

CALF Security Framework

Technical Specification Version 1.0

Contents

1. [Overview 3](#_Toc457314708)
2. [Framework User Personas 3](#_Toc457314709)
3. [Technical Requirements 3](#_Toc457314710)
4. [Not-in-Scope 4](#_Toc457314711)
5. [Non-Technical Requirements 4](#_Toc457314712)
6. [Framework Feature Areas 5](#_Toc457314713)

[Security Management 5](#_Toc457314714)

[Security Services 5](#_Toc457314715)

[Security Authorization Extensibility 6](#_Toc457314716)

[Security Auditing and Monitoring 6](#_Toc457314717)

1. [Framework Design 7](#_Toc457314718)

[Security Management Architecture 7](#_Toc457314719)

[Security Services Architecture 8](#_Toc457314720)

[Framework Access Control 9](#_Toc457314721)

[Security Auditing and Monitoring 9](#_Toc457314722)

[Security Flow 10](#_Toc457314723)

[Security Management Sequence Diagram 10](#_Toc457314724)

[Security Services Sequence Diagram 11](#_Toc457314725)

[Menu Navigation Sequence Diagram 12](#_Toc457314726)

[Security Auditing Sequence Diagram 12](#_Toc457314727)

1. [Framework Implementation 13](#_Toc457314728)

[Database Schema 13](#_Toc457314729)

[Identity Relationships 13](#_Toc457314730)

[Application Relationships 14](#_Toc457314731)

[Role-Permission Relationship 14](#_Toc457314732)

[Security Management Classes 15](#_Toc457314733)

[Import Security Manifest Schema 16](#_Toc457314734)

[Import Security Assignments Schema 17](#_Toc457314735)

[Export Security Manifest Schema 18](#_Toc457314736)

[Export Security Assignments Schema 19](#_Toc457314737)

[Security Services Endpoints 20](#_Toc457314738)

1. [Application Programming Model 20](#_Toc457314739)
2. [Security Authorization Provider Extension 21](#_Toc457314740)
3. [Security Services API 21](#_Toc457314741)

[GetClaims 21](#_Toc457314742)

[GetClaimsRequest 21](#_Toc457314743)

[GetClaimsResponse 21](#_Toc457314744)

[IsInRole 22](#_Toc457314745)

[GetRoleMembershipRequest 22](#_Toc457314746)

[GetRoleMembershipResponse 22](#_Toc457314747)

[GetMenus 22](#_Toc457314748)

[GetMenusRequest 22](#_Toc457314749)

[GetMenusResponse 22](#_Toc457314750)

1. [Security Management API 23](#_Toc457314751)
2. [Security Import-Export API 24](#_Toc457314752)
3. [Security Auditing API 24](#_Toc457314753)
4. [Scalability and Reliability 25](#_Toc457314754)

[Security Caching 25](#_Toc457314755)

[Security Exception Monitoring 25](#_Toc457314756)

1. [Deployment Model 25](#_Toc457314757)
2. [Workstream and Enterprise Lifecycle 26](#_Toc457314758)

|  |  |  |
| --- | --- | --- |
| Author | Status | Date |
| S. Compean | Creation | 7/21/2016 |
| S. Compean | Incorporated demo feedback | 7/26/2016 |
|  |  |  |

# Overview

This technical specification describes the objects, design and implement for the CALF Security Architecture part of the FHLB Common Core Library. The document serves as a valuable reference to guide construction and verification of framework feature capabilities.

# Framework User Personas

|  |  |
| --- | --- |
| User Persona | Description |
| .NET Developer | FLHB .NET Developers will use the security framework to define and configure security aspects for an ASP .NET MVC application. The developer will be able to define application identities, roles and permissions as well as assignments between these entities. The developer will also be able to define application menu navigation. |
| Security Admin | FHLB Security Administrators will use the security framework to manage application security configuration. The administrator will be able to import and export security metadata via a security manifest file. The administrator uses the framework to manage application security configuration across server environments. |

# Technical Requirements

The technical requirements for the CALF Security Framework are defined in the following table.

|  |  |  |
| --- | --- | --- |
| ID | Requirement | Notes |
| 1 | Framework needs to support defining an Application |  |
| 2 | Framework needs to support defining Application Identities |  |
| 3 | Framework needs to support defining Application Roles |  |
| 4 | Framework needs to support defining Application Menus |  |
| 5 | Framework needs to support defining Application Permissions |  |
| 6 | Framework needs to support assigning Identities to Roles |  |
| 7 | Framework needs to support assigning Roles to Permissions |  |
| 8 | Framework needs to support persisting Application definitions |  |
| 9 | Framework needs to support persisting Application Roles |  |
| 10 | Framework needs to support persisting Application Menus |  |
| 11 | Framework needs to support persisting Application Permissions |  |
| 12 | Framework needs to support persisting Identity-to-Role mappings |  |
| 13 | Framework needs to support persisting Role-to-Permissions mappings |  |
| 14 | Framework should provide extended authorization data to .NET apps |  |
| 15 | Framework should audit security configuration changes |  |
| 16 | Framework should report exceptions for monitoring |  |
| 17 | Framework should enable importing application security definitions |  |
| 18 | Framework should enable importing application security permissions |  |
| 19 | Framework should enable exporting application security definitions |  |
| 20 | Framework should enable exporting application security permissions |  |
| 21 | Framework should define declarative markup for security definitions |  |
| 22 | Framework should define declarative markup for security permissions |  |
| 23 | Framework should provide menu rendering data to .NET apps |  |
|  |  |  |

# Not-in-Scope

The following items have been determined to be out-of-scope:

|  |  |  |
| --- | --- | --- |
| ID | Requirement | Notes |
| 1 | Framework does not need to support Active Directory management |  |
| 2 | Framework does not need to support user authentication nor SSO |  |
| 3 | Framework does not need to support Web services access control |  |
| 4 | Framework does not need to support applications for external users |  |
|  |  |  |

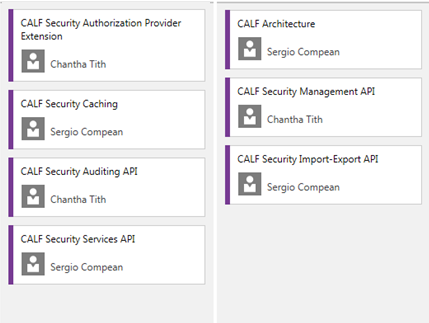
# Non-Technical Requirements

The following non-technical requirements have been identified:

|  |  |  |
| --- | --- | --- |
| ID | Requirement | Notes |
| 1 | Framework should provide great ease of use for .NET developers |  |
| 2 | Framework should provide authorization provider extensibility |  |
| 3 | Framework should be high performance and scalable |  |
| 4 | Framework services should not be chatty |  |
| 5 | Framework should apply caching as needed |  |
|  |  |  |

# Framework Feature Areas

The CALF Security Framework feature areas have been organized into the following categories. The categories have been grouped into two major types: management and services. Management features provide the design-time declarative functions to define security metadata. Services deliver run-time developer functions that primarily augment the capabilities provided by the native security authorization provider used by ASP .NET MVC applications.



## Security Management

Security Management provides features to define and persist security metadata and privilege assignments. The management features give the .NET developer the ability to declaratively define an application’s identities, roles and permissions. Assignments between identities and roles as well as between roles and permissions are enabled. These relationships are created using a manifest file which is processed by the management functions. The security metadata is persisted to an application security container database.

