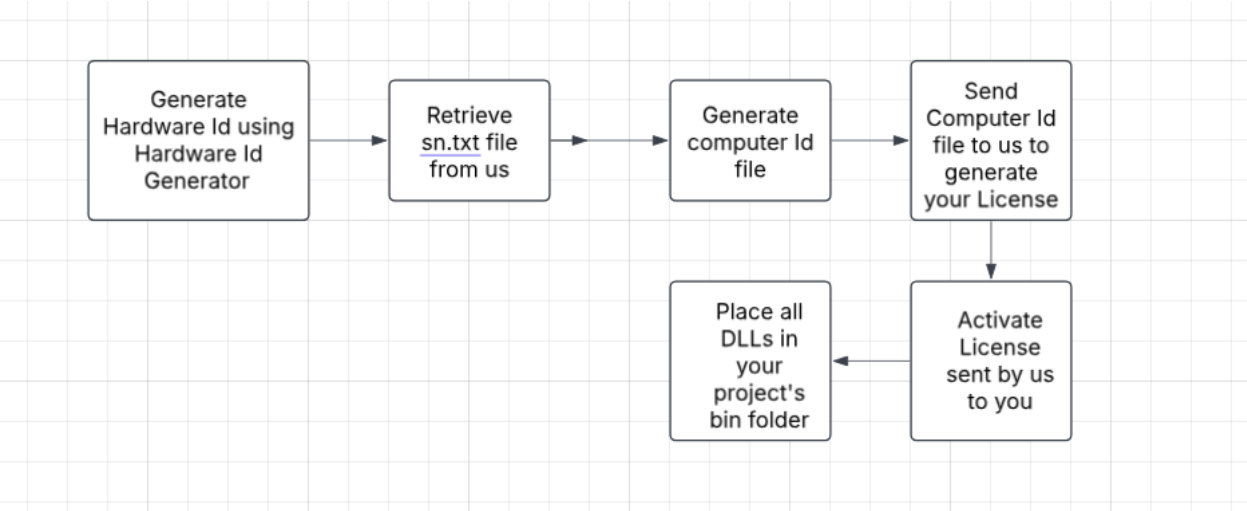
Overall Flow:



Prerequisites:

1. Generating a Hardware ID

1. Navigate to the Artifacts folder in your root directory.
2. Open the HardwareId Generator folder.
3. Double-click on MxFace.SDK.DeviceIdGenerator.exe.
4. Note down the generated Hardware ID displayed in the format:
   Example: **Generated Device Id: MQAS5AGX3VWKYGH9ZNR07EHSA4BMDKE0JDXH7WQGAH8WH33BJ8W0**
5. Provide the retrieved **Hardware ID** to Mantra for generating your License File.

Example: **Generated Device Id: MQAS5AGX3VWKYGH9ZNR07EHSA4BMDKE0JDXH7WQGAH8WH33BJ8W0**

2. Activating License Files

1. Navigate to the `Artifacts` folder in your root directory.
2. Open Command Prompt as **Run As Administrator**.

3. Copy the path of the `IdGen` folder (e.g., `D:\Activation\IDGen`) and change the Command Prompt directory to this folder using:

   ```
   Ex: cd /d D:\Activation\IDGen
   ```

4. Ensure the `IdGen` folder contains `id_gen.exe` and `sn.txt` files.
5. Run the following command in Command Prompt:

   ```
   id_gen.exe sn.txt hardware.id
   ```

6. Provide the retrieved **Hardware ID** to us for generating your License File.
7. Navigate back to the **Activation** folder in Command Prompt using:

   ```
   cd ..
   ```

8. Once in the Activation folder, run the command:

   ```
   pg.exe -install
   ```

Cloning the Sample Project from GitHub

1. Clone the sample project from GitHub.
2. Open the cloned project in **Visual Studio/Visual Studio Code**.
3. Open the terminal and run:

   ```
   dotnet build
   ```

4. Open the **Project Folder** in File Explorer and navigate to:

   ```
   \Artifacts\Activation\bin
   ```

5. Copy all **DLL** files from this folder.
6. Navigate to:

   ```
   \bin
   ```

7. Paste all copied DLL files here.

8. You will receive a **license file** named `Face.lic`. Place this file in the **bin** folder where you just pasted the DLL files.

3. System Requirements

1. x86-64 (64-bit) processors are highly recommended.
2. AVX2 support is highly recommended.
3. Intel Xeon with Intel AVX2 and AVX-512, Intel Core processors with Intel AVX2
4. 8 vCPUs on virtualized environment on Intel Xeon processors
5. Minimum 4 GB of free RAM
6. Minimum 256 GB free storage for storing the extracted templates
7. Temporary internet access to activate the license

3.1 Linux specific requirements

- Linux 4.9 kernel or newer is required.
- glibc 2.24 or newer GStreamer 1.10.x or newer with gst-plugin-base and gst-plugin-good is required for face capture using camera/webcam or rtsp video.
- libgudev-1.0 230 or newer (for camera and/or microphone usage)
- alsa-lib 1.1.6 or newer (for voice capture)
- gcc 6.3 or newer
- GNU Make 3.81 or newer
- Java SE JDK 8 or newer
- Python 3.x

3.2 Microsoft Windows specific requirements

- Microsoft Windows 7 / 8 / 10 / 11.
- Microsoft .NET framework 4.5 (for .NET components usage)
- Microsoft Visual Studio 2012 or newer
- Microsoft DirectX 9.0 or later (for face capture using camera/webcam)
- Java SE JDK 8 or newer
- Python 3.x

Database Setup:

You need to connect to Database in order to enroll, search and match faces.

