Software Requirements Specification (SRS) - MicroBiz Hub

1. Introduction

1.1. Purpose

The purpose of this document is to specify the functional and non-functional requirements for "MicroBiz Hub," a simple Enterprise Resource Planning (ERP) web application. This system is designed to provide Small and Medium Enterprises (SMEs) with essential tools for managing inventory, sales orders, basic invoicing, and user access, centralizing operations that are often fragmented across manual processes or disparate tools.

1.2. Scope

"MicroBiz Hub" will initially focus on providing core functionalities in the following modules:

- **Inventory Management:** Product catalog, stock tracking, and basic stock adjustments.
- **Sales Order & Basic Invoicing:** Customer management, sales order creation, and simple invoice generation.
- **User Management & Authentication:** Secure user registration, login, and role-based access control (RBAC).

Future enhancements, not covered in this initial scope, include comprehensive financial accounting, detailed reporting, advanced CRM, supply chain management, and HR modules.

1.3. Definitions, Acronyms, and Abbreviations

- **SME:** Small and Medium Enterprise
- **ERP:** Enterprise Resource Planning
- **CRUD:** Create, Read, Update, Delete
- **API:** Application Programming Interface
- **RBAC:** Role-Based Access Control
- **HTTP:** Hypertext Transfer Protocol
- **JWT:** JSON Web Token
- **SQL Server:** Microsoft SQL Server
- **ASP.NET Core 8:** Microsoft's open-source, cross-platform framework for building modern, cloud-based, internet-connected applications.
- **Angular 18:** A platform and framework for building single-page client applications using HTML and TypeScript.

- **EF Core:** Entity Framework Core, an object-relational mapper for .NET.

1.4. References

- ASP.NET Core Documentation: https://docs.microsoft.com/en-us/aspnet/core/
- Angular Documentation: https://angular.io/docs
- SQL Server Documentation: https://docs.microsoft.com/en-us/sql/sql-server/

1.5. Overview

The remainder of this document outlines the overall description of MicroBiz Hub, including its general capabilities, user characteristics, and constraints. Following this, detailed specific requirements for each module are provided, including functional requirements, data models, and implementation strategies for the backend (ASP.NET Core 8) and frontend (Angular 18).

2. Overall Description

2.1. Product Perspective

MicroBiz Hub is a standalone web application, designed to be deployed as a full-stack solution. The frontend (client-side) will be developed using Angular 18, communicating with a backend (server-side) API developed using ASP.NET Core 8. Data persistence will be managed by SQL Server.

2.2. Product Functions

The system will provide the following primary functions:

- Secure user authentication and authorization.
- Management of product inventory (creation, viewing, updating, deleting products and their stock).
- Management of customer information.
- Creation and tracking of sales orders.
- Generation of basic invoices for sales orders.

2.3. User Characteristics

The system is intended for use by SME employees and owners, including:

- **Administrators:** Full system access, including user management.
- **Sales Users:** Access to customer and sales order management.
- **Inventory Users:** Access to product and inventory management.
- **General Users:** Can view limited information (e.g., their own profile).

Users are expected to have basic computer literacy and familiarity with web

applications.

2.4. Constraints

- **Technology Stack:** ASP.NET Core 8 for backend, Angular 18 for frontend, SQL Server for database.
- **Security:** Role-Based Access Control (RBAC) must be implemented to restrict access to functionalities based on user roles.
- **Performance:** The system should be reasonably responsive for typical SME operations (e.g., fast loading times for lists, quick form submissions).
- **Browser Compatibility:** Must be compatible with modern web browsers (Chrome, Firefox, Edge, Safari).

2.5. Assumptions and Dependencies

- Users will have stable internet connectivity to access the web application.
- The application will run on a server environment capable of hosting ASP.NET Core applications and SQL Server databases.
- Frontend assets will be served by the ASP.NET Core application or a separate web server.

