DOCLER HOLDING

ShazamKit in practice

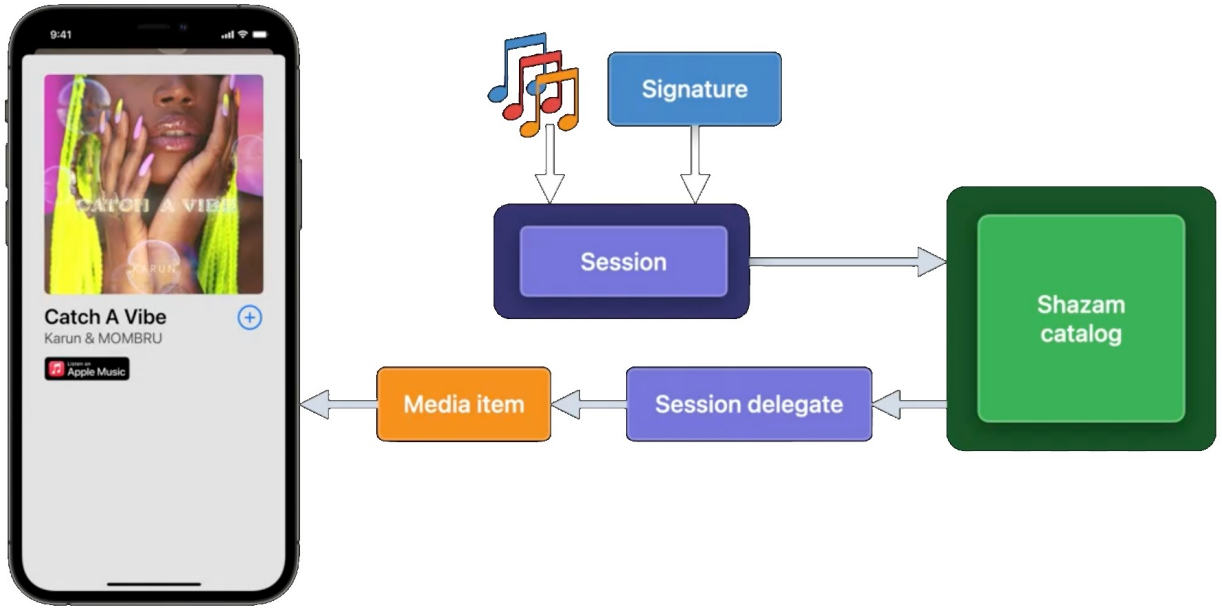
Horváth Martina
martina.horvath@doclerholding.com

Introducing ShazamKit

- Audio recognition system owned by Apple
- Pre-recorded or continuous audio using an online Shazam catalog
- Custom catalog
- Law rule changes
- Basic conception is easy to understand

ShazamKit

How it works



- A catalog stores the reference signatures and their associated metadata, or media items such as the song title and other details
- Signatures are a compact lossy representation of the original audio, it cannot be recreated from a signature, there exists only enough data to confirm that the signature matches a certain piece of audio
- A custom catalog has its own reference signatures and their associated metadata

Getting started on Android

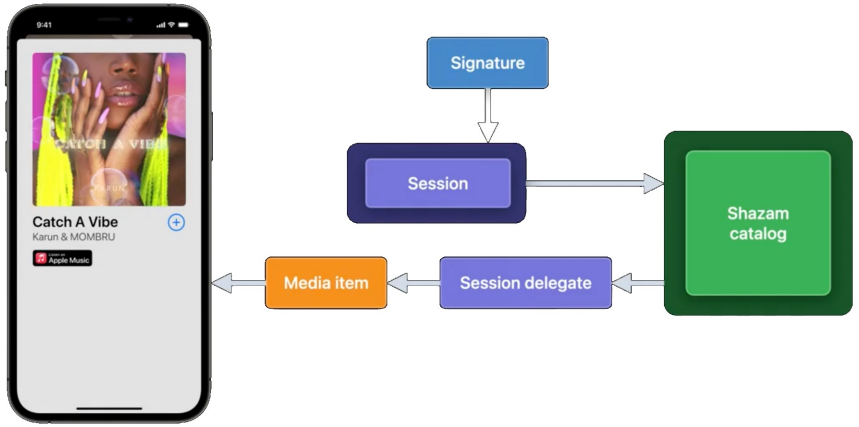
Requirements

- Minimum API version: 23 (iOS 15)
- Kotlin Coroutines

  More methods are suspend functions, they should be called only from a coroutine or another suspend function
- OkHttp and Retrofit libraries
- ShazamKit comes in the form of an Android Archive (AAR) file

