M2M Authentication Flow

1. Schema - server store the secrets shared to the app

```
CREATE TABLE [dbo].[ExternalAppAuthDetails](
    [ExternalApplicationId] [int] IDENTITY(1,1) NOT NULL,
    [ApplicationName] [nvarchar](80) NULL,
    [ClientId] [nvarchar](80) NULL,
    [ClientSecret] [nvarchar](128) NULL,
    [IsActive] [bit] NOT NULL,
    [CreatedBy] [varchar](256) NOT NULL,
    [CreatedDate] [datetime] NOT NULL,
    [ModifiedBy] [varchar](256) NULL,
    [ModifiedDate] [datetime] NULL
)
GO
```

a.

```
CREATE TABLE [dbo].[ExternalAppAuthDetails](
    [ExternalApplicationId] [int] IDENTITY(1,1) NOT NULL,
    [ApplicationName] [nvarchar](80) NULL,
    [ClientId] [nvarchar](80) NULL,
    [ClientSecret] [nvarchar](128) NULL,
    [IsActive] [bit] NOT NULL,
    [CreatedBy] [varchar](256) NOT NULL,
    [CreatedDate] [datetime] NOT NULL,
    [ModifiedBy] [varchar](256) NULL,
    [ModifiedDate] [datetime] NULL
)
GO
```

2. Below authentication server code is used to register to the application

```
1 reference | dileep.ravula, 54 days ago | 3 authors, 3 changes
public async Task<bool> RegisterExternalApp(RegisterExternalAppRequest externalAppDetails)
{
    if (externalAppDetails == null)
    {
        throw new CustomException(ErrorConstants.ExternalAppRegistrationFailed, System.Net.HttpStatusCode.BadRequest);
    }

    externalAppDetails.ClientSecret = Convert.ToBase64String(RandomNumberGenerator.GetBytes(64));
    if (string.IsNullOrWhiteSpace(externalAppDetails.ClientId))
    {
        externalAppDetails.ClientId = Guid.NewGuid().ToString();
    }
    else
    {
        var isExternalAppExists = await _externalAppAuthRepository.IsExternalAppExists(externalAppDetails.ClientId);

        if (isExternalAppExists)
        {
            throw new CustomException(ErrorConstants.ExternalAppAlreadyExists, System.Net.HttpStatusCode.Conflict);
        }
    }

    var externalAppAuthDetails = new ExternalAppAuthDetail()
    {
        IsActive = true,
        ApplicationName = externalAppDetails.ApplicationName,
        ClientId = externalAppDetails.ClientId,
        ClientSecret = externalAppDetails.ClientSecret,
        CreatedBy = "dbo",
        CreatedDate = DateTime.UtcNow
    };

    _ribbonDbContext.ExternalAppAuthDetails.Add(externalAppAuthDetails);
    var result = _ribbonDbContext.SaveChanges();
    return result != 0;
}
```

a.

```
public async Task<bool> RegisterExternalApp(RegisterExternalAppRequest externalAppDetails)
        {
            externalAppDetails.ClientSecret = Convert.ToBase64String(RandomNumberGenerator.GetBytes(64));
            if (string.IsNullOrWhiteSpace(externalAppDetails.ClientId))
```

```
            {
                externalAppDetails.ClientId = Guid.NewGuid().ToString();
            }
            else
            {
                var isExternalAppExists = await _externalAppAuthRepository.IsExternalAppExists(externalApp
                if (isExternalAppExists)
                {
                    throw new CustomException(ErrorConstants.ExternalAppAlreadyExists, System.Net.HttpStat
                }
            }

            var externalAppAuthDetails = new ExternalAppAuthDetail()
            {
                IsActive = true,
                ApplicationName = externalAppDetails.ApplicationName,
                ClientId = externalAppDetails.ClientId,
                ClientSecret = externalAppDetails.ClientSecret,
                CreatedBy = "dbo",
                CreatedDate = DateTime.UtcNow
            };

            _ribbonDbContext.ExternalAppAuthDetails.Add(externalAppAuthDetails);
            var result = _ribbonDbContext.SaveChanges();
            return result != 0;
        }
```

