ПРОГРАММНЫЙ МОДУЛЬ ОБРАБОТКИ ИНФОРМАЦИИ С ОТВЕТЧИКА ЛЕТАТЕЛЬНОГО АППАРАТА

ТЕКСТ ПРОГРАММЫ

ПМ АЗН

Содержание

[1 Класс запуска ВС ДЭП 3](#_Toc102477310)

[2 Контроллер работы с учетными записями 10](#_Toc102477311)

[3 Контроллер авторизации 12](#_Toc102477312)

[4 Контроллер работы с почтовыми серверами 15](#_Toc102477313)

[5 Контроллер получения писем 17](#_Toc102477314)

[6 Контроллер изменения писем 19](#_Toc102477315)

[7 Контроллер регистрации пользователей 20](#_Toc102477316)

# Класс запуска ВС ДЭП

using System.Reflection;

using Iris.Api.Controllers.ConnectionsControllers;

using Iris.Configuration;

using Iris.Database;

using Iris.Services.AccountsService;

using Iris.Services.AuthService;

using Iris.Services.ClaimsPrincipalHelperService;

using Iris.Services.ConnectionProtocolHelperService;

using Iris.Services.FormatLettersService;

using Iris.Services.ImapClientService;

using Iris.Services.LettersService;

using Iris.Services.MailServersService;

using Iris.Services.Pop3ClientService;

using Iris.Services.RegistrationService.cs;

using Iris.Services.UserService;

using Iris.Stores.AuthRequestStore;

using Iris.Stores.ServiceConnectionStore;

using Iris.Stores.TokensStore;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using Serilog;

using Serilog.Events;

// Logger Configuration

Log.Logger = new LoggerConfiguration()

.MinimumLevel.Override("Microsoft.AspNetCore", LogEventLevel.Information)

.Enrich.FromLogContext()

.WriteTo.Console()

.CreateBootstrapLogger();

var log = Log.ForContext<Program>();

var config = Config.BuildConfig();

var builder = WebApplication.CreateBuilder(args);

#region serilog configuration

var logTemplateConsole = "[{Level:u3}] <{ThreadId}> :: {Message:lj}{NewLine}{Exception}";

var logTemplateFile =

"{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level:u3}] <{ThreadId}> :: {Message:lj}{NewLine}{Exception}";

if (!Directory.Exists(config.Logger.FilePath))

try

{

Directory.CreateDirectory(config.Logger.FilePath);

log.Information($"create directory {config.Logger.FilePath} for logs");

}

catch

{

log.Error("Can't find and create directory for logs");

return;

}

builder.Host.UseSerilog((context, services, configuration) => configuration

.ReadFrom.Configuration(context.Configuration)

.ReadFrom.Services(services)

.Enrich.FromLogContext()

.Enrich.WithThreadId()

.WriteTo.Console(outputTemplate: logTemplateConsole)

.WriteTo.File(

outputTemplate: logTemplateFile,

path: Path.Combine(config.Logger.FilePath, config.Logger.FileName),

shared: true,

rollingInterval: RollingInterval.Day,

fileSizeLimitBytes: config.Logger.LimitFileSize

)

);

#endregion

// Add services to the container.

builder.Services.AddControllers();

builder.Services

.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(config.AuthConfig.JwtSecurityKey),

ValidateIssuer = true,

ValidIssuer = config.AuthConfig.JwtIssuer,

ValidateAudience = true,

ValidAudience = config.AuthConfig.JwtAudience,

RequireExpirationTime = true,

ValidateLifetime = true,

ClockSkew = TimeSpan.FromMinutes(1)

};

options.Events = new JwtBearerEvents

{

OnMessageReceived = \_ =>

{

if (string.IsNullOrEmpty(\_.Token))

{

var fromAuth = \_.Request.Query["auth"];

if (!string.IsNullOrEmpty(fromAuth)) \_.Token = fromAuth;

var fromAccessToken = \_.Request.Query["access\_token"];

if (!string.IsNullOrEmpty(fromAccessToken)) \_.Token = fromAccessToken;

}

return Task.CompletedTask;

}

};

});

builder.Services.AddAuthorization(

options =>

{

options.DefaultPolicy = new AuthorizationPolicyBuilder()

.AddAuthenticationSchemes(JwtBearerDefaults.AuthenticationScheme)

.RequireAuthenticatedUser()

.Build();

});

