Table of Contents

General Description

SPN offers two Single Sign-On (SSO) solutions to integrate clients' corporate authentication systems:

- **Web Self-Service SSO**: SAML 2.0 integration for web applications
- **SPN Mobile App SSO**: OAuth 2.0/MSAL integration for mobile applications

Web Self-Service SSO

Table of Contents

Self-Service General Description

The SSO (Single Sign-On) solution for SPN Self-Service allows clients to integrate their corporate authentication system with the Self-Service platform, providing a unified and secure user experience.

Main Benefits

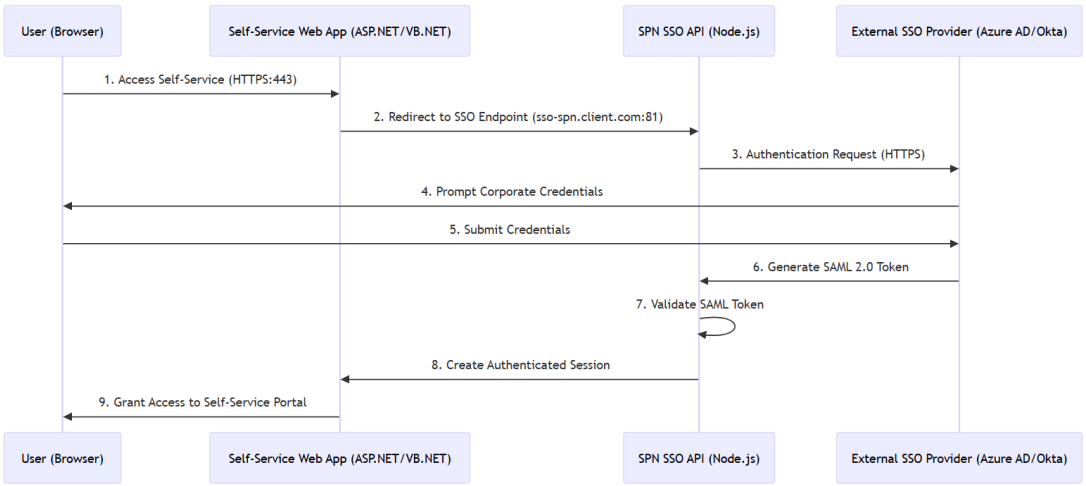
- **Unified authentication**: Users utilize their existing corporate credentials
- **Enhanced security**: Centralization of authentication and authorization policies
- **User experience**: Elimination of multiple logins and passwords
- **Compliance**: Adherence to corporate security policies

Self-Service Solution Architecture

```
┌──────────────────────────────────────────────────┐
│              Client (Windows Server)             │
├──────────────────────────────────────────────────┤
│  1. Self-Service Web App (ASP.NET/VB.NET)        │
│     - Port: 80/443                               │
│     - Main user interface                        │
│                    ↕                             │
│  2. SPN SSO API (Node.js)                        │
│     - Port: 81/443                               │
│     - SAML processing                            │
│     - Domain: sso-spn.client.com                 │
│                    ↕                             │
│  3. Client SSO Provider (Azure AD / Okta)        │
│     - Configured and managed by client           │
│     - Corporate user authentication              │
└──────────────────────────────────────────────────┘
```

Authentication Flow

1. User accesses Self-Service
2. Self-Service redirects to SPN SSO API
3. API redirects to client's SSO provider
4. User authenticates with corporate credentials
5. SSO provider sends SAML token to API
6. API processes token and validates information
7. User is redirected to Self-Service with active session



Supported SSO Providers

Microsoft Azure AD

- **Protocol**: SAML 2.0
- **User format**: Corporate email
- **Configuration**: Enterprise Application
- **Ideal for**: Organizations with Microsoft 365 ecosystem

Okta

- **Protocol**: SAML 2.0
- **User format**: Company ID or email
- **Configuration**: SAML 2.0 Web App
- **Ideal for**: Organizations with hybrid or multi-cloud identities

Self-Service Client Responsibilities

The Client is Responsible for:

Infrastructure and Hosting

- Provide Windows server to host Self-Service and SPN SSO API
- Maintain and administer server infrastructure
- Configure and maintain network connectivity
- Manage backups and disaster recovery

SSO Provider

- Completely manage their SSO environment (Azure AD, Okta, etc.)
- Configure SAML application in their SSO provider
- Assign users to SSO application
- Maintain active users and manage access
- Provide test credentials for validation