3. Below authentication server code is used to login (acquire access token) in to the application

```
1 reference | juhitha, 61 days ago | 2 authors, 2 changes
public async Task<LoginResponse> ExternalAppLogin(ExternalAppLoginRequest loginRequest)
{
    var applicationUser = await _ribbonDbContext.ExternalAppAuthDetails.FirstOrDefaultAsync(_ => !string.IsNullOrWhiteSpace
      (loginRequest.ClientId) && _.ClientId == loginRequest.ClientId);

    var applicationClaims = new Dictionary<string, object>
    {
        { JwtRegisteredClaimNames.Sub, applicationUser.ClientId },
        { Constants.UserNameClaim, applicationUser.ApplicationName }
    };

    var key = Encoding.UTF8.GetBytes(_externalApplicationJWTSettings.Key);

    var signingInCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature);

    var tokenHandler = new JwtSecurityTokenHandler();
    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Claims = applicationClaims,
        Expires = DateTime.UtcNow.AddMinutes(_externalApplicationJWTSettings.DurationInMinutes),
        SigningCredentials = signingInCredentials,
        Audience = _externalApplicationJWTSettings.Audience,
        Issuer = _externalApplicationJWTSettings.Issuer,
        IssuedAt = DateTime.UtcNow
    };

    var ctoken = tokenHandler.CreateToken(tokenDescriptor);
    var token = tokenHandler.WriteToken(ctoken);

    return new LoginResponse()
    {
        IsLoginSuccess = true,
        TokenResponse = new TokenResponse()
        {
            Token = token,
            Expiration = _externalApplicationJWTSettings.DurationInMinutes
        }
    };
}
```

a.

```
public async Task<LoginResponse> ExternalAppLogin(ExternalAppLoginRequest loginRequest)
        {
            var applicationUser = await _ribbonDbContext.ExternalAppAuthDetails.FirstOrDefaultAsync(_ => !
            var applicationClaims = new Dictionary<string, object>
            {
                { JwtRegisteredClaimNames.Sub, applicationUser.ClientId },
                { Constants.UserNameClaim, applicationUser.ApplicationName }
            };
```

```
            var key = Encoding.UTF8.GetBytes(_externalApplicationJWTSettings.Key);
            var signingInCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgor
            var tokenHandler = new JwtSecurityTokenHandler();
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Claims = applicationClaims,
                Expires = DateTime.UtcNow.AddMinutes(_externalApplicationJWTSettings.DurationInMinutes),
                SigningCredentials = signingInCredentials,
                Audience = _externalApplicationJWTSettings.Audience,
                Issuer = _externalApplicationJWTSettings.Issuer,
                IssuedAt = DateTime.UtcNow
            };

            var ctoken = tokenHandler.CreateToken(tokenDescriptor);
            var token = tokenHandler.WriteToken(ctoken);

            return new LoginResponse()
            {
                IsLoginSuccess = true,
                TokenResponse = new TokenResponse()
                {
                    Token = token,
                    Expiration = _externalApplicationJWTSettings.DurationInMinutes
                }
            };
        }
```

```
"ExternalAppJwtTokenSettings": {
  "Key": "A1CF4B7DC4C4175B6618DE4F55CA4A1CF4B7DC4C4175B6618DE4F55CA4A1CF4B7DC4C4175B6618DE4F55CA4",
  "Issuer": "https://localhost:7291/",
  "Audience": "IdentityUser",
  "DurationInMinutes": 30
},
```

