



Mangosuthu
University of Technology

UMLAZI - KWAZULU NATAL

P.O. Box 12363 Jacobs 4026 Durban Tel: 031 907 7111 Fax: 031 907 2892

System Development

Project 2024:

System Implementation Document

Programme	Information systems 2
Group (Department)	Information and Communication Technology – Second Year
Number of group	21 groups
Type of project	System Development project
Client	MR Msane

INFORMATION SYSTEMS 2: System development PROJECT

Examiner: KM Ngcobo

Internal Moderator: Mrs F.R. Pillay

Due Date: documentation – 20/09/2024 (Professionally documented)

Instructor: KM Ngcobo

Student	Number
Khumalo BZS	22363276
Mgabhi NT	21718444
Mthethwa MS	22223731
Nxumalo NA	22323727
Mbambo N	22208482

Table of Contents

1	Introduction	5
1.1	Implemented Features:	6
1.2	Limitations:	6
1.3	Future Developments:	7
2	Implementation Environment	8
2.1	Hardware Specifications:	8
2.2	Software and Tools:	9
2.3	Programming Languages:	9
2.4	Third-Party Libraries/Frameworks:	9
3	Implementation Approach	11
3.1	Methodology:	11
3.2	Justification for Agile:	11
4	Coding and Development	14
4.1	Core Functionality	14
4.2	Modularization	15
4.3	Project Structure:	16
5	Database Implementation	18
5.1	Schema Implementation	18
5.2	Queries and Procedures	24
	Data Management: Explain how data validation, storage, and retrieval have been handled.	25
6	Testing and Debugging	28
7	Security Considerations	30
7.1	Authentication:	30
7.2	Encryption:	30
7.3	Hashing:	31
7.4	Input Validation:	31
7.5	Penetration Testing:	31
8	Deployment	32
8.1	Deployment	32
8.2	Installation/Setup Instructions	32
9	User Manual and Documentation	34

10	Future Enhancements	34
10.1	E-commerce Checkout and Payment Integration:	34
10.2	Static Files Placement:.....	35
10.3	Customer Relationship Management (CRM) Enhancements:.....	35
10.4	User Interface (UI) and User Experience (UX):	36
10.5	Internationalization and Localization:	37
10.6	Automated Testing and Continuous Integration:.....	37
11	Conclusion	37
11.1	Major Achievements	38
11.2	Challenges Encountered and Solutions.....	38

1 Introduction

Objective: Briefly describe the system's purpose and goals.

The primary objective of the system is to streamline and optimize the business operations of Best Brightness Pongola Shop by implementing a comprehensive, efficient solution. This web-based application, developed using ASP.NET, will address several key business needs critical to the shop's success.

The system will feature inventory management, ensuring real-time tracking of stock levels, automated reordering, and effective oversight of product availability. Customer management capabilities will allow for the efficient handling of customer information, enhancing personalized service and customer relationship management (CRM). Additionally, the system will incorporate sales tracking, providing detailed reports and insights into sales performance, revenue trends, and profit margins.

A significant aspect of the system is its e-commerce integration, enabling the shop to expand its business online. This will allow customers to browse products, place orders, and make payments through a user-friendly web interface. By providing a seamless connection between the physical store and online platform, the system will enhance the shop's digital presence and improve customer engagement.

Overall, the system aims to increase operational efficiency, improve decision-making through real-time data, and drive business growth by expanding the shop's reach to a broader online audience.

Scope: What parts of the system have been implemented in this phase? Highlight any limitations or future developments.

Scope of the System Implementation:

In this phase of the Best Brightness Pongola Shop system development, several critical features have been implemented to address the shop's immediate business needs. Below is a breakdown of the implemented parts, as well as current limitations and planned future developments:

1.1 Implemented Features:

Inventory Management:

The system tracks stock levels in real-time, allowing the shop to monitor inventory status.

Basic automated notifications for low stock are implemented to streamline reordering processes.

Customer Management:

A customer database has been set up, allowing the shop to store and manage customer information.

Basic customer profiles with contact details and order histories have been integrated, supporting enhanced customer service.

Sales Tracking:

The system records sales transactions, providing insights into daily sales performance.

Sales reports can be generated to track overall revenue and identify top-selling products.

E-commerce Platform (Initial Setup):

An online storefront has been integrated, allowing customers to browse products and view stock availability.

A shopping cart feature has been implemented to allow customers to select products for purchase.

1.2 Limitations:

Inventory Automation: While low-stock notifications are in place, full automation for reordering from suppliers has not yet been integrated.

Advanced Customer Relationship Management (CRM): Current customer management functionality is limited to storing basic information and order history. Advanced CRM features, such as customer segmentation and personalized recommendations, are planned for future phases.

E-commerce Features: The online payment gateway is not yet fully functional. Users can browse and add items to the cart, but online checkout and payment processing have not been integrated.

Sales Analytics: While sales tracking is functional, advanced analytics such as trend prediction and sales forecasting are not yet available.

1.3 Future Developments:

E-commerce Expansion: Integration with a secure payment gateway and the implementation of user accounts for seamless online shopping experiences.

Supplier Integration: Automation of inventory restocking, allowing for direct communication with suppliers when stock levels are low.

Enhanced CRM Tools: Adding features like customer loyalty programs, targeted promotions, and personalized marketing strategies.

Comprehensive Reporting & Analytics: Incorporating more advanced reporting tools and dashboards for deeper insights into business performance, including predictive analytics and sales trends.

This phase provides a solid foundation for the Best Brightness Pongola Shop system, with core functionalities in place. However, future phases will focus on refining these features and expanding capabilities to meet the full scope of business needs

2 Implementation Environment

Hardware Specifications: Detail the hardware used for development (e.g., servers, workstations, testing devices).