3. Specific Requirements

3.1. Module: User Management & Authentication

3.1.1. Functional Requirements

- **FR.UM.1: User Registration:**
  - Users (initially only Admins) shall be able to register new accounts with a unique email and password.
  - Passwords shall meet minimum complexity requirements (e.g., minimum length, mix of characters).
- **FR.UM.2: User Login:**
  - Registered users shall be able to log in using their email and password.
  - Upon successful login, the system shall provide a JWT for authenticated API access.
- **FR.UM.3: Role Assignment:**
  - Administrators shall be able to assign predefined roles (Admin, SalesUser, InventoryUser) to users.
  - A user can have multiple roles.
- **FR.UM.4: User Listing:**
  - Administrators shall be able to view a list of all registered users with their associated roles.

- **FR.UM.5: User Deactivation (Soft Delete):**
  - Administrators shall be able to deactivate a user account, preventing them from logging in, without deleting their data.
- **FR.UM.6: User Profile:**
  - Authenticated users shall be able to view their own basic profile details.
- **FR.UM.7: Logout:**
  - Authenticated users shall be able to log out of the system.

3.1.2. Data Model (SQL Server)

- **AspNetUsers Table:** (Managed by ASP.NET Core Identity)
  - Id (PK, GUID)
  - Email (Unique)
  - PasswordHash
  - UserName
  - IsActive (BIT, for soft delete)
  - ... (other standard Identity columns)
- **AspNetRoles Table:** (Managed by ASP.NET Core Identity)
  - Id (PK, GUID)
  - Name (Unique, e.g., 'Admin', 'SalesUser', 'InventoryUser')
- **AspNetUserRoles Table:** (Managed by ASP.NET Core Identity, Many-to-Many relationship)
  - UserId (FK to AspNetUsers.Id)
  - RoleId (FK to AspNetRoles.Id)

3.1.3. Implementation Strategy

- **Backend (ASP.NET Core 8):**
  - **Authentication:** Implement JWT Bearer Token authentication.
  - **Authorization:** Use [Authorize] attributes on controllers/actions and [Authorize(Roles = "RoleName")] for role-based access.
  - **ASP.NET Core Identity:** Utilize Identity for user and role management. Configure it with EF Core for SQL Server.
  - **API Endpoints:**
    - POST /api/auth/register: For admin to register new users (initially).
    - POST /api/auth/login: Authenticate user, return JWT.
    - GET /api/users: Get list of users (Admin only).
    - PUT /api/users/{id}/roles: Update user roles (Admin only).
    - PUT /api/users/{id}/deactivate: Deactivate user (Admin only).
    - GET /api/users/profile: Get current user's profile.
  - **Services:** UserService to encapsulate business logic for user management.

    - **DTOs:** LoginDto, RegisterDto, UserDto, UserRoleUpdateDto.
- **Frontend (Angular 18):**
    - **AuthService:** Handle login/logout, store/retrieve JWT, manage user authentication state.
    - **AuthGuard:** Protect routes based on authentication status and user roles.
    - **Components:**
        - LoginComponent: Form for user login.
        - RegisterComponent: Form for registering new users (Admin-only access to this component).
        - UserListComponent: Display table of users, actions for roles/deactivation (Admin only).
        - UserProfileComponent: Display current user details.
    - **Routing:** Set up routes for login, dashboard, user management, etc., with appropriate guards.
    - **Forms:** Reactive Forms for login and user registration/updates, including validation.

3.2. Module: Inventory Management

3.2.1. Functional Requirements

- **FR.INV.1: Product Creation:**
    - Users with InventoryUser or Admin role shall be able to add new products.
    - Required fields: Name, SKU (unique), Unit Price, Cost Price, Current Stock (initial value).
    - Optional fields: Description, Minimum Stock.
- **FR.INV.2: Product Listing:**
    - Users with InventoryUser or Admin role shall be able to view a list of all products.
    - The list shall display Name, SKU, Current Stock, Unit Price.
    - The list shall highlight products where Current Stock <= Minimum Stock (if Minimum Stock is defined).
    - Shall support basic search by Name/SKU and sorting.
- **FR.INV.3: Product Details & Update:**
    - Users with InventoryUser or Admin role shall be able to view detailed information for a specific product.
    - Shall be able to update product details (Name, Description, Unit Price, Cost Price, Minimum Stock).
- **FR.INV.4: Product Deletion:**
    - Users with InventoryUser or Admin role shall be able to delete a product (logical delete recommended, but simple hard delete is fine for minimal

scope).

