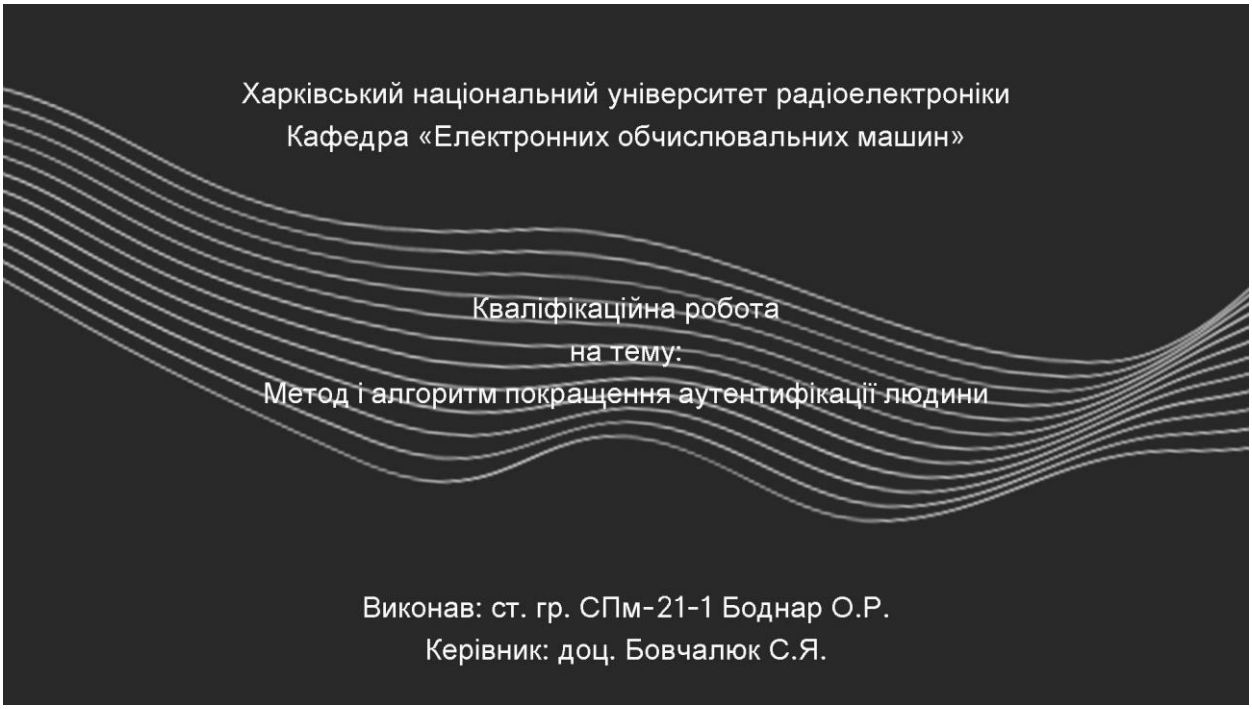
ДОДАТОК А

Графічний матеріал кваліфікаційної роботи



Мета і задачі дослідження

МЕТОЮ КВАЛІФІКАЦІЙНОЇ РОБОТИ Є ДОСЛІДЖЕННЯ СУЧАСНИХ МЕТОДІВ І АЛГОРИТМІВ АУТЕНТИФІКАЦІЇ ЛЮДИНИ, ВИЗНАЧЕННЯ ЇХНІХ ПЕРЕВАГ І НЕДОЛІКІВ, ТА ПОКРАЩЕННЯ ФУНКЦІОНУВАННЯ, ШЛЯХОМ УВЕДЕННЯ ДОДАТКОВИХ ЕЛЕМЕНТІВ ЗАХИСТУ І РОЗРОБКИ МОБІЛЬНОГО ЗАСТОСУНКУ, ЩО РЕАЛІЗУЄ ДОДАТКОВИЙ ФУНКЦІОНАЛ.

Часткові завдання дослідження:

- Провести аналіз різноманітних технологій і алгоритмів аутентифікації;
- Виконати огляд сучасних середовищ розробки, технологій створення застосунків для мобільних пристроїв та хмарних біометричних сервісів;
- Розробити архітектуру програмного продукту та навести опис технічної реалізації застосунку;
- Навести опис інтерфейсу користувача та використання застосунку сторонніми сервісами у якості двохфакторної аутентифікації.

Сучасні методи аутентифікації

ПАРОЛЬНА АУТЕНТИФІКАЦІЯ

Користувач може використовувати унікальний пароль для підтвердження своєї особистості.

БІОМЕТРИЧНА АУТЕНТИФІКАЦІЯ

Користувач може використовувати біометричні дані такі, як геометрія обличчя, відбитки пальців, тембр голосу для аутентифікація.

БАГАТОФАКТОРНА АУТЕНТИФІКАЦІЯ

Для більшої безпеки користувач може використовувати декілька методів аутентифікації послідовно.

Пристрої необхідні для збору біометричних даних

Камера, або спеціальний датчик для збору даних про обличчя користувача.

Мікрофон для збору даних про голос користувача.

Датчик для збору даних про відбитки пальців користувача.

Оскільки всі ці пристрої для персональних комп'ютерів та деяких ноутбуків є периферією, яку має не кожен користувач, доцільно припустити, що мобільний пристрій є найзручнішим способом збору біометричних даних.

Технології використані для імплементації додатку

- Клієнтська частина - React Native (мова програмування TypeScript)
- Серверна частина - ASP.NET Core (мова програмування C#)
- Хмарні сервіси - Firebase Cloud Messaging, Azure Cognitive Services

Firebase Cloud Messaging

Firebase - хмарний провайдер компанії Google.

