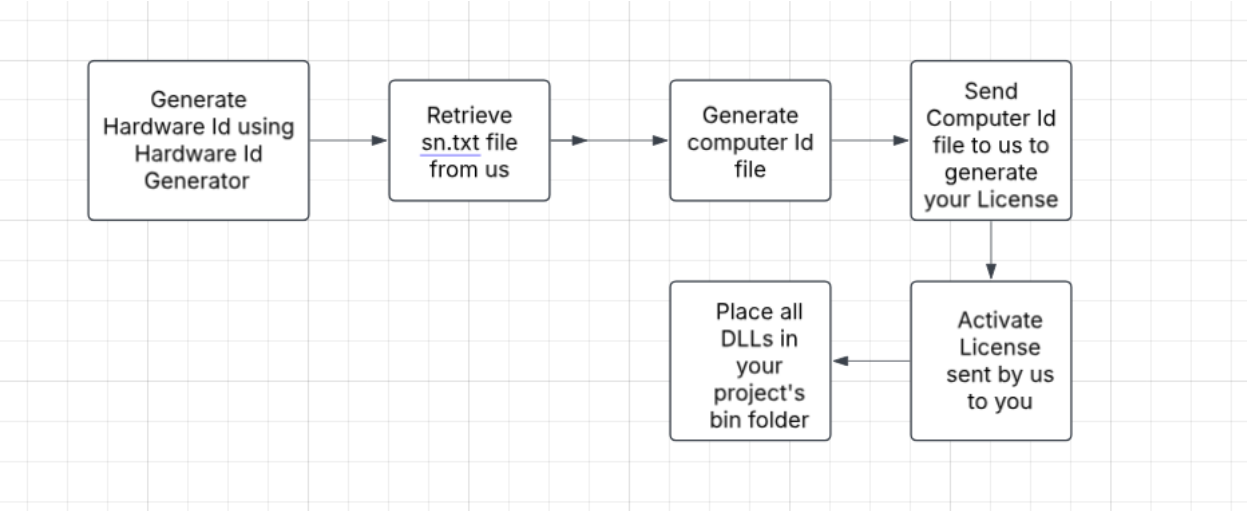
Overall Flow:



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 us for generating your License File.

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. Navigate back to the **Activation** folder in Command Prompt using:

```
cd ..
```

7. 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 `Fingerprint.lic`. Place this file in the **bin** folder where you just pasted the DLL files.

Activating the Windows Service

1. Navigate to `Artifacts/Activation/Windows Service`.
2. Double-click on `MFScanClientService`.
3. When prompted with **Yes/No**, click **Yes** to install the service.
4. If you are using the **MIS100V2 Fingerprint Device**, install its drivers from:

`Artifacts/Activation/Drivers`

Database Setup:

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

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 which has below fields in your table with table name: "__FingerprintSubjects"

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 fingerprints 😊

Interfaces Overview:

The **MxFace Fingerprint SDK** provides comprehensive fingerprint management capabilities through three main interfaces:

- **ISearch**
- **IDevice**
- **ICaptureService**

This documentation details each interface and its methods.

Interface Details

ISearch Interface

Handles core fingerprint operations including enrollment, verification, and searching.

Enroll Method

```
Task Enroll(byte[] source, string externalId, string group)
```

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

Parameters:

- `source`: Raw fingerprint image data in bytes.
- `externalId`: Unique identifier for the fingerprint record.
- `group`: 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(byte[] source, byte[] target)
```

Purpose: Performs direct comparison between two fingerprint samples.

Parameters:

- `source`: First fingerprint data for comparison.
- `target`: Second fingerprint data to compare against.

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(byte[] source, string group)
```

Purpose: Searches database for matching fingerprints.

Parameters:

- `source`: Fingerprint data to search for.
- `group`: Optional 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.

IDevice Interface

Manages fingerprint devices.

GetConnectedDevices Method

```
Task GetConnectedDevices(List devices)
```

Purpose: Identifies all physically connected fingerprint devices.

Parameters:

- `devices`: List to be populated with connected device names.

Returns:

- Number of connected devices found.

Features:

- Real-time device detection.
- Connection status validation.
- Device name parsing.

GetSupportedDevices Method

```
Task GetSupportedDevices(List deviceList)
```

Purpose: Lists all supported device models.

Parameters:

- `deviceList`: List to be populated with supported device names.

Returns:

- Count of supported devices.

Features:

- Model compatibility checking.
- Support validation.
- Comprehensive device listing.

Init Method

```
Task Init(string productName)
```

Purpose: Initializes specified fingerprint device.

Parameters:

- `productName`: Name/model of device to initialize.

Returns:

- `0`: Successful initialization.
- `-1`: Initialization failure.

Features:

- Device-specific initialization.
- Error handling.
- License validation.

GetDeviceInfoAsync Method

```
Task GetDeviceInfoAsync(string deviceName)
```

Purpose: Retrieves detailed device information.

Parameters:

- `deviceName`: Target device name.

Returns:

- `Device` object containing:
    - Error code.
    - Device specifications.
    - Status information.

Features:

- Detailed device diagnostics.
- Connection status checking.
- Comprehensive error reporting.

ICaptureService Interface

Manages fingerprint capture operations.

StartCaptureAsync Method

```
Task StartCaptureAsync(int Timeout = 10, int MinimumQuality = 60)
```

Purpose: Initiates fingerprint capture process.

Parameters:

- `Timeout`: Maximum capture duration in milliseconds.
- `MinimumQuality`: Required quality threshold (1-100).

Returns:

- `CaptureViewModel` containing:
  - Captured fingerprint data.
  - Quality metrics.
  - Capture status.

Features:

- Quality control.
- Timeout management.
- Auto-capture on quality achievement.
- License validation.

StopCaptureAsync Method

```
Task StopCaptureAsync()
```

Purpose: Forcefully stops ongoing capture process.

Returns:

- Status code indicating stop success/failure.

Features:

- Immediate capture termination.
- Resource cleanup.
- Device state reset.

Implementation Requirements

System Requirements

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

Technical Requirements

- HTTP client for device communication.
- Async operation support.
- Database connectivity.
- Proper error handling implementation.

Best Practices

- Always initialize device before capture.
- Implement proper error handling.
- Check license status before operations.
- Manage device resources properly.
- Use appropriate quality thresholds.
- Implement proper group management.
- Handle timeouts appropriately.

Usage Examples

Basic Device Initialization

```
IDevice deviceService = new DeviceService(httpClient, logger);
await deviceService.Init("DeviceName");
```

Fingerprint Capture

```
ICaptureService captureService = new
FingerprintCapturingService(httpClient);
var captureResult = await captureService.StartCaptureAsync(timeout:
15, minimumQuality: 75);
```

Search Implementation

```
ISearch searchService = new SearchService(configuration);
var searchResults = await
searchService.Search(captureResult.FingerprintData, "GroupA");
```

End of Documentation