2.1 Hardware Specifications:

Development Workstations:

Processor: Intel Core i5 (or equivalent) for optimal performance during development.

RAM: 16GB DDR4 to handle the demands of compiling code, running development environments, and testing.

Storage: 512GB SSD for fast access to development files and tools.

Graphics: Integrated GPU sufficient for typical ASP.NET development tasks.

OS: Windows 10/11 for compatibility with Visual Studio and other Microsoft development tools.

Servers:

Development Server: Local server running Windows Server 2019 for hosting the ASP.NET application during development.

Testing Server: Virtual machine simulating the production environment, equipped with IIS (Internet Information Services) to mimic the live deployment.

Database Server: A dedicated server running SQL Server 2019 for managing and storing development databases.

Testing Devices:

Desktop: Various Windows-based machines for browser compatibility testing.

Software and Tools: List the development tools, programming languages, database systems, and other software used.

2.2 Software and Tools:

Development Tools:

Visual Studio 2022: Primary Integrated Development Environment (IDE) for developing the ASP.NET project.

SQL Server Management Studio (SSMS): Used for database management, querying, and performance tuning.

2.3 Programming Languages:

C#: Primary language for backend logic and ASP.NET Core MVC.

HTML5/CSS3: For designing the frontend of the web application.

JavaScript: Used for client-side scripting and interactions (along with libraries like jQuery).

T-SQL: For database interaction with SQL Server.

Database Systems:

SQL Server 2019: Used for storing inventory data, customer information, sales records, and managing transactions.

Entity Framework Core: Used as the ORM (Object-Relational Mapping) tool for interacting with the database.

Third-Party Libraries/Frameworks: Document any external libraries, frameworks, or APIs used in the project.

2.4 Third-Party Libraries/Frameworks:

Bootstrap 5: Frontend CSS framework used for responsive and mobile-first design.

jQuery: Simplifies DOM manipulation and event handling for more interactive web elements.

Entity Framework Core: ORM framework used for database operations in a more object-oriented manner.

NUnit/xUnit: For unit testing the backend code to ensure the reliability of features.

AutoMapper: Used for simplifying object-to-object mapping in business logic.

Stripe API: Planned future integration for handling secure online payments in the e-commerce module.

Version Control: Explain how the version control system (e.g., Git) was utilized during development.

3 Implementation Approach

Methodology: Describe the development methodology (e.g., Agile, Waterfall) and justify its choice for this project.

3.1 Methodology:

The development of the Best Brightness Pongola Shop system followed the Agile methodology, a flexible, iterative approach to software development.

3.2 Justification for Agile:

Flexibility and Adaptability: The scope of the project evolved based on ongoing feedback from stakeholders. Agile allowed us to continuously adapt to changing business needs and priorities.

Frequent Iterations: Development was carried out in sprints, with frequent reviews and deliverables, allowing for incremental progress and regular feedback. This ensured that key functionalities like inventory management, customer handling, and sales tracking were developed and tested early.

Early Testing: Testing and bug fixing were integrated into every sprint, reducing the likelihood of serious issues during the final stages of development.

Phases of Implementation: Break down the implementation into phases or stages (e.g., data layer development, user interface development, integration testing).

Phase 1: Requirements Gathering and Planning

Conducted discussions with stakeholders to gather detailed business requirements.

Created a project backlog, breaking down features like inventory management, customer management, and sales tracking into user stories.

Phase 2: Data Layer Development

Database Design: Developed the database schema, defining tables for products, customers, and transactions.

Entity Framework Integration: Established the data models and set up Entity Framework Core to handle database interactions.

Initial Data Seeding: Populated the database with sample data for testing and demonstration.

Phase 3: User Interface (UI) Development

Wireframing and Prototyping: Designed the basic layout of the system, focusing on user-friendliness for both the shop staff and online customers.

Frontend Implementation: Developed the web interface using ASP.NET Core MVC, integrating Bootstrap for responsive design.

Basic E-commerce Frontend: Created the initial product browsing and shopping cart pages for the e-commerce module.

Phase 4: Core Feature Development

Inventory Management: Implemented real-time stock tracking and notifications for low stock levels.

Customer Management: Built functionality for managing customer profiles and tracking purchase histories.

Sales Tracking: Added the ability to log sales transactions and generate basic sales reports.

E-commerce Integration (Initial Setup): Set up basic product browsing and cart functionality on the web portal.

Phase 5: Testing and Integration

Unit Testing: Used NUnit to test individual components and Postman to validate API endpoints.

Integration Testing: Verified that the various system components (inventory, sales, customer management) worked together without issues.

User Acceptance Testing (UAT): Conducted testing sessions with stakeholders to ensure the system met business requirements.

Phase 6: Final Refinement and Deployment Preparation

Bug Fixes and Performance Optimization: Addressed issues found during testing and optimized the system for performance.

Documentation: Prepared user manuals and technical documentation.

Deployment Preparation: Configured the system for deployment on a production server, including ensuring database backups and the setup of environment variables.

Future Phases (Planned):

Phase 7: E-commerce Checkout and Payment Integration: Future development will include integrating a secure payment gateway (e.g., Stripe) to complete the online shopping functionality.

Phase 8: Advanced Reporting & Analytics: Implement more detailed analytics features, including sales forecasting and business trend reports.

Division of Work: If applicable, describe how the work was divided among team members.

4 Coding and Development

Core Functionality: Provide an overview of the major features and functionalities implemented. This includes the backend logic, front-end interface, database integrations, etc.

Modularization: Explain how the system is broken down into modules or components and how these interact.

