

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2025 р.

Вебзастосунок для автоматичного підбору вакансій на основі резюме та адаптації резюме за допомогою нейромереж для ІТ-галузі (комплексна тема).

Вебзастосунок та агрегація вакансій

Текст програми

КПІ.ІІ-1122.045440.03.12

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Катерина ЛІЩУК

Нормоконтроль:

_____ Катерина ЛІЩУК

Виконавець:

_____ Юрій РЯБОВ

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду

<https://github.com/YuraRiabov/KolybaResume>

Файл **DouVacancyAggregatorService.cs**

Реалізація функціональної задачі збору вакансій з сайту dou.ua

```
using System.Net;
using System.Net.Http.Json;
using AutoMapper;
using KolybaResume.BLL.Models;
using KolybaResume.BLL.Services.Abstract;
using KolybaResume.BLL.Services.Base;
using KolybaResume.BLL.Services.Utility;
using KolybaResume.Common.Enums;
using KolybaResume.DAL.Context;
using KolybaResume.DAL.Entities;
using Microsoft.EntityFrameworkCore;

namespace KolybaResume.BLL.Services.Aggregators;

public class DouVacancyAggregatorService(KolybaResumeContext context,
IMapper mapper) : BaseService(context, mapper), IAggregator
{
    public async Task<List<Vacancy>> Aggregate()
    {
        var isFirstRun = !_context.Vacancies.Any(v => v.Source ==
VacancySource.Dou);
        var companyLinks = await _context.Companies.Select(c =>
c.Url).Take(1500).ToListAsync();
        var addedVacancies = new List<Vacancy>();
        var allVacanciesIds = new List<int>();

        try
        {
            foreach (var link in companyLinks)
            {
                var vacancies = await GetVacancies($"{link}vacancies/export/",
isFirstRun, allVacanciesIds);
                await _context.Vacancies.AddRangeAsync(vacancies);
                await _context.SaveChangesAsync();
                addedVacancies.AddRange(vacancies);
            }
        }
        finally
```

```

    {
        var vacanciesToDelete = (await _context.Vacancies.ToListAsync())
            .Where(v => v.Source == VacancySource.Dou &&
                !allVacanciesIds.Contains(DouVacancyIdExtractor.GetId(v.Url)));

        _context.Vacancies.RemoveRange(vacanciesToDelete);
        await _context.SaveChangesAsync();
    }

    return addedVacancies;
}

private async Task<Vacancy[]> GetVacancies(string url, bool isFirstRun,
List<int> allVacanciesIds)
{
    var handler = new HttpClientHandler
    {
        UseCookies = true,
        CookieContainer = new CookieContainer(),
        AutomaticDecompression = DecompressionMethods.GZip |
DecompressionMethods.Deflate
    };

    using var client = new HttpClient(handler);

    client.DefaultRequestHeaders.UserAgent.ParseAdd(
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64) " +
        "AppleWebKit/537.36 (KHTML, like Gecko) " +
        "Chrome/114.0.0.0 Safari/537.36"
    );

    client.DefaultRequestHeaders.Accept.ParseAdd("application/json,
text/javascript, */*; q=0.01");
    client.DefaultRequestHeaders.TryAddWithoutValidation("X-Requested-
With", "XMLHttpRequest");

    using var request = new HttpRequestMessage(HttpMethod.Get, url);

    var response = await client.SendAsync(request);
    response.EnsureSuccessStatusCode();

    var vacancies = await
response.Content.ReadFromJsonAsync<VacancyModel[]>();
    allVacanciesIds.AddRange(vacancies?.Select(v =>

```

```

        DouVacancyIdExtractor.GetId(v.Link)) ?? []);
        return _mapper.Map<Vacancy[]>(vacancies?.Where(v => isFirstRun || v.Date
> DateTime.Today.AddDays(-1)));
    }
}

