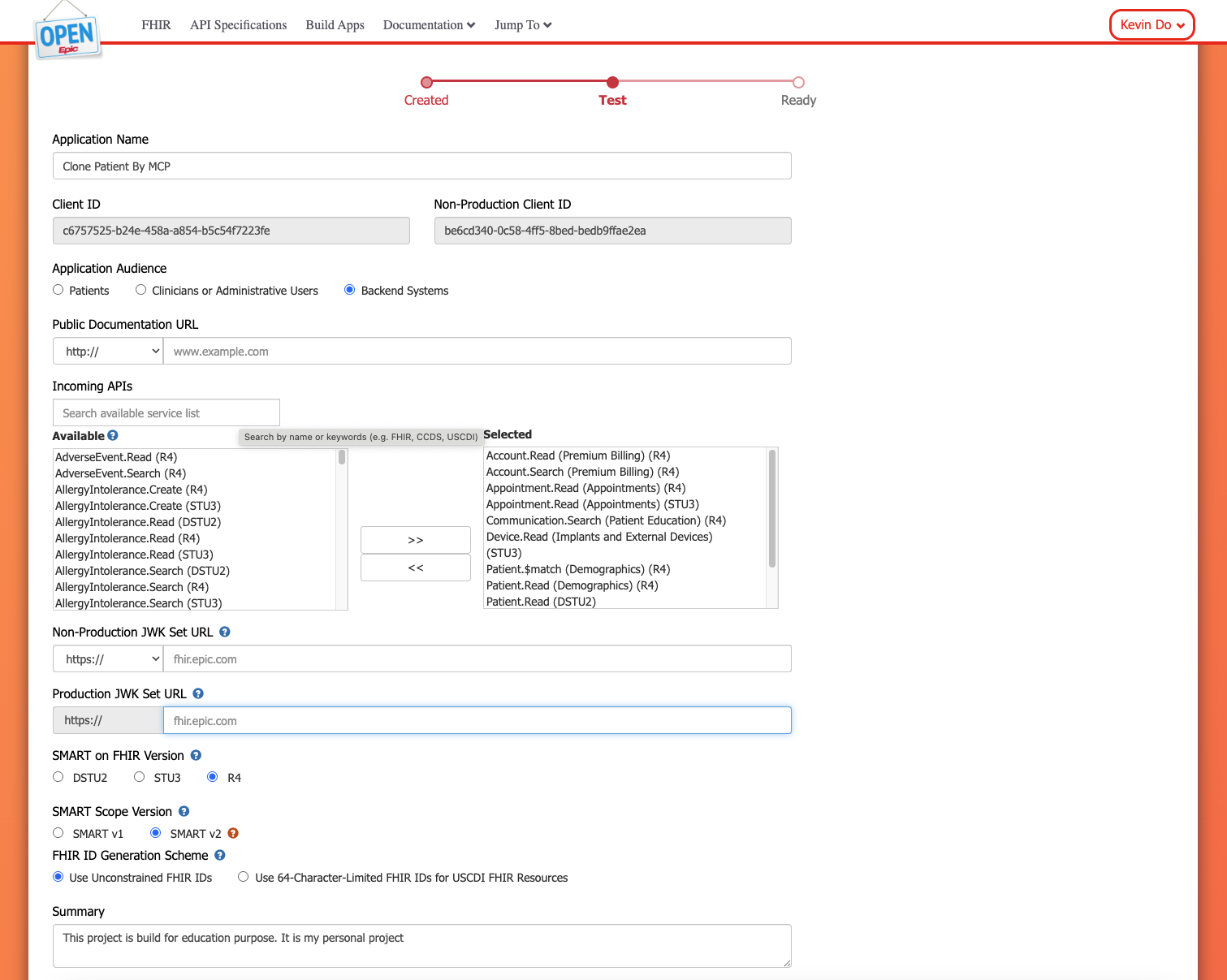
# Epic setup

## Create sandbox app

Go to Build App -> Create



Enter the application name and select the API you want to use. We’ll use the **Patient.Read (R4)** API to perform a patient search.