Цей хмарний провайдер надає велику кількість сервісів, серед яких багато для роботи з мобільними пристроями.

Firebase Cloud Messaging надає можливість надсилати push-нотифікації на мобільні пристрої користувачів.

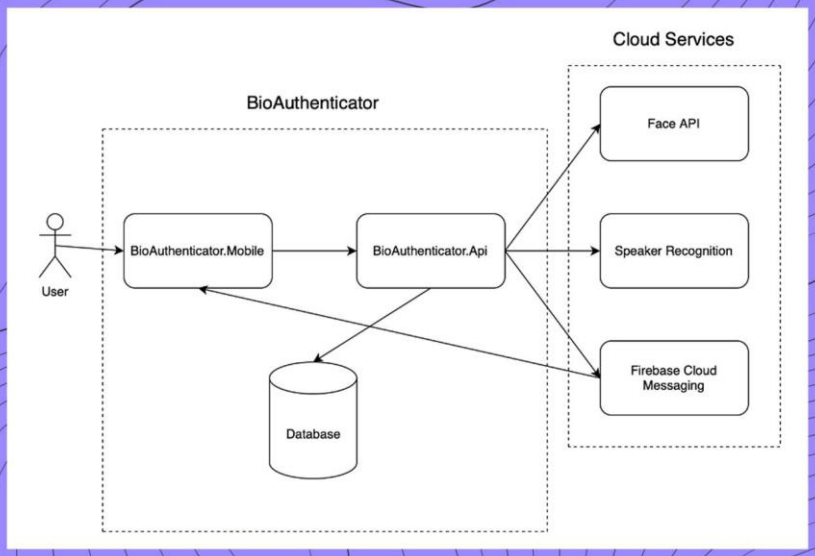
Azure Cognitive Services

Azure - хмарний провайдер компанії Microsoft.

Face API - сервіс, що є частиною Azure Cognitive Service групи для роботи з фото та відео з обличчям, його розпізнаванням, порівнянням, тощо.

Speaker Recognition - сервіс, що є частиною тієї ж групи, який надає можливості опрацювання голосу людини.

Архітектура розробленого додатку



Основні терміни якими оперує додаток

ORGANIZATION

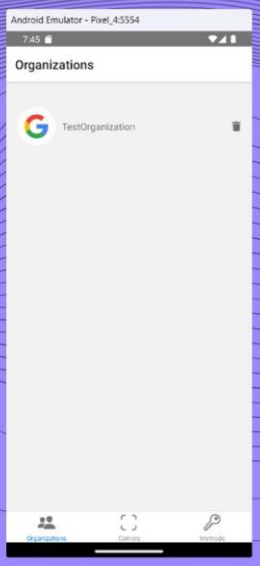
Організація, або ж просто сервіс, що використовує розроблений додатоку якості другого методу двохфакторнох аутентифікації.

ORGANIZATION USER

Користувач з точки зору тієї, чи іншої органцізації. Кожен користувач закріплений за певною організацією, та може бути запрошений для підтвердження аутентифікації за допомогою мобільного пристрою.

APP INSTALLATION

Інстанс додатку, встановлений у пристрої. Пристрій може бути прив'язаний до безлічі користувачів різних організацій. Коли користувачу необхідно підтвердити свою особистість, він отримує повідомлення на прив'язаний пристрій



ІНТЕРФЕЙС КОРИСТУВАЧА МОБІЛЬНОГО ДОДАТКУ БІОМЕТРИЧНОЇ АУТЕНТИФІКАЦІЇ

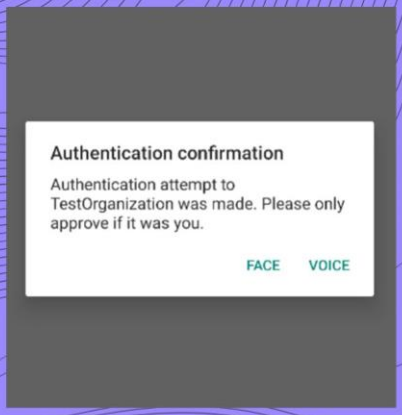
Сторінка **Organizations** вміщує в собі список усих організації, що були пов'язані з поточним пристроєм та використовують його для підтвердження аутентифікації певних користувачів.

З цієї сторінки можна перейти на сторінку **Camera**, що надає можливість відсканувати код для прив'язки організації, та сторінку **Methods**, де можна налаштувати бажані методи аутентифікації.

ПІДТВЕРДЖЕННЯ ОСОБИСТОСТІ КОРИСТУВАЧА

Коли сервісу, що використовує додаток, необхідно підтвердити особистість користувача, він надсилає відповідний запит до API, який в свою чергу генерує та надсилає сповіщення на мобільний пристрій відповідного користувача.

Отримавши повідомлення, користувач має змогу обрати один з налаштованих методів для підтвердження особистості. Коли процес завершено, початковий сервіс отримує результати перевірки.