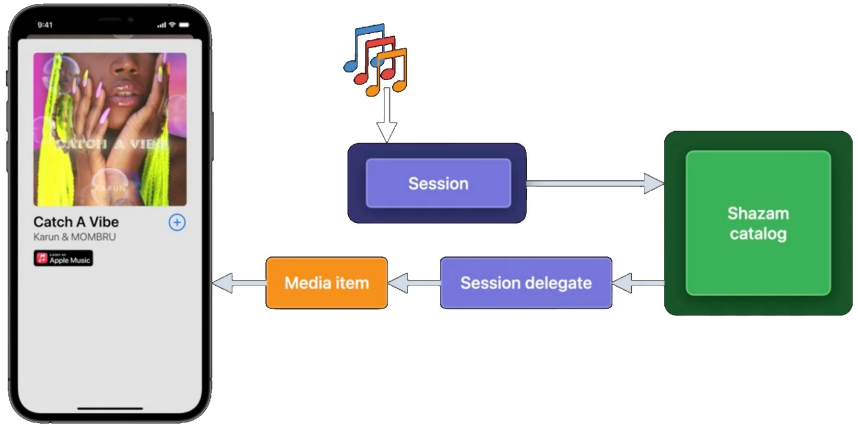
  Decompiled files, source code is obfuscated
- Apple Developer token for using ShazamCatalog
- The audio format must be PCM 16bit MONO in one of the following sample rates: 48000Hz, 44100Hz, 32000Hz, 16000Hz

Audio recognition



with pre-recorded audio

```
createSession()
session.match(signature)
```

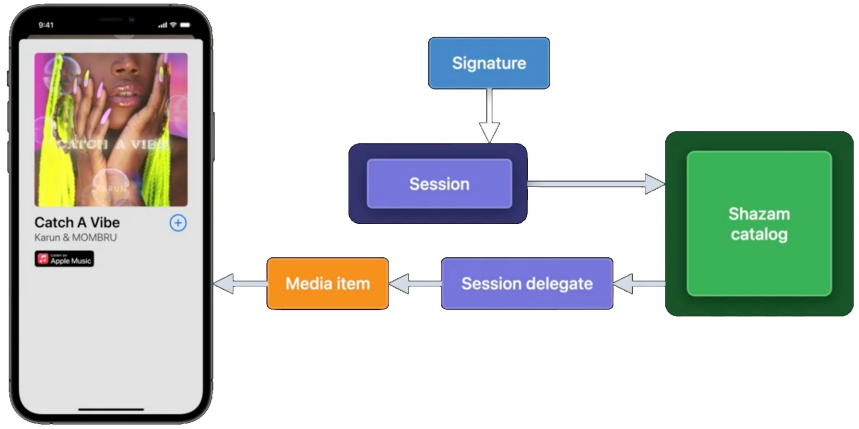


with continuous audio

```
createStreamingSession()
session.matchStream(audio_bytes, …)
```

Code overview with pre-recorded audio

```
val signatureGenerator = (createSignatureGenerator(AudioSampleRateInHz.SAMPLE_RATE_48000) as Success).data

signatureGenerator.append(pcm16_mono_audio_bytes, meaningful_length_in_bytes, System.currentTimeMillis())

val signature = signatureGenerator.generateSignature()

val catalog = ShazamKit.createShazamCatalog(developerTokenProvider)

when (val result = ShazamKit.createSession(catalog)) {
    is ShazamKitResult.Success -> {
        val session = result.data
        val matchResult = session.match(signature)

        // Handle match result
    }
    is ShazamKitResult.Failure -> // Handle error
}
```