Import and export functions will allow the security metadata in the container to be inserted and extracted so that they can be migrated to additional server environments.

## Security Services

Security Services provide the run-time features that enable ASP .NET applications to access augmented authorization provider properties that can be used for access control. The services inject additional claims properties to the security principal so that code logic can check values to determine execution flow. The security services will be transparent to .NET developers and work more in the background to surface the augmented security role and claim data.

Security services will also deliver menu rendering metadata so the application can construct navigation user interface elements. The services will handle context-aware menu requests to provide the corresponding menu elements needed for proper rendering.

## Security Authorization Extensibility

The Security Framework will make extensive use of extending the native security principal attributes to surface custom authorization attributes. The .NET developer should not have to be aware of the extra functions to bring this data into the ASP .NET application. The framework will seamless inject the custom security metadata into the native authorization provider interface by overriding default behavior.

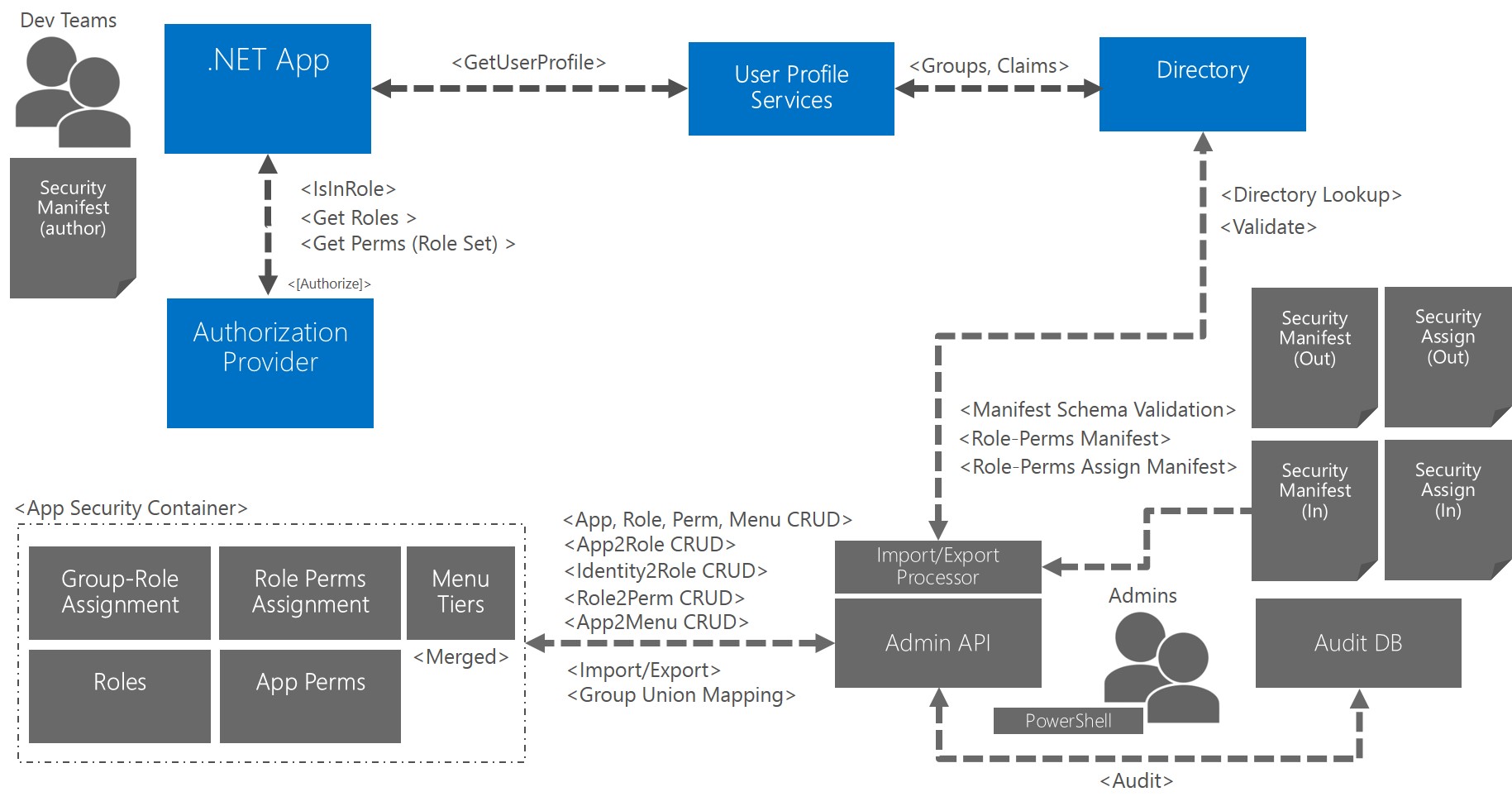
## Security Auditing and Monitoring

The Security Framework auditing and monitoring features will ensure configuration changes made to the application security container are tracked with detailed information to support an IT audit. The auditing captures information such as identity of user making the security configuration change, the delta in the security metadata as well as timestamp data. The framework captures old and new values of the security metadata to maintain a journal of changes made by either .NET developers or security administrators.

Security exceptions will be monitored by using a logging mechanism to the Windows Event Log. Exceptions can be tracked to enable proactive remedy action to prevent unauthorized access to framework features or security authorization issues at run-time in ASP .NET MVC applications.

# Framework Design

## Security Management Architecture



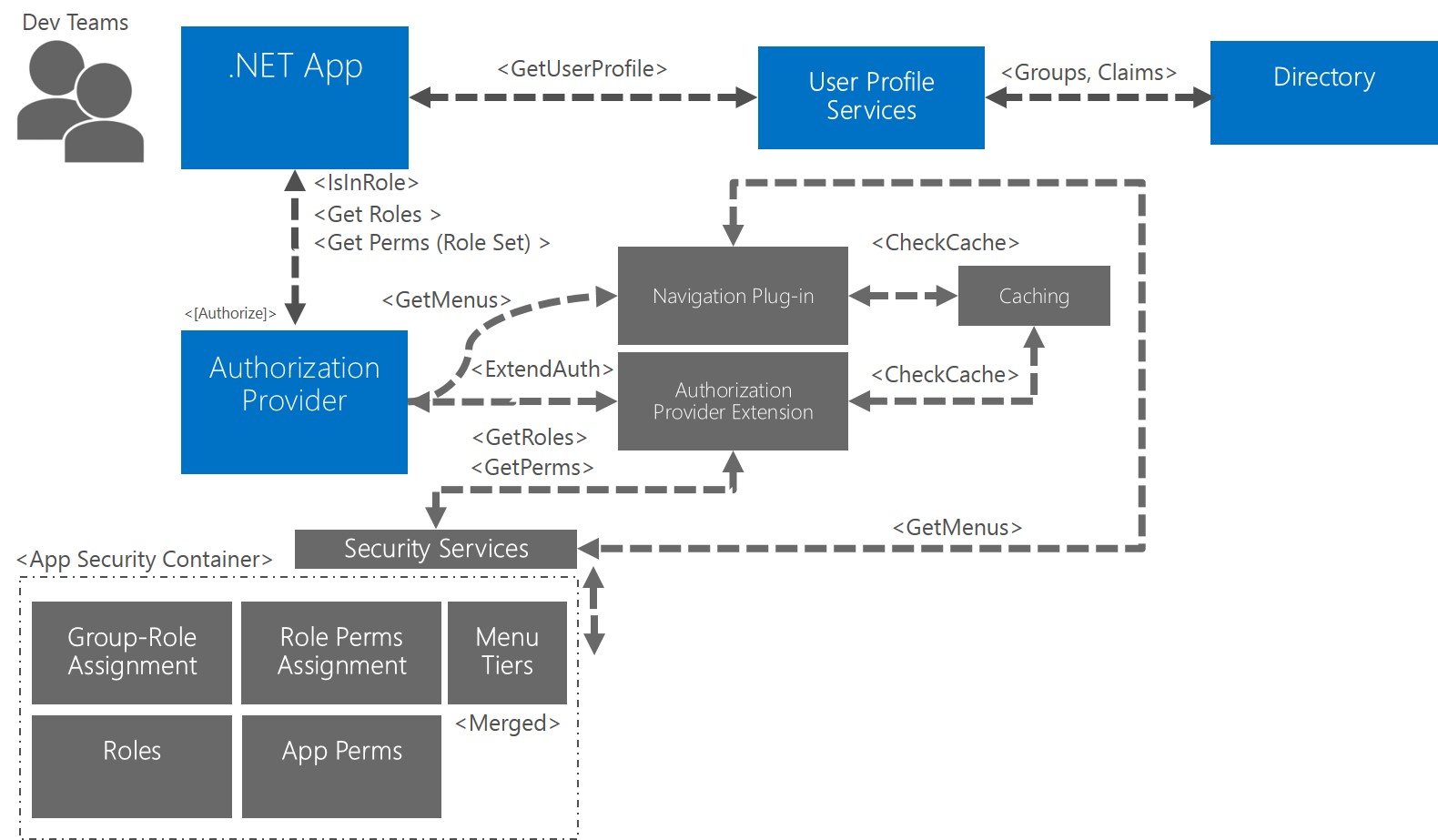
The Security Management architecture is defined by the following major components:

* Security Manifest
* Security Assignment Manifest
* Import/Export Processor
* PowerShell Scripts
* Admin API
* App Security Container
* Audit DB

The Security Manifest defines the application security objects such as identities, roles and permissions. The Security Assignment Manifest defines the assignments and mappings between these objects. The Admin API can be used to import and export metadata to and from the App Security Container via the Import/Export Processor. The Import/Export processor is invoked using administrative PowerShell scripts. The App Security Container responsibility is to serve as the persistent store for security configuration. The Admin API will also be responsible for tracking change requests made by journaling the deltas to the Audit DB.

.NET Developers will author security manifests using an XML markup language. Security administrators should also be able to import and export App Security Container metadata to migrate the security configuration to additional environments.

## Security Services Architecture



The Security Services architecture is defined by the following major components:

* Authorization Provider Extension
* Extended Claims
* Navigation Plug-in
* Security Services
* Caching
* App Security Container

The Authorization Provider Extension will seamless deliver extended claims to the .NET application without any extra API being necessary for the developer. The Extended Claims will surface using the native identity and role management programming model in the .NET framework.

The Navigation Plug-in will retrieve the working menu navigation set for a given application. The plug-in provides the menu rendering information needed by the application to present the navigation model in the user interface.

The Security Services will provide a SOA-style API using WCF to consuming .NET applications via the Authorization Provider Extension. The services design will be stateless to facilitate scale-out in a Web server farm environment. The services implement a façade repository pattern into the App Security Container.

Caching will be implemented to provide performance and throughput enhancement to increase scalability. Appropriate caching policies will be applied to ensure staleness does not become an issue for critical security information.

## Framework Access Control

Security framework functionality will apply a gatekeeper pattern to safeguard against unauthorized usage. Several techniques will be evaluated and applied in specific layers and components in the architecture. Some options available for implementing framework access control include:

* Code-signing
* Header custom authentication
* X.509 certificates
* Evidenced-based authorization
* Code access policies

These techniques will be assessed for level of effort to develop and maintain in an enterprise setting.

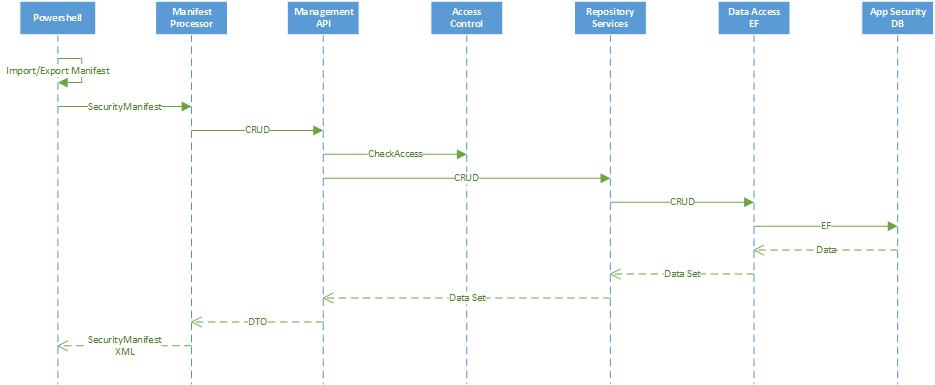
## Security Auditing and Monitoring

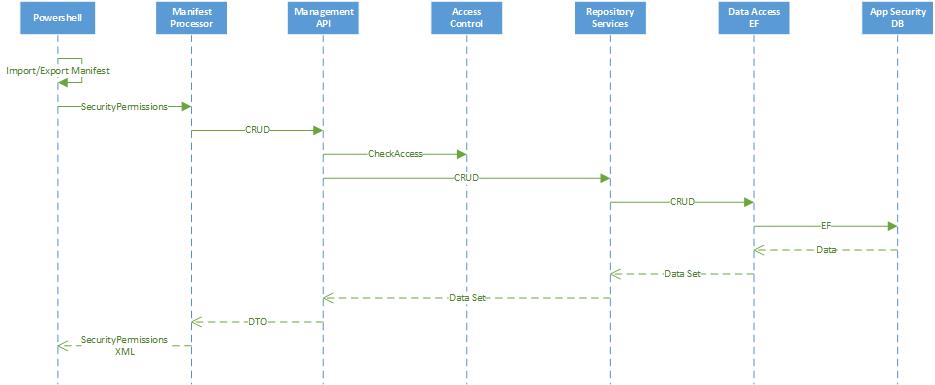
Security auditing and monitoring will enable tracking for security configuration changes and exceptions. Monitoring will be important to be able to detect unauthorized access to security framework APIs as well as run-time exceptions in .NET applications.

Auditing will provide a simple journaling mechanism to track deltas in security metadata. The journaling will persist old and new values to be able to view the changes made to support an IT audit.