Апробація результатів дослідження

Черкаський державний
технологічний університет

Військова Академія Збройних Сил
Азербайджанської республіки

Університет технології і гуманітарних наук
(м. Бельсько-Бяла, Польща)

Національний технічний університет
"Харківський політехнічний інститут"

Харківський національний
університет радіоелектроніки

ДП «Південний державний проектно-конструкторський
та науково-дослідний інститут авіаційної промисловості»

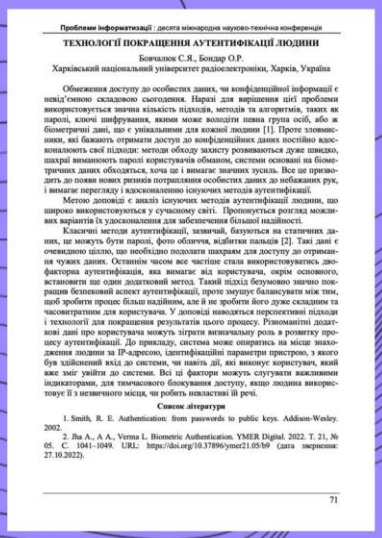
**ПРОБЛЕМИ
ІНФОРМАТИЗАЦІЇ**

ТЕЗИ ДОПОВІДЕЙ ДЕСЯТОЇ МІЖНАРОДНОЇ
НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

24 – 25 листопада 2022 року

Том 1

Черкаси – Баку – Бельсько-Бяла – Харків – 2022



ТЕХНОЛОГІЇ ПОКРАЩЕННЯ АУТЕНТИФІКАЦІЇ ЛЮДИНИ

Бовчалюк С.Я., Бондар О.Р.
Харківський національний університет радіоелектроніки, Харків, Україна

Обмеження доступу до особистих даних, чи конфіденційної інформації є невід'ємною складовою сьогодення. Наразі для вирішення цієї проблеми використовується значна кількість підходів, методів та алгоритмів, таких як паролі, ключі шифрування, якими може володіти певна група осіб, або ж біометричні дані, що є унікальними для кожної людини [1]. Проте зловмисники, які бажають отримати доступ до конфіденційних даних постійно вдосконалюють свої підходи: методи обходу захисту розвиваються дуже швидко, шахраї вимагають паролі користувачів обманом, системи основані на біометричних даних обходяться, хоча це і вимагає значних зусиль. Все це призводить до появи нових ризиків потрапляння особистих даних до небажаних рук, і вимагає перегляду і вдосконаленню існуючих методів аутентифікації.

Метою доповіді є аналіз існуючих методів аутентифікації людини, що широко використовуються у сучасному світі. Пропонується розгляд можливих варіантів їх удосконалення для забезпечення більшої надійності.

Класичні методи аутентифікації, зазвичай, базуються на статичних даних, це можуть бути паролі, фото обличчя, відбитки пальців [2]. Такі дані є очевидною ціллю, що необхідно подолати шахраям для доступу до отримання чужих даних. Останнім часом все частіше стали використовуватись двофакторна аутентифікація, яка вимагає від користувача, окрім основного, встановити ще один додатковий метод. Такий підхід безумовно значно покращив безпековий аспект аутентифікації, проте змушує балансувати між тим, щоб зробити процес більш надійним, але й не зробити його дуже складним та часовитратним для користувача. У доповіді наводяться перспективні підходи і технології для покращення результатів цього процесу. Різноманітні додаткові дані про користувача можуть зіграти визначальну роль в розвитку процесу аутентифікації. До прикладу, система може опиратись на місце знаходження людини за IP-адресою, ідентифікаційні параметри пристрою, з якого був здійснений вхід до системи, чи навіть дії, які виконує користувач, який вже зміг увійти до системи. Всі ці фактори можуть слугувати важливими індикаторами, для тимчасового блокування доступу, якщо людина використовує її з незвичного місця, чи робить невластиві їй речі.

Список літератури

1. Smith, R. E. Authentication: from passwords to public keys. Addison-Wesley. 2002.

2. Jha A., A A., Verma L. Biometric Authentication. YMER Digital. 2022. T. 21, № 05. С. 1041–1049. URL: https://doi.org/10.37896/ymer21.05/b9 (дата звернення: 27.10.2022).

Висновки

У процесі виконання кваліфікаційної роботи було наведено обґрунтування актуальності потреби покращення аутентифікації людини, проведено аналіз парольної аутентифікації користувача і біометричних даних, як методу аутентифікації. Зроблено аналіз мультифакторної аутентифікації і виконано обґрунтування переваг використання мобільного пристрою для проведення процесу аутентифікації. У роботі наведено огляд хмарних біометричних сервісів, сучасних середовищ розробки та технологій для створення застосунків для мобільних пристроїв, також наведено обґрунтування обраних технологій для їх розробки. Виконано розробку архітектури програмного продукту і надано опис його технічної реалізації. Зроблено опис інтерфейсу користувача мобільного застосунку, показано процедуру використання застосунку сторонніми сервісами у якості двохфакторної аутентифікації.

ДОДАТОК Б

Наукові публікації за темою кваліфікаційної роботи

Черкаський державний
технологічний університет

Військова Академія Збройних Сил
Азербайджанської республіки

Університет технології і гуманітарних наук
(м. Бельсько-Бяла, Польща)

Національний технічний університет
"Харківський політехнічний інститут"

Харківський національний
університет радіоелектроніки

ДП «Південний державний проектно-конструкторський
та науково-дослідний інститут авіаційної промисловості»

ПРОБЛЕМИ ІНФОРМАТИЗАЦІЇ

ТЕЗИ ДОПОВІДЕЙ ДЕСЯТОЇ МІЖНАРОДНОЇ
НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

24 – 25 листопада 2022 року

Том 1

Черкаси – Баку – Бельсько-Бяла – Харків – 2022

ТЕХНОЛОГІЇ ПОКРАЩЕННЯ АУТЕНТИФІКАЦІЇ ЛЮДИНИ

Бовчалюк С.Я., Бондар О.Р.
Харківський національний університет радіоелектроніки, Харків, Україна

Обмеження доступу до особистих даних, чи конфіденційної інформації є невід'ємною складовою сьогодення. Наразі для вирішення цієї проблеми використовується значна кількість підходів, методів та алгоритмів, таких як паролі, ключі шифрування, якими може володіти певна група осіб, або ж біометричні дані, що є унікальними для кожної людини [1]. Проте зловмисники, які бажають отримати доступ до конфіденційних даних постійно вдосконалюють свої підходи: методи обходу захисту розвиваються дуже швидко, шахраї виманюють паролі користувачів обманом, системи основані на біометричних даних обходяться, хоча це і вимагає значних зусиль. Все це призводить до появи нових ризиків потрапляння особистих даних до небажаних рук, і вимагає перегляду і вдосконаленню існуючих методів аутентифікації.