4. Below external application code is used to request authentication login (acquire access token) in to the application
   a. This call should be made with clientId and clientsceret generated using authentication server registration endpoint.

```
1 reference | Yuva C, 8 days ago | 1 author, 1 change
private async Task<ExternalAppLoginResponse> GetRibbonAuthenticationToken(PatientInformationRequest patientInformation)
{

    var loginRequest = new ExternalAppLoginRequest
    {
        ClientId = _ribbonConfigurations.AuthenticationConfigurations.ClientId,
        ClientSecret = _ribbonConfigurations.AuthenticationConfigurations.ClientSecret,
        EHRType = _identity.FindFirst(CommonConstants.EHRTypeClaim)?.Value,
        TenantName = _identity.FindFirst(CommonConstants.TenantNameClaim)?.Value,
        TenantApplicationId = _identity.FindFirst(CommonConstants.TenantApplicationInformationIdClaim)?.Value,
        SessionId = _identity.FindFirst(CommonConstants.SessionIdClaim)?.Value ?? Guid.NewGuid().ToString(),
        PatientInformation = patientInformation
    };
    var loginResponse = await _httpRequestHandler.PostHttpRequest
      (_ribbonConfigurations.AuthenticationConfigurations.AuthenticationUrl, loginRequest);
    if (loginResponse.StatusCode != System.Net.HttpStatusCode.OK)
    {
        return null;
    }
    string loginResponseContent = await loginResponse.Content.ReadAsStringAsync();
    var loginResponseObject = JsonConvert.DeserializeObject<APIResponse<ExternalAppLoginResponse>>(loginResponseContent);
    return loginResponseObject.Data;
}
```

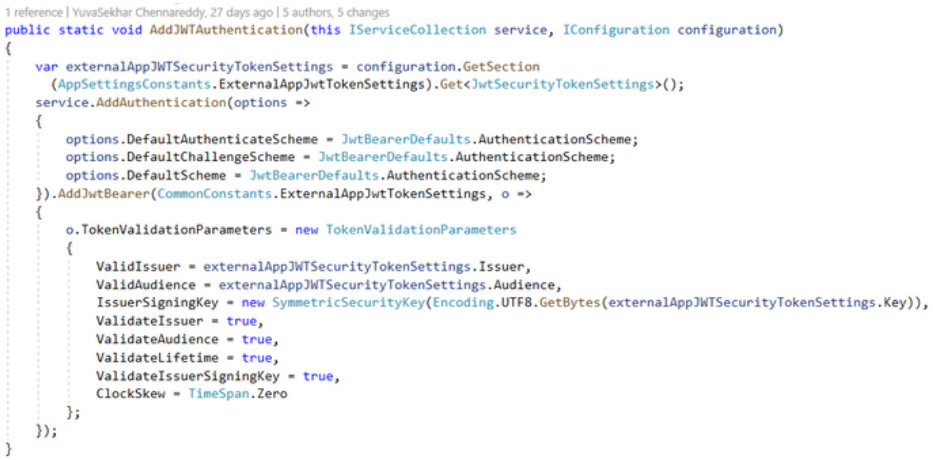
```
"RibbonConfigurations": {
  "AuthenticationConfigurations": {
    "AuthenticationUrl": "https://localhost:7291/api/ExternalAppAuthentication/LoginApplication",
    "ClientId": "deeca889-e0b4-426e-a541-5db292a42d33",
    "ClientSecret": "7r1NYg1s+A+j2Iw/5erc15O9EubGjcrDRJcdpQZMp1xj+7RKmyWfLM1fzoW7xyKo07JvHCAsnZrc5SB7Lpj8Ng=="
  }
},
```