Audio manipulation

1. Get the media file format (sample rate, channel count: 2, sample size: 2, audio track index)

Precondition: audio is a raw file with PCM 16 bit encoding (wav)

```
val extractor = MediaExtractor()
extractor.setDataSource(resources.openRawResourceFd(R.raw.music))

val audioTrackFormat = extractor.getTrackFormat(audioTrackIndex)
val sampleRate = audioTrackFormat.getInteger(MediaFormat.KEY_SAMPLE_RATE)
val channelCount = audioTrackFormat.getInteger(MediaFormat.KEY_CHANNEL_COUNT)
val sampleSizeInByte = audioTrackFormat.getInteger(MediaFormat.KEY_PCM_ENCONDING)
```

Audio manipulation

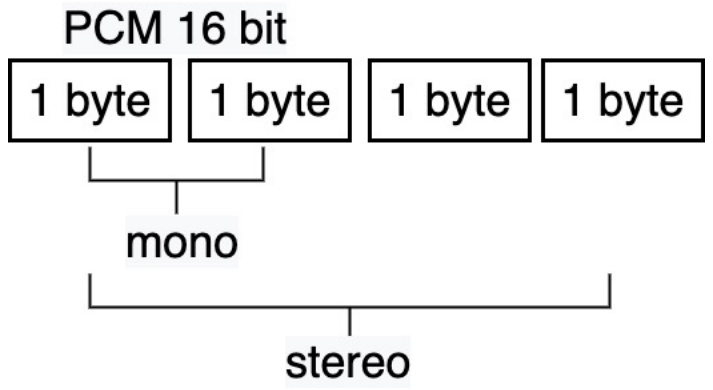
2. Read data with Media Extractor (inputBuffer)

```
val maxDurationInSeconds = catalog.maximumQuerySignatureDurationInMs.toInt() / 1000
val inputBuffer = ByteBuffer.allocate(sampleRate * maxDurationInSeconds * channelCount * sampleSizeInBytes)
val monoBuffer = ByteBuffer.allocate(sampleRate * maxDurationInSeconds * sampleSizeInBytes)

while (extractor.readSampleData(inputBuffer, 0) >= 0) {

  // 3. Format stereo audio to mono (monoBuffer)
  // 4. Pass converted audio to ShazamKit

}
```

Audio manipulation

3. Format stereo audio to mono (monoBuffer)

```
while (inputBuffer.remaining() > channelCount * sampleSizeInBytes) {

    // put first part into the monoInputBuffer from inputBuffer
    monoBuffer.putShort(inputBuffer.short)
    // getShort(): reads the next two bytes at buffer's current position, and then increments it

    // simply get bytes for mono audio
    inputBuffer.short
}
```



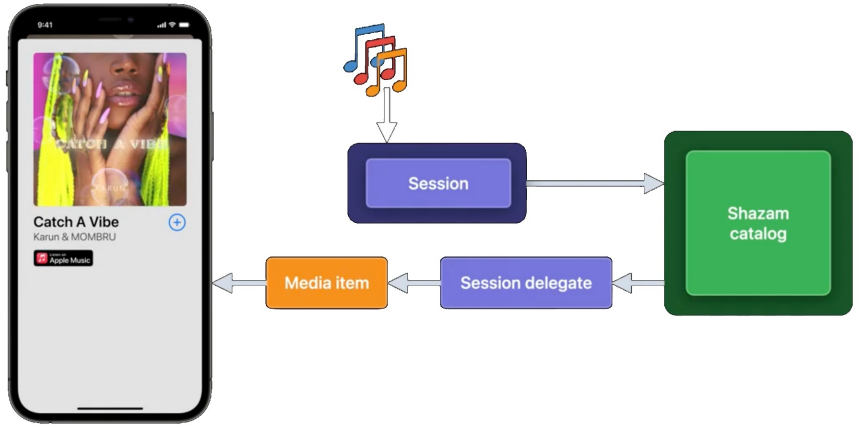
Audio manipulation

4. Pass converted audio to ShazamKit

```
while (extractor.readSampleData(inputBuffer, 0) >= 0) {

    if (monoBuffer.remaining() < inputBuffer.remaining()) {

        val signatureGenerator = (createSignatureGenerator(SAMPLE_RATE_44100) as Success).data
        signatureGenerator.append(monoBuffer.array(), monoBuffer.position(), System.currentTimeMillis())

        val signature = signatureGenerator.generateSignature()

        val matchResult = session.match(signature)
        // Handle match result

        monoBuffer.clear()
    }

    // 3. Format stereo audio to mono
}
```