- **FR.INV.5: Stock Adjustment:**
  - Users with InventoryUser or Admin role shall be able to manually adjust (add or deduct) Current Stock for a product.
  - Each adjustment must include a numeric quantity and a reason (e.g., "Received shipment," "Damaged goods," "Inventory count adjustment"). This implies an StockAdjustment entity.

3.2.2. Data Model (SQL Server)

- **Products Table:**
  - Id (PK, INT/GUID)
  - Name (NVARCHAR(255), NOT NULL)
  - SKU (NVARCHAR(50), UNIQUE, NOT NULL)
  - Description (NVARCHAR(MAX), NULL)
  - UnitPrice (DECIMAL(18, 2), NOT NULL)
  - CostPrice (DECIMAL(18, 2), NOT NULL)
  - CurrentStock (INT, NOT NULL, DEFAULT 0)
  - MinimumStock (INT, NULL)
  - IsDeleted (BIT, NOT NULL, DEFAULT 0, for logical delete)
  - CreatedAt (DATETIME2, NOT NULL)
  - UpdatedAt (DATETIME2, NOT NULL)
- **StockAdjustments Table:**
  - Id (PK, INT/GUID)
  - ProductId (FK to Products.Id, NOT NULL)
  - AdjustmentQuantity (INT, NOT NULL) -- Positive for add, negative for deduct
  - Reason (NVARCHAR(255), NOT NULL)
  - AdjustedByUserId (FK to AspNetUsers.Id, NOT NULL)
  - AdjustedAt (DATETIME2, NOT NULL)

3.2.3. Implementation Strategy

- **Backend (ASP.NET Core 8):**
  - **Entities:** C# classes Product, StockAdjustment.
  - **ApplicationDbContext:** Add DbSet<Product>, DbSet<StockAdjustment>.
  - **Services:** ProductService (encapsulates product CRUD, stock updates) and StockAdjustmentService. Business logic for CurrentStock updates should reside in ProductService triggered by StockAdjustmentService or SalesOrderService.
  - **API Endpoints:**
    - GET /api/products: Get all products (with optional search/filter params).

        - GET /api/products/{id}: Get single product.
        - POST /api/products: Create new product.
        - PUT /api/products/{id}: Update product.
        - DELETE /api/products/{id}: Delete product (or mark as deleted).
        - POST /api/products/{id}/adjust-stock: For stock adjustments.
    - **DTOs:** ProductDto, ProductCreateDto, ProductUpdateDto, StockAdjustmentDto.
    - **Authorization:** [Authorize(Roles = "Admin,InventoryUser")] on relevant controllers/actions.
- **Frontend (Angular 18):**
    - **ProductService:** Angular service for HTTP calls to product API endpoints.
    - **Components:**
        - ProductListComponent: Displays product table, search, sorting. Uses *ngFor. Highlights low stock.
        - ProductFormComponent: Reactive form for creating/editing products.
        - ProductDetailComponent: Displays single product details.
        - StockAdjustmentModalComponent: A modal form for recording stock adjustments (opened from ProductListComponent or ProductDetailComponent).
    - **Routing:** /products, /products/new, /products/:id.
    - **Forms:** Reactive Forms with validation for product and stock adjustment inputs.