```

Файл **PostJobVacancyAggregatorService.cs**

Реалізація функціональної задачі збору вакансій з сайту postjobfree.com

```

using AutoMapper;
using KolybaResume.BLL.Services.Abstract;
using KolybaResume.BLL.Services.Base;
using KolybaResume.BLL.Services.Scrappers;
using KolybaResume.Common.Enums;
using KolybaResume.DAL.Context;
using KolybaResume.DAL.Entities;
using Microsoft.EntityFrameworkCore;

namespace KolybaResume.BLL.Services.Aggregators;

public class PostJobVacancyAggregatorService(KolybaResumeContext context,
IMapper mapper) : BaseService(context, mapper), IAggregator
{
    private static readonly string[] CategoryQueries = [
        "\"mobile developer\"",
        "\"customer support\"",
        "\"project manager\"",
        "devops",
        "\"data analyst\"",
        "qa",
        "\"sales manager\"",
        "\"ux designer\"",
    ];

    public async Task<List<Vacancy>> Aggregate()
    {
        var isFirstRun = !_context.Vacancies.Any(v => v.Source ==
VacancySource.PostJob);
        var addedVacancies = new List<Vacancy>();
        var allVacanciesLinks = new List<string>();

        try
        {
            foreach (var query in CategoryQueries)
            {

```

```

        var vacancies = PostJobFreeVacanciesScraper.Scrape(query);
        allVacanciesLinks.AddRange(vacancies.Select(v => v.Link));

        var vacanciesToAdd = _mapper.Map<Vacancy[]>(vacancies.Where(v
=> isFirstRun || v.Date > DateTime.Today.AddDays(-1)));
        await _context.Vacancies.AddRangeAsync(vacanciesToAdd);
        await _context.SaveChangesAsync();
        addedVacancies.AddRange(vacanciesToAdd);
    }
}
finally
{
    var vacanciesToDelete = (await _context.Vacancies.ToListAsync())
        .Where(v => v.Source == VacancySource.PostJob &&
            !allVacanciesLinks.Contains(v.Url));

    _context.Vacancies.RemoveRange(vacanciesToDelete);
    await _context.SaveChangesAsync();
}

return addedVacancies;
}
}

```

Файл PostJobFreeVacanciesScraper.cs

Реалізація функціональної задачі скрапінгу вакансій з сайту postjobfree.com за фразою пошуку

```

using System.Globalization;
using KolybaResume.BLL.Models;
using KolybaResume.Common.Enums;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;

namespace KolybaResume.BLL.Services.Scrappers;

public static class PostJobFreeVacanciesScraper
{
    private const string BaseUrl = "https://www.postjobfree.com/jobs";

    public static List<VacancyModel> Scrape(string query)
    {
        var vacanciesList = new List<VacancyModel>();
        var options = new ChromeOptions();

```

```

options.AddArgument("--headless");
var driver = new ChromeDriver(options);
var wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));

try
{
    var firstPageUrl =
    $"{{BaseUrl}}?q={{Uri.EscapeDataString(query)}}&r=100&p=1";
    driver.Navigate().GoToUrl(firstPageUrl);
    wait.Until(drv => drv.FindElements(By.CssSelector(".pager")).Count !=
0);

    var pagerLinksCount = driver
        .FindElements(By.CssSelector(".pager"))
        .Where(a =>
        {
            var paginationText = a.Text.Trim();
            return !string.Equals(paginationText, "Previous",
StringComparison.OrdinalIgnoreCase)
                && !string.Equals(paginationText, "Next",
StringComparison.OrdinalIgnoreCase);
        })
        .Count();

    int totalPages = Math.Min(pagerLinksCount, 5);

    for (int page = 1; page <= totalPages; page++)
    {
        var listUrl =
        $"{{BaseUrl}}?q={{Uri.EscapeDataString(query)}}&r=100&p={{page}}";
        driver.Navigate().GoToUrl(listUrl);

        wait.Until(drv => drv.FindElements(By.CssSelector("h3 > a")).Count !=
0);

        var jobLinks = driver
            .FindElements(By.CssSelector("h3 > a"))
            .Select(elem => elem.GetAttribute("href"))
            .Where(link => !string.IsNullOrEmpty(link) &&
link.Contains("postjobfree.com/job"))
            .ToList();

        if (jobLinks.Count == 0)
        {
            break;

```

```

    }

    foreach (var link in jobLinks)
    {
        try
        {
            var vacancy = new VacancyModel
            {
                Link = link,
                Source = VacancySource.PostJob,
                Category = query
            };

            driver.Navigate().GoToUrl(link);
            wait.Until(drv => drv.FindElement(By.TagName("h1")));

            vacancy.Title = driver.FindElement(By.TagName("h1")).Text;
            vacancy.Location =
                SafeFindText(By.CssSelector(".colorLocation"), driver);
            vacancy.Salary = SafeFindText(By.CssSelector(".colorSalary"),
                driver);

            vacancy.Date = DateTime.ParseExact(
                SafeFindText(By.CssSelector(".colorDate"), driver),
                "MMMM d, yyyy",
                CultureInfo.InvariantCulture
            );
            vacancy.Description =
                driver.FindElement(By.CssSelector(".normalText")).Text.Trim();

            vacanciesList.Add(vacancy);
        }
        catch (Exception)
        {
            Console.WriteLine($"Unable to get vacancy, link: {link}");
        }
    }
}
finally
{
    driver.Quit();
}

return vacanciesList;
}