   b.

```
private async Task<ExternalAppLoginResponse> GetRibbonAuthenticationToken(PatientInformationRequest patien
        {
            var loginRequest = new ExternalAppLoginRequest
            {
                ClientId = _ribbonConfigurations.AuthenticationConfigurations.ClientId,
```

```
                ClientSecret = _ribbonConfigurations.AuthenticationConfigurations.ClientSecret,
                EHRType = _identity.FindFirst(CommonConstants.EHRTypeClaim)?.Value,
                TenantName = _identity.FindFirst(CommonConstants.TenantNameClaim)?.Value,
                TenantApplicationId = _identity.FindFirst(CommonConstants.TenantApplicationInformationIdCl
                SessionId = _identity.FindFirst(CommonConstants.SessionIdClaim)?.Value ?? Guid.NewGuid().T
                PatientInformation = patientInformation
            };
            var loginResponse = await _httpRequestHandler.PostHttpRequest(_ribbonConfigurations.Authentica
            if (loginResponse.StatusCode != System.Net.HttpStatusCode.OK)
            {
                return null;
            }
            string loginResponseContent = await loginResponse.Content.ReadAsStringAsync();
            var loginResponseObject = JsonConvert.DeserializeObject<APIResponse<ExternalAppLoginResponse>>
            return loginResponseObject.Data;
        }
```

5. Once token is obtained the external application can use it to access the endpoints in the resource server (who has set-up with the same authentication server and jwt keys)



```
1 reference | YuvaSekhar Chennareddy, 27 days ago | 5 authors, 5 changes
public static void AddJWTAuthentication(this IServiceCollection service, IConfiguration configuration)
{
    var externalAppJWTSecurityTokenSettings = configuration.GetSection
      (AppSettingsConstants.ExternalAppJwtTokenSettings).Get<JwtSecurityTokenSettings>();
    service.AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
    }).AddJwtBearer(CommonConstants.ExternalAppJwtTokenSettings, o =>
    {
        o.TokenValidationParameters = new TokenValidationParameters
        {
            ValidIssuer = externalAppJWTSecurityTokenSettings.Issuer,
            ValidAudience = externalAppJWTSecurityTokenSettings.Audience,
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(externalAppJWTSecurityTokenSettings.Key)),
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ClockSkew = TimeSpan.Zero
        };
    });
}
```

a.

```
public static void AddJWTAuthentication(this IServiceCollection service, IConfiguration configuration)
        {
            var externalAppJWTSecurityTokenSettings = configuration.GetSection(AppSettingsConstants.Extern
            service.AddAuthentication(options =>
            {
                options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
                options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
                options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
            }).AddJwtBearer(CommonConstants.ExternalAppJwtTokenSettings, o =>
            {
                o.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidIssuer = externalAppJWTSecurityTokenSettings.Issuer,
                    ValidAudience = externalAppJWTSecurityTokenSettings.Audience,
                    IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(externalAppJWTSecur
                    ValidateIssuer = true,
                    ValidateAudience = true,
                    ValidateLifetime = true,
                    ValidateIssuerSigningKey = true,
                    ClockSkew = TimeSpan.Zero
                };
            });
        }
```

```
"ExternalAppJwtTokenSettings": {
  "Key": "A1CF4B7DC4C4175B6618DE4F55CA4A1CF4B7DC4C4175B6618DE4F55CA4A1CF4B7DC4C4175B6618DE4F55CA4",
  "Issuer": "https://localhost:7291/",
  "Audience": "IdentityUser",
  "DurationInMinutes": 30
},
```

```
[Route("api/[controller]")]
[ApiController]
[Authorize(AuthenticationSchemes = CommonConstants.ExternalAppJwtTokenSettings)]
3 references | Yuva Chennareddy, 11 days ago | 1 author, 1 change | 1 work item
public class ExternalAppInfoController : BaseController<ExternalAppInfoController>
{
    private readonly IStratificationRankService _stratificationRankService;
    private readonly IFileService _fileService;

    0 references | Yuva Chennareddy, 11 days ago | 1 author, 1 change | 1 work item
    public ExternalAppInfoController(IStratificationRankService stratificationRankService, IFileService fileService,
      ILogger<ExternalAppInfoController> logger) ...

    /// <summary>
    /// Get Patient health and risk info
    /// </summary>
    /// <param name="getPatientInformationRequest"></param>
    /// <returns></returns>
    [HttpGet("GetPatientHealthInfo")]
    0 references | Yuva Chennareddy, 11 days ago | 1 author, 1 change | 1 work item
    public async Task<IActionResult> GetPatientHealthInfo()...
```

6. EOF