## Using a JWT to Obtain an Access Token for a Backend Service

### Create public private key for JWT token : OpenSSL

You can create a new private key named privatekey.pem using OpenSSL with the following command:

| openssl genrsa -**out** /path\_to\_key/privatekey.pem 2048 |
| --- |

Make sure the key length is at least 2048 bits.

For backend apps, you can export the public key to a base64 encoded X.509 certificate named publickey509.pem using this command:

| openssl req -**new** -x509 -key /path\_to\_key/privatekey.pem -**out** /path\_to\_key/publickey509.pem -subj '/CN=myapp' |
| --- |

Where '/CN=myapp' is the subject name (for example the app name) the key pair is for. The subject name does not have a functional impact in this case but it is required for creating an X.509 certificate.

### Publish public key

You must implement and expose an endpoint that Epic can call to obtain your public key, which is used to verify your JWT signatures.

**Note:** Youcan use ngrok to expose endpoint to allow Epic call and verified private key **at least once**

| **using** System.Security.Cryptography.X509Certificates; **using** DotNetEnv;  **var** builder = WebApplication.CreateBuilder(args); **var** app = builder.Build();  Env.Load("../.env");  **var** keyId = Environment.GetEnvironmentVariable("EPIC:KEYID");  **string** home = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile); **string** pemPath = Path.Combine(home, "publickey509.pem");  **string** pem = File.ReadAllText(pemPath); **var** cert = X509Certificate2.CreateFromPem(pem);   *// Extract RSA public key* **using** **var** rsa = cert.GetRSAPublicKey();  *// Export modulus/exponent for JWK* **var** parameters = rsa.ExportParameters(false);  **string** **Base64UrlEncode**(**byte**[] input) =>  Convert.ToBase64String(input)  .TrimEnd('=')  .Replace('+', '-')  .Replace('/', '\_');  **var** jwk = **new** {  kty = "RSA",  kid = keyId,  use = "sig",  alg = "RS384", *// must match your signing algorithm*  n = Base64UrlEncode(parameters.Modulus),  e = Base64UrlEncode(parameters.Exponent) };  app.MapGet("/.well-known/jwks.json", () => **new** { keys = **new**[] { jwk } });  app.Run(); |
| --- |

### Create JWT and make http request call

| **string** home = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile); **string** pemPath = Path.Combine(home, "privatekey.pem"); **string** pem = File.ReadAllText(pemPath); **var** rsa = RSA.Create(); rsa.ImportFromPem(pem);  **var** credentials = **new** SigningCredentials(**new** RsaSecurityKey(rsa), SecurityAlgorithms.RsaSha384);  **var** payload = **new** JwtPayload {  { "iss", \_config["EPIC:CLIENT\_ID"] },  { "sub", \_config["EPIC:CLIENT\_ID"] },  { "aud", \_config["EPIC:AUDIENCE"] },  { "kid", \_config["EPIC:KID"]},   { "jti", Guid.NewGuid().ToString() },  { "exp", DateTimeOffset.UtcNow.AddMinutes(4).ToUnixTimeSeconds() } }; |
| --- |

**Note:** The exp claim in the JWT must be set to a time no more than **5 minutes** after the token is issued. A value between **1 and 4 minutes** is recommended to ensure the JWT remains valid.