Code Structure: Provide an explanation of the overall code organization, including naming conventions, file structure, and the use of design patterns.

Coding and Development

4.1 Core Functionality

The system encompasses a comprehensive set of features and functionalities that include the following key components:

Backend Logic:

Sales and Inventory Management: The core logic handles CRUD (Create, Read, Update, Delete) operations for managing sales, inventory, and supplier data. It includes functionalities to add, update, delete, and retrieve data related to stock, orders, and suppliers.

Predictive Analytics: A Python-based forecasting model was integrated to analyze historical sales data and predict future inventory needs. This model runs periodically or on-demand, generating forecast data and visualizations.

User Authentication: Implements secure user authentication, allowing users to log in, manage their profiles, and access different parts of the system based on their roles (e.g., admin, staff).

API Integration: Provides APIs for interacting with the system's data, including endpoints for retrieving sales data, inventory status, and forecast results.

Front-End Interface:

User Dashboard: A dynamic user interface that displays key metrics such as current inventory levels, recent sales, and forecasted demand. Uses charts and graphs to provide a visual overview of data.

Forms and Views: Interactive forms allow users to manage stock, create orders, manage pickups, and handle supplier information. These forms are built using Razor views in ASP.NET Core.

Visualizations: Incorporates data visualizations, such as bar charts and line graphs, to display sales trends, inventory levels, and forecasting results. These visualizations are rendered dynamically using libraries like Plotly.

Database Integrations:

Database Management: Integrates with a SQL-based database (e.g., SQL Server) to store sales, inventory, orders, and supplier data. Uses Entity Framework Core for ORM (Object-Relational Mapping) to interact with the database.

Data Persistence: Ensures data integrity through transactions and leverages database migrations for schema changes.

Data Retrieval: Implements efficient queries to retrieve data for display on the front end, such as filtering and searching inventory items, generating reports, and displaying historical data.

4.2 Modularization

The system is broken down into various modules or components to promote maintainability, scalability, and separation of concerns:

Authentication Module:

Handles user login, registration, and access control.

Ensures that different user roles have access to appropriate sections of the system.

Sales and Inventory Module:

Manages stock levels, sales transactions, and order processing.

Provides APIs for creating, updating, and deleting inventory records.

Forecasting Module:

Runs the forecasting model and manages the generated forecast data.

Integrates with the front-end to display visualizations of forecasted sales and inventory needs.

Supplier Management Module:

Manages supplier information, including adding, editing, and removing suppliers.

Tracks supplier performance and provides reports related to supplier activities.

Reporting and Visualization Module:

Generates reports based on sales, inventory, and supplier data.

Provides visualizations using charts and graphs for an intuitive understanding of data trends.

Common Utilities and Services:

Includes services for logging, error handling, and utility functions used across various modules.

Ensures consistency in data handling, such as date formatting and data validation.

Code Structure

The overall code organization follows a clean and modular structure, adhering to best practices and design patterns:

4.3 Project Structure:

Controllers: Contains all the controllers for handling HTTP requests and defining actions for different routes (e.g., SalesController, InventoryController).

Models: Defines the data models and entities that represent the system's core data structures (e.g., Product, Order, Supplier). These models are used with Entity Framework for database interactions.

Views: Holds Razor views, which define the UI components and the layout of different pages in the application. Organized into folders based on the related controller (e.g., Views/Sales, Views/Inventory).

Services: Contains services that encapsulate business logic, such as InventoryService for managing stock levels and ForecastingService for handling predictive analytics.

Data: Includes the database context and data access logic. The ApplicationDbContext class manages database connections and entities using Entity Framework Core.

Static Files: wwwroot directory holds static assets like CSS, JavaScript files, images, and the forecast outputs (e.g., forecast_data.csv, forecast_plot.png).

Naming Conventions:

Classes: PascalCase (e.g., InventoryController, ProductModel).

Methods: PascalCase (e.g., GetInventory, CreateOrder).

Variables: camelCase (e.g., inventoryService, orderDetails).

Views and Folders: Organized based on functionality (e.g., Views/Sales, Views/Inventory).

Design Patterns:

MVC (Model-View-Controller): The application follows the MVC pattern, separating the application into Models (data), Views (UI), and Controllers (business logic).

Repository Pattern: Used for data access, providing a layer between the application and the database, enabling loose coupling and easier unit testing.

Dependency Injection: Services are injected into controllers to manage dependencies, allowing for better modularity and testability.

Asynchronous Programming: Implements async methods for database operations to improve performance and responsiveness, especially when handling large datasets.

By following this structured approach, the system ensures maintainability, scalability, and a clear separation of concerns, facilitating future enhancements and debugging.

Code Snippets/Examples: Include important code snippets that demonstrate critical functionality (e.g., key algorithms, database interactions, authentication mechanisms).

5 Database Implementation

Schema Implementation: Provide details about the database schema (tables, relationships, etc.) and any modifications made during implementation.

Queries and Procedures: Discuss any complex SQL queries, stored procedures, or triggers implemented in the system.

5.1 Schema Implementation

Overview: The database schema was designed to support the system's key functionalities, including managing sales data, inventory, and forecasting needs. The schema was structured to maintain data integrity, optimize performance, and ensure scalability.