Метою доповіді є аналіз існуючих методів аутентифікації людини, що широко використовуються у сучасному світі. Пропонується розгляд можливих варіантів їх удосконалення для забезпечення більшої надійності.

Класичні методи аутентифікації, зазвичай, базуються на статичних даних, це можуть бути паролі, фото обличчя, відбитки пальців [2]. Такі дані є очевидною ціллю, що необхідно подолати шахраям для доступу до отримання чужих даних. Останнім часом все частіше стала використовуватись двофакторна аутентифікація, яка вимагає від користувача, окрім основного, встановити ще один додатковий метод. Такий підхід безумовно значно покращив безпековий аспект аутентифікації, проте змушує балансувати між тим, щоб зробити процес більш надійним, але й не зробити його дуже складним та часовитратним для користувача. У доповіді наводяться перспективні підходи і технології для покращення результатів цього процесу. Різноманітні додаткові дані про користувача можуть зіграти визначальну роль в розвитку процесу аутентифікації. До прикладу, система може опиратись на місце знаходження людини за IP-адресою, ідентифікаційні параметри пристрою, з якого був здійснений вхід до системи, чи навіть дії, які виконує користувач, який вже зміг увійти до системи. Всі ці фактори можуть слугувати важливими індикаторами, для тимчасового блокування доступу, якщо людина використовує її з незвичного місця, чи робить невластиві їй речі.

Список літератури

1. Smith, R. E. Authentication: from passwords to public keys. Addison-Wesley. 2002.

2. Jha A., A A., Verma L. Biometric Authentication. YMER Digital. 2022. T. 21, № 05. C. 1041–1049. URL: https://doi.org/10.37896/ymer21.05/b9 (дата звернення: 27.10.2022).