SSL Certificates

- Acquire valid SSL certificate for API domain
- Install and maintain certificate in IIS
- Renew certificates before expiration

User Linking

- Identify linking method (email or company ID)
- Provide mapping of SSO users with Self-Service users
- Maintain user synchronization between systems

Self-Service Technical Requirements

Server and Software

- Windows Server 2016 or higher
- Internet Information Services (IIS) 8.5+
- Node.js 16 LTS or higher
- iisnode module for IIS
- .NET Framework 4.7.2+ (for Self-Service)

Network and Connectivity

- Internet access for communication with SSO providers
- Port 80/443 available for Self-Service
- Additional port (e.g., 81/443) for SPN SSO API
- Firewall configured to allow HTTPS traffic
- DNS configured for API subdomain

SSL Certificate

- Valid SSL certificate for SSO API domain
- .pfx format with private key included
- Complete certificate chain if required
- Valid for at least 12 months

SSO Provider

- Active tenancy in Azure AD or Okta
- Administrative permissions to configure SAML applications
- Users configured and active in system
- Appropriate licensing based on number of users

Information Required by SPN

For API Configuration

From SSO Provider

- SSO provider metadata XML
- X.509 certificate (if not included in metadata)
- Provider Entity ID
- Single Sign-On URL
- Single Logout URL (if available)

From Client Environment

- Definitive domain for SSO API (e.g., sso-spn.client.com)
- Preferred linking method (email or company ID)
- Test user information for validation

Access for Installation

- Remote access to production server
- Local administrator credentials for server
- Agreed maintenance window for installation

Self-Service Implementation Process

Phase 1: Planning and Design

Duration: 1-2 days
Responsible: SPN + Client

- Definition of client-specific architecture
- SSO API domain selection
- User linking method identification
- Implementation window planning

Phase 2: Client Preparation

Duration: 3-5 days
Responsible: Client

- SSL certificate acquisition and configuration
- Windows server preparation
- SAML application configuration in SSO provider
- Provider metadata and certificates generation

Phase 3: Development and Configuration

Duration: 2-3 days
Responsible: SPN

- SPN SSO API configuration with client information
- Configuration files preparation
- Testing in development environment
- Installation package preparation

Phase 4: Installation and Deployment

Duration: 1 day
Responsible: SPN + Client

- Prerequisites installation on server
- SPN SSO API deployment
- IIS and SSL certificates configuration
- Integration with existing Self-Service

Phase 5: Testing and Validation

Duration: 1-2 days
Responsible: SPN + Client

- Connectivity and authentication testing
- User linking validation
- Complete login flow testing
- Resolution of identified issues

Phase 6: Go-Live and Support

Duration: 1 day + ongoing support
Responsible: SPN + Client

- Production activation
- Initial system monitoring
- Client administrator training
- Final configuration documentation

Self-Service Client Configurations

In Azure AD

Client must configure an Enterprise Application with:

- **Identifier (Entity ID)**: SSO API domain
- **Reply URL**: API SSO ACS endpoint
- **Sign on URL**: API SSO login endpoint
- **Attribute mappings**: Email, name, and user ID
- **User assignment**: Users who will have access

In Okta

Client must configure a SAML 2.0 Web App with:

- **Single sign on URL**: API SSO ACS endpoint
- **Audience URI**: SSO API domain
- **Name ID format**: EmailAddress or Unspecified
- **Attribute statements**: User attribute mapping
- **Application assignments**: Users or groups with access

User Configuration

Email Linking (typical Azure AD)

- Self-Service users must have email matching Azure AD
- Verify emails are unique and valid
- Synchronize email changes between systems

Company ID Linking (typical Okta)

- Self-Service users must have company ID field
- Map corporate IDs with Self-Service users
- Maintain consistency in ID format

Self-Service Validation and Testing

Connectivity Tests

- Verify access to SSO provider URLs
- Confirm DNS resolution of API domain
- Validate SSL certificate installed correctly

Authentication Tests

- Successful login with test user
- Verify information received in SAML token
- Confirm correct redirection to Self-Service

Linking Tests

- Validate SSO users link correctly
- Test user not found scenarios
- Verify permissions and roles in Self-Service

Session Tests

- Confirm appropriate session duration
- Test logout and session termination
- Validate behavior with multiple tabs/windows

Self-Service Support and Maintenance

SPN Responsibilities

- SPN SSO API technical support
- Security and functionality updates
- API logs and error monitoring
- Documentation and troubleshooting guides

Client Responsibilities

- SSO provider maintenance
- User and access management
- SSL certificate renewal
- Server infrastructure maintenance
- Configuration backup and recovery