Tables and Structure:

Customers

	Column Name	Data Type	Allow Nulls
🔑	CustomerId	int	<input type="checkbox"/>
	FirstName	nvarchar(50)	<input type="checkbox"/>
	LastName	nvarchar(50)	<input type="checkbox"/>
	Email	nvarchar(255)	<input type="checkbox"/>
	Password	nvarchar(100)	<input type="checkbox"/>
	RegistrationDate	datetime	<input type="checkbox"/>
	Phone	nvarchar(100)	<input checked="" type="checkbox"/>
	Address	nvarchar(200)	<input checked="" type="checkbox"/>
	DateOfBirth	datetime	<input checked="" type="checkbox"/>
	Gender	nvarchar(10)	<input checked="" type="checkbox"/>

Carts:

- 🔑 UserId (PK, nvarchar(255), not null)
- 📋 CustomerId (int, not null)
- 📋 Discount (decimal(18,2), not null)

CartItems:

- Id (PK, int, not null)
- ProductId (int, not null)
- ProductName (nvarchar(255), not null)
- ImagePath (nvarchar(255), null)
- Price (float, not null)
- Quantity (int, not null)
- CartUserId (nvarchar(255), not null)
- Total (Computed, float, not null)
- CartId (nvarchar(255), null)
- Discount (float, null)

Orders:

- OrderId (PK, int, not null)
- OrderDate (datetime2(7), not null)
- TotalAmount (float, not null)
- CustomerId (int, not null)
- ShippingAddress (nvarchar(200), not null)
- Subtotal (float, not null)
- Tax (float, not null)
- Total (float, not null)
- IsConfirmed (bit, null)
- Feedback (nvarchar(max), null)
- IsCanceled (bit, not null)
- DeliveryDate (datetime2(7), null)

OrderItem:

- OrderItemId (PK, int, not null)
- OrderId (int, not null)
- ProductId (int, not null)
- ProductName (nvarchar(max), not null)
- Price (float, not null)
- Quantity (int, not null)

Payments:

- PaymentId (PK, int, not null)
- CustomerId (int, not null)
- OrderId (int, not null)
- CardName (nvarchar(50), not null)
- CardNumber (nvarchar(16), not null)
- ExpiryDate (nvarchar(5), not null)
- CVV (nvarchar(3), not null)
- BillingAddress (nvarchar(max), not null)
- PaymentDate (datetime2(7), not null)

QuotationRequests:

- Id (PK, int, not null)
- FullName (nvarchar(max), not null)
- Email (nvarchar(max), not null)
- ProductId (int, not null)
- Quantity (int, not null)
- Message (nvarchar(max), not null)

Quotations:

- QuotationId (PK, int, not null)
- ProductDetails (nvarchar(max), not null)
- Quantity (int, not null)
- CustomerName (nvarchar(100), not null)
- CustomerEmail (nvarchar(100), not null)
- CustomerPhone (nvarchar(20), not null)
- CustomerAddress (nvarchar(200), not null)
- RequestDate (datetime, not null)
- Feedback (nvarchar(500), null)

Reviews:

- Id (PK, int, not null)
- ProductId (int, not null)
- Content (nvarchar(max), not null)
- DateSubmitted (datetime2(7), not null)
- Rating (decimal(18,2), not null)

WarehouseRegisterViewModels:

- Id (PK, int, not null)
- Name (nvarchar(max), not null)
- Email (nvarchar(max), not null)
- Password (nvarchar(max), not null)
- ConfirmPassword (nvarchar(max), null)

WarehouseLoginViewModels:

- Id (PK, nvarchar(450), not null)
- Email (nvarchar(450), not null)
- Password (nvarchar(max), not null)

WarehouseUsers:

- Id (PK, int, not null)
- Name (nvarchar(max), not null)
- Email (nvarchar(max), not null)
- PasswordHash (nvarchar(max), not null)

WarehouseProducts:

- Id (PK, nvarchar(450), not null)
- Name (nvarchar(max), not null)
- Quantity (int, not null)
- Supplier (nvarchar(max), not null)
- ReceivedDate (datetime2(7), not null)
- ImageUrl (nvarchar(max), not null)
- OriginalQuantity (int, not null)

WarehouseOrders:

- Id (PK, nvarchar(450), not null)
- ProductId (nvarchar(max), not null)
- Quantity (int, not null)
- Supplier (nvarchar(max), not null)
- OrderDate (datetime2(7), not null)
- Status (nvarchar(max), not null)

Pickups:

- Id (PK, nvarchar(450), not null)
- ProductId (nvarchar(max), not null)
- Quantity (int, not null)
- PickupDate (datetime2(7), not null)
- Status (nvarchar(max), not null)
- DeclineReason (nvarchar(max), not null)

Stocks:

- Id (PK, nvarchar(450), not null)
- Name (nvarchar(max), not null)
- OriginalQuantity (int, not null)
- Quantity (int, not null)
- Supplier (nvarchar(max), not null)
- ReceivedDate (datetime2(7), not null)
- ImageBase64 (nvarchar(max), not null)

StockItems:

- PK StockItemId (PK, int, not null)
- ProductName (nvarchar(max), not null)
- QuantityOrdered (int, not null)
- UnitPrice (decimal(18,2), not null)

Suppliers:


- PK Id (PK, nvarchar(450), not null)
- Name (nvarchar(max), not null)
- ContactInfo (nvarchar(max), not null)
- Address (nvarchar(max), not null)
- AverageDeliveryTime (float, null)
- AverageRating (float, null)
- FulfillmentAccuracy (float, null)


SupplierRatings:

- PK Id (PK, int, not null)
- SupplierId (FK, nvarchar(450), not null)
- StoremanId (nvarchar(max), not null)
- Rating (int, not null)
- Comments (nvarchar(max), not null)
- RecordedDate (datetime2(7), not null)

SupplierPerformances:

- PK Id (PK, int, not null)
- SupplierId (FK, nvarchar(450), not null)
- OrderId (int, not null)
- DeliveryTime (float, not null)
- FulfillmentAccuracy (bit, not null)
- ProductQualityRating (float, null)
- RecordedDate (datetime2(7), not null)

f4afe4e03aa06c1.Be...ss - dbo.OrderItem -> X			
	Column Name	Data Type	Allow Nulls
	OrderItemId	int	<input type="checkbox"/>
	OrderId	int	<input type="checkbox"/>
	ProductId	int	<input type="checkbox"/>
	ProductName	nvarchar(MAX)	<input type="checkbox"/>
	Price	float	<input type="checkbox"/>
	Quantity	int	<input type="checkbox"/>

f4afe4e03aa06c1.Be...ghtness - dbo.Sales -> X			
	Column Name	Data Type	Allow Nulls
	Id	int	<input type="checkbox"/>
	ProductId	int	<input type="checkbox"/>
	Quantity	int	<input type="checkbox"/>
	PricePerUnit	decimal(18, 2)	<input type="checkbox"/>
	TotalPrice	decimal(18, 2)	<input type="checkbox"/>
	SaleDate	datetime2(7)	<input type="checkbox"/>
	SalespersonId	nvarchar(MAX)	<input type="checkbox"/>
	InvoiceId	int	<input checked="" type="checkbox"/>

f4afe4e03aa06c1.Be...ess - dbo.Products -> X			
	Column Name	Data Type	Allow Nulls
	StockLevel	int	<input type="checkbox"/>
	Price	decimal(18, 2)	<input type="checkbox"/>
	LastUpdatedDate	datetime	<input type="checkbox"/>
	LastUpdatedBy	nvarchar(100)	<input checked="" type="checkbox"/>
	SalespersonId	nvarchar(MAX)	<input type="checkbox"/>
	Category	nvarchar(MAX)	<input type="checkbox"/>
	Rating	decimal(18, 2)	<input type="checkbox"/>
	ReviewCount	int	<input type="checkbox"/>
	QuotationRequestId	int	<input checked="" type="checkbox"/>
	Discount	decimal(5, 2)	<input checked="" type="checkbox"/>
	ImagePath	nvarchar(255)	<input checked="" type="checkbox"/>

f4afe4e03aa06c1.Be...ss - dbo.StockItems -> X			
	Column Name	Data Type	Allow Nulls
🔑	StockItemId	int	<input type="checkbox"/>
	ProductName	nvarchar(MAX)	<input type="checkbox"/>
	QuantityOrdered	int	<input type="checkbox"/>
	UnitPrice	decimal(18, 2)	<input type="checkbox"/>

f4afe4e03aa06c1.Be...htness - dbo.Orders -> X			
	Column Name	Data Type	Allow Nulls
🔑	OrderId	int	<input type="checkbox"/>
	OrderDate	datetime2(7)	<input type="checkbox"/>
	TotalAmount	float	<input type="checkbox"/>
	CustomerId	int	<input type="checkbox"/>
	ShippingAddress	nvarchar(200)	<input type="checkbox"/>
	Subtotal	float	<input type="checkbox"/>
	Tax	float	<input type="checkbox"/>
	Total	float	<input type="checkbox"/>
	IsConfirmed	bit	<input checked="" type="checkbox"/>
	Feedback	nvarchar(MAX)	<input checked="" type="checkbox"/>
	IsCanceled	bit	<input type="checkbox"/>
	DeliveryDate	datetime2(7)	<input checked="" type="checkbox"/>

Modifications During Implementation:

Normalization: The schema was normalized to the third normal form (3NF) to reduce data redundancy and ensure data integrity.

Indexes: Indexes were added on frequently queried fields, such as SaleDate and ProductID, to improve query performance.

Constraints: Foreign key constraints were applied to enforce referential integrity between related tables.

5.2 Queries and Procedures

Overview: The system implemented various SQL queries, stored procedures, and triggers to manage data efficiently and support the application's core functionality.

Data Management: Explain how data validation, storage, and retrieval have been handled.

Backup and Recovery: Detail the backup and recovery mechanisms implemented for the database.

Data Management

Data Validation

Data validation is crucial to ensure that the system processes only accurate and meaningful data. The following mechanisms were implemented to validate data before it is stored in the database:

Server-Side Validation:

ASP.NET Core: Implemented validation within the models and controllers to ensure data integrity.

Data Annotations: Used attributes like [Required], [Range], and [StringLength] in the model classes to validate input data. For example, the Sales model ensured UnitsSold was a positive integer.

Custom Validation: Additional checks were added in the controller actions to ensure that data like dates and quantities adhered to specific business rules.

Python Scripts: Data used for forecasting was validated using Pandas to handle missing values, incorrect data types, and outliers before analysis. For example:

Date Validation: Ensured date fields were in the correct format using `pd.to_datetime`.

Handling Missing Values: Used methods like `fillna` to address any missing values.

Client-Side Validation:

Implemented using JavaScript and HTML5 attributes for basic checks, ensuring immediate feedback for users during data entry.

ASP.NET Core's built-in validation was also used to ensure consistent validation rules on both the client and server sides.

Data Storage

Data storage was designed to be efficient, secure, and reliable, supporting the core functionalities of sales and inventory management:

Database:

SQL Server: Chosen as the primary database for its robustness and support for complex queries and stored procedures. Data related to sales, products, inventory, and suppliers was stored here.

Table Structure: Proper indexing was used to improve query performance, especially for frequently accessed tables like Sales and Inventory.

Data Consistency:

Transactions: Used SQL transactions in stored procedures to ensure atomic operations. For example, when a new sale was added, both the Sales and Inventory tables were updated in a single transaction to maintain consistency.

Foreign Key Constraints: Ensured referential integrity across related tables, such as linking ProductID in the Sales table to the Products table.