```

```

private static string SafeFindText(By by, ChromeDriver driver)
{
    try
    {
        return driver.FindElement(by).Text.Trim();
    }
    catch (NoSuchElementException)
    {
        return string.Empty;
    }
}
}

```

Файл PostJobFreeVacancyScraper.cs

Реалізація функціональної задачі скрапінгу тексту вакансій за посиланням з postjobfree.com

```

using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;

namespace KolybaResume.BLL.Services.Scrappers;

public class PostJobFreeVacancyScraper
{
    public async Task<string> Scrape(string url)
    {
        var options = new ChromeOptions();
        options.AddArgument("--headless");
        using var driver = new ChromeDriver(options);

        try
        {
            await driver.Navigate().GoToUrlAsync(url);

            var wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));
            var description = wait.Until(d =>
                d.FindElement(By.CssSelector(".normalText")));

            return description.Text.Trim();
        }
        finally
        {
            driver.Quit();
        }
    }
}

```



```

    }
}
}

```

Файл **DouCompanyScrapper.cs**

Реалізація функціональної задачі збору посилань на компанії з сайту dou.ua
using OpenQA.Selenium;

```

using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;
using SeleniumExtras.WaitHelpers;

```

```

namespace KolybaResume.BLL.Services.Scrappers;

```

```

public static class DouCompanyScrapper
{

```

```

    public static string[] Scrape()
    {

```

```

        var options = new ChromeOptions();
        options.AddArgument("--headless");
        using var driver = new ChromeDriver(options);

```

```

        driver.Navigate().GoToUrl("https://jobs.dou.ua/companies/");

```

```

        var wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));

```

```

        while (true)
        {

```

```

            try
            {

```

```

                var moreButton =

```

```

wait.Until(ExpectedConditions.ElementToBeClickable(
    By.LinkText("Більше компаній")
));

```

```

        var count = driver.FindElements(By.CssSelector("a.cn-a")).Count;

```

```

        if (count > 1500)
        {
            break;
        }

```

```

        moreButton.Click();

```

```

        wait.Until(drv =>

```

```

        drv.FindElements(By.CssSelector("a.cn-a")).Count > count
    );
}
catch (WebDriverTimeoutException)
{
    break;
}
catch (NoSuchElementException)
{
    break;
}
}

return driver.FindElements(By.CssSelector("a.cn-a"))
    .Select(e => e.GetAttribute("href"))
    .Distinct()
    .ToArray!();
}
}

```

Файл TextExtractorService.cs

Реалізація функціональної задачі зчитування тексту резюме

```

using System.Text;
using DocumentFormat.OpenXml.Packaging;
using NPOI.HWPF;
using NPOI.HWPF.Extractor;
using UglyToad.PdfPig;

namespace KolybaResume.BLL.Services;

public static class TextExtractorService
{
    public static string ReadPdf(Stream stream)
    {
        var sb = new StringBuilder();
        using (var document = PdfDocument.Open(stream))
        {
            foreach (var page in document.GetPages())
            {
                sb.AppendLine(page.Text);
            }
        }
    }
}

```

```

        return sb.ToString();
    }

    public static string ReadDocx(Stream stream)
    {
        using var document = WordprocessingDocument.Open(stream, false);
        return document.MainDocumentPart!.Document.Body!.InnerText;
    }

    public static string ReadDoc(Stream stream)
    {
        var document = new HWPFDocument(stream);
        var extractor = new WordExtractor(document);
        return extractor.Text;
    }
}

