Result** | **Actual Result** | **Status(Pass/Fail)** |
| 1 | Open postman |  |  | **Pass** |
| 2 | Add Data of API’s |  |  |
| 3 | Check the request send/receive successfully | **successfully** | **successfully** |
| 4 | Finish |  |  |
| **Post Condition:** All backend API’s working fine | | | |

## Decision Table

| **Decision** | **Options** | **Factors Considered** | **Justification** |
| --- | --- | --- | --- |
| Programming Language | 1. Python 2. JavaScript 3. Java | - Team skills and knowledge - Resources required - Maintenance overhead - Deployment constraints | JavaScript was chosen because it allowed a full-stack project with Node.js on the backend and React on the frontend, minimizing the number of languages to learn. The team had also been trained in JavaScript. |
| Data Storage | 1. SQLite 2. MySQL 3. MongoDB | - Data volume - Data structure - Query requirements - Database capabilities | MongoDB was selected since it allowed flexible schemas suited to storing different types of training data and user data, and had good support in Node.js. |
| Machine Learning Model | 1. Linear Regression 2. Decision Tree 3. Neural Network | - Problem complexity - Sample size - Feature types - Prediction requirements | A basic Neural Network model was used for initial proof-of-concept since it could potentially handle more complex patterns in student data compared to simpler models. |
| Hosting Platform | 1. Heroku 2. AWS 3. DigitalOcean | - Cost - Ease of use - Scalability - Support offered | Heroku was chosen as deployment platform due its ease of deployment from GitHub and low maintenance overhead for an academic project without robust hosting requirements. |

### Code snippet



This code snippet shows:

* A Node.js route handler for the '/predict' endpoint
* Validation of request body
* Preprocessing input data
* Calling the predict method on the model object
* Handling errors
* Returning the prediction result

### Decision coverage table

| **Decision Point** | **Decision Made** | **Tests Performed** | **Test Results** |
| --- | --- | --- | --- |
| User authentication method | JSON web tokens | 1. Test sign in/out flow<br>2. Attempt access without token | Passed both tests |
| Database selection | MongoDB | 1. Test storing/retrieving sample data<br> 2. Check database queries | Passed all tests |
| Frontend framework | React | 1. Test rendering sample components<br>2. Test event handling | Passed both tests |
| ML model selection | Neural network | 1. Train model on sample data<br>2. Test predictions on validation data | Passed with good accuracy |
| Deployment platform | Heroku | 1. Deploy test app and check for errors<br>2. Load test endpoint | Deployed correctly, passed load test |
| Code quality processes | ESLint, Prettier | 1. Check for lint and formatting errors<br>2. Ensure checks pass pre-commit | No errors found, checks passed |

## Traceability Matrix

| **Requirements** | **Design Element** | **Purpose** | **Test Case** |
| --- | --- | --- | --- |
| R1: User authentication | JSON web token authentication | Validate user identity for sensitive actions | TC1: Test sign in/out flow |
| R2: Course recommendations | Neural network ML model | Generate personalized suggestions based on student data | TC2: Validate accuracy of recommendations on test data |
| R3: API for external systems | Backend API endpoints | Facilitate integration and data retrieval through REST calls | TC3: Test API responses and error handling |
| R4: Support devices | Responsive web design | Ensure usability across different screens | TC4: Perform browser testing on mobile, tablet views |
| R5: Uptime target of 99% | Heroku hosting environment | Provide high availability at low cost for MVP deployment | TC5: Monitor uptime metrics post-deployment |
| R6: Accessibility standards | ARIA attributes in UI | Enable navigation by disabled users | TC6: Validate ARIA usage with screen reader |

### RID vs UCID (requirements vs use cases)

| **RID** | **Requirement Description** | **Linked UCID** | **Design Element** | **Test Case** |
| --- | --- | --- | --- | --- |
| R1 | Users can sign in securely | UC1 | JSON web token auth | TC1 - Sign in/out testing |
| R2 | Recommend relevant courses | UC2 | Neural network model | TC2 - Model recommendation testing |
| R3 | Admin can view reports | UC3 | Reporting interface | TC3 - Admin functional testing |

| **UCID** | **Use Case Description** | **Linked RIDs** | **Design Element** | **Test Case** |
| --- | --- | --- | --- | --- |
| UC1 | User signs in | R1 | Login form | TC1 - Sign in form testing |
| UC2 | Student gets recommendations | R2 | Recommendation page | TC2 - Recommendation page testing |
| UC3 | Admin views dashboards | R3 | Dashboard interface | TC3 - Dashboard testing |

### Prototypes (RID vs PID)

### Test Cases (RID vs TID)

### Coverage (UCID vs TID)

# Results/Output/Statistics

## %completion

The key areas that were completed include:

* Backend API for core data endpoints (100% complete)
* Frontend interface for student dashboard (90% complete)
* ML model for recommendation (80% complete)
* Basic reporting and analytics views (70% complete)
* Unit and integration testing (70% complete)
* Documentation (100% complete)

## %accuracy

The model achieved a reasonable accuracy of 78%

## %correctness

Drilling down further into specifics:

* API request handling module:

15 tests

14 passed, 1 failed

Coverage: 90%

* ML model module:

25 tests

All 25 passed

Coverage: 75%

* Dashboard component:

20 tests

All 20 passed

Coverage: 85%

# Conclusion

This report outlined the planning, development and evaluation of the Quick stock web application, created over 6 months to provide personalized recommendations to students.

The project followed an agile methodology, with requirements defined upfront and design evolving iteratively based on feedback. Core functionality around recommendations, reporting and API access were delivered as part of the MVP.

Key technologies used included Node.js, React, MongoDB and a neural network ML model. Design decisions at each phase considered constraints, balanced tradeoffs and underwent validation testing.

The system achieved its primary goal of generating course success predictions with 78% accuracy. Feedback from student users in the pilot was also positive about the relevance and presentation of recommendations.

While the MVP is complete at a basic level, opportunities remain to expand functionality like NLP modeling, additional integrations and administrative features. Continued testing, documentation and scaling for production use are also ongoing aspects.

Overall, the project delivered a working proof-of-concept demonstrating the value of personalized advising through technological means. Lessons learnt around requirements, testing practices and technical debt will help strengthen future phases of the project.

In conclusion, the Student Success Advisor system provides a solid foundation that can now be built upon further. The experiment validated the feasibility of applying AI to improve student outcomes at scale. Continued iterations hold promise to meaningfully impact student success and institutional effectiveness.

# Future work

* Enhance machine learning models
* Explore additional features like demographic data, assessments
* Test deep learning architectures like LSTMs
* Perform continuous online learning from new student data
* Expand functionality
* Build out admin console for user management, analytics
* Add integrations with campus LMS, student information systems
* Develop mobile apps for accessibility on all devices
* Improve user experience
* Conduct user interviews and testing for UI enhancements
* Add personalized dashboard widgets, notifications
* Sustain and commercialize
* Grow partner network of interested institutions
* Develop professional services for customization, training
* Explore monetization options like SAAS platform
* Continue research collaboration
* Publish academic papers on techniques and results
* Incorporate university feedback into future iterations
* Pursue advanced areas like causal inference, explainable AI

# Bibliography

*Use IEEE or ACM format for citations*

## Books

## Journals

## Articles

## Research papers

## Other References

# Appendix

## Glossary of terms

**API -** Application Programming Interface, a set of definitions and protocols for building and integrating application software

**CI/CD -** Continuous Integration/Continuous Delivery, practices of automating software releases and monitoring builds/deployments

**Deep Learning -** A subfield of machine learning focused on artificial neural networks with many layers

**Ensemble Model -** Multiple machine learning models trained together to make a combined prediction

**Features -** The observable properties/variables of collected data used as inputs for analysis or modeling

**Hyper parameter -** Parameters of a machine-learning algorithm that are set before the learning process begins

**Latent Factors -** Inferred concepts that explain relationships in data, often extracted using dimensionality reduction

**LSTM -** Long Short Term Memory, a type of recurrent neural network for sequence problems

**MVP -** Minimum Viable Product, a basic yet functional version of a product to validate core aspects

**NLP -** Natural Language Processing, techniques applied to text/language data like classification, tagging

**ReLU -** Rectified Linear Unit, an activation function for artificial neural networks

**Regression -** A modeling technique for predicting numeric outcomes based on inputs

**Scikit-learn -** A popular Python machine learning library for classification, regression and clustering

## Pre-requisites

**Technical prerequisites:**

* Programming experience in JavaScript/Node.js, Python or similar language
* Familiarity with frameworks like React, Django, Flask
* Knowledge of databases like MongoDB, SQL
* Understanding of machine learning concepts and libraries like Scikit-learn, TensorFlow
* Experience with devops tools such as Docker, Kubernetes

**Data prerequisites:**

* Demographic and assessment data (optional)
* Cleaned and prepared dataset in CSV/JSON format

**Infrastructure prerequisites:**

* Cloud computing account on AWS/Azure/GCP for deploying services
* Version control system (GitHub/Bitbucket)
* Continuous integration/deployment (Travis/GitLab CI)
* Monitoring and logging (Datadog/ELK)

**Other prerequisites:**

* Institutional approval and support for project
* Research ethics board clearance
* Funding/budget as required
* Personnel for development, operations, research