Data Retrieval

Data retrieval was optimized for performance and usability, ensuring that users can efficiently access the required information:

Data Access Layer (DAL):

Entity Framework Core (EF Core): Used as the Object-Relational Mapper (ORM) in the ASP.NET Core application to simplify data access and management.

LINQ Queries: Utilized LINQ to interact with the database, allowing for complex queries to retrieve data efficiently. For example:

Fetching sales data for a specific date range for forecasting purposes.

Retrieving low stock products to trigger reorder processes.

API Integration:

Developed APIs within the ASP.NET Core application to retrieve data for external modules, such as the Python forecasting scripts, ensuring seamless data exchange between components.

Backup and Recovery

Backup Mechanisms

To safeguard the data against loss, regular backups were implemented using the following strategies:

Automated Backups:

SQL Server Backup Jobs: Scheduled automated backups using SQL Server Agent to create full backups daily and differential backups every few hours.

Backup Storage: Backups were stored in a secure location, with retention policies ensuring recent and historical data were available if needed.

On-Demand Backups:

Provided the ability to perform manual backups before critical operations, like system updates or schema changes, to protect against accidental data loss.

Recovery Mechanisms

Recovery procedures were established to ensure data could be restored quickly in case of data loss or corruption:

Restoration Process:

SQL Server Management Studio (SSMS): Used SSMS to perform data restoration from the most recent backup file in case of a failure.

Point-in-Time Recovery: Enabled the capability to restore the database to a specific point in time, using transaction log backups to recover from data corruption or accidental deletions.

Disaster Recovery Plan:

Offsite Backups: Stored copies of backups at an offsite location to ensure data could be recovered in the event of a catastrophic failure at the primary data centre.

Testing and Verification: Regularly tested the backup and recovery processes to ensure that they were functioning correctly and data could be restored without issues.

This comprehensive approach to data management, backup, and recovery ensured that the system was robust, resilient, and capable of handling both routine operations and unexpected data-related incidents effectively.

6 Testing and Debugging

Unit Testing: Outline how individual components or functions were tested during development.

Integration Testing: Describe how the system was tested as modules were combined to ensure functionality as a whole.

User Testing/Acceptance Testing: If applicable, provide details of any end-user testing or acceptance criteria used.

Bug Tracking: Explain the bug tracking process and tools used. Provide a list of resolved issues during implementation.

Testing and Debugging

Unit Testing

Overview: Unit testing involves validating that individual components or functions of the system operate correctly in isolation. This ensures that each part of the system performs its intended function.

Testing Frameworks:

For ASP.NET Core, unit tests were conducted using frameworks such as xUnit, NUnit, or MSTest.

Python scripts and forecasting models were tested using testing frameworks like unittest or pytest.

Testing Approach:

ASP.NET Core:

Controllers were tested to ensure actions return expected results and handle various scenarios correctly.

Services were verified to confirm that business logic was implemented correctly.

Data access methods were tested to ensure proper interaction with the database.

Python:

Data preparation functions were tested for accurate data cleaning and formatting.

Forecasting models were validated to ensure they produce reliable predictions and handle different inputs appropriately.

Integration Testing

Overview: Integration testing focuses on verifying that different modules of the system work together as intended. This type of testing ensures that combined functionalities across components are correctly implemented.

Testing Approach:

ASP.NET Core:

Database interactions were tested to ensure proper execution of CRUD operations and data retrieval.

End-to-end scenarios involving multiple components were tested to validate complete workflows.

Python:

Integration tests ensured that Python scripts used for forecasting correctly interacted with the ASP.NET Core application, including data exchange and processing.

User Testing/Acceptance Testing

Overview: User testing or acceptance testing ensures the system meets the needs and expectations of end users. This involves real users interacting with the system to validate functionality and usability.

Testing Approach:

Scenario-Based Testing: Users performed specific tasks, such as generating forecasts and downloading files, to verify the system's functionality.

Feedback Collection: Collected user feedback on usability, performance, and feature accuracy.

Acceptance Criteria:

Forecast Accuracy: Ensured forecasts generated matched expected results based on historical data.

UI Functionality: Confirmed that user interface elements, such as download buttons and visualizations, functioned correctly and provided accurate data.

7 Security Considerations

Authentication and Authorization: Explain the security measures implemented for user authentication and access control.

7.1 Authentication:

ASP.NET Identity: Utilized ASP.NET Identity for user authentication, which includes support for secure login mechanisms and user account management. Users are authenticated using a combination of usernames and hashed passwords.

Password Hashing: Passwords are hashed using a secure hashing algorithm (e.g., PBKDF2) before storage in the database. This ensures that even if the database is compromised, user passwords remain secure.

Authorization:

Role-Based Access Control (RBAC): Implemented role-based authorization to restrict access to various parts of the application based on user roles (e.g., admin, manager, customer). Users are assigned roles, and permissions are enforced based on these roles.

Claims-Based Authorization: Used claims-based authorization to manage user permissions at a finer granularity. Claims are used to define user attributes and permissions, ensuring access control decisions are both flexible and secure.

Data Protection: Describe how sensitive data is protected (e.g., encryption, hashing, input validation).

7.2 Encryption:

Data at Rest: Sensitive data stored in the database is encrypted using Transparent Data Encryption (TDE) or column-level encryption in SQL Server to protect it from unauthorized access.

Encryption of Sensitive Information: Specific fields such as credit card numbers or personal identification details are encrypted before storing them in the database to add an extra layer of security.

7.3 Hashing:

Password Hashing: As mentioned, passwords are hashed using a secure algorithm to prevent plaintext storage. Techniques such as salting are used to add randomness to the hashes, mitigating the risk of rainbow table attacks.

