**Dealer Management System**

Table of Contents

[1. Introduction 3](#_Toc193318786)

[2. System Overview 3](#_Toc193318787)

[3. High-Level Architecture 3](#_Toc193318788)

[4. Component Interaction 5](#_Toc193318789)

[4.1 User (Browser) 5](#_Toc193318790)

[4.2 Razor Pages (Frontend) 5](#_Toc193318791)

[4.3 ASP.NET Core API (Backend) 6](#_Toc193318792)

[4.4 Azure Active Directory (Azure AD) 6](#_Toc193318793)

[4.5 SQL Database 7](#_Toc193318794)

[4.6 Azure Blob Storage 7](#_Toc193318795)

[4.7 External APIs / 3rd Party Services 7](#_Toc193318796)

[5. Security Considerations 7](#_Toc193318797)

[6.Tech Stack Selection and Justification 8](#_Toc193318798)

[6.1. .NET Core 8](#_Toc193318799)

[6.2. Razor Pages 9](#_Toc193318800)

[6.3. Azure Active Directory (Azure AD) 9](#_Toc193318801)

[6.4. Azure SQL Database 9](#_Toc193318802)

[6.5. Azure Blob Storage 10](#_Toc193318803)

[6.6. API Security: OAuth 2.0 & JWT 10](#_Toc193318804)

[7. Deployment Playbook and Infrastructure as code scripts 10](#_Toc193318805)

[7.1. Infrastructure Setup 10](#_Toc193318806)

[7.2. Application Deployment 10](#_Toc193318807)

[7.3. CI/CD Execution 10](#_Toc193318808)

[7.4. Post-Deployment Checks 11](#_Toc193318809)

[8. Cloud vs. On-Premises Deployment Considerations 11](#_Toc193318810)

[8.1. Cloud Deployment (Azure) 11](#_Toc193318811)

[8.2. On-Premises Deployment 11](#_Toc193318812)

[8.3. Hybrid Approach 12](#_Toc193318813)

[9. Multi-Region and High Availability Deployment Strategy 12](#_Toc193318814)

[9.1 Multi-Region Deployment 12](#_Toc193318815)

[9.2. High Availability Strategy 12](#_Toc193318816)

# Introduction

The goal is to design a system that allows dealers to manage user management, inventory, sales, and other business processes, with secure authentication and authorization mechanisms through SSO and MFA.

# System Overview

The Dealer Management System is a web-based application that enables dealerships to manage their day-to-day operations. The system will allow users (dealers, admins, and support staff) to access functionalities based on their roles. It will leverage Azure AD for user identity management and ensure secure access through SSO and MFA.

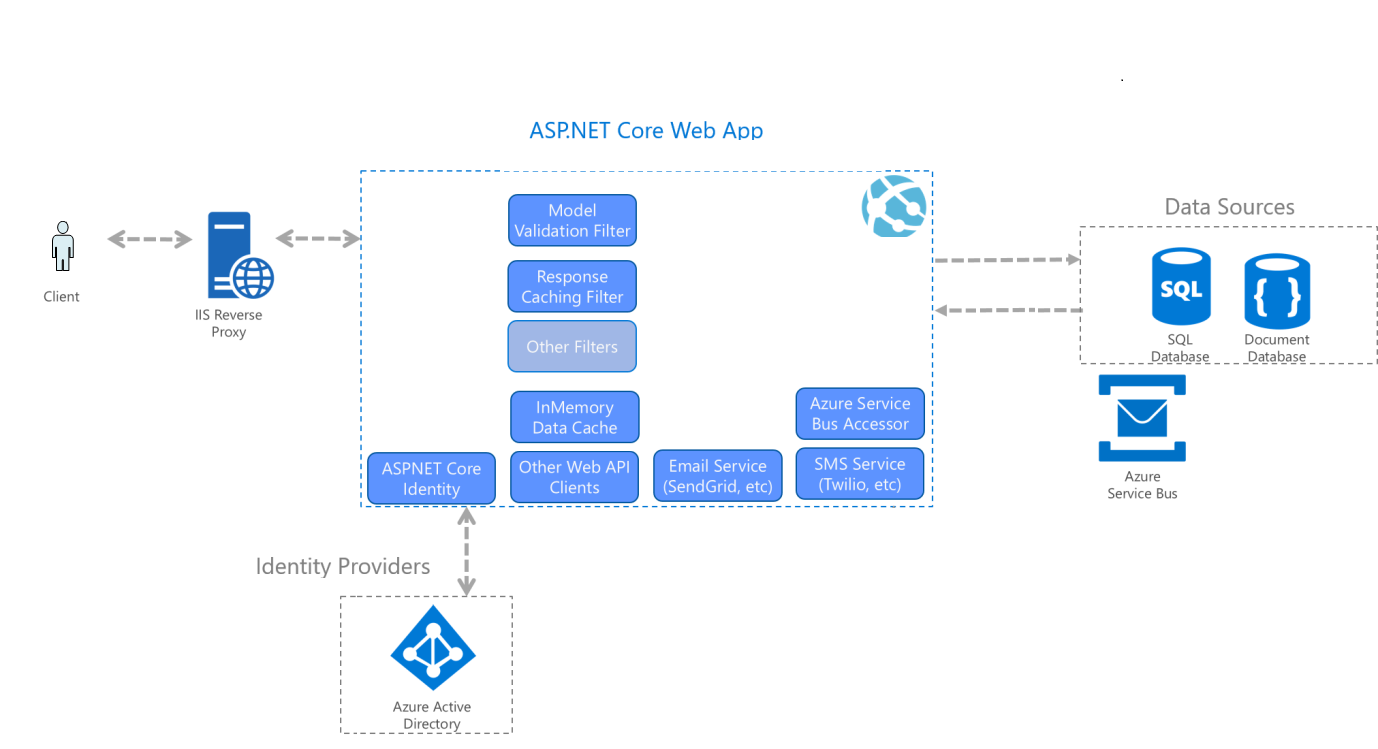
Key features of the DMS:

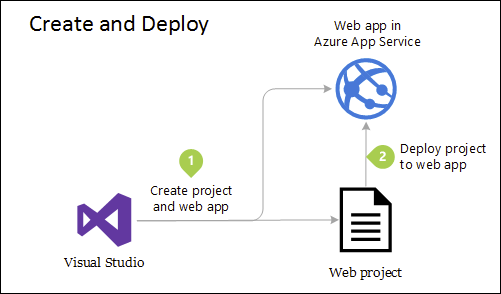
* User authentication and authorization via Azure AD (SSO & MFA).
* Dealer and customer management.
* Inventory management.
* Parts and Supply management.
* Finance and Accounting.
* Order management.
* Reporting and analytics.

# 3. High-Level Architecture

The architecture of the DMS consists of several layers and components:

[User (Browser)] <-> [Razor Pages (Frontend)] <-> [ASP.NET Core API (Backend)] <-> [Azure Active Directory (Identity Management)] <-> [Database (SQL/NoSQL)] <-> [External APIs / 3rd Party Services] <-> [Azure Blob Storage (Document Storage)]





Core Components:

* **Frontend (Razor Pages)**: UI layer for interacting with users.
* **Backend (ASP.NET Core API)**: Business logic layer handling user requests and interactions.
* **Azure Active Directory (Azure AD)**: Manages authentication (SSO & MFA), user roles, and permissions.
* **SQL Database**: Stores business data, including dealer profiles, inventory, customer information, orders, etc.
* **Azure Blob Storage**: Stores documents and images (e.g., inventory pictures, invoices, contracts).
* **External APIs/3rd Party Services**: Integration with external services for payments, inventory data, etc.

# 4. Component Interaction

# 4.1 User (Browser)

* **Role**: The user (e.g., a dealer, admin, or support staff) accesses the system using a browser.
* **Interaction with Razor Pages**: The user interacts with the frontend UI to view and manage their data (inventory, orders, customers).
* **Authentication Flow**: When accessing the system, the user is redirected to Azure AD for authentication via **SSO** and **MFA**.
  1. **Login Initiation**: The user tries to access the DMS via the browser. If the user isn't logged in, they are redirected to **Azure AD**.
  2. **SSO & MFA**: Azure AD handles the **Single Sign-On** (SSO) and **Multi-Factor Authentication** (MFA) processes, ensuring the user identity is validated before granting access.
  3. **User Session**: Once authenticated, Azure AD issues a token (JWT) that is used for subsequent requests. The token is stored in the browser (via cookies or local storage).

# 4.2 Razor Pages (Frontend)

* **Role**: Razor Pages is the user interface layer where dealers and users interact with the DMS.
* **Interaction with Azure AD**: Upon receiving a valid authentication token from Azure AD, Razor Pages ensures the user's session is active and that they have access to the appropriate resources based on their roles.
* **Interaction with ASP.NET Core API**: The frontend interacts with the backend via API calls to fetch and manage data such as inventory, customer profiles, orders, etc.
  + The frontend makes REST API calls to the backend to retrieve data and display it in the UI.
  + API calls are secured with the JWT token from Azure AD, ensuring that only authorized users can access specific resources.

# 4.3 ASP.NET Core API (Backend)

* **Role**: The backend is responsible for handling the business logic of the Dealer Management System.
* **Interaction with Azure AD**: The backend uses Azure AD for **OAuth 2.0** authorization to validate user tokens (JWT). Every API request contains an authorization header with the JWT token.
* **API Security**: The API is protected using **Azure AD** authentication. It validates the token to ensure that the request is coming from an authenticated user.
* **Business Logic**: The backend processes requests, such as:
  + CRUD operations on dealer data (inventory, customers, orders).
  + Business rules such as inventory checks, order processing, etc.
* **Interaction with Database**: The backend communicates with the database (SQL/NoSQL) to retrieve, update, and delete data. This could involve querying tables or collections for data like customer profiles, inventory details, and transactions.