Support Escalation

- **Level 1**: SPN SSO API issues → SPN Support
- **Level 2**: SSO provider issues → Client administrator
- **Level 3**: Infrastructure issues → Client IT team

Monitoring and Alerts

- Authentication logs in SPN SSO API
- Monitoring certificates near expiration
- SSO provider connectivity alerts
- Active user and session metrics

Technical and Architectural Arguments

SSL Certificate Arguments

Non-Negotiable Technical Requirement

- Modern browsers block "Mixed Content" (HTTPS → HTTP)
- Azure AD and Okta automatically reject HTTP endpoints for SAML
- SAML 2.0 protocol requires encryption in transit by standard
- Without HTTPS, there is literally no possible communication with SSO providers

Actual User Flow

- User from their PC → Self-Service (HTTPS) → User redirected to SSO API
- The call comes from the user's browser, NOT from the server
- It's external communication that requires public DNS and valid certificate

Arguments Against Separate Server

Increased Security Risks

- **Expanded attack surface**: Two servers = double exposure
- **Firewall complexity**: More ports and rules to manage
- **Patch synchronization**: Two operating systems to maintain
- **Distributed logs**: More complex and error-prone auditing

Communication Problems

- Additional latency between Self-Service and SSO API
- Internal network dependency that can fail
- Complexity in troubleshooting connectivity issues
- More failure points in authentication chain

False Compliance

- Separating related components does NOT improve compliance
- Real compliance requires encryption, auditing, and access controls
- Having two servers may violate “defense in depth” principles
- It’s less secure architecture, not more secure

Technical Architecture Arguments

Why Separate API from Legacy .NET

- **Old .NET Framework**: Doesn’t support modern SAML libraries
- **Node.js for SAML**: Updated and actively maintained libraries
- **Separation of concerns**: UI (.NET) vs Authentication (Node.js)
- **Maintainability**: Easier to update specific component

Why Same Server is Optimal

- **Functional cohesion**: Both components are part of the same solution
- **Internal network**: Ultra-fast server-to-server communication
- **Unified management**: Single server to patch and monitor
- **Reduced cost**: Single Windows Server license

Responses to Specific Objections

"Why do we need public domain?"

- Azure AD/Okta need to send SAML responses back
- The user is who navigates between systems, not the server
- Internal DNS is not accessible to external SSO providers
- It’s a SAML protocol requirement, not SPN preference

"Compliance requires separation"

- **False**: Compliance requires controls, not physical separation
- Separation without technical purpose introduces more risks
- Better compliance = fewer components to secure
- Audits prefer simple and well-documented architectures

"What if API fails?"

- On separate server: Self-Service becomes totally inaccessible
- On same server: Localized problem, easier diagnosis
- Centralized logs allow rapid event correlation
- Faster recovery with co-located components

Recommended Technical Proposal

Secure Architecture on Same Server

- IIS with two sites: Port 80/443 (Self-Service) + Port 81/443 (API)
- Internal firewall: Only allow 80/443 public, 81/443 restricted
- SSL certificate with SAN for both domains if necessary
- Unified logs in Event Viewer and centralized files

Secure Network Configuration

- API on non-standard port (81) reduces exposure
- Same server = internal traffic doesn't leave network
- Split-brain DNS: Internal points to localhost, external to server
- Optional WAF for additional protection of both endpoints

Mobile App SSO

Table of Contents

Mobile General Description

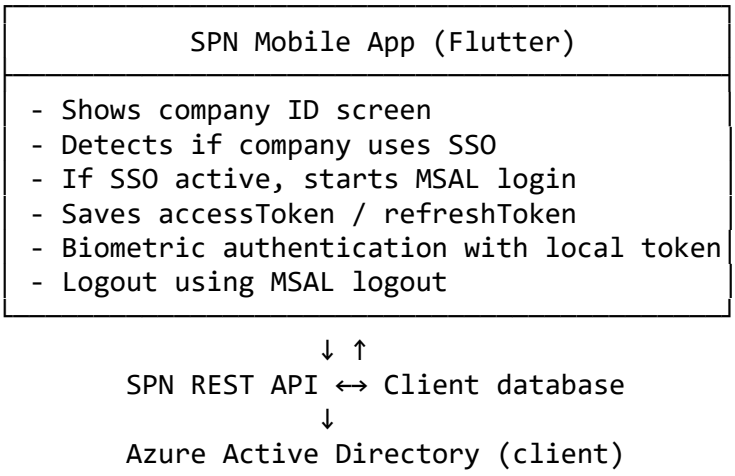
The Single Sign-On (SSO) solution for SPN Mobile application allows clients to authenticate their users through Azure Active Directory, providing a unified and secure access experience from mobile devices.

