ДОДАТОК А

Програмний код:

AccountController.

```
[Route("api/[controller]")]
   [ApiController]
   public class AccountController : Controller
   {
      private readonly IAccountService accountService;
      private EigenFaceRecognizer recognizer;
      private Image<Bgr, Byte> bgrFrame = null;

      public AccountController(IAccountService service)
      {
         this.accountService = service;
      }

      [HttpPost("registration")]
      public async Task<IActionResult> Registration(RegistrationModel model)
      {
         var result = await this.accountService.RegisterAsync(model);
         return this.Ok(result);
      }

      [HttpPost("login")]
      public async Task<IActionResult> Login(LoginModel model)
      {
         var applicationUser = await this.accountService.LogInAsync(model, model.Password);
         return this.Ok(applicationUser);
      }

      [HttpPost("logout")]
      public async Task<IActionResult> Logout()
      {
         await this.accountService.LogOutAsync();
         return this.Ok();
      }

      [HttpPost("biometrics")]
      public async Task<IActionResult> Biometrics([FromForm] FileModel file)
      {
         Bitmap bitmapImage;
         using (var memoryStream = new MemoryStream())
         {
            await file.FormFile.CopyToAsync(memoryStream);
            using (var img = Image.FromStream(memoryStream))
            {
               bitmapImage = (Bitmap)img;
               Rectangle rectangle = new Rectangle(0, 0, bitmapImage.Width, bitmapImage.Height);
```

```
                BitmapData bmpData = bitmapImage.LockBits(rectangle, ImageLockMode.ReadWrite, bit-
mapImage.PixelFormat);

                Image<Bgr, byte> outputImage = new Image<Bgr, byte>(bitmapImage.Width, bit-
mapImage.Height, bmpData.Stride, bmpData.Scan0);
                recognizer = new EigenFaceRecognizer(2);
                FaceRecognizer.PredictionResult result = recognizer.Predict(outputImage.Resize(100, 100,
Inter.Cubic));
                return Ok(result);
            }
        }
    }
  }

  public class FileModel
  {
    public IFormFile FormFile { get; set; }
          }

        AccountService

public class AccountService : IAccountService
  {
    private readonly IAccountRepository accountRepository;

    private readonly IUserManager userManager;

    public AccountService(IAccountRepository accountRepository, IUserManager userManager)
    {
      this.accountRepository = accountRepository;
      this.userManager = userManager;
    }

    public async Task<UserModel> RegisterAsync(RegistrationModel model)
    {
      var response = new UserModel();
      if (model == null)
      {
        response.Errors.Add(ErrorConstants.ModelIsNull);
        return response;
      }

      var existingUser = await this.accountRepository.GetByEmailAsync(model.Email);
      if (existingUser != null)
      {
        response.Errors.Add(ErrorConstants.UserWithSameEmailExists);
        return response;
      }

      var config = new MapperConfiguration(cfg => cfg.CreateMap<RegistrationModel, Application-
User>()
          .ForMember("UserName", opt => opt.MapFrom(x => x.Login)));
      var mapper = new AutoMapper.Mapper(config);
      var applicationUser = mapper.Map<RegistrationModel, ApplicationUser>(model);
      var result = await this.userManager.CreateAsync(applicationUser, model.Password);
      if (!result)
```

```
    {
        response.Errors.Add(ErrorConstants.IncorrectInput);
        return response;
    }

    await this.userManager.AddToRoleAsync(applicationUser);
    return response;
}

public async Task<UserModel> LogInAsync(LoginModel model, string password)
{
    var response = new UserModel();
    if (model == null)
    {
        response.Errors.Add(ErrorConstants.ModelIsNull);
        return response;
    }

    var existingUser = await this.accountRepository.GetByEmailAsync(model.Email);
    if (existingUser == null)
    {
        response.Errors.Add(ErrorConstants.IncorrectInput);
        return response;
    }

    var result = await this.userManager.LogInAsync(existingUser, password);
    if (!result)
    {
        response.Errors.Add(ErrorConstants.IncorrectPassword);
        return response;
    }

    var config = new MapperConfiguration(cfg => cfg.CreateMap<ApplicationUser, UserModel>()
        .ForMember("Login", opt => opt.MapFrom(x => x.UserName)));
    var mapper = new AutoMapper.Mapper(config);
    response = mapper.Map<ApplicationUser, UserModel>(existingUser);
    return response;
}

public Task LogOutAsync()
{
    return Task.Run(() =>
    {
        return this.userManager.LogOutAsync();
    });
}

public async Task<UserModel> GetUserByIdAsync(long id)
{
    var response = new UserModel();
    var existingUser = await this.accountRepository.GetByIdAsync(id);
    if (existingUser == null)
    {
        response.Errors.Add(ErrorConstants.ImpossibleToFindUser);
        return response;
    }
```

```
            var config = new MapperConfiguration(cfg => cfg.CreateMap<ApplicationUser, UserModel>()
                .ForMember("Login", opt => opt.MapFrom(x => x.UserName)));
            var mapper = new AutoMapper.Mapper(config);
            response = mapper.Map<ApplicationUser, UserModel>(existingUser);
            return response;
        }
                }

         AccountRepository.

public class AccountRepository : IAccountRepository
    {
        private readonly UserManager<ApplicationUser> userManager;

        private readonly ApplicationContext.ApplicationContext applicationContext;

        public AccountRepository(UserManager<ApplicationUser> userManager, ApplicationContext.Ap-
plicationContext applicationDbContext)
        {
            this.userManager = userManager;
            this.applicationContext = applicationDbContext;
        }

        public async Task<ApplicationUser> GetByIdAsync(long id)
        {
            var user = await this.applicationContext.Users.FindAsync(id);
            return user;
        }

        public async Task<ApplicationUser> GetByEmailAsync(string email)
        {
            var user = await this.applicationContext.Users.Where(x => x.Email == email).FirstOrDefaultA-
sync();
            return user;
        }

        public async Task<bool> CreateRoleAsync(string roleName)
        {
            var role = new ApplicationRole() { Name = roleName };
            await this.applicationContext.Roles.AddAsync(role);
            var result = await this.applicationContext.SaveChangesAsync();
            return result > 0;
        }
                }
```