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "Iris-Api",

Version = "v1"

});

var executingLocation = Assembly.GetExecutingAssembly().Location;

var xmlName = $"{Path.GetFileNameWithoutExtension(executingLocation)}.xml";

var xmlPath = Path.Combine(Path.GetDirectoryName(executingLocation), xmlName);

c.IncludeXmlComments(xmlPath);

});

var dbContext = new DatabaseContext("Data Source=Database\\Database.db");

#region AdminUser

if (!dbContext.Users.Any(\_ => \_.Name == "admin"))

{

dbContext.Users.Add(new User

{

Name = "admin",

Password = "admin",

IsAdmin = true,

Token = "GCAC4UPWTN6MS552"

});

dbContext.SaveChanges();

}

#endregion

builder.Services.AddSingleton(config);

builder.Services.AddSingleton(dbContext);

builder.Services.AddSingleton<IServerConnectionStore, ServerConnectionStore>();

builder.Services.AddSingleton<IAuthRequestsStore, AuthRequestsStore>();

builder.Services.AddSingleton<ITokensStore, TokensStore>();

builder.Services.AddSingleton<IUserService, UserService>();

builder.Services.AddSingleton<IAuthService, AuthService>();

builder.Services.AddSingleton<IConnectionProtocolHelperService, ConnectionProtocolHelperService>();

builder.Services.AddScoped<IClaimsPrincipalHelperService, ClaimsPrincipalHelperService>();

builder.Services.AddSingleton<IMailServersService, MailServersService>();

builder.Services.AddScoped<ILetterService, LetterService>();

builder.Services.AddScoped<IFormatLettersSevice, FormatLettersSevice>();

builder.Services.AddScoped<IRegistrationService, RegistrationService>();

builder.Services.AddScoped<IPop3ClientService, Pop3ClientService>();

builder.Services.AddScoped<IImapClientService, ImapClientService>();

builder.Services.AddScoped<IAccountsService, AccountsService>();

var app = builder.Build();

#region AddRequiredServers

var mailServersService = app.Services.GetRequiredService<IMailServersService>();

try

{

mailServersService.NewMailServer(new MailServerContract

{

Host = "imap.mail.ru",

Port = 993,

Name = "VK",

IsPrivate = false

});

}

catch

{

// ignored

}

try

{

mailServersService.NewMailServer(new MailServerContract

{

Host = "pop.mail.ru",

Port = 995,

Name = "VK",

IsPrivate = false

});

}

catch

{

// ignored

}

try

{

mailServersService.NewMailServer(new MailServerContract

{

Host = "imap.yandex.ru",

Port = 993,

Name = "Яндекс",

IsPrivate = false

});

}

catch

{

// ignored

}

try

{

mailServersService.NewMailServer(new MailServerContract

{

Host = "pop.yandex.ru",

Port = 995,

Name = "Яндекс",

IsPrivate = false

});

}

catch

{

// ignored

}

try

{

mailServersService.NewMailServer(new MailServerContract

{

Host = "imap.gmail.com",

Port = 993,

Name = "Google",

IsPrivate = false

});

}

catch

{

// ignored

}

try

{

mailServersService.NewMailServer(new MailServerContract

{

Host = "pop.gmail.com",

Port = 995,

Name = "Google",

IsPrivate = false

});

}

catch

{

// ignored

}

try

{

mailServersService.NewMailServer(new MailServerContract

{

Host = "outlook.office365.com",

Port = 993,

Name = "Outlook",

IsPrivate = false

});

}

catch

{

// ignored

}

try

{

mailServersService.NewMailServer(new MailServerContract

{

Host = "outlook.office365.com",

Port = 995,

Name = "Outlook",

IsPrivate = false

});

}

catch

{

// ignored

}

#endregion

app.UseSerilogRequestLogging();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

app.UseSwagger();

app.UseSwaggerUI(c => { c.SwaggerEndpoint("/swagger/v1/swagger.json", "Iris-Api"); });

app.Use(async (context, next) =>

{

try

{

if (context.Response.HasStarted)

Log.Information(

"Current user identity: {Name} AuthenticationType: {AuthenticationType}",

context.User.Identity.Name, context.User.Identity.AuthenticationType);

}

catch (Exception exc)

{

Log.Error("{Message}", exc.Message);

}

await next();

});

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints => { endpoints.MapControllers(); });

app.Run();