Code overview with continuous audio

```
val catalog = ShazamKit.createShazamCatalog(developerTokenProvider)

when (val result = ShazamKit.createStreamingSession(
    catalog, SAMPLE_RATE_44100,
    // buffer size - set the buffer size using AudioTrack
    AudioTrack.getMinBufferSize(
        44100, AudioFormat.CHANNEL_OUT_MONO, AudioFormat.ENCODING_PCM_16BIT)
)) {
     is Success -> {
        val session = result.data

        recordingFlow().collect { byteBuffer, bufferSize, timestamp ->
            result.data.matchStream(
                monoByteBuffer.array(),
                monoByteBuffer.position(), // a buffer's position - the next element to be read or written
                timestamp
            )
        }

        session.recognitionResults().collect { matchResult -> /* Handle matchResult*/ }
    }
    // Handle Error
}
```



Documentation about recording from Microphone

Apple Developer token

- To authenticate as a trusted member of the Apple Developer Program
- Only for using Shazam Catalog
- API supports the JSON Web Token (JWT)

JWT token

- Consists of three parts separated by dots:

{{Header}.{Payload}.{Signature}}

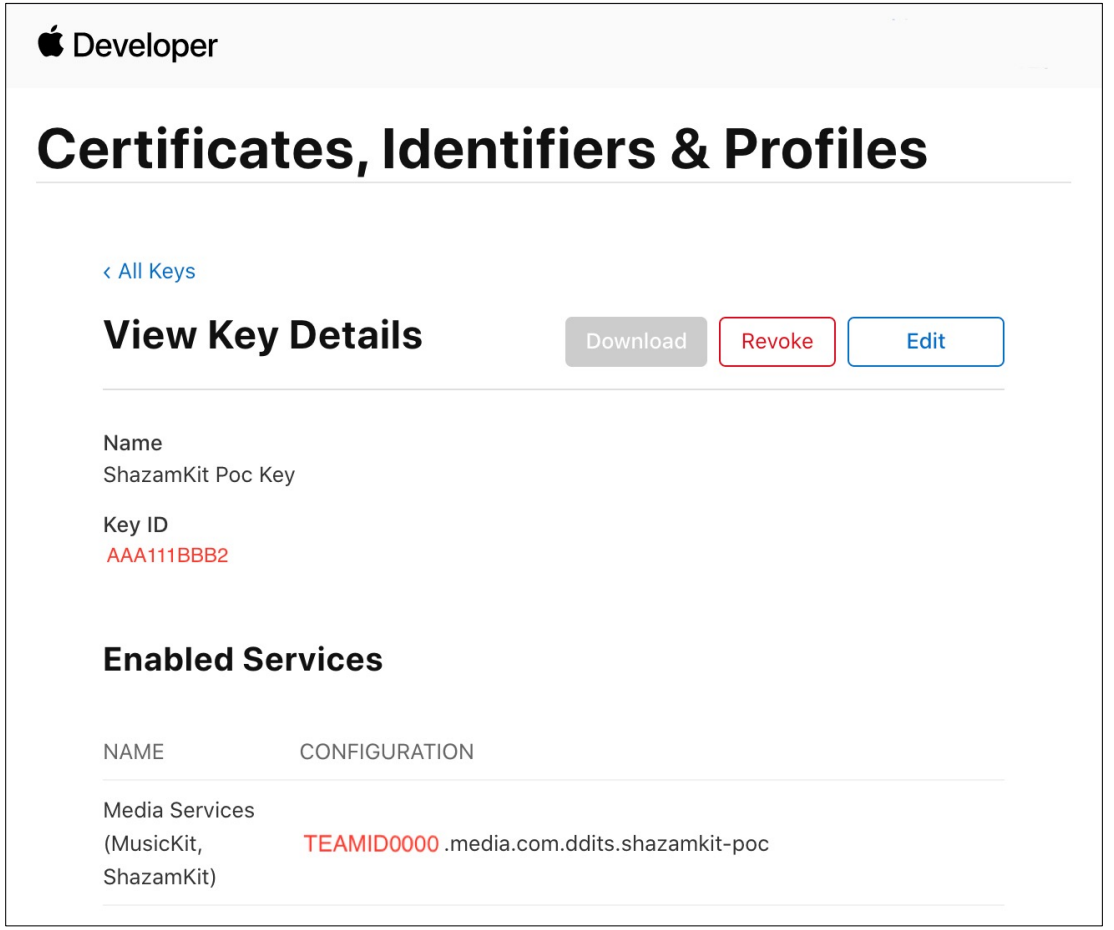
- Steps to create this token:
  1. Get the private key from your Apple Developer account
  2. Construct the JSON object
  3. Sign it with the private key