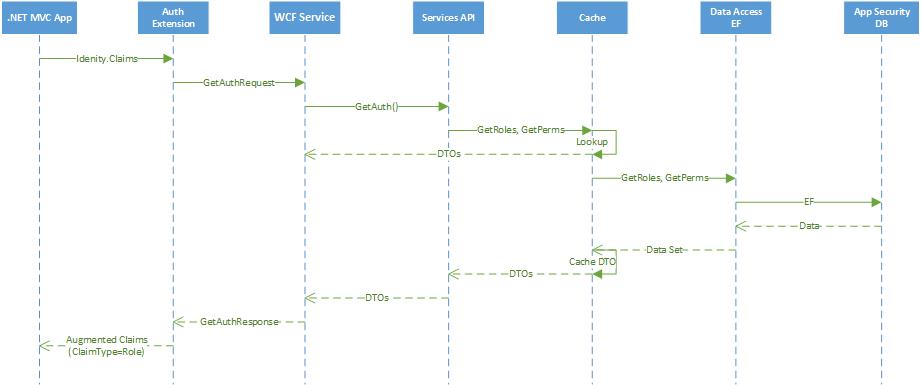
## Security Flow

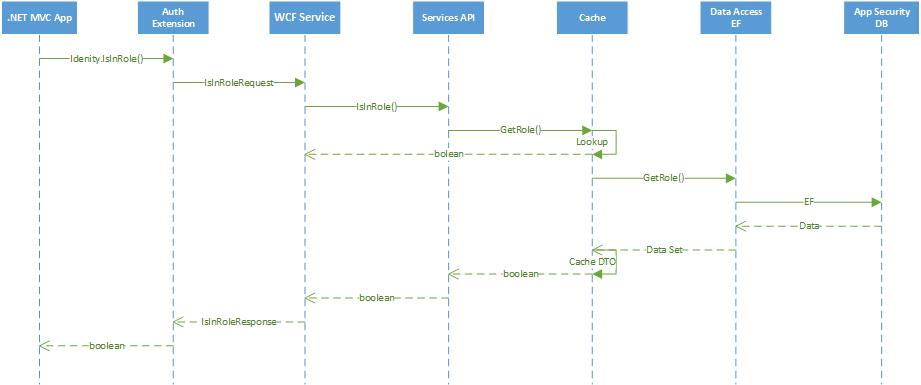
### Security Management Sequence Diagram