Let's set up the Database in your system:

1. Install ODBC Driver:
    a. If you're using PostgreSQL database, download "**psqlodbc_x64.msi**" from https://www.postgresql.org/ftp/odbc/releases/REL-17_00_0004-mimalloc/
    b. If you're using MySQL database, download "Windows (x86, 64-bit), MSI Installer" from https://dev.mysql.com/downloads/connector/odbc/
    c. If you're using MsSQL database, click on "Download Microsoft ODBC Driver 18 for SQL Server (x64)" from https://learn.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server?view=sql-server-ver16

2. After installing driver for your database, double click on Newley installed application and set it up.
3. Then, press Windows button on your keyboard and type "odbc Data Sources (64-bit)" and open the application.
4. Click on the Add button on the right side on odbc, Select a driver named: PostgreSQL ANSI(x64)
5. Add your database configuration and Test the connection.
6. If Successful, create the below table on your database by running below query:

Column Name	Data Type	Constraints	Default Value
Id	Integer	NOT NULL, PRIMARY KEY	NULL
SubjectId	character varying(45)		NULL
Template	bytea		NULL
Group	character varying(45)		NULL
ClientId	integer		NULL

And you're ready to enroll, search and match faces 😊

Interface Overview:

The **MxFace Face SDK** provides comprehensive face management capabilities through it's main interface:

- **IFaceProcessor**

This documentation details interface and its methods.

Interface Details

IFaceProcessor Interface

Handles all face operations like Upload, enrollment, verification, and searching.

Upload Method

```
Task<UploadFaceResponse> Upload(UploadFaceRequest model)
```

Purpose: Detect and measure facial features like, age and gender.

Request:

- The `UploadFaceRequest` object contains the following fields:

Field Name	Type	Required	Description
`encoded_image`	`string`	Yes	Base64-encoded image of the face.
`MinimumDistanceBetweenEyeThreshold`	`double`	No	Minimum required distance between the eyes (optional).

Returns:

- The `UploadFaceResponse` object contains the following fields:

Field Name	Type	Description
`FaceAnalytics`	`object`	facial analytics in images facial analytics in images.
`Quality`	`double`	The quality score of the uploaded face image (scale 0-100).
`FaceRectangle`	`object`	Bounding of the detected face in the image.

- ## FaceAnalytics Object

Field Name	Type	Description
`Age`	`string`	Estimated age of the detected face.
`Gender`	`string`	Detected gender of the face (e.g., Male, Female).
`Emotion`	`string`	Predominant emotion detected on the face (e.g., Happiness, Sadness).

- ## FaceRectangle Object

Field Name	Type	Description
`x`	`int`	X-coordinate of the face rectangle.
`y`	`int`	Y-coordinate of the face rectangle.
`height`	`int`	Height of the detected face.
`width`	`int`	Width of the detected face.

Key Features:

- Extracts age, gender, and emotion from the detected face.
- Provides a quality score for the uploaded face image.
- Identifies the face and returns its bounding box coordinates.
- License validation.
- Comprehensive error handling.

Enroll Method

```
Task Enroll(string source, string externalId, string group)
```

Purpose: Stores a new face in the database for future matching.

Parameters:

Name	Type	Required	Description
source	string	Yes	Raw face image data in base64 format.

externalId	string	Yes	Unique identifier for the face record.
group	string	Yes	Organizational group identifier for categorization.

Returns:

- `EnrollmentResponse` object containing:
  - Success status (`200` for success, `400` for duplicates).
  - Detailed result message.
  - Error information if applicable.

Key Features:

- Duplicate checking before enrollment.
- License validation.
- Group-based organization.
- Comprehensive error handling.

Verify Method

```
Task Verify(string source, string target)
```

Purpose: Performs direct comparison between two face samples.

Parameters:

Name	Type	Required	Description
source	string	Yes	First face data for comparison in base64 format.
target	string	Yes	Second face data to compare against in base64 format.

Returns:

- `MatchResponse` containing:
  - Matching score.
  - Status message.
  - Error code if applicable.

Features:

- One-to-one comparison.
- Quality-based matching.
- Detailed score reporting.

Search Method

```
Task<List> Search(string source, string group)
```

Purpose: Searches database for matching faces.

Parameters:

Name	Type	Required	Description
source	string	Yes	Face data to search for in base64 format.
group	string	Yes	group filter for targeted searching.

Returns:

- `List<SearchResponse>` objects containing:
  - Matching scores.
  - Up to 5 best matches.
  - Match details.

Features:

- Group filtering capability.
- Multiple match support.
- Score-based result ranking.

Implementation Requirements

System Requirements

- Valid SDK license.
- Database configuration for search/enrollment operations.

Technical Requirements

- Async operation support.
- Database connectivity.
- Proper error handling implementation.

Best Practices

- Implement proper error handling.
- Check license status before operations.
- Use appropriate quality images.
- Implement proper group management.

Usage Examples

FaceProcessor Implementation

```
IFaceProcessor enrollService = new FaceProcessor(configuration);
var enrollResults = await enrollService.Enroll(image1,
enrollmentRequest.ExternalId, enrollmentRequest.Group);

IFaceProcessor searchService = new FaceProcessor(configuration);
var searchResults = await searchService.Search(image1,
searchRequest.Group);
```