ДОДАТОК В

Код мобільного застосунку

В.1 EnrollmentService

```
using System.Text.Json;
using BioAuthenticator.Core.Enums;
using BioAuthenticator.Core.Exceptions;
using BioAuthenticator.Core.Interfaces;
using BioAuthenticator.Core.Models;
using BioAuthenticator.Core.Models.Options;
using BioAuthenticator.Infrastructure.Data;
using Microsoft.AspNetCore.Http;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Options;
using QRCoder;

namespace BioAuthenticator.Infrastructure.Services;

/// <inheritdoc/>
public class EnrollmentService : IEnrollmentService
{
    private readonly IFaceService _faceService;
    private readonly ISpeechService _speechService;
    private readonly AppDbContext _context;
    private readonly ApplicationSettingsOptions _appSettings;
    private readonly QRCodeGenerator _qrGenerator;

    /// <summary>
    /// Creates a new instance of <see
cref="EnrollmentService"/>.
    /// </summary>
    /// <param name="faceService">Service that provides face
data management functionality.</param>
    /// <param name="speechService">Service that provides voice
data management functionality.</param>
    /// <param name="context">Database context.</param>
    /// <param name="appSettings">Application settings
options.</param>
    /// <param name="qrGenerator">Class for generating QR
codes.</param>
    public EnrollmentService(IFaceService faceService,
ISpeechService speechService,
        AppDbContext context,
IOptions<ApplicationSettingsOptions> appSettings,
QRCodeGenerator qrGenerator)
    {
```

```
        _faceService = faceService;
        _speechService = speechService;
        _context = context;
        _appSettings = appSettings.Value;
        _qrGenerator = qrGenerator;
    }

    public async Task<AuthMethodEnrollmentResponse>
EnrollAsync(string appInstallationId, AuthMethod method,
        IFormFile file, string? passphrase)
    {
        var appInstallation =
            await
_context.AppInstallations.FirstOrDefaultAsync(x =>
x.InstallationId == appInstallationId);

        if (appInstallation is null)
        {
            throw new
BioAuthenticatorEntityNotFoundException(nameof(AppInstallation),
                nameof(AppInstallation.InstallationId),
appInstallationId);
        }

        return method switch
        {
            AuthMethod.Face => await
EnrollNewFaceAsync(appInstallation, file),
            AuthMethod.Voice => await
EnrollNewVoiceAsync(appInstallation, file, passphrase),
            _ => throw new BioAuthenticatorException($"Invalid
auth method: {method}")
        };
    }

    /// <inheritdoc/>
    public async Task<byte[]> GetEnrollmentQrCodeAsync(Guid
organizationId, Guid organizationUserId)
    {
        var organizationUser = await _context.OrganizationUsers
            .Include(x => x.Organization)
            .FirstOrDefaultAsync(x => x.OrganizationId ==
organizationId && x.Id == organizationUserId);

        if (organizationUser is null)
        {
            throw new
BioAuthenticatorEntityNotFoundException(nameof(OrganizationUser)
, new Dictionary<string, string>
            {
                { nameof(OrganizationUser.OrganizationId),
organizationId.ToString() },
                { nameof(OrganizationUser.Id),
```

```
organizationUserId.ToString() }
            });
        }

        var userEnrollment = new OrganizationUserEnrollment
        {
            IsEnrollmentCompleted = false,
            OrganizationUserId = organizationUserId,
            EnrollmentCode = Guid.NewGuid(),
            EnrollmentCodeGeneratedDate = DateTime.UtcNow
        };

        await
_context.OrganizationUserEnrollments.AddAsync(userEnrollment);
        await _context.SaveChangesAsync();

        var code = new
EnrollmentCode(organizationUser.Organization.Name,
organizationId, organizationUserId,
            userEnrollment.EnrollmentCode);

        var qrCodeData =
_qrGenerator.CreateQrCode(JsonSerializer.Serialize(code),
QRCodeGenerator.ECCLevel.Q);

        return new PngByteQRCode(qrCodeData).GetGraphic(20);
    }

    /// <inheritdoc/>
    public async Task EnrollNewAppInstallationForUserAsync(Guid
organizationId, Guid organizationUserId, Guid enrollmentCode,
        string appInstallationId)
    {
        var enrollment = await
_context.OrganizationUserEnrollments
            .Include(x => x.OrganizationUser)
            .FirstOrDefaultAsync(x =>
                x.OrganizationUser!.OrganizationId ==
organizationId && x.OrganizationUserId == organizationUserId &&
                x.EnrollmentCode == enrollmentCode);

        if (enrollment is null)
        {
            throw new
BioAuthenticatorEntityNotFoundException(nameof(OrganizationUserE
nrollment),
                new Dictionary<string, string>
                {
                    {

$"{nameof(OrganizationUserEnrollment.OrganizationUser)}.{nameof(
OrganizationUser.OrganizationId)}",
                        organizationId.ToString()
```

```
                    },
                    {
nameof(OrganizationUserEnrollment.OrganizationUserId),
organizationUserId.ToString() },
                    {
nameof(OrganizationUserEnrollment.EnrollmentCode),
enrollmentCode.ToString() }
                });
        }

        if (enrollment.IsEnrollmentCompleted)
        {
            throw new
BioAuthenticationInvalidEnrollmentException(
                $"The provided enrollment code was already
used");
        }

        if ((DateTime.UtcNow -
enrollment.EnrollmentCodeGeneratedDate).TotalHours >=
            _appSettings.UserEnrollmentCodeExpirationInHours)
        {
            throw new
BioAuthenticationInvalidEnrollmentException("The provided
enrollment code expired");
        }

        var appInstallation = await
_context.AppInstallations.FirstOrDefaultAsync(x =>
x.InstallationId == appInstallationId);

        if (appInstallation is null)
        {
            throw new
BioAuthenticatorEntityNotFoundException(nameof(AppInstallation),
                nameof(AppInstallation.InstallationId),
appInstallationId);
        }

        enrollment.OrganizationUser!.AppInstallation =
appInstallation;
        enrollment.IsEnrollmentCompleted = true;

        await _context.SaveChangesAsync();
    }

    /// <inheritdoc/>
    public async Task ResetAuthMethod(string installationId,
AuthMethod method)
    {
        var appInstallation =
            await
_context.AppInstallations.FirstOrDefaultAsync(x =>
```

```
x.InstallationId == installationId);

        if (appInstallation is null)
        {
            throw new
BioAuthenticatorEntityNotFoundException(nameof(AppInstallation),
                nameof(AppInstallation.InstallationId),
installationId);
        }

        switch (method)
        {
            case AuthMethod.Face:
                await ResetFaceMethodAsync(appInstallation);
                break;
            case AuthMethod.Voice:
                await ResetVoiceMethodAsync(appInstallation);
                break;
            default:
                throw new BioAuthenticatorException($"Invalid
auth method: {method}");
        }
    }

    private async Task ResetFaceMethodAsync(AppInstallation
installation)
    {
        if (!installation.FaceProfileId.HasValue)
        {
            return;
        }

        await
_faceService.DeletePersonAsync(installation.FaceProfileId.Value)
;
        installation.FaceProfileId = null;
        await _context.SaveChangesAsync();
    }

    private async Task ResetVoiceMethodAsync(AppInstallation
installation)
    {
        if
(string.IsNullOrWhiteSpace(installation.SpeechProfileId))
        {
            return;
        }

        await
_speechService.DeleteProfileAsync(installation.SpeechProfileId);
        installation.SpeechProfileId = null;
        await _context.SaveChangesAsync();
    }
```

```
    private async Task<AuthMethodEnrollmentResponse>
EnrollNewFaceAsync(AppInstallation appInstallation, IFormFile
faceImage)
    {
        if (appInstallation.FaceProfileId is not null)
        {
            throw new
BioAuthenticationInvalidEnrollmentException(
                "Provided app installation already has face
enrolled.");
        }

        var personGuid = await
_faceService.CreatePersonAsync(appInstallation.InstallationId);
        var result = await
_faceService.AssignFaceToPersonAsync(personGuid, faceImage);

        if (result.Result ==
AuthMethodEnrollmentResult.EnrollmentFailed)
        {
            await _faceService.DeletePersonAsync(personGuid);
            return result;
        }

        appInstallation.FaceProfileId = personGuid;
        await _context.SaveChangesAsync();
        return result;
    }

    private async Task<AuthMethodEnrollmentResponse>
EnrollNewVoiceAsync(AppInstallation appInstallation, IFormFile
voiceRecord, string? passphrase)
    {
        if (appInstallation.SpeechProfileId is null)
        {
            var profileId = await
_speechService.CreateProfileAsync();
            appInstallation.SpeechProfileId = profileId;
            await _context.SaveChangesAsync();
        }

        if (passphrase is null)
        {
            throw new BioAuthenticatorException("Passphrase is
required for voice enrollment");
        }

        return await
_speechService.EnrollProfileAsync(appInstallation.SpeechProfileI
d, voiceRecord, passphrase);
    }
}
```