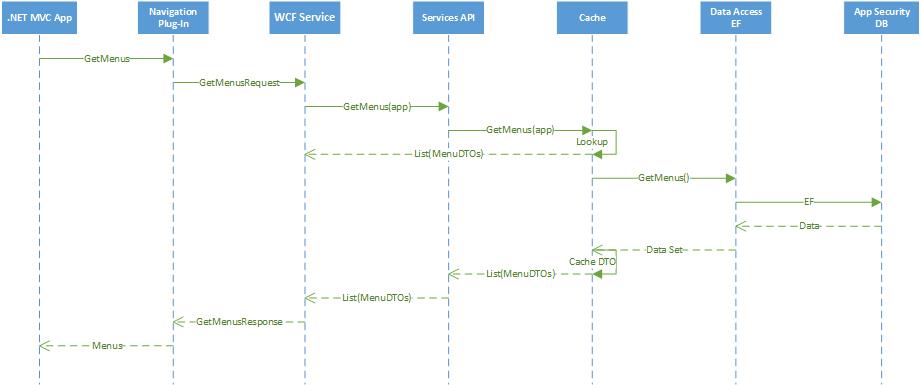


### Security Services Sequence Diagram

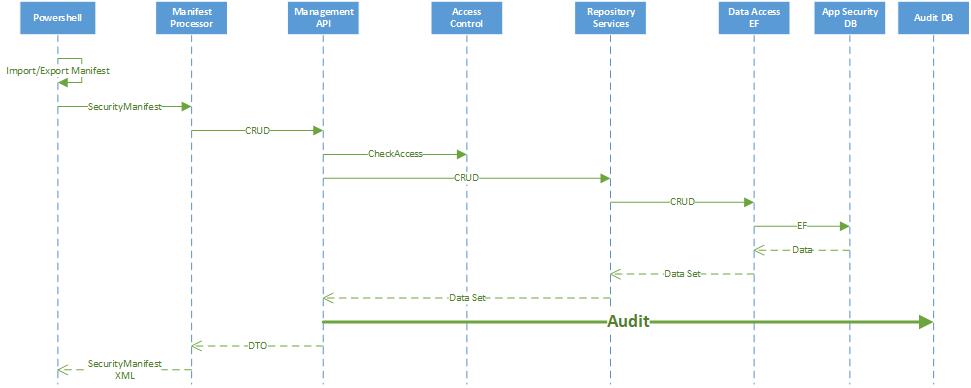




### Menu Navigation Sequence Diagram



### Security Auditing Sequence Diagram

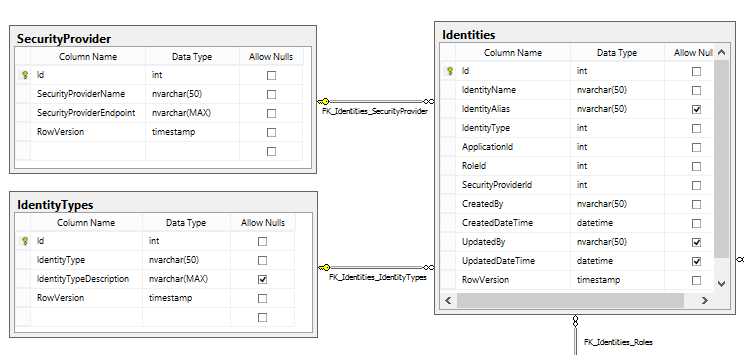


# Framework Implementation

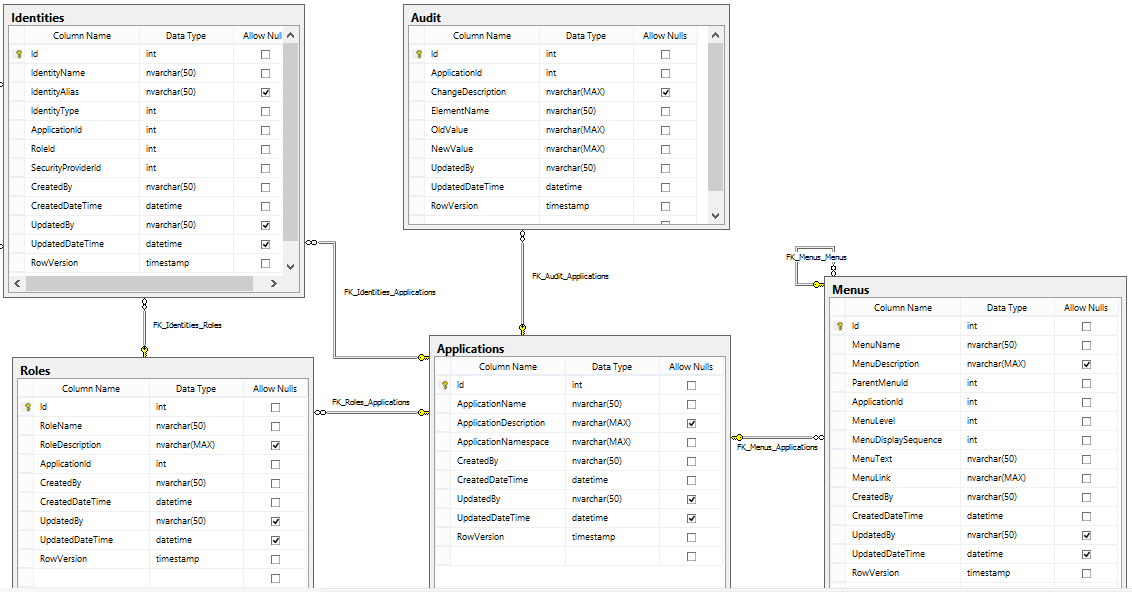
## Database Schema



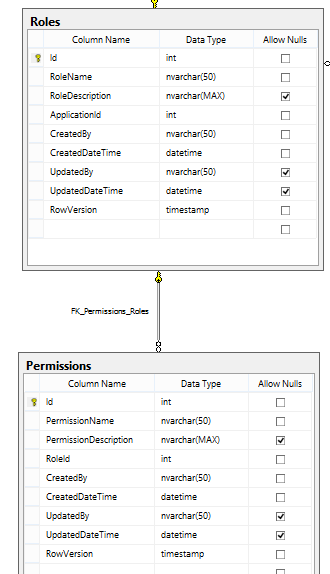
### Identity Relationships



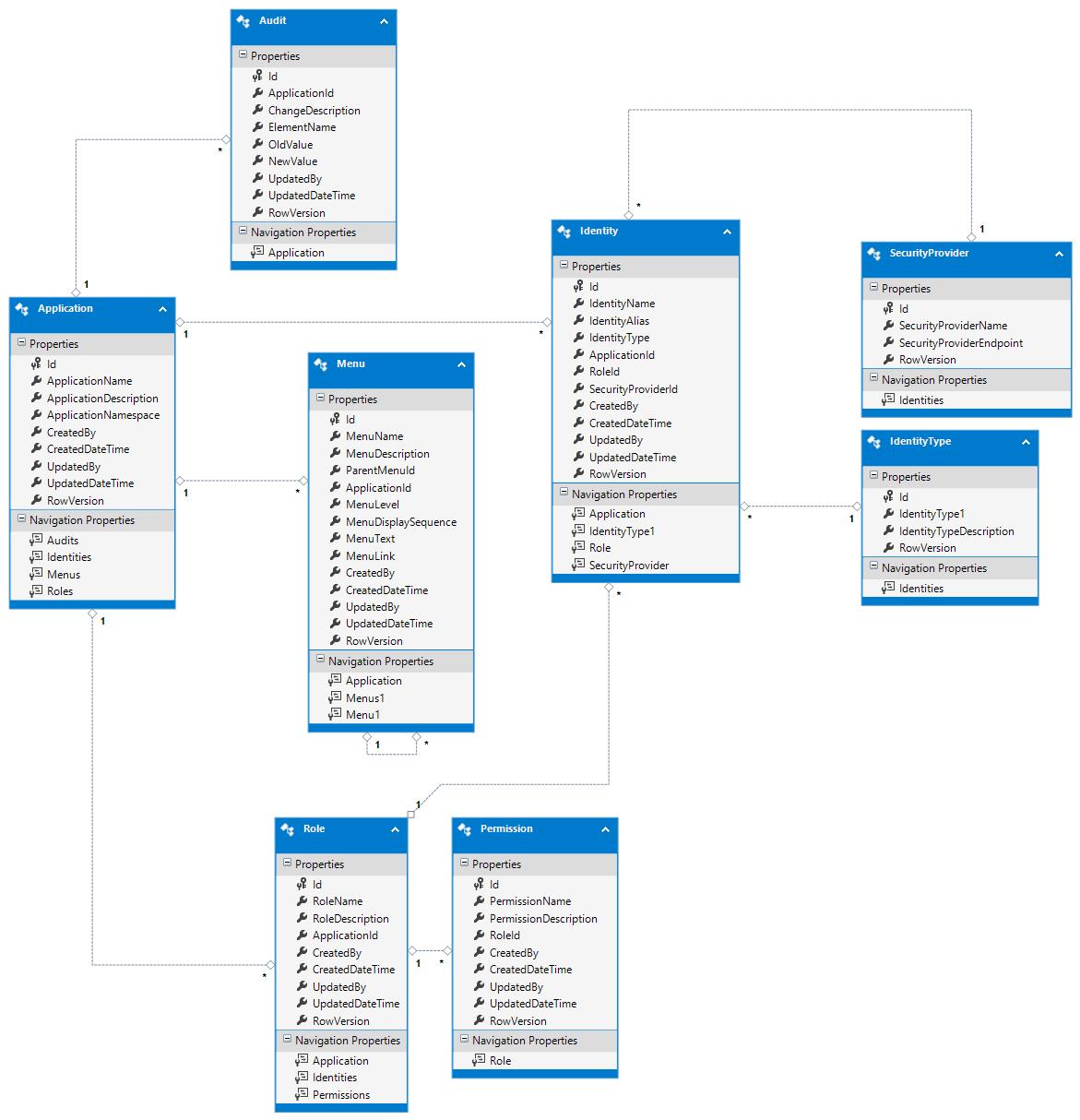
### Application Relationships



### Role-Permission Relationship



## Security Management Classes



## Import Security Manifest Schema

<?xml version="1.0" encoding="utf-8"?>

<xsd:schema id="SecurityManifestImport"

targetNamespace="http://www.fhlb.com/AppSecurity/SecurityManifestImport.xsd"

elementFormDefault="qualified"

xmlns="http://www.fhlb.com/AppSecurity/SecurityManifestImport.xsd"

xmlns:tns="http://www.fhlb.com/AppSecurity/SecurityManifestImport.xsd"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

>

<xsd:element name="comment" type="xsd:string" />

<xsd:element name="application" type="tns:ApplicationType" />

<xsd:element name="menus" type="tns:Menus" />

<xsd:complexType name="ApplicationType">

<xsd:sequence>

<xsd:element name="applicationName" type="xsd:string" />

<xsd:element name="applicationDescription" type="xsd:string" />

<xsd:element name="applicationNamespace" type="xsd:string" />

<xsd:element name="roles" type="tns:Roles" />

<xsd:element name="menus" type="tns:Menus" />

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Roles">

<xsd:sequence>

<xsd:element name="role" minOccurs="0" maxOccurs="unbounded">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="roleName" type="xsd:string" />

<xsd:element name="roleDescription" type="xsd:string" />

<xsd:element name="permissions" type="Permissions" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Permissions">

<xsd:sequence>

<xsd:element name="permission" minOccurs="0" maxOccurs="unbounded">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="permissionName" type="xsd:string" />

<xsd:element name="permissionDescription" type="xsd:string" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Menus">

<xsd:sequence>

<xsd:element name="menu" minOccurs="0" maxOccurs="unbounded">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="menuName" type="xsd:string" />

<xsd:element name="menuDescription" type="xsd:string" />

<xsd:element name="menuText" type="xsd:string" />

<xsd:element name="menuLink" type="xsd:string" />

<xsd:element name="menuDisplaySequence" type="xsd:integer" />

<xsd:element ref="menus" minOccurs="0" maxOccurs="unbounded" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

</xsd:schema

## Import Security Assignments Schema

<?xml version="1.0" encoding="utf-8"?>

<xsd:schema id="SecurityAssignmentsImport"

targetNamespace="http://www.fhlb.com/AppSecurity/SecurityAssignmentsImport.xsd"

elementFormDefault="qualified"

xmlns="http://www.fhlb.com/AppSecurity/SecurityAssignmentsImport.xsd"

xmlns:tns="http://www.fhlb.com/AppSecurity/SecurityAssignmentsImport.xsd"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

>

<xsd:element name="comment" type="xsd:string" />

<xsd:element name="application" type="tns:ApplicationType" />

<xsd:complexType name="ApplicationType">

<xsd:sequence>

<xsd:element name="applicationName" type="xsd:string" />

<xsd:element name="applicationDescription" type="xsd:string" />

<xsd:element name="applicationNamespace" type="xsd:string" />

<xsd:element name="identity" type="tns:Identities" />

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Identities">

<xsd:sequence>

<xsd:element name="identity" minOccurs="0" maxOccurs="unbounded">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="identityName" type="xsd:string" />

<xsd:element name="identityDescription" type="xsd:string" />

<xsd:element name="identityAlias" type="xsd:string" />

<xsd:element name="permissions" type="Permissions" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Permissions">