3.3. Module: Sales Order & Basic Invoicing

3.3.1. Functional Requirements

- **FR.SALES.1: Customer Creation:**
    - Users with SalesUser or Admin role shall be able to add new customers.
    - Required fields: Name, Email.
    - Optional fields: Phone, Address.
- **FR.SALES.2: Customer Listing:**
    - Users with SalesUser or Admin role shall be able to view a list of all customers.
    - Shall support basic search by Name/Email and sorting.
- **FR.SALES.3: Customer Details & Update:**
    - Users with SalesUser or Admin role shall be able to view and update customer details.
- **FR.SALES.4: Sales Order Creation:**
    - Users with SalesUser or Admin role shall be able to create new sales orders.
    - Each sales order must be linked to an existing customer.
    - Users shall be able to add multiple line items, each referencing a product and

quantity.
    - The system shall validate if sufficient stock is available for each product. If not, the order cannot be created, or a warning is issued.
    - Upon successful creation, the CurrentStock of affected products shall be reduced.
    - The system shall automatically calculate the line item total and the overall order total.
- **FR.SALES.5: Sales Order Listing:**
    - Users with SalesUser or Admin role shall be able to view a list of all sales orders.
    - The list shall display Order ID, Customer Name, Order Date, Total Amount, and Status.
    - Shall support basic search by Order ID/Customer Name and sorting.
- **FR.SALES.6: Sales Order Details & Status Update:**
    - Users with SalesUser or Admin role shall be able to view detailed information for a specific sales order, including all line items.
    - Shall be able to update the order status (e.g., "New," "Processing," "Completed," "Cancelled").
    - If an order is "Cancelled," the stock deduction should be reversed.
- **FR.SALES.7: Basic Invoice Generation (View Only):**
    - For sales orders with a "Completed" status, users with SalesUser or Admin role shall be able to view a simple invoice representation on screen.
    - The invoice shall include: Customer details, Order ID, Order Date, list of products (Name, Quantity, Unit Price, Line Total), and Grand Total. (No PDF generation for minimal scope).

3.3.2. Data Model (SQL Server)

- **Customers Table:**
    - Id (PK, INT/GUID)
    - Name (NVARCHAR(255), NOT NULL)
    - Email (NVARCHAR(255), NULL)
    - Phone (NVARCHAR(50), NULL)
    - Address (NVARCHAR(500), NULL)
    - IsDeleted (BIT, NOT NULL, DEFAULT 0)
    - CreatedAt (DATETIME2, NOT NULL)
    - UpdatedAt (DATETIME2, NOT NULL)
- **SalesOrders Table:**
    - Id (PK, INT/GUID)
    - CustomerId (FK to Customers.Id, NOT NULL)

    - OrderDate (DATETIME2, NOT NULL)
    - Status (NVARCHAR(50), NOT NULL, e.g., 'New', 'Processing', 'Completed', 'Cancelled')
    - TotalAmount (DECIMAL(18, 2), NOT NULL)
    - OrderedByUserId (FK to AspNetUsers.Id, NOT NULL)
    - CreatedAt (DATETIME2, NOT NULL)
    - UpdatedAt (DATETIME2, NOT NULL)
- **OrderItems Table:** (Line items for SalesOrder)
    - Id (PK, INT/GUID)
    - SalesOrderId (FK to SalesOrders.Id, NOT NULL)
    - ProductId (FK to Products.Id, NOT NULL)
    - Quantity (INT, NOT NULL)
    - UnitPriceAtTimeOfOrder (DECIMAL(18, 2), NOT NULL) -- Store price at order time to prevent future price changes affecting old orders.
    - LineTotal (DECIMAL(18, 2), NOT NULL)