B.2 FaceService

```
using System.Net;
using BioAuthenticator.Core.Enums;
using BioAuthenticator.Core.Interfaces;
using BioAuthenticator.Core.Models;
using BioAuthenticator.Core.Models.Options;
using Microsoft.AspNetCore.Http;
using Microsoft.Azure.CognitiveServices.Vision.Face;
using Microsoft.Azure.CognitiveServices.Vision.Face.Models;
using Microsoft.Extensions.Options;

namespace BioAuthenticator.Infrastructure.Services;

/// <inheritdoc/>
public class FaceService : IFaceService
{
    private readonly IFaceClient _faceClient;
    private readonly FaceRecognitionApiOptions
_faceServiceOptions;


    /// <summary>
    /// Creates a new instance of <see cref="FaceService"/>.
    /// </summary>
    /// <param name="faceClient">Face client provided by Azure
Cognitive Services.</param>
    /// <param name="azureServicesOptions">Options that contain
azure services configuration.</param>
    public FaceService(IFaceClient faceClient,
IOptions<AzureServicesOptions> azureServicesOptions)
    {
        _faceClient = faceClient;
        _faceServiceOptions =
azureServicesOptions.Value.FaceRecognitionApi;
    }

    /// <inheritdoc/>
    public async Task<bool> VerifyAsync(Guid personId, IFormFile
image)
    {
        var faces = await GetFacesFromImageAsync(image);

        foreach (var detectedFace in faces.Where(x =>
x.FaceId.HasValue))
        {
            var result =
                await
_faceClient.Face.VerifyFaceToPersonAsync(detectedFace.FaceId!.Va
lue, personId,
                    _faceServiceOptions.PersonGroupId);
```

```
            if (result.IsIdentical)
            {
                return true;
            }
        }

        return false;
    }

    /// <inheritdoc/>
    public async Task<Guid> CreatePersonAsync(string personName)
    {
        if (!await
DoesPersonGroupExistAsync(_faceServiceOptions.PersonGroupId))
        {
            await
CreatePersonGroupAsync(_faceServiceOptions.PersonGroupId);
        }

        var person = await
_faceClient.PersonGroupPerson.CreateAsync(_faceServiceOptions.Pe
rsonGroupId, personName);

        return person.PersonId;
    }

    ///<inheritdoc/>
    public async Task<AuthMethodEnrollmentResponse>
AssignFaceToPersonAsync(Guid personId, IFormFile image)
    {
        if ((await GetFacesFromImageAsync(image)).Count != 1)
        {
            return new AuthMethodEnrollmentResponse
            {
                Result =
AuthMethodEnrollmentResult.EnrollmentFailed,
                Message = "Image should always contain 1 face
only"
            };
        }

        await using var imageStream = image.OpenReadStream();
        await
_faceClient.PersonGroupPerson.AddFaceFromStreamAsync(_faceServic
eOptions.PersonGroupId, personId, imageStream);

        return new AuthMethodEnrollmentResponse { Result =
AuthMethodEnrollmentResult.EnrollmentCompleted };
    }

    /// <inheritdoc/>
    public async Task DeletePersonAsync(Guid personId)
```

```
    {
        var person = await
_faceClient.PersonGroupPerson.GetAsync(_faceServiceOptions.Perso
nGroupId, personId);

        if (person is null)
        {
            return;
        }

        await
_faceClient.PersonGroupPerson.DeleteAsync(_faceServiceOptions.Pe
rsonGroupId, personId);
    }

    private async Task CreatePersonGroupAsync(string
personGroupId)
    {
        await _faceClient.PersonGroup.CreateAsync(personGroupId,
"Main BioAuthenticator person group");
    }


    private async Task<bool> DoesPersonGroupExistAsync(string
personGroupId)
    {
        try
        {
            return await
_faceClient.PersonGroup.GetAsync(personGroupId) is not null;
        }
        catch (APIErrorException e)
        {
            if (e.Response.StatusCode ==
HttpStatusCode.NotFound)
            {
                return false;
            }

            throw;
        }
    }

    private async Task<IList<DetectedFace>>
GetFacesFromImageAsync(IFormFile image)
    {
        await using var imageStream = image.OpenReadStream();
        return await
_faceClient.Face.DetectWithStreamAsync(imageStream);
    }
}
```