<xsd:sequence>

<xsd:element name="permission" minOccurs="0" maxOccurs="unbounded">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="permissionName" type="xsd:string" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

</xsd:schema>

## Export Security Manifest Schema

<?xml version="1.0" encoding="utf-8"?>

<xsd:schema id="SecurityManifestExport"

targetNamespace="http://www.fhlb.com/AppSecurity/SecurityManifestExport.xsd"

elementFormDefault="qualified"

xmlns="http://www.fhlb.com/AppSecurity/SecurityManifestExport.xsd"

xmlns:tns="http://www.fhlb.com/AppSecurity/SecurityManifestExport.xsd"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

>

<xsd:element name="comment" type="xsd:string" />

<xsd:element name="exportDate" type="xsd:dateTime" />

<xsd:element name="exportUser" type="xsd:string" />

<xsd:element name="application" type="tns:ApplicationType" />

<xsd:element name="menus" type="tns:Menus" />

<xsd:complexType name="ApplicationType">

<xsd:sequence>

<xsd:element name="applicationName" type="xsd:string" />

<xsd:element name="applicationDescription" type="xsd:string" />

<xsd:element name="applicationNamespace" type="xsd:string" />

<xsd:element name="roles" type="tns:Roles" />

<xsd:element name="menus" type="tns:Menus" />

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Roles">

<xsd:sequence>

<xsd:element name="role" minOccurs="0" maxOccurs="unbounded">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="roleName" type="xsd:string" />

<xsd:element name="roleDescription" type="xsd:string" />

<xsd:element name="permissions" type="Permissions" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Permissions">

<xsd:sequence>

<xsd:element name="permission" minOccurs="0" maxOccurs="unbounded">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="permissionName" type="xsd:string" />

<xsd:element name="permissionDescription" type="xsd:string" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Menus">

<xsd:sequence>

<xsd:element name="menu" minOccurs="0" maxOccurs="unbounded">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="menuName" type="xsd:string" />

<xsd:element name="menuDescription" type="xsd:string" />

<xsd:element name="menuText" type="xsd:string" />

<xsd:element name="menuLink" type="xsd:string" />

<xsd:element name="menuDisplaySequence" type="xsd:integer" />

<xsd:element ref="menus" minOccurs="0" maxOccurs="unbounded" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

</xsd:schema

## Export Security Assignments Schema

<?xml version="1.0" encoding="utf-8"?>

<xsd:schema id="SecurityAssignmentsExport"

targetNamespace="http://www.fhlb.com/AppSecurity/SecurityAssignmentsExport.xsd"

elementFormDefault="qualified"

xmlns="http://www.fhlb.com/AppSecurity/SecurityAssignmentsExport.xsd"

xmlns:tns="http://www.fhlb.com/AppSecurity/SecurityAssignmentsExport.xsd"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

>

<xsd:element name="comment" type="xsd:string" />

<xsd:element name="exportDate" type="xsd:dateTime" />

<xsd:element name="exportUser" type="xsd:string" />

<xsd:element name="application" type="tns:ApplicationType" />

<xsd:complexType name="ApplicationType">

<xsd:sequence>

<xsd:element name="applicationName" type="xsd:string" />

<xsd:element name="applicationDescription" type="xsd:string" />

<xsd:element name="applicationNamespace" type="xsd:string" />

<xsd:element name="identity" type="tns:Identities" />

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Identities">

<xsd:sequence>

<xsd:element name="identity" minOccurs="0" maxOccurs="unbounded">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="identityName" type="xsd:string" />

<xsd:element name="identityDescription" type="xsd:string" />

<xsd:element name="identityAlias" type="xsd:string" />

<xsd:element name="permissions" type="Permissions" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="Permissions">

<xsd:sequence>

<xsd:element name="permission" minOccurs="0" maxOccurs="unbounded">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="permissionName" type="xsd:string" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

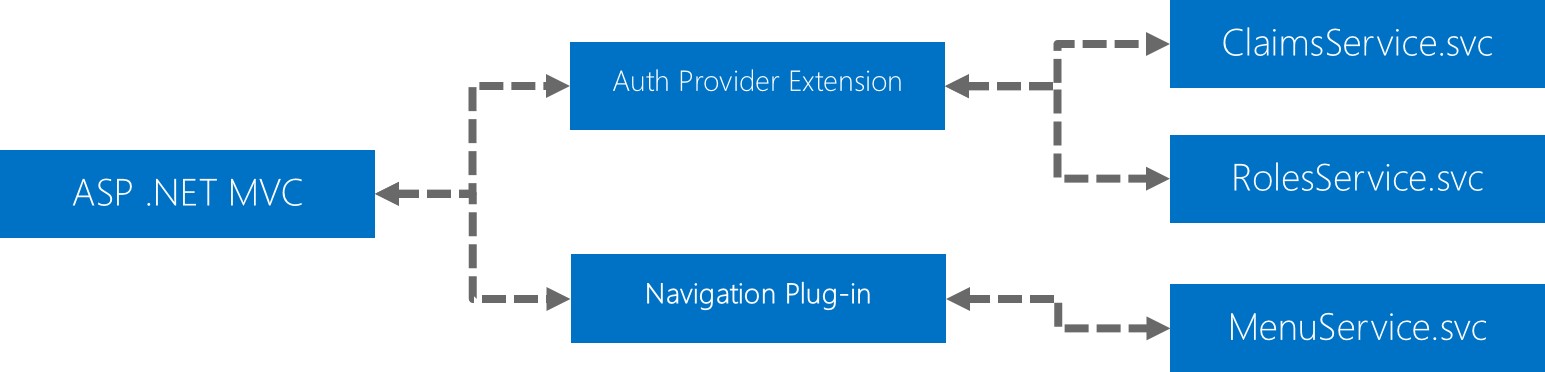
</xsd:sequence>

</xsd:complexType>

</xsd:schema>

## Security Services Endpoints

The security services endpoints will be implemented as WCF services and .svc URIs. The authentication provider extension will be the consumer for these Web services, surfacing security data to the ASP .NET MVC app. The MenuService.svc provides user interface elements.