| **var** header = **new** JwtHeader(credentials);  **var** jwt = **new** JwtSecurityToken(header, payload);    **var** handler = **new** JwtSecurityTokenHandler();  **var** assertion = handler.WriteToken(jwt);    **var** content = **new** FormUrlEncodedContent(**new**[]  {  **new** KeyValuePair<**string**, **string**>("grant\_type", "client\_credentials"),  **new** KeyValuePair<**string**, **string**>("client\_id", \_config["EPIC:CLIENT\_ID"]),  **new** KeyValuePair<**string**, **string**>("client\_assertion\_type", "urn:ietf:params:oauth:client-assertion-type:jwt-bearer"),  **new** KeyValuePair<**string**, **string**>("client\_assertion", assertion )  });      **var** response = **await** \_client.PostAsync("https://fhir.epic.com/interconnect-fhir-oauth/oauth2/token", content);  **var** responseBody = **await** response.Content.ReadAsStringAsync();  **if** (!response.IsSuccessStatusCode)  {  Console.WriteLine($"Epic token request failed: {response.StatusCode} - {responseBody}");  **throw** **new** Exception($"Epic token request failed: {response.StatusCode} - {responseBody}");  }  **using** **var** doc = JsonDocument.Parse(responseBody);  **return** doc.RootElement.GetProperty("access\_token").GetString(); } |
| --- |

# MCP Server

## MCP server tools:

| **using** System.ComponentModel; **using** System.Globalization; **using** ModelContextProtocol.Server;  **namespace** **McpServer.Tools**;  [**McpServerToolType**] **public** **class** **GetPatientDataTool** {  **private** **readonly** IEpicClient \_epicClient;  **private** **readonly** IPostgresService \_postgresService;   **public** **GetPatientDataTool**(IEpicClient epicClient, IPostgresService postgresService)  {  \_epicClient = epicClient;  \_postgresService = postgresService;  }   [**McpServerTool, Description("This tools help get access token")**]  **public** **async** Task<**string**> **GetAccessToken**()  {  **try**  {  **return** **await** \_epicClient.GetAccessToken();  }  **catch** (Exception ex)  {  **return** ex.Message;  }  }   [**McpServerTool, Description(  "This tool helps get patient data, extract json result string to show how many patient found and " +  "detail of information found, after that prompt to the user if they want to use the tool to persist data into postgres, keep result of this tools to be the input of Persist Data Tool call")**]  **public** **async** Task<**string**> **GetPatientInformation**(**string** family, **string** given, **string** birthdate, **string** accessToken)  {  **try**  {  **if** (**string**.IsNullOrEmpty(family) || **string**.IsNullOrEmpty(given) || **string**.IsNullOrEmpty(birthdate))  {  **return** "Bad request these field cannot be null";  }   **if** (!DateTime.TryParseExact(  birthdate,  "yyyy-MM-dd",  CultureInfo.InvariantCulture,  DateTimeStyles.None,  **out** \_))  {  **return** "Bad request the birth date format must be yyyy-MM-dd";  }     **return** **await** \_epicClient.GetInformationOfPatient(family, given, birthdate, accessToken);    }  **catch** (Exception ex)  {  **return** ex.Message;  }  }   [**McpServerTool,  Description(  "This tool help persistence data for patient. This use input which is the result from Get Patient Information tool and also use given and family name from this call. This tool cannot be called if user haven't made any successfully call to register")**]  **public** **async** Task<**string**> **PersistPatientInformation**(**string** family, **string** given, **string** jsonResult)  {  **try**  {  **return** **await** \_postgresService.PersistPatentData(family, given, jsonResult);  }  **catch** (Exception ex)  {  **return** ex.Message;  }  } } |
| --- |