B.3 SpeechService

```
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Speaker;
using BioAuthenticator.Core.Enums;
using BioAuthenticator.Core.Interfaces;
using BioAuthenticator.Core.Models;
using BioAuthenticator.Infrastructure.Extensions;
using BioAuthenticator.Infrastructure.Helpers;
using Microsoft.AspNetCore.Http;
using Microsoft.CognitiveServices.Speech;

namespace BioAuthenticator.Infrastructure.Services;

/// <inheritdoc/>
public class SpeechService : ISpeechService
{
    private readonly VoiceProfileClient _voiceClient;
    private readonly SpeechApiRecognizersFactory
_speechApiRecognizersFactory;

    /// <summary>
    /// Creates a new instance of <see cref="SpeechService"/>.
    /// </summary>
    /// <param name="voiceClient">Voice profile client provided
by Azure Cognitive Services.</param>
    /// <param name="speechApiRecognizersFactory">Helper factory
class that initialize different recognizers.</param>
    public SpeechService(VoiceProfileClient voiceClient,
SpeechApiRecognizersFactory speechApiRecognizersFactory)
    {
        _voiceClient = voiceClient;
        _speechApiRecognizersFactory =
speechApiRecognizersFactory;
    }

    /// <inheritdoc/>
    public async Task<string> CreateProfileAsync()
    {
        using var profile = await
_voiceClient.CreateProfileAsync(VoiceProfileType.TextIndependent
Verification, "en-us");
        return profile.Id;
    }

    /// <inheritdoc/>
    public async Task DeleteProfileAsync(string profileId)
    {
        using var profile = new VoiceProfile(profileId);
        await _voiceClient.DeleteProfileAsync(profile);
    }
```

```
    /// <inheritdoc/>
    public async Task<AuthMethodEnrollmentResponse>
EnrollProfileAsync(string profileId, IFormFile voice, string
passphrase)
    {
        using var audioInputStream = await
CreateAudioInputStreamFromStreamAsync(voice);
        using var audioConfig =
AudioConfig.FromStreamInput(audioInputStream);

        if (!await VerifyPassphraseAsync(audioConfig,
passphrase))
        {
            return new AuthMethodEnrollmentResponse
            {
                Result =
AuthMethodEnrollmentResult.EnrollmentFailed,
                Message = "Recognized text and passphrase does
not match"
            };
        }

        using var profile = new VoiceProfile(profileId,
VoiceProfileType.TextIndependentVerification);
        var result = await
_voiceClient.EnrollProfileAsync(profile, audioConfig);

        return new AuthMethodEnrollmentResponse
        {
            Result = result.RemainingEnrollmentsCount > 0
                ?
AuthMethodEnrollmentResult.EnrollmentInProgress
                :
AuthMethodEnrollmentResult.EnrollmentCompleted,
            RemainingEnrollmentsCount =
result.RemainingEnrollmentsCount
        };
    }

    /// <inheritdoc/>
    public async Task<bool> VerifyAsync(string profileId,
IFormFile voice, string passphrase)
    {
        using var audioInputStream = await
CreateAudioInputStreamFromStreamAsync(voice);
        using var audioConfig =
AudioConfig.FromStreamInput(audioInputStream);

        return await VerifyPassphraseAsync(audioConfig,
passphrase) && await VerifyProfileAsync(audioConfig, profileId);
    }

    private async Task<bool> VerifyPassphraseAsync(AudioConfig
```

```
audioConfig, string passphrase)
    {
        var speechRecognizer =
_speechApiRecognizersFactory.CreateSpeechRecognizer(audioConfig)
;
        var speechToTextResult = await
speechRecognizer.RecognizeOnceAsync();

        return speechToTextResult.Reason ==
ResultReason.RecognizedSpeech &&
               speechToTextResult.Text.Equals(passphrase,
StringComparison.OrdinalIgnoreCase);
    }

    private async Task<bool> VerifyProfileAsync(AudioConfig
audioConfig, string profileId)
    {
        var speechRecognizer =
_speechApiRecognizersFactory.CreateSpeakerRecognizer(audioConfig
);
        var model = SpeakerVerificationModel.FromProfile(new
VoiceProfile(profileId,
VoiceProfileType.TextIndependentVerification));
        var result = await
speechRecognizer.RecognizeOnceAsync(model);

        return result.Score > 0.8;
    }
    private async Task<AudioInputStream>
CreateAudioInputStreamFromStreamAsync(IFormFile voice)
    {
        await using var voiceStream = voice.OpenReadStream();
        var audioInputStream =
AudioInputStream.CreatePushStream();

        var voiceBytes = await voiceStream.ReadAllBytesAsync();
        audioInputStream.Write(voiceBytes, voiceBytes.Length);

        return audioInputStream;
    }
}
```