# 4.4 Azure Active Directory (Azure AD)

* **Role**: Azure AD provides the identity management platform for the DMS system. It handles the authentication (SSO & MFA) and authorization of users.
* **Authentication**: Azure AD authenticates users based on their credentials (username, password, MFA) and provides access tokens (JWT) to allow users to access resources securely.
* **Role-Based Access Control (RBAC)**: Azure AD allows for role-based access control, enabling the system to assign specific roles to users (e.g., dealer, admin, support staff) and restrict access based on those roles.
* **SSO & MFA**: Azure AD facilitates **Single Sign-On** for users across different applications and **Multi-Factor Authentication** to ensure secure access.

# 4.5 SQL Database

* **Role**: The database stores the core data for the Dealer Management System.
  + **SQL Database**: For relational data (e.g., dealer profiles, customer information, orders).
* **Interaction**: The backend (ASP.NET Core API) communicates with the database to persist and retrieve data necessary for managing dealers, customers, and inventory.

# 4.6 Azure Blob Storage

* **Role**: Azure Blob Storage is used to store unstructured data such as images, documents, contracts, or receipts related to inventory or orders.
* **Interaction**:
  + Users upload documents or images via Razor Pages, which are then stored in Azure Blob Storage.
  + The backend API may also retrieve documents from Azure Blob Storage when requested by users (e.g., displaying inventory images or order receipts).

# 4.7 External APIs / 3rd Party Services

* **Role**: Integration with external systems (e.g., payment gateways, shipping providers, external inventory services) to enhance functionality.
* **Interaction**: The backend API interacts with these external services using REST APIs, typically for functionalities like processing payments, tracking shipments, or retrieving external inventory data.

# 5. Security Considerations

* **SSO & MFA**: Azure AD ensures that only authenticated users can access the system. Multi-Factor Authentication adds an extra layer of security, protecting against unauthorized access even if a user's password is compromised.
* **Role-Based Access**: Azure AD assigns users to roles, and the backend enforces access control based on those roles to ensure that users can only access the data and features that are relevant to them.
* **API Security**: The backend ensures that all API calls are protected by validating the JWT tokens issued by Azure AD.
* **Data Security**: Sensitive data is encrypted both at rest (in Azure Storage and Database) and in transit (via HTTPS).

# 6.Tech Stack Selection and Justification

* **User Management & Access Control** (Azure AD SSO, MFA, Role-Based Access)
* **Vehicle Inventory & Sales Management** (CRUD operations, data storage)
* **Service & Maintenance Management** (Customer records, scheduling)
* **Parts & Supply Chain Management** (Stock updates, supplier management)
* **Finance & Accounting Integration** (Invoices, payments, tax calculations)
* **Security & Compliance** (Data encryption, GDPR compliance)
* **Frontend:** .NET Core Razor Pages
* **Backend:** ASP.NET Core Web API
* **Authentication:** Azure Active Directory (SSO & MFA)
* **Database:** Azure SQL Database
* **Hosting:** Azure App Services
* **DevOps:** Azure DevOps
* **Monitoring:** Azure Application Insights

# 6.1. .NET Core

* **Justification**:
  + **Cross-Platform**: .NET Core is cross-platform, allowing the application to run on multiple platforms (Windows, Linux, macOS).
  + **Performance**: .NET Core offers high performance, making it a suitable choice for enterprise-level applications like DMS that may require handling many concurrent users and large datasets.
  + **Security**: .NET Core has built-in security features such as data protection, authentication mechanisms (OAuth, JWT), and secure handling of requests.
  + **Ecosystem**: .NET Core provides a rich ecosystem of libraries and tools for building and maintaining robust, secure applications.
  + **Long-Term Support**: Microsoft provides long-term support for .NET Core, ensuring that security patches and updates are regularly released.

# 6.2. Razor Pages

* **Justification**:
  + **Server-Side Rendering (SSR)**: Razor Pages allows for seamless integration of dynamic server-side rendering with clean separation of concerns, making it ideal for building interactive web applications like a DMS.
  + **Built on .NET Core**: Razor Pages is part of the .NET Core ecosystem, enabling smooth integration with other components of the system (API, security).
  + **Security**: Razor Pages automatically handles encoding for output, protecting against Cross-Site Scripting (XSS) attacks, and supports features like CSRF protection.
  + **Ease of Use**: Razor Pages offers a simple and efficient way to build web interfaces with minimal configuration, making it a good choice for rapid development.