# Контроллер работы с учетными записями

using Iris.Services.AccountsService;

using Iris.Services.ClaimsPrincipalHelperService;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace Iris.Api.Controllers.AccountsControllers;

/// <summary>

/// Контроллер работы с учетными записями

/// </summary>

[Authorize]

public class AccountsController : Controller

{

private readonly IAccountsService \_accountsService;

private readonly IClaimsPrincipalHelperService \_claimsPrincipalHelperService;

/// <summary>

/// .ctor

/// </summary>

public AccountsController(IAccountsService accountsService,

IClaimsPrincipalHelperService claimsPrincipalHelperService)

{

\_accountsService = accountsService;

\_claimsPrincipalHelperService = claimsPrincipalHelperService;

}

/// <summary>

/// Добавить новую учетную запись

/// </summary>

/// <param name="contract">Запрос добавления новой учетной записи</param>

[HttpPost("~/api/accounts/add")]

[AllowAnonymous]

[ProducesResponseType(typeof(void), 200)]

public IActionResult AddNewAccount([FromBody] AccountRequestContract contract)

{

var userId = \_claimsPrincipalHelperService.GetUserId(User);

\_accountsService.AddNewAccount(userId, contract);

return Ok();

}

/// <summary>

/// Удалить учетную запись

/// </summary>

/// <param name="accId">Id учетной записи</param>

[HttpDelete("~/api/accounts/{accId}")]

[AllowAnonymous]

[ProducesResponseType(typeof(void), 200)]

public IActionResult RemoveAccount(int accId)

{

var userId = \_claimsPrincipalHelperService.GetUserId(User);

\_accountsService.RemoveAccount(userId, accId);

return Ok();

}

}

# Контроллер авторизации

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using Iris.Api.Results;

using Iris.Database;

using Iris.Services.AuthService;

using Iris.Services.ClaimsPrincipalHelperService;

using Iris.Stores.AuthRequestStore;

using Iris.Stores.TokensStore;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using ILogger = Serilog.ILogger;

namespace Iris.Api.Controllers.AuthControllers;

/// <summary>

/// Контроллер авторизации

/// </summary>

[Authorize]

public class AuthController : Controller

{

private static readonly ILogger Log = Serilog.Log.ForContext<AuthController>();

private readonly IAuthRequestsStore \_authRequestsStore;

private readonly IAuthService \_authService;

private readonly IClaimsPrincipalHelperService \_claimsPrincipalHelperService;

private readonly ITokensStore \_tokensStore;

/// <summary>

/// .ctor

/// </summary>

public AuthController(IAuthRequestsStore authRequestsStore,

IAuthService authService, ITokensStore tokensStore,

IClaimsPrincipalHelperService claimsPrincipalHelperService)

{

\_authRequestsStore = authRequestsStore;

\_authService = authService;

\_tokensStore = tokensStore;

\_claimsPrincipalHelperService = claimsPrincipalHelperService;

}

/// <summary>

/// Инициализировать авторизацию

/// </summary>

/// <returns>Id запроса авторизации</returns>

[HttpPost("~/api/authorize")]

[AllowAnonymous]

[ProducesResponseType(typeof(int), 201)]

public IActionResult InitAuth()

{

var request = \_authRequestsStore.CreateRequest();

return Created(

$"/api/authorize/{Uri.EscapeDataString(request.Id)}",

request.Id

);

}

/// <summary>

/// Выполнить авторизацию

/// </summary>

/// <param name="id">Id запроса авторизации</param>

/// <param name="authRequest">Контракт запроса авторизации</param>

/// <returns>Контракт ответа авторизации</returns>

[HttpPut("~/api/authorize/{id}")]

[AllowAnonymous]

[ProducesResponseType(typeof(AuthResponseContract), 200)]

public IActionResult ExecuteAuthorization(string id, [FromBody] AuthRequestContract authRequest)

{

var operationRequest = \_authRequestsStore.FindRequest(id);

if (operationRequest == null) return new AuthErrorResult(message: "Ошибка авторизации");

ClaimsIdentity identity;

User user;

try

{

(identity, user) = \_authService.Authorize(operationRequest, authRequest);

}

catch

{

Log.Error($"Ошибка авторизации пользователя {authRequest.Login}");

throw;

}

if (identity == null)

{

Log.Error($"Ошибка авторизации пользователя {authRequest.Login}");

return new AuthErrorResult();