## Epic Client

| **using** System.IdentityModel.Tokens.Jwt; **using** System.Net.Http.Headers; **using** System.Security.Cryptography; **using** System.Text.Json; **using** Microsoft.IdentityModel.Tokens;  **namespace** **McpServer**;  **public** **class** **EpicClient** : **IEpicClient** {  **private** **readonly** IConfiguration \_config;  **private** **readonly** HttpClient \_client;  **private** **readonly** ILogger<EpicClient> \_logger;   **public** **EpicClient**(IConfiguration config, HttpClient client, ILogger<EpicClient> logger)  {  \_config = config;  \_client = client;  \_logger = logger;  }    **public** **async** Task<**string**> **GetAccessToken**()  {  **string** home = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile);  **string** pemPath = Path.Combine(home, "privatekey.pem");  **string** pem = File.ReadAllText(pemPath);  **var** rsa = RSA.Create();  rsa.ImportFromPem(pem);    **var** credentials = **new** SigningCredentials(**new** RsaSecurityKey(rsa), SecurityAlgorithms.RsaSha384);    **var** payload = **new** JwtPayload  {  { "iss", \_config["EPIC:CLIENT\_ID"] },  { "sub", \_config["EPIC:CLIENT\_ID"] },  { "aud", \_config["EPIC:AUDIENCE"] },  { "kid", \_config["EPIC:KID"]},   { "jti", Guid.NewGuid().ToString() },  { "exp", DateTimeOffset.UtcNow.AddMinutes(4).ToUnixTimeSeconds() }  };    **var** header = **new** JwtHeader(credentials);  **var** jwt = **new** JwtSecurityToken(header, payload);    **var** handler = **new** JwtSecurityTokenHandler();  **var** assertion = handler.WriteToken(jwt);    **var** content = **new** FormUrlEncodedContent(**new**[]  {  **new** KeyValuePair<**string**, **string**>("grant\_type", "client\_credentials"),  **new** KeyValuePair<**string**, **string**>("client\_id", \_config["EPIC:CLIENT\_ID"]),  **new** KeyValuePair<**string**, **string**>("client\_assertion\_type", "urn:ietf:params:oauth:client-assertion-type:jwt-bearer"),  **new** KeyValuePair<**string**, **string**>("client\_assertion", assertion )  });      **var** response = **await** \_client.PostAsync("https://fhir.epic.com/interconnect-fhir-oauth/oauth2/token", content);  **var** responseBody = **await** response.Content.ReadAsStringAsync();  **if** (!response.IsSuccessStatusCode)  {  Console.WriteLine($"Epic token request failed: {response.StatusCode} - {responseBody}");  **throw** **new** Exception($"Epic token request failed: {response.StatusCode} - {responseBody}");  }  **using** **var** doc = JsonDocument.Parse(responseBody);  **return** doc.RootElement.GetProperty("access\_token").GetString();  }   **public** **async** Task<**string**> **GetInformationOfPatient**(**string** family, **string** given, **string** birthdate, **string** accessToken)  {  **if** (**string**.IsNullOrEmpty(\_config["EPIC:HOST"]))  {  \_logger.LogError("Cannot get the EPIC:HOST environment key value");  **throw** **new** Exception("The epic host is not found in the environment");  }     \_client.DefaultRequestHeaders.Add("Accept", "application/fhir+json");  \_client.DefaultRequestHeaders.Authorization = **new** AuthenticationHeaderValue("Bearer", accessToken);    **string** url = \_config["EPIC:HOST"] + "api/FHIR/R4/Patient?" + $"family={family}&given={given}&birthdate={birthdate}";  \_logger.LogInformation("Epic host request received: {url}", url);  **var** response = **await** \_client.GetAsync(url);  **return** **await** response.Content.ReadAsStringAsync();  } } |
| --- |

**Note:** The Patient Search method requires family, given and birthdate as the minimum parameters.

## PostgresService

| **using** McpServer.DbContext; **using** McpServer.Model;  **namespace** **McpServer**;  **public** **class** **PostgresService** : **IPostgresService** {  **private** **readonly** ApplicationDbContext \_context;   **public** **PostgresService**(ApplicationDbContext context)  {  \_context = context;  }  **public** **async** Task<**string**> **PersistPatentData**(**string** family, **string** given, **string** data)  {  **var** patientData = **new** Patient() { FamilyName = family, GivenName = given, Data = data };  **await** \_context.Patients.AddAsync(patientData);  **await** \_context.SaveChangesAsync();  **return** "Successfully persisted patient data";  } } |
| --- |

# Flow Builder configuration

## MCP Tools HTTP SSE

| Name = whatever\_name URL = HOST/sse |
| --- |