B.4 AuthenticationService

```
using BioAuthenticator.Core.Enums;
using BioAuthenticator.Core.Exceptions;
using BioAuthenticator.Core.Interfaces;
using BioAuthenticator.Core.Models;
using BioAuthenticator.Infrastructure.Data;
using Microsoft.AspNetCore.Http;
using Microsoft.EntityFrameworkCore;

namespace BioAuthenticator.Infrastructure.Services;

/// <inheritdoc/>
public class AuthenticationService : IAuthenticationService
{
    private readonly IFaceService _faceService;
    private readonly ISpeechService _speechService;
    private readonly AppDbContext _context;
    private readonly IMobileMessagingService _messagingService;

    /// <summary>
    /// Creates a new instance of the <see
cref="AuthenticationService"/> class.
    /// </summary>
    /// <param name="faceService">The face service.</param>
    /// <param name="speechService">The speech service.</param>
    /// <param name="context">The database context.</param>
    /// <param name="messagingService">The mobile messaging
service.</param>
    public AuthenticationService(IFaceService faceService,
ISpeechService speechService, AppDbContext context,
        IMobileMessagingService messagingService)
    {
        _faceService = faceService;
        _speechService = speechService;
        _context = context;
        _messagingService = messagingService;
    }

    /// <inheritdoc/>
    public async Task<bool> AuthenticateAsync(string
appInstallationId, Guid organizationUserId, AuthMethod method,
IFormFile file,
        string? passphrase)
    {
        var installation = await
_context.AppInstallations.FirstOrDefaultAsync(x =>
x.InstallationId == appInstallationId);

        if (installation is null)
        {
            throw new
```

```
BioAuthenticatorEntityNotFoundException(nameof(AppInstallation),
                nameof(AppInstallation.InstallationId),
appInstallationId);
        }

        var result = method switch
        {
            AuthMethod.Face => await
AuthenticateFaceAsync(installation, file),
            AuthMethod.Voice => await
AuthenticateVoiceAsync(installation, file, passphrase),
            _ => throw new BioAuthenticatorException("Invalid
authentication method")
        };

        var authHistoryRecord =
            await
_context.AuthenticationHistories.FirstOrDefaultAsync(x =>
                x.OrganizationUserId == organizationUserId &&
                x.AuthenticationAttemptState ==
AuthenticationAttemptState.InProgress);

        if (authHistoryRecord is null)
        {
            throw new
BioAuthenticatorEntityNotFoundException(nameof(AuthenticationHis
tory),
                new Dictionary<string, string>
                {
                    {
nameof(AuthenticationHistory.OrganizationUserId),
organizationUserId.ToString() },
                    {

nameof(AuthenticationHistory.AuthenticationAttemptState),

AuthenticationAttemptState.InProgress.ToString()
                    }
                });
        }

        authHistoryRecord.AuthenticationAttemptState =
            result ? AuthenticationAttemptState.Approved :
AuthenticationAttemptState.Denied;
        await _context.SaveChangesAsync();

        return result;
    }

    /// <inheritdoc/>
    public async Task<bool>
RequireAuthenticationApprovalAsync(Guid organizationId, Guid
organizationUserId)
```

```
    {
        var user = await _context.OrganizationUsers
            .Include(x => x.AppInstallation)
            .Include(x => x.Organization)
            .FirstOrDefaultAsync(x =>
            x.OrganizationId == organizationId && x.Id ==
organizationUserId);

        if (user is null)
        {
            throw new
BioAuthenticatorEntityNotFoundException(nameof(OrganizationUser)
, new Dictionary<string, string>
            {
                { nameof(OrganizationUser.OrganizationId),
organizationId.ToString() },
                { nameof(OrganizationUser.Id),
organizationUserId.ToString() }
            });
        }

        if (user.AppInstallation?.NotificationServiceToken is
null)
        {
            throw new BioAuthenticatorException("User has no app
installation assigned");
        }

        await
MarkAllActiveAuthenticationAttemptsCanceledAsync(organizationUse
rId);

        await SendAuthenticationRequestNotification(user);

        var historyRecord = new AuthenticationHistory
        {
            AuthenticationAttemptDate = DateTime.UtcNow,
            AuthenticationAttemptState =
AuthenticationAttemptState.InProgress,
            OrganizationUserId = organizationUserId
        };

        await
_context.AuthenticationHistories.AddAsync(historyRecord);
        await _context.SaveChangesAsync();

        return await
WaitForUserAuthApprovalAsync(historyRecord);
    }

    private async Task
SendAuthenticationRequestNotification(OrganizationUser user)
    {
```

```
        await
_messagingService.SendDataAsync(user.AppInstallation.Notificatio
nServiceToken,
            new Dictionary<string, string>
            {
                { "ORGANIZATION_NAME", user.Organization.Name },
                { "ORGANIZATION_USER_GUID", user.Id.ToString() }
            });
    }

    private async Task
MarkAllActiveAuthenticationAttemptsCanceledAsync(Guid
organizationUserId)
    {
        var recordsToCancel = await
_context.AuthenticationHistories.Where(x =>
            x.OrganizationUserId == organizationUserId &&
            x.AuthenticationAttemptState ==

AuthenticationAttemptState.InProgress).ToListAsync();

        recordsToCancel.ForEach(x =>
x.AuthenticationAttemptState =
AuthenticationAttemptState.Canceled);
        await _context.SaveChangesAsync();
    }

    private async Task<bool>
WaitForUserAuthApprovalAsync(AuthenticationHistory
historyRecord, int timeoutInSeconds = 40, int
pollIntervalInSeconds = 2)
    {
        return await Task.Run(async () =>
        {
            while ((DateTime.UtcNow -
historyRecord.AuthenticationAttemptDate).TotalSeconds <
timeoutInSeconds)
            {
                await
_context.Entry(historyRecord).ReloadAsync();

                if (historyRecord.AuthenticationAttemptState !=
AuthenticationAttemptState.InProgress)
                {
                    return
historyRecord.AuthenticationAttemptState ==
AuthenticationAttemptState.Approved;
                }

                await Task.Delay(pollIntervalInSeconds * 1000);
            }

            historyRecord.AuthenticationAttemptState =
```

```
AuthenticationAttemptState.TimedOut;
            await _context.SaveChangesAsync();
            return false;
        });
    }

    private async Task<bool>
AuthenticateFaceAsync(AppInstallation installation, IFormFile
file)
    {
        if (installation.FaceProfileId is null)
        {
            throw new BioAuthenticatorException("Face method is
not available for this installation");
        }

        return await
_faceService.VerifyAsync(installation.FaceProfileId.Value,
file);
    }

    private async Task<bool>
AuthenticateVoiceAsync(AppInstallation installation, IFormFile
file, string? passphrase)
    {
        if (installation.SpeechProfileId is null)
        {
            throw new BioAuthenticatorException("Voice method is
not available for this installation");
        }

        if (passphrase is null)
        {
            throw new BioAuthenticatorException("Passphrase is
required for voice authentication");
        }

        return await
_speechService.VerifyAsync(installation.SpeechProfileId, file,
passphrase);
    }
}
```