}

var (token, expires) = \_authService.GenerateToken(identity.Claims, user);

var roles = identity.Claims.Where(\_ => \_.Type == identity.RoleClaimType)

.Select(\_ => \_.Value).ToList();

var userIdVal = identity.Claims.First(c => c.Type == JwtRegisteredClaimNames.Sid).Value;

var userId = int.Parse(userIdVal);

\_tokensStore.AddOrUpdate(userId.ToString(), token);

Log.Information($"Пользователь {user.Name} авторизован");

return Ok(new AuthResponseContract

{

UserId = userIdVal,

Login = identity.Name,

Token = token,

Roles = roles,

TokenType = JwtBearerDefaults.AuthenticationScheme

});

}

/// <summary>

/// Де-авторизация

/// </summary>

[HttpPost("~/api/authorize/deauth")]

[ProducesResponseType(typeof(OkResult), 200)]

public IActionResult DeAuth()

{

var userId = \_claimsPrincipalHelperService.GetUserId(User);

\_tokensStore.Remove(userId.ToString());

Log.Information($"User {userId} is de-auth");

Response?.Headers?.Add("Clear-Site-Data", "\"cache\", \"cookies\", \"storage\"");

return Ok();

}

/// <summary>

/// Проверка авторизированности

/// </summary>

/// <returns></returns>

[HttpGet("~/api/authorize/isauth")]

[AllowAnonymous]

[ProducesResponseType(typeof(bool), 200)]

public IActionResult IsAuth()

{

return Ok(User?.Identity?.IsAuthenticated ?? false);

}

}

# Контроллер работы с почтовыми серверами

using Iris.Database;

using Iris.Services.ClaimsPrincipalHelperService;

using Iris.Services.MailServersService;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace Iris.Api.Controllers.ConnectionsControllers;

/// <summary>

/// Контроллер работы с почтовыми серверами

/// </summary>

[Authorize]

public class MailServersController : Controller

{

private readonly IClaimsPrincipalHelperService \_claimsPrincipalHelperService;

private readonly IMailServersService \_mailServersService;

/// <summary>

/// .ctor

/// </summary>

public MailServersController(IMailServersService mailServersService,

IClaimsPrincipalHelperService claimsPrincipalHelperService)

{

\_mailServersService = mailServersService;

\_claimsPrincipalHelperService = claimsPrincipalHelperService;

}

/// <summary>

/// Получить список аккаунтов к почтовым серверам пользователя

/// </summary>

[HttpGet("~/api/connections/mailservers/accounts/users")]

[ProducesResponseType(typeof(IEnumerable<MailServerAccountContract>), 200)]

public IActionResult GetMailServersAccounts()

{

var userId = \_claimsPrincipalHelperService.GetUserId(User);

var serverAccounts = \_mailServersService.GetMailServerAccounts(userId);

return Ok(serverAccounts);

}

/// <summary>

/// Получить список доступных почтовых серверов

/// </summary>

[HttpGet("~/api/connections/mailservers/accounts/availables")]

[ProducesResponseType(typeof(IEnumerable<MailServer>), 200)]

public IActionResult GetAvailableMailServers()

{

var userId = \_claimsPrincipalHelperService.GetUserId(User);

var serverAccounts = \_mailServersService.GetAvailableMailServers(userId);

return Ok(serverAccounts);

}

/// <summary>

/// Добавить новый почтовый сервер

/// </summary>

/// <param name="mailServerContract">Контракт добавления сервера</param>

[HttpPost("~/api/connections/mailservers")]

[ProducesResponseType(typeof(void), 200)]

public IActionResult AddUserMailService([FromBody] MailServerContract mailServerContract)

{

\_mailServersService.NewMailServer(mailServerContract);

return Ok();

}

}

# Контроллер получения писем

using Iris.Services.ClaimsPrincipalHelperService;

using Iris.Services.FormatLettersService;

using Iris.Services.LettersService;

using Iris.Services.LettersService.Contracts;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace Iris.Api.Controllers.LettersControllers

{

/// <summary>

/// Контроллер получения писем

/// </summary>

[Authorize]

public class GetLettersController : Controller

{

private readonly ILetterService \_letterService;

private readonly IFormatLettersSevice \_formatLettersSevice;

private readonly IClaimsPrincipalHelperService \_claimsPrincipalHelperService;