7.4 Input Validation:

Server-Side Validation: All user inputs are validated on the server side to prevent injection attacks, such as SQL injection and cross-site scripting (XSS). Input validation ensures that only valid and expected data is processed.

Client-Side Validation: Complementary client-side validation is also implemented to provide immediate feedback to users and improve the user experience. This helps in catching errors before they reach the server.

Data Masking:

Masking Sensitive Data: Sensitive data, such as user credit card numbers, is masked in user interfaces and logs to prevent unauthorized exposure. For example, displaying only the last four digits of a credit card number

Security Testing: Provide details of any security testing performed (e.g., penetration testing, vulnerability scanning).

7.5 Penetration Testing:

Internal Penetration Testing: Performed by the development team to identify potential security issues from within the organization, ensuring that internal threats are also considered.

8 Deployment

Deployment Plan: Explain how the system was deployed or is intended to be deployed in the production environment.

Installation/Setup Instructions: Provide detailed steps for installing and configuring the system on a production server.

8.1 Deployment

Deployment Plan

The system has been successfully deployed to the production environment following a carefully planned process to ensure stability, performance, and accessibility. The deployment plan involved setting up the production server, configuring the ASP.NET Core application, and ensuring that all dependencies and services were properly integrated.

Server Setup: The production server was provisioned with the necessary hardware and software specifications to support the application. This included an environment compatible with ASP.NET Core and any additional libraries required for the predictive analytics components.

Application Deployment: The application was deployed using a CI/CD pipeline to ensure a smooth and consistent deployment process. This included building the application, running tests, and deploying it to the production server.

Static Files and Resources: All static files, including the forecast plot images and CSV files, were placed in the appropriate directories (wwwroot) to ensure they are correctly served by the application.

Configuration Management: Environment-specific configurations, such as connection strings, API keys, and logging levels, were set up using environment variables and configuration files to ensure the application behaves correctly in the production environment.

Testing and Validation: Post-deployment, the system was thoroughly tested in the production environment to ensure that all functionalities, including data forecasting, visualizations, and file downloads, were working as expected.

8.2 Installation/Setup Instructions

The system has been installed and configured on the production server following these detailed steps:

Server Preparation:

Installed the required runtime for ASP.NET Core.

Set up a web server (e.g., IIS, Nginx) to host the application.

Configured the server with the necessary security measures, including firewalls and SSL certificates for HTTPS.

Application Deployment:

Deployed the application to the server using the CI/CD pipeline. This involved pulling the latest code from the repository, building the application, and deploying it to the designated directory on the server.

Ensured that all dependencies (e.g., .NET runtime, Python environment for forecasting scripts) were installed and correctly configured.

Static Files Configuration:

Placed the static files (forecast_plot.png, forecast_data.csv) in the wwwroot folder to ensure they are accessible to the application.

Configured the web server to serve static files from the wwwroot directory.

Environment Configuration:

Configured environment-specific settings, such as database connections and file paths, using the appsettings.json file and environment variables.

Set up logging and error handling to capture and monitor any issues in the production environment.

Testing:

Verified the deployment by accessing the application in a live environment.

Tested key features, including the forecasting visualizations, CSV download functionality, and overall system performance.

Monitoring and Maintenance:

Set up monitoring tools to keep track of application performance, server health, and error logs.

Scheduled regular maintenance tasks, such as backups and updates, to ensure the system remains stable and secure.

This deployment strategy ensures the application runs smoothly in the production environment, providing a seamless user experience while maintaining security and reliability.

9 User Manual and Documentation

User Guide: Include a brief user manual explaining how the end-users can navigate and use the system.

Technical Documentation: Provide any additional technical documentation needed for maintenance, debugging, or further development.

10 Future Enhancements

Pending Features: List any features that were planned but not implemented in this phase.

10.1 E-commerce Checkout and Payment Integration:

Online Payment Processing: Integrating a secure payment gateway (e.g., Stripe, PayPal) to handle online transactions, allowing customers to make payments directly through the website.

Order Confirmation and Receipt Generation: Implementing functionality to send order confirmation emails and generate electronic receipts for completed purchases.

Advanced Reporting and Analytics:

Detailed Sales Reports: Creating advanced reports and dashboards to analyze sales trends, customer behavior, and inventory turnover.

Predictive Analytics: Implementing tools to forecast sales and inventory needs based on historical data.

Overview

Predictive analytics has been integrated into the ASP.NET Core application to forecast sales and inventory needs based on historical data. The following outlines the process and the current setup.

Data Preparation and Forecasting

Historical Data: Historical sales and inventory data was collected, cleaned, and formatted for analysis.

Forecasting Tools: A Python-based forecasting tool (e.g., Prophet) was used to analyze the data and generate forecasts.

Outputs: Forecast results and visualizations were saved as forecast_data.csv and forecast_plot.png.

Integration into ASP.NET Core

10.2 Static Files Placement:

Files Added: The generated CSV file (forecast_data.csv) and the plot image (forecast_plot.png) were placed in the wwwroot folder of the ASP.NET Core project to make them accessible as static files.

Controller Configuration:

Download Action: A controller action was implemented to handle requests for downloading the forecast CSV file. This action reads the file from the wwwroot directory and serves it to the user.

Example Action:

The action ensures the file exists before serving it and handles cases where the file might not be found.

View Updates:

Visualization Display: The view was updated to display the forecast plot image by linking to the image file stored in the wwwroot folder.

Download Button: A button was added to allow users to download the CSV file. The button links to the download action in the controller.

Testing and Validation:

File Access: The files were tested to ensure they are correctly served and accessible.

Functionality Checks: The download button and visualization display were tested to confirm they function as expected.

Key Features

Forecast Visualization: Users can view the forecast plot image directly within the application.