```

Файл UserService.cs

Реалізація функціональної задачі управління користувачами і їх резюме

```

using AutoMapper;
using FirebaseAdmin.Auth;
using KolybaResume.BLL.Extensions;
using KolybaResume.BLL.Services.Abstract;
using KolybaResume.BLL.Services.Base;
using KolybaResume.Common.DTO.User;
using KolybaResume.DAL.Context;
using KolybaResume.DAL.Entities;
using Microsoft.AspNetCore.Http;
using Microsoft.EntityFrameworkCore;

namespace KolybaResume.BLL.Services;

public class UserService(KolybaResumeContext context, IMapper mapper,
    FirebaseAuth firebaseAuth, IHttpContextAccessor httpContextAccessor,
    IMachineLearningApiService apiService) : BaseService(context, mapper),
    IUserService
{
    public async Task<UserDto> GetCurrent()
    {
        var currentUser = await GetCurrentInternal();

        await AddClaims(currentUser.Uid, currentUser.Id);
    }
}

```

```

        var currentUserDto = _mapper.Map<UserDto>(currentUser);
        return currentUserDto;
    }

    public async Task<bool> CheckExisting(string email)
    {
        return await _context.Users.AnyAsync(u => u.Email == email);
    }

    public async Task<UserDto> Create(NewUserDto userDto)
    {
        if (userDto is null)
        {
            throw new ArgumentNullException(nameof(userDto), "New user cannot be null");
        }

        var userEntity = await _context.Users.FirstOrDefaultAsync(u =>
u.Email.Equals(userDto.Email));
        if (userEntity is not null)
        {
            return _mapper.Map<UserDto>(userEntity);
        }

        var newUser = _mapper.Map<NewUserDto, User>(userDto);
        var user = (await _context.Users.AddAsync(newUser)).Entity;
        await _context.SaveChangesAsync();

        await AddClaims(user.Uid, user.Id);

        return _mapper.Map<User, UserDto>(user);
    }

    public async Task AddResume(string text)
    {
        var userId = (await GetCurrentInternal()).Id;
        var existingResume = await _context.Resumes.FirstOrDefault(r =>
r.UserId == userId);

        if (existingResume is not null)
        {
            _context.Resumes.Remove(existingResume);
        }
    }

```

```

var resume = new Resume
{
    Text = text,
    UserId = userId
};

await _context.Resumes.AddAsync(resume);
await _context.SaveChangesAsync();

if (await apiService.NotifyResumeCreated(resume.Id))
{
    return;
}

_context.Resumes.Remove(resume);
await _context.SaveChangesAsync();

throw new Exception("Could not add resume");
}

public async Task<long> GetResumeId()
{
    var user = await GetCurrentInternal();
    return user.Resume?.Id ?? 0;
}

private async Task AddClaims(string? uid, long? id)
{
    if (uid is null || id is null)
    {
        return;
    }

    var userRecord = await firebaseAuth.GetUserAsync(uid);

    if (userRecord.CustomClaims.ContainsKey("id"))
    {
        return;
    }

    var userClaims = new Dictionary<string, object>
    {
        { "id", id }
    };
};

```

```

        await firebaseAuth.SetCustomUserClaimsAsync(uid, userClaims);
    }

    private async Task<User> GetCurrentInternal()
    => await _context.Users.Include(u => u.Resume).FirstOrDefaultAsync(u =>
u.Uid == GetCurrentId())
        ?? throw new KeyNotFoundException("User doesn't exist");

    private string? GetCurrentId()
    {
        var userId = httpContextAccessor.HttpContext.User.GetUid();
        return userId;
    }
}

```

Файл VacancyService.cs

Реалізація функціональної задачі отримання вакансій та адаптації резюме

```

using AutoMapper;
using KolybaResume.BLL.Models;
using KolybaResume.BLL.Services.Abstract;
using KolybaResume.BLL.Services.Base;
using KolybaResume.BLL.Services.Scrappers;
using KolybaResume.BLL.Services.Utility;
using KolybaResume.Common.DTO.Vacancy;
using KolybaResume.Common.Enums;
using KolybaResume.DAL.Context;
using Microsoft.EntityFrameworkCore;

namespace KolybaResume.BLL.Services;

public class VacancyService(
    KolybaResumeContext context,
    IMapper mapper,
    IMachineLearningApiService apiService,
    IUserService userService) : BaseService(context, mapper), IVacancyService
{
    public async Task<VacancyTextDto> ParseVacancy(string vacancyUrl)
    {
        if (vacancyUrl.Contains("jobs.dou.ua"))
        {
            var vacancies = await _context.Vacancies.Where(v => v.Source ==