# 6.3. Azure Active Directory (Azure AD)

* **Justification**:
  + **Identity and Access Management**: Azure AD provides secure authentication and authorization, allowing for easy management of user identities and roles.
  + **SSO and MFA**: Azure AD simplifies the implementation of SSO and MFA, improving security and user experience.
  + **Scalability**: Azure AD is highly scalable, handling large numbers of users and offering enterprise-grade features.
  + **Integration**: It integrates easily with .NET Core applications and supports industry standards like OAuth 2.0 and OpenID Connect for secure API authorization.

# 6.4. Azure SQL Database

* **Justification**:
  + **Azure SQL Database**: A fully managed relational database that scales seamlessly, supports high availability, and integrates well with .NET Core applications.
  + **Security**: Azure SQL Database offers built-in encryption, automatic backups, and security features like managed firewalls and advanced threat protection.

# 6.5. Azure Blob Storage

* **Justification**:
  + **Unstructured Data Storage**: Azure Blob Storage is ideal for storing unstructured data such as images, documents, and contracts related to dealer operations.
  + **Scalability**: Blob Storage can scale to handle large amounts of data, making it a robust solution for storing documents and images that may be uploaded by dealers.

# 6.6. API Security: OAuth 2.0 & JWT

* **Justification**:
  + **Security**: OAuth 2.0 and JWT ensure secure token-based authentication and authorization for accessing APIs. This eliminates the need for session management and offers better scalability for APIs.
  + **Integration**: Azure AD seamlessly integrates with OAuth 2.0 and JWT to secure API access and verify user identities.

# 7. Deployment Playbook and Infrastructure as code scripts

# 7.1. Infrastructure Setup

* Ensure Azure subscription access.
* Run Terraform apply or az deployment group created with Bicep.
* Verify resource creation in Azure Portal.

# 7.2. Application Deployment

* Build and publish the application: dotnet publish -c Release -o ./publish
* Deploy using Azure CLI: az webapp deployment source config-zip --resource-group dms-rg --name dms-app-service --src ./publish.zip

# 7.3. CI/CD Execution

* Push code to Azure DevOps repository.
* Run pipeline (DMS.yml).
* Ensure build and deployment success.

# 7.4. Post-Deployment Checks

* Verify the application URL in Azure.
* Check Azure Application Insights for logs.
* Ensure authentication with Azure AD.

# 8. Cloud vs. On-Premises Deployment Considerations

# 8.1. Cloud Deployment (Azure)

**Pros:**

* **Scalability**: Easily scale resources up/down.
* **Managed Services**: Reduced operational overhead with Azure App Service, Azure SQL, etc.
* **Security & Compliance**: Built-in security features like Azure AD, Defender, and compliance with GDPR.
* **High Availability**: Multi-region failover with built-in disaster recovery.
* **Cost Efficiency**: The pay-as-you-go model minimizes upfront infrastructure costs.

**Cons:**

* **Ongoing Costs**: Subscription-based models can be expensive over time.
* **Latency**: Depending on region availability, some operations may have slight latency.
* **Vendor Lock-in**: Dependency on Azure-specific services and tools.

# 8.2. On-Premises Deployment

**Pros:**

* **Full Control**: Direct control over hardware, software, and security policies.
* **Fixed Costs**: No ongoing subscription fees, beneficial for long-term cost planning.
* **Data Residency**: Ensures sensitive data stays within internal infrastructure.
* **Customization**: Tailored infrastructure and security configurations.

**Cons:**

* **High Initial Costs**: Requires significant upfront investment in hardware, licenses, and networking.
* **Maintenance Overhead**: Dedicated IT team required for server, database, and network management.
* **Scalability Challenges**: Scaling infrastructure requires additional physical hardware.
* **Disaster Recovery Complexity**: Requires backup strategies and secondary data centers for redundancy.

# 8.3. Hybrid Approach

For organizations that require both **cloud scalability** and **on-premises control**, a **hybrid deployment** can be implemented:

* **Azure Arc**: Manage on-prem resources with Azure services.
* **Hybrid Identity**: Combine Azure AD with on-premises Active Directory.
* **Data Synchronization**: Use Azure Hybrid Connections for seamless data exchange.

# 9. Multi-Region and High Availability Deployment Strategy

# 9.1 Multi-Region Deployment

* Deploy application replicas in multiple Azure regions (e.g., West Europe, North Europe).
* Use **Azure Traffic Manager** or **Azure Front Door** for global load balancing.
* Configure geo-redundant storage for databases (Azure SQL Geo-Replication, Cosmos DB multi-region writes).

# 9.2. High Availability Strategy

* Use **Azure App Service Premium Plan** for autoscaling and high availability.
* Deploy with **Availability Zones** for redundancy.
* Enable **Azure Load Balancer** for backend failover.
* Implement **Database Failover Groups** for automatic failover.