eyJhbGciOiJFUzI1NiIsInR5cCI6Ikd
dXVCIsImtpZCI6IkE3VDc3NEoyN
FkifQ.eyJpYXQiOjE2MjgwNzcyOT
MsImV4cCI6MTY0MzYydTI5Mywi
aXNzIjoiWFJSTjRaMldMOSJ9.msz
4SGIjrfI4ws8w1QifQrCQScSty0PeE
lYYTjMfkFi5d8_09QjtliitPEN_215F
8HUJLd5PAFtWEr1odHmB3Q

JWT token

1. Get the private key from your Apple Developer account
    1. Create a Apple Developer account, sign in
    2. Create private key
    3. Download it
    4. Get the 10-character key identifier
    5. Get the team id

Please find the detailed description here.



JWT token

2. Construct the JSON Object

```
{
    "alg": "ES256",
    "kid": "AAA111BBB2"
}
{
    "iss": "TEAMID0000",
    "iat": 1625486922,
    "exp": 1636114122
}
```

Header

alg: The encryption algorithm to encrypt the token - Apple supports ES256

kid: key id

Payload

iss: team id

iat: the time at which the token was generated in UTC

exp: expiration time – not greater than 15777000 (6 months in seconds)

JWT token

3. Sign it with private key using Node.js

Encrypt the token using the Elliptic Curve Digital Signature Algorithm (ECDSA) with the P-256 curve and the SHA-256 hash algorithm

```
const fs = require("fs");
const jwt = require("jsonwebtoken");

const privateKey = fs.readFileSync("private_key.p8").toString(); // Private key
const jwtToken = jwt.sign(
        {},
        privateKey,
        {
            algorithm: "ES256",
            expiresIn: "180d",
            issuer: "TEAMID000", // Team ID
            header: {
            alg: "ES256",
            kid: "AAA111BBB2" // ShazamKit key ID
            }
        }
);
```

Using Custom Catalog

- A CustomCatalog can be accepted both in a Session or a StreamingSession
- Audio can not only music, but any voice - Apple shared this starter project to present its utility in practice
- Custom catalog means collections of manually generated signatures
- Signatures can be created in advance and loaded from anywhere as InputStream

```
val customCatalog = ShazamKit.createCustomCatalog().apply { addFromCatalog(inputStream) }

val session = (ShazamKit.createSession(catalog) as Success).data
val matchResult = session.match(signature)
// signature: audio's signature which has to be matched with custom catalog

when (matchResult) { /* Handle matchResult */ }
```

Links

- ShazamKit overview
- ShazamKit on Android
- Create an Apple private key to access ShazamKit API
- Android documentation from Apple
- Buffers

DOCLER HOLDING

DOCLER HOLDING

H-1101 BUDAPEST
EXPO TÉR 5-7.

WWW.DOCLERHOLDING.HU