```

```

VacancySource.Dou).ToListAsync();
    var vacancy = vacancies.FirstOrDefault(v =>
        DouVacancyIdExtractor.Compare(v.Url, vacancyUrl));

    if (vacancy != null)
    {
        return new VacancyTextDto
        {
            Text = vacancy.CleanedText,
        };
    }
}

if (vacancyUrl.Contains("www.postjobfree.com/job"))
{
    return new VacancyTextDto
    {
        Text = await new PostJobFreeVacancyScraper().Scrape(vacancyUrl)
    };
}

throw new ArgumentException("Invalid URL");
}

public async Task<VacancyDto[]> Get()
{
    var resumeId = await userService.GetResumeId();

    var scores = await apiService.GetVacancyScores(resumeId);

    var vacancies =
        (await _context.Vacancies.ToListAsync()).Where(v => scores.Any(score
=> score.VacancyId == v.Id));
    var dtos = _mapper.Map<VacancyDto[]>(vacancies);

    foreach (var dto in dtos)
    {
        dto.Score = scores.First(score => score.VacancyId == dto.Id).Score;
    }

    return dtos.OrderByDescending(d => d.Score).ToArray();
}

public async Task<AdaptationResponseDto> AdaptResume(string vacancyText,
bool shouldClean = true)

```

```

    {
        var resumeId = await userService.GetResumeId();

        return _mapper.Map<AdaptationResponseDto>(await
        apiService.GetResumeAdaptation(new ResumeAdaptationRequest
        {
            ResumeId = resumeId,
            VacancyText = vacancyText,
            Clean = shouldClean
        }));
    }

    public async Task<AdaptationResponseDto> AdaptResume(long vacancyId)
    {
        var vacancyText = (await _context.Vacancies.FirstOrDefaultAsync(v => v.Id
        == vacancyId))??.CleanedText;

        if (vacancyText == null)
        {
            throw new ArgumentException("Vacancy not found");
        }

        return await AdaptResume(vacancyText, false);
    }
}

```

Файл AggregatorJob.cs

Реалізація функціональної задачі збору вакансій та надсилання імейлів

```

using KolybaResume.BLL.Models;
using KolybaResume.BLL.Services.Abstract;
using KolybaResume.DAL.Context;
using KolybaResume.DAL.Entities;
using Microsoft.EntityFrameworkCore;
using Quartz;

namespace KolybaResume.Jobs;

public class AggregatorJob(IEnumerable<IAggregator> aggregators,
    KolybaResumeContext dbContext, IEmailService emailService,
    IMachineLearningApiService apiService) : IJob
{

```



```

public async Task Execute(IJobExecutionContext context)
{
    var addedVacancies = new List<Vacancy>();

    foreach (var aggregator in aggregators)
    {
        addedVacancies.AddRange(await aggregator.Aggregate());
    }

    var scores = new List<VacancyScoreResponse>();

    foreach (var batch in addedVacancies.Chunk(96))
    {
        scores.AddRange(await apiService.NotifyVacanciesUpdated(batch.Select(v
=> v.Id).ToArray()));
    }

    foreach (var userScore in scores.GroupBy(s => s.UserId))
    {
        var user = await dbContext.Users.FirstOrDefaultAsync(u => u.Id ==
userScore.Key);

        var relevantVacancies = userScore
            .Where(us => us.Score > 60)
            .Select(us => addedVacancies.First(v => v.Id ==
us.VacancyId)).ToArray();

        if (relevantVacancies.Length != 0 && user != null)
        {
            await emailService.SendAsync(
                user.Email,
                user.Name,
                "New relevant vacancies",
                string.Join(Environment.NewLine, relevantVacancies.Select(v =>
$"{v.Title}: {v.Url}")));
        }
    }
}

```

Файл ScrapperJob.cs

Реалізація функціональної задачі збору посилань на компанії при першому запуску серверу

```

using KolybaResume.BLL.Services.Abstract;
using KolybaResume.BLL.Services.Scrappers;

namespace KolybaResume.Jobs;

public class ScrapperJob(IServiceProvider services) : IHostedService
{
    public async Task StartAsync(Cancellation_token cancellation_token)
    {
        try
        {
            using var scope = services.CreateScope();
            var companyService =
scope.ServiceProvider.GetRequiredService<ICompanyService>();

            if (await companyService.HasCompanies())
            {
                return;
            }

            var companyLinks = DouCompanyScrapper.Scrape();
            await companyService.Create(companyLinks);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex);
        }
    }

    public Task StopAsync(Cancellation_token cancellation_token) =>
Task.CompletedTask;
}

```