3.3.3. Implementation Strategy

- **Backend (ASP.NET Core 8):**
    - **Entities:** C# classes Customer, SalesOrder, OrderItem.
    - **ApplicationDbContext:** Add DbSet<Customer>, DbSet<SalesOrder>, DbSet<OrderItem>.
    - **Services:** CustomerService, SalesOrderService. SalesOrderService will contain complex logic:
        - Stock validation and deduction/reversion upon order creation/cancellation.
        - Calculation of order totals.
    - **API Endpoints:**
        - GET /api/customers: Get all customers.
        - GET /api/customers/{id}: Get single customer.
        - POST /api/customers: Create new customer.
        - PUT /api/customers/{id}: Update customer.
        - GET /api/salesorders: Get all sales orders.
        - GET /api/salesorders/{id}: Get single sales order details (including items).
        - POST /api/salesorders: Create new sales order.
        - PUT /api/salesorders/{id}/status: Update sales order status.
        - GET /api/salesorders/{id}/invoice: Get invoice details for a sales order.
    - **DTOs:** CustomerDto, CustomerCreateDto, CustomerUpdateDto, SalesOrderDto, SalesOrderCreateDto (with nested OrderItemCreateDto), SalesOrderStatusUpdateDto, InvoiceDto.
    - **Authorization:** [Authorize(Roles = "Admin,SalesUser")] on relevant

controllers/actions.

- **Frontend (Angular 18):**
  - **CustomerService:** Angular service for HTTP calls to customer API endpoints.
  - **SalesOrderService:** Angular service for HTTP calls to sales order/invoice API endpoints.
  - **Components:**
    - CustomerListComponent: Displays customer table, search.
    - CustomerFormComponent: Reactive form for creating/editing customers.
    - SalesOrderListComponent: Displays sales order table, search, status.
    - SalesOrderFormComponent: Complex reactive form for creating sales orders (including dynamic adding/removing of product line items, product lookup, stock validation display).
    - SalesOrderDetailComponent: Displays sales order details.
    - InvoiceViewComponent: Displays the basic invoice for a completed sales order.
  - **Routing:** /customers, /customers/new, /customers/:id, /salesorders, /salesorders/new, /salesorders/:id, /salesorders/:id/invoice.
  - **Forms:** Reactive Forms with validation for customer and sales order inputs. Implement dynamic form arrays for order items.

4. Non-Functional Requirements

4.1. Security

- **NFR.SEC.1:** All communication between frontend and backend shall be secured using HTTPS.
- **NFR.SEC.2:** User passwords shall be hashed and salted before storage.
- **NFR.SEC.3:** All API endpoints requiring authentication shall enforce it using JWT tokens.
- **NFR.SEC.4:** Role-based access control shall be strictly enforced on the backend to prevent unauthorized access to data or functionalities.

4.2. Performance

- **NFR.PERF.1:** List pages (e.g., Products, Sales Orders, Customers) should load within 2 seconds for up to 1000 records.
- **NFR.PERF.2:** Form submissions (e.g., creating a product, sales order) should complete within 1.5 seconds.

4.3. Usability

- **NFR.US.1:** The user interface shall be intuitive and easy to navigate for users with basic computer literacy.

- **NFR.US.2:** Error messages shall be clear, concise, and helpful to the user.
- **NFR.US.3:** The application shall provide visual feedback for user actions (e.g., loading indicators, success/error toasts).

4.4. Maintainability

- **NFR.MAINT.1:** Codebase shall follow clear architectural patterns (e.g., Clean Architecture principles for backend, modular Angular structure).
- **NFR.MAINT.2:** Code shall be well-commented and conform to industry-standard coding practices.
- **NFR.MAINT.3:** Dependency injection shall be used consistently throughout the application.

5. Future Enhancements (Beyond Initial Scope)

- Comprehensive reporting with filtering and export to PDF/Excel.
- Dashboard visualizations (charts for sales trends, inventory value).
- Purchase Management module (tracking purchases from suppliers).
- Basic Accounting (Journal Entries, Ledger, Payment Reconciliation).
- User notifications (e.g., email alerts for low stock).
- Integrations with payment gateways (mock initially).
- Advanced search and filtering capabilities.
- Audit logs for all major data modifications.
- Soft deletion for all entities, with an admin recovery option.
- Customizable dashboards.

This SRS provides a solid foundation for your "MicroBiz Hub" project. By focusing on these core, minimal features, you can build a functional and impressive application that showcases your full-stack capabilities with ASP.NET Core 8 and Angular 18.