CSV Download: Users can download the forecast data in CSV format for further analysis.

10.3 Customer Relationship Management (CRM) Enhancements:

Loyalty Programs: Developing features for customer loyalty programs, including reward points and discounts for repeat customers.

Personalized Marketing: Implementing personalized marketing campaigns based on customer purchase history and preferences.

Supplier Integration:

Automated Reordering: Setting up integration with suppliers for automated reordering based on inventory levels and sales forecasts.

Supplier Management: Adding functionality to manage supplier information, track orders, and handle supplier performance.

Enhanced Security Features:

Two-Factor Authentication (2FA): Implementing 2FA for user accounts to provide an additional layer of security.

Role-Based Access Control (RBAC) Enhancements: Expanding the RBAC system to include more granular permissions and roles.

Improvements: Suggest areas of the system that could be improved in future iterations, identified limitations.

10.4 User Interface (UI) and User Experience (UX):

Design Enhancements: Improving the aesthetics and usability of the web application based on user feedback and usability testing.

Accessibility: Ensuring the application meets accessibility standards (e.g., WCAG) to provide a better experience for users with disabilities.

Performance Optimization:

Load Times: Optimizing server response times and reducing load times for web pages to improve user experience.

Database Queries: Analyzing and optimizing database queries to enhance performance and reduce latency.

Scalability:

Infrastructure Scaling: Planning for infrastructure scaling to handle increased traffic and data volume as the business grows.

Microservices Architecture: Considering a move to microservices architecture to improve scalability and maintainability.

10.5 Internationalization and Localization:

Multi-language Support: Adding support for multiple languages to cater to a diverse customer base.

Currency and Regional Settings: Implementing support for different currencies and regional settings for international customers.

Integration with Third-Party Services:

Social Media Integration: Adding features to integrate with social media platforms for marketing and customer engagement.

Shipping and Logistics: Integrating with shipping and logistics providers to streamline order fulfillment and tracking.

10.6 Automated Testing and Continuous Integration:

Automated Testing: Implementing automated testing frameworks to improve code quality and reduce the risk of bugs.

Continuous Integration/Continuous Deployment (CI/CD): Setting up CI/CD pipelines to automate the build, test, and deployment processes.

11 Conclusion

Summarize the implementation phase, highlighting the major achievements, challenges encountered, and how they were addressed.

11.1 Major Achievements

Comprehensive Sales and Inventory Management: The system effectively manages sales, inventory, orders, and supplier information. It provides a seamless interface for users to perform essential tasks like updating stock, creating orders, and managing suppliers, ensuring efficient business operations.

Predictive Analytics Integration: One of the standout achievements is the integration of predictive analytics using historical sales data. By leveraging a forecasting model, the system can predict future sales trends and inventory needs, helping businesses make informed decisions and optimize stock levels.

Dynamic Visualizations: The implementation of data visualizations, such as line graphs, bar charts, and forecast plots, offers an intuitive way for users to understand complex data trends. This feature greatly enhances the usability of the system, allowing for a clear representation of sales and inventory patterns.

User Authentication and Security: The system includes robust user authentication and role-based access control, ensuring that sensitive data is protected and only accessible to authorized users. This security layer enhances the reliability and trustworthiness of the application.

Seamless Deployment and Integration: The system was successfully deployed to a production environment with all necessary configurations. The integration of Python-based forecasting within the ASP.NET Core application was a significant achievement, showcasing the flexibility and power of combining different technologies.

Data Export and File Management: The ability to export forecast results and other data to CSV files provides users with an easy way to download and analyze data offline. The implementation of file management, including saving forecast plots and data in the server's file system, adds to the system's practical utility.

11.2 Challenges Encountered and Solutions

Data Integration and Cleaning: Integrating historical sales data for predictive analytics required careful data cleaning and transformation. Initially, some datasets lacked proper date formats and had inconsistent values, leading to errors during processing. This was addressed by implementing data preprocessing steps, including handling missing values, correcting date formats, and ensuring data consistency.

Forecasting Model Integration: Incorporating the Python-based forecasting model into the ASP.NET Core environment posed a technical challenge due to differences in runtime environments. This was overcome by using the matplotlib library for generating plots and saving them as static images in the wwwroot directory. Additionally, the use of Python scripts was managed through command-line execution within the .NET environment.

Visualizations: Implementing dynamic and interactive visualizations required careful consideration of various JavaScript libraries and their integration with Razor views. To address this, libraries like Chart.js were used to render visualizations on the client side, ensuring compatibility with the ASP.NET Core framework and providing an intuitive user experience.

File Management and Download Functionality: Ensuring that users could download forecast results and visualizations involved configuring the system to handle file storage and retrieval properly. This required managing file paths, serving static files, and providing download links in the user interface. These functionalities were addressed by correctly setting up the wwwroot folder and using controller actions to handle file downloads.

Deployment Configuration: Deploying the system to a production server involved setting up the server environment, configuring static files, and managing security concerns like HTTPS and CORS policies. These challenges were addressed by following best practices for ASP.NET Core deployment, including configuring the web server, setting environment variables, and ensuring proper SSL configurations for secure communications.

Summary

The implementation phase of this system was marked by significant accomplishments in developing a robust sales and inventory management solution with integrated predictive analytics. The combination of an intuitive user interface, data visualizations, secure authentication, and seamless integration of forecasting tools provides a comprehensive solution for businesses. While challenges such as data integration, model implementation, and deployment complexities were encountered, they were effectively addressed through careful planning, modular design, and the use of appropriate technologies.

Overall, the successful completion of this phase has resulted in a powerful and versatile application capable of meeting the dynamic needs of inventory management and sales forecasting.