This integration uses the MSAL (Microsoft Authentication Library) library, designed to facilitate authentication from Flutter apps to Azure AD.

Main Benefits

- **Authentication unification**: Use of Azure AD corporate credentials
- **Security**: Valid tokens issued by Azure with automatic refresh support
- **Optimized experience**: Smooth authentication with PIN or biometrics
- **Cross-platform compatibility**: Compatible with Android and iOS

Mobile Solution Architecture



General Flow

1. User enters company ID
2. App queries API to know if company uses SSO
3. If uses SSO:
   - MSAL login is initiated
   - User enters corporate credentials
   - Azure AD responds with tokens
4. If doesn't use SSO: traditional login with ID/password is shown
5. In subsequent openings, app attempts authentication with PIN or biometrics, validating stored token

SSO Provider Used

Microsoft Azure Active Directory

- **Protocol**: OAuth 2.0 / OpenID Connect
- **Library used**: MSAL for Flutter
- **User format**: Corporate email
- **Integration method**: App registration in Azure as Public client/native app

Mobile Client Responsibilities

Azure AD Registration

- Register "SPN Mobile" mobile app in Azure
- Configure redirections:
  - `msauth:///auth`
- Establish Client ID and Tenant ID
- Assign allowed users to registered app

Information Provision

- Provide the following data for each company in database:
  - SSO_Client_ID
  - SSO_Tenant_ID
  - SSO_Enable (boolean)

Platform-specific Configurations

- **For Android**:
  - Register SHA-1 of app signature
- **For iOS**:
  - Bundle ID and custom URI schemes

Mobile Technical Requirements

Flutter App

- Minimum Flutter version: 2.17.0
- Key dependency: azure_oauth: ^2.3.1
- Other dependencies:
  - shared_preferences
  - local_auth (for biometrics)
  - flutter_bloc (state management)

Mobile Information Required by SPN

For each company wanting SSO integration:

- Client ID of app in Azure
- Directory Tenant ID
- SSO status (active/inactive)
- SHA-1 signature (Android)
- Bundle ID (iOS)
- Authorized URI scheme
- Test user

Mobile Implementation Process

Phase 1: Design and Preparation

- Login flow within Flutter app is defined
- Companies using SSO are identified

Phase 2: Azure AD Registration

- Each client must register app with appropriate configuration
- Required technical values are provided (SHA, Bundle ID, etc.)

Phase 3: App Implementation

- Company screen (company ID) is created
- Conditional login is implemented (SSO vs Traditional)
- Token is stored and validated locally in each session
- Refresh token is used silently

Phase 4: Testing and Validation

- Both flows are tested (SSO and traditional)
- Token and persistence are validated
- Biometrics and fallback to MSAL login are tested

Mobile Client Configurations

In Azure AD

- Register App as Public Client
- Establish:
  - Redirect URI: `msauth:///auth`
  - Client ID: generated by Azure
  - Tenant ID: corporate AD identifier
- Assign permissions (delegated):
  - openid, profile, email, offline_access
- Configure allowed users

Mobile Validation and Testing

Authentication Tests

- Verify successful MSAL login
- Validate stored tokens
- Simulate expiration and renew with refresh token

Fallback Tests

- Simulate token error → force new login
- Validate general message on credential error

Biometric Tests

- Activate biometric login (if available)
- Validate that login is not required every time

Mobile Support and Maintenance

SPN

- Maintains MSAL integration logic
- Support for authentication errors
- MSAL library updates in future versions

Client

- Maintains app registered in Azure
- Updates certificates or permissions when necessary
- Provides test users for changes

Escalation

- **Level 1**: Errors within SPN App → SPN Support
- **Level 2**: Azure problems (registration, permissions) → Client IT team
- **Level 3**: Global authentication issues → Microsoft Support

Support and Maintenance

Security Considerations

Best Practices

- SSL certificates always valid and updated
- SAML tokens with limited lifetime
- Audit logs enabled in all components
- Limited access only to authorized users

Final Considerations

For Mobile App

- Each client should have a separate registered app in Azure
- Use of refresh token reduces need for constant login
- App doesn't store passwords, only valid tokens issued by Azure
- Logout process executes complete session closure in Azure

Document version 1.0 – September 2025