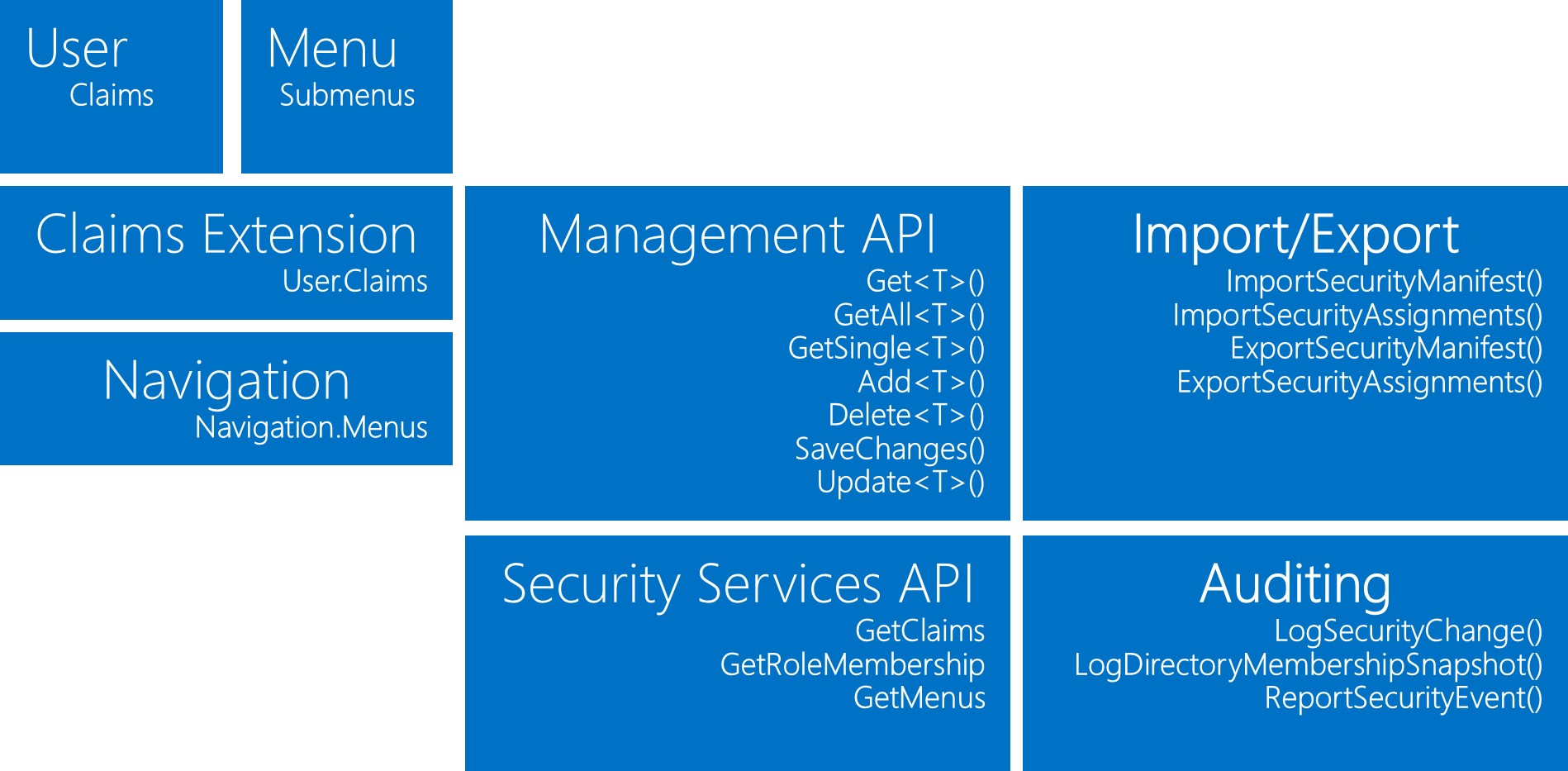


# 

# Application Programming Model

The application programming model for ASP .NET MVC application using the CALF Security Framework should deliver transparent and seamless access to security and menu navigation configuration to .NET developers.

The security permissions will be surfaced via claims augmentation to the application’s security principal. The menu configuration is delivered via a plug-in component (.NET assembly) that is securely accessible to the ASP .NET MVC application.



# Security Authorization Provider Extension

The authorization provider extension will primarily be responsible for filling in custom claim data to facilitate role and permission privileges in the .NET application. The code snippet below illustrates how custom claims can be injected into the security principal profile.

// Note the authenticationType must match the one defined in CookieAuthenticationOptions.AuthenticationType

var userIdentity = await manager.CreateIdentityAsync(this, DefaultAuthenticationTypes.ApplicationCookie);

// Add custom user claims here

userIdentity.AddClaim(new Claim("myCustomClaim", "value of claim"));

return userIdentity;

# 

# Security Services API

Security Services API is implemented using a request-response Message Exchange Pattern (MEP). The MessageContracts define the messaging interfaces for the WCF-based service endpoints.

## GetClaims

### GetClaimsRequest

[MessageContract]

public class GetClaimsequest : MessageRequest

{

[MessageBodyMember]

public string UserId { get; set; }

}

### GetClaimsResponse

[MessageContract]

public class GetClaimsMessageResponse : MessageResponse

{

[MessageBodyMember]

public ClaimsDTO Advance { get; set; }

[MessageBodyMember]

public IEnumerable<ClaimsDTO> ClaimsList { get; set; }

}

## IsInRole

### GetRoleMembershipRequest

[MessageContract]

public class GetRoleMembershipMessageRequest : MessageRequest

{

[MessageBodyMember]

public string UserId { get; set; }

[MessageBodyMember]

public string Role { get; set; }

}

### GetRoleMembershipResponse

[MessageContract]

public class GetRoleMembershipMessageResponse : MessageResponse

{

[MessageBodyMember]

public boolean IsInRole { get; set; }

}

## GetMenus

### GetMenusRequest

[MessageContract]

public class GetMenusMessageRequest : MessageRequest

{

[MessageBodyMember]

public int AppId { get; set; }

}

### GetMenusResponse

[MessageContract]

public class GetMenusMessageResponse : MessageResponse

{

[MessageBodyMember]

public MenuDTO Advance { get; set; }

[MessageBodyMember]

public IEnumerable<MenuDTO> MenuList { get; set; }

}

# Security Management API

Security Management API is implemented with .NET Entity Framework using a generic repository pattern. The Management API provides CRUD services for App Security Container entities.

public class AppSecurityRepository : IAppSecurityRepository

{

private readonly AppSecurityEntities \_db = new AppSecurityEntities();

public IEnumerable<T> Get<T>(Expression<Func<T, bool>> predicate) where T : class

{

return \_db.Set<T>().AsExpandable().Where(predicate);

}

public IEnumerable<T> GetAll<T>() where T : class

{

return \_db.Set<T>();

}

public T GetSingle<T>(Expression<Func<T, bool>> predicate) where T : class

{

return \_db.Set<T>().AsExpandable().Where(predicate).FirstOrDefault();

}

public T Add<T>(T entity) where T : class

{

\_db.Set<T>().Add(entity);

return entity;

}

public void Delete<T>(Expression<Func<T, bool>> predicate) where T : class

{

\_db.Set<T>().Remove(\_db.Set<T>().Find(predicate));

}

public void SaveChanges()

{

\_db.SaveChanges();

}

public void Update<T>(T entity) where T : class

{

\_db.Entry(entity).State = EntityState.Modified;

}

#region IDisposable Support

private bool disposedValue = false; // To detect redundant calls

protected virtual void Dispose(bool disposing)

{

if (!disposedValue)

{

if (disposing)

{

\_db.Dispose();

}

disposedValue = true;

}

}

public void Dispose()

{

Dispose(true)

}

#endregion

}

# Security Import-Export API

Security Import-Export API is implemented via an XML markup processor that will validate security manifest and security assignment input XML files against corresponding schemas.