/// <summary>

/// .ctor

/// </summary>

public GetLettersController(ILetterService letterService, IFormatLettersSevice formatLettersSevice, IClaimsPrincipalHelperService claimsPrincipalHelperService)

{

\_letterService = letterService;

\_formatLettersSevice = formatLettersSevice;

\_claimsPrincipalHelperService = claimsPrincipalHelperService;

}

/// <summary>

/// Получить письма по запросу

/// </summary>

/// <param name="lettersRequest"></param>

/// <returns></returns>

[HttpGet("~/api/letters")]

[ProducesResponseType(typeof(IEnumerable<LetterContract>), 200)]

public IActionResult GetLetters([FromQuery] LettersRequest lettersRequest)

{

var userId = \_claimsPrincipalHelperService.GetUserId(User);

var letters = \_letterService.GetLetters(userId, lettersRequest);

return Ok(letters);

}

/// <summary>

/// Получить письма по запросу

/// </summary>

/// <param name="lettersRequest">Запрос писем</param>

/// <param name="format">Формат</param>

/// <returns></returns>

[HttpGet("~/api/{format}/letters")]

[ProducesResponseType(typeof(IEnumerable<LetterContract>), 200)]

public IActionResult GetLetters(string format, [FromQuery] LettersRequest lettersRequest)

{

var userId = \_claimsPrincipalHelperService.GetUserId(User);

var needFormat = \_formatLettersSevice.GetFormat(format);

var letters = \_letterService.GetLetters(userId, lettersRequest);

var formattedLetters = \_formatLettersSevice.FormatLetters(letters, needFormat);

return Ok(formattedLetters);

}

}

}

# Контроллер изменения писем

using Iris.Services.ClaimsPrincipalHelperService;

using Iris.Services.LettersService;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace Iris.Api.Controllers.LettersControllers;

/// <summary>

/// Контроллер изменения писем

/// </summary>

[Authorize]

public class UpdateLettersController : Controller

{

private readonly IClaimsPrincipalHelperService \_claimsPrincipalHelperService;

private readonly ILetterService \_letterService;

/// <summary>

/// .ctor

/// </summary>

public UpdateLettersController(ILetterService letterService,

IClaimsPrincipalHelperService claimsPrincipalHelperService)

{

\_letterService = letterService;

\_claimsPrincipalHelperService = claimsPrincipalHelperService;

}

/// <summary>

/// Установить флаг

/// </summary>

/// <param name="accId">Id учетной записи</param>

/// <param name="letterId">Id письма</param>

/// <param name="flag">Флаг</param>

[HttpPost("~/api/letters/accaunt/{accId}/letter/{letterId}/flag/{flag}")]

[ProducesResponseType(typeof(OkResult), 200)]

public IActionResult ChangeFlag(int accId, string letterId, int flag)

{

var userId = \_claimsPrincipalHelperService.GetUserId(User);

\_letterService.ChangeFlag(userId, accId, letterId, flag);

return Ok();

}

/// <summary>

/// Удалить письио

/// </summary>

/// <param name="accId">Id учетной записи</param>

/// <param name="letterId">Id письма</param>

[HttpDelete("~/api/letters/accaunt/{accId}/letter/{letterId}")]

[ProducesResponseType(typeof(OkResult), 200)]

public IActionResult RemoveLetter(int accId, string letterId)

{

var userId = \_claimsPrincipalHelperService.GetUserId(User);

\_letterService.RemoveLetter(userId, accId, letterId);

return Ok();

}

}

# Контроллер регистрации пользователей

using Iris.Api.Controllers.RegistrationControllers;

using Iris.Services.RegistrationService.cs;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace Iris.Api.Controllers.RegistrationController;

/// <summary>

/// Контроллер регистрации пользователей

/// </summary>

[Authorize]

public class RegistrationController : Controller

{

private readonly IRegistrationService \_registrationService;

/// <summary>

/// .ctor

/// </summary>

public RegistrationController(IRegistrationService registrationService)

{

\_registrationService = registrationService;

}

/// <summary>

/// Зарегисрировать пользователя

/// </summary>

/// <param name="contract">Контракт создания пользователя</param>

[HttpPost("~/api/registration")]

[AllowAnonymous]

[ProducesResponseType(typeof(RegistrationResponseContract), 200)]

public IActionResult RegisterUser([FromBody] RegistrationRequestContract contract)

{

return Ok(\_registrationService.RegisterUser(contract));

}

}