The import API will read in the XML files and update the App Security Container with the security information. The export API will emit XML files containing security manifest and assignment data based on parameters requested in the PowerShell script. The import/export API will support the migration of the App Security Container configuration changes across server environments.

public class AppSecurityImporter

{

public boolean ImportSecurityManifest(string manifestPath)

{

}

public boolean ImportSecurityAssignments(string manifestPath)

{

}

}

public class AppSecurityExporter

{

public boolean ExportSecurityManifest(string appName)

{

}

public boolean ExportSecurityAssignments(string appName)

{

}

}

# Security Auditing API

Security Auditing API provides journaling history of security configuration changes made in the App Security Container. The API will leverage FHLB.Common.Core logging services using Log4NET instrumentation.

using FHLB.Common.Core;

public class AppSecurityAuditor

{

public boolean LogSecurityChange(string appName, Dictionary oldValues, Dictionary newValues)

{

}

public boolean LogDirectoryMembershipSnapshot(string appName, string userName)

{

}

public void ReportSecurityEvent(string event)

{

}Get

}

# Scalability and Reliability

## Security Caching

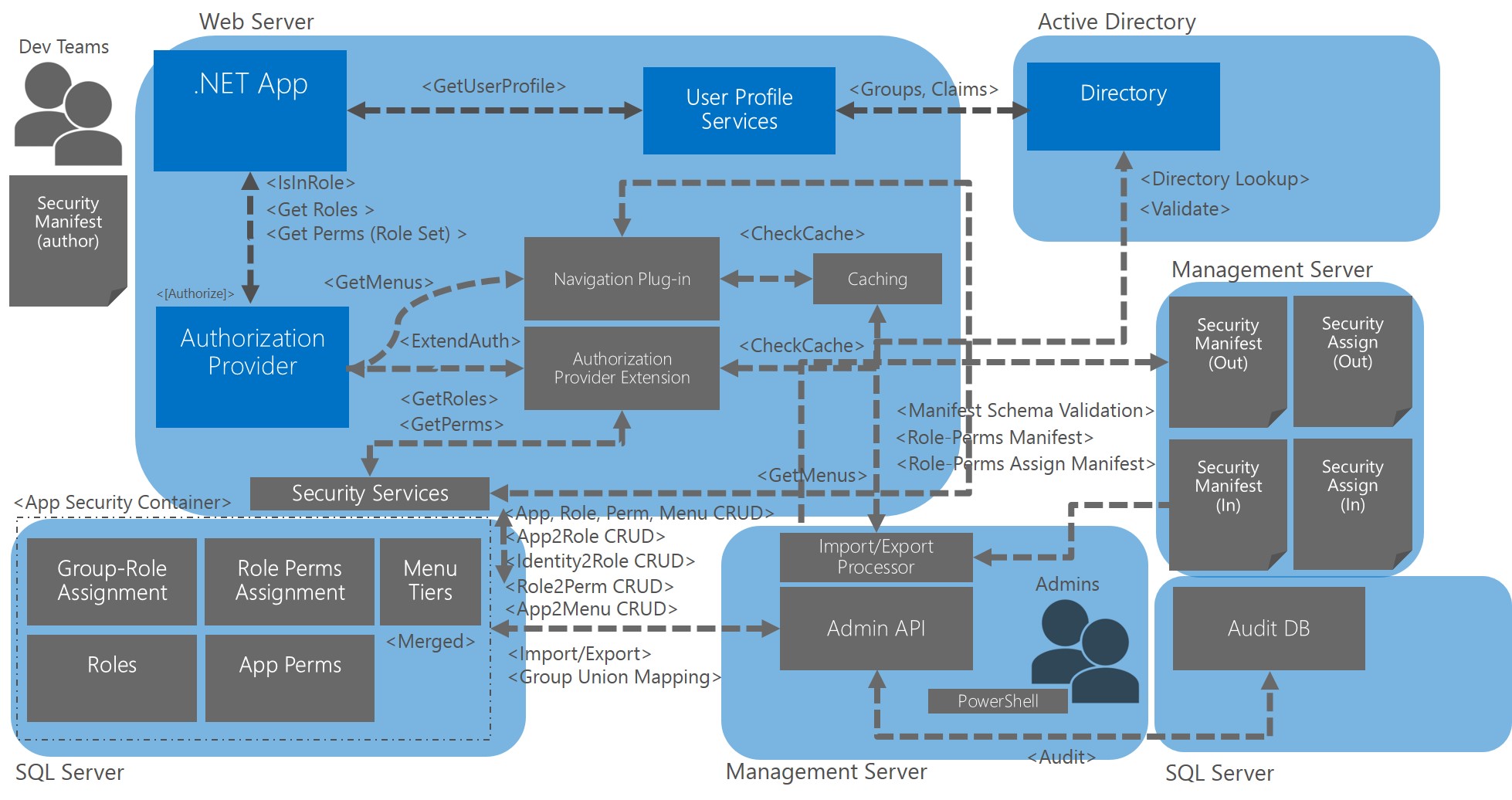
Security caching will leverage the facilities provided by FHLB.Common.Core library to maintain cached data with appropriate refresh policies to ensure integrity of application state. The security caching will be designed to be configurable so that policies can be dynamically changed. Caching will help to save I/O latency incurred from requesting the security data from the App Security Container.

## Security Exception Monitoring

Security exception monitoring will be key to ensure high reliability in order to take on proactive measures for remediation. Security exceptions will be reported to the Windows Event Log using Log4Net logging framework.

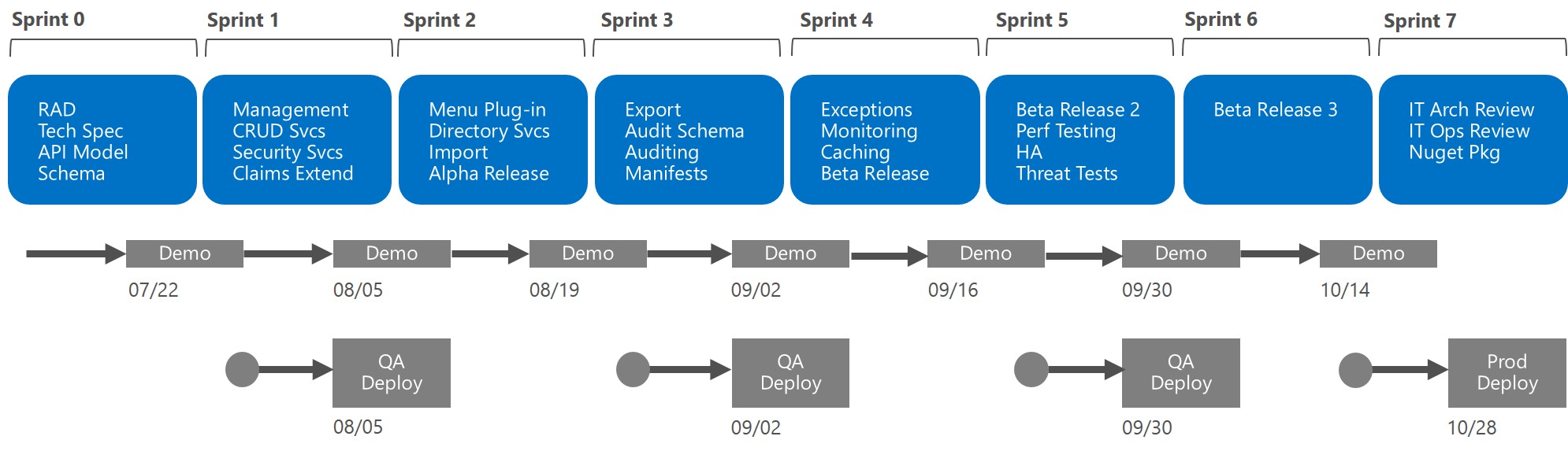
# Deployment Model

The following model illustates the physical server deployment locations for the components in the architecture.



# Workstream and Enterprise Lifecycle

The workstream structure and projected timeline below provides delivery schedule for CALF Security Framework services. The framework delivery is expected to go through one alpha and three beta cycles in advance of production release.



The enterprise lifecycle stakeholders need to be incorporated in the design and review process for the security framework to ensure readiness for production acceptance. An enterprise lifecycle journey map will be developed to illustrate how the framework should be adapted across IT processes and organization, including operations, change management and security.