ДОДАТОК А

Лістинг коду

Файл DataService.cs

```
using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Text;
using System.Threading.Tasks;
using MySql.Data.MySqlClient;
using Diploma.Models;
using Amazon;
using Amazon.S3;
using System.Security.Cryptography;
using Amazon.S3.Transfer;
using Amazon.S3.Model;
using Avalonia.Controls;
using DocumentFormat.OpenXml.Packaging;
using DocumentFormat.OpenXml.Wordprocessing;
using iText.Kernel.Pdf.Canvas.Parser;
using iText.Kernel.Pdf;
using System.Diagnostics;
using System.IO.Compression;
using System.IO;
using System.Linq;

namespace Diploma.Services
{
    public class DataService
    {
        public string ConnectionString { get; } =
"server=candidates.cl6ekk8qmumc.eu-north-
1.rds.amazonaws.com;port=3306;user=admin;database=candidates_db;password=sS23#44
56.^&;";

        public async Task<List<Candidate>> LoadCandidatesFromDatabase()
        {
            List<Candidate> candidates = new List<Candidate>();

            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                await connection.OpenAsync();
```

```
                string query = @"
                SELECT cv.file_name, cv.cv_path
                FROM candidate_cv cv";

                using (MySqlCommand command = new MySqlCommand(query,
connection))
                {
                    using (DbDataReader reader = await
command.ExecuteReaderAsync())
                    {
                        while (await reader.ReadAsync())
                        {
                            candidates.Add(new Candidate
                            {
                                Name =
reader.GetString(reader.GetOrdinal("file_name")),
                                CvPath =
reader.GetString(reader.GetOrdinal("cv_path")),
                            });
                        }
                    }
                }
            }

            return candidates;
        }

        public async Task<List<Candidate>> FilterCandidates(List<string>
selectedFilters)
        {
            List<Candidate> candidates = new List<Candidate>();

            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                await connection.OpenAsync();

                StringBuilder queryBuilder = new StringBuilder();
                queryBuilder.Append("SELECT cv.file_name, cv.cv_path ");
                queryBuilder.Append("FROM candidate_cv cv ");

                if (selectedFilters.Count > 0)
                {
```

```
                    queryBuilder.Append("WHERE ");
                    for (int i = 0; i < selectedFilters.Count; i++)
                    {
                        queryBuilder.Append($"cv.cv_data LIKE @filterData{i} ");
                        if (i < selectedFilters.Count - 1)
                        {
                            queryBuilder.Append("AND ");
                        }
                    }
                }

                using (MySqlCommand command = new
MySqlCommand(queryBuilder.ToString(), connection))
                {
                    for (int i = 0; i < selectedFilters.Count; i++)
                    {
                        command.Parameters.AddWithValue($"@filterData{i}",
$"%{selectedFilters[i]}%");
                    }

                    using (DbDataReader reader = await
command.ExecuteReaderAsync())
                    {
                        while (await reader.ReadAsync())
                        {
                            candidates.Add(new Candidate
                            {
                                Name =
reader.GetString(reader.GetOrdinal("file_name")),
                                CvPath =
reader.GetString(reader.GetOrdinal("cv_path")),
                            });
                        }
                    }
                }
            }

            return candidates;
        }

        public async Task<List<Diploma.Models.Filter>> LoadFiltersFromDatabase()
        {
```

```
            List<Diploma.Models.Filter> filters = new
List<Diploma.Models.Filter>();

            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                await connection.OpenAsync();

                string query = "SELECT filter_data FROM filters";

                using (MySqlCommand command = new MySqlCommand(query,
connection))
                {
                    using (DbDataReader reader = await
command.ExecuteReaderAsync())
                    {
                        while (await reader.ReadAsync())
                        {
                            filters.Add(new Diploma.Models.Filter
                            {
                                FilterData =
reader.GetString(reader.GetOrdinal("filter_data"))
                            });
                        }
                    }
                }
            }

            return filters;
        }

        public async Task<bool> IsFilterUnique(string filterData)
        {
            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                await connection.OpenAsync();

                string query = "SELECT COUNT(*) FROM filters WHERE filter_data =
@filterData";

                using (MySqlCommand command = new MySqlCommand(query,
connection))
```

```
                {
                    command.Parameters.AddWithValue("@filterData", filterData);
                    int count = Convert.ToInt32(await
command.ExecuteScalarAsync());
                    return count == 0;
                }
            }
        }

        public async Task SaveFilter(string filterData)
        {
            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                string query = "INSERT INTO filters (filter_data, created_at)
VALUES (@filter_data, @created_at)";
                using (MySqlCommand command = new MySqlCommand(query,
connection))
                {
                    command.Parameters.AddWithValue("@filter_data", filterData);
                    command.Parameters.AddWithValue("@created_at",
DateTime.Now);
                    await connection.OpenAsync();
                    await command.ExecuteNonQueryAsync();
                }
            }
        }

        public async Task<string> DownloadFileFromS3(string s3FilePath, string
bucketName, RegionEndpoint region)
        {
            string localFilePath = Path.Combine(Path.GetTempPath(),
Path.GetFileName(s3FilePath));

            using (var s3Client = new AmazonS3Client(region))
            {
                GetObjectRequest request = new GetObjectRequest
                {
                    BucketName = bucketName,
                    Key = s3FilePath
                };
```

```
                using      (GetObjectResponse      response      =      await
s3Client.GetObjectAsync(request))
                {
                    await response.WriteResponseStreamToFileAsync(localFilePath,
false, default);
                }
            }

            return localFilePath;
        }

        public async Task ExtractFilesFromZip(Window parentWindow)
        {

            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.AllowMultiple = false;
            openFileDialog.Filters.Add(new  FileDialogFilter  {  Name  =  "Zip
files", Extensions = { "zip" } });

            string[] result = await openFileDialog.ShowAsync(parentWindow);

            if (result == null || result.Length == 0)
            {
                return;
            }

            string zipFilePath = result[0];

            var region = RegionEndpoint.EUNorth1;
            using (var s3Client = new AmazonS3Client(region))
            {
                string s3BucketName = "ccvs";

                string    tempFolderPath    =    Path.Combine(Path.GetTempPath(),
Guid.NewGuid().ToString());
                Directory.CreateDirectory(tempFolderPath);
                ZipFile.ExtractToDirectory(zipFilePath, tempFolderPath);

                string[]   files   =   Directory.GetFiles(tempFolderPath,   "*",
SearchOption.AllDirectories);

                string[] validExtensions = { ".pdf", ".docx" };
```

```
                foreach (var file in files)
                {
                    var extension = Path.GetExtension(file).ToLowerInvariant();

                    if (validExtensions.Contains(extension))
                    {
                        string                  uniqueFileName                    =
$"{Guid.NewGuid()}_{Path.GetFileName(file)}";

                        await      UploadFileToS3(s3Client,      s3BucketName,
uniqueFileName, file);

                        string cvData = ReadFileContent(file);

                        SaveCandidateCv(Path.GetFileName(file),          cvData,
uniqueFileName);
                    }
                    else
                    {
                        try
                        {
                            File.Delete(file);
                        }
                        catch (Exception ex)
                        {

                        }
                    }
                }

                Directory.Delete(tempFolderPath, true);
            }
        }

        private   async   Task   UploadFileToS3(AmazonS3Client   s3Client,   string
bucketName, string s3FileName, string filePath)
        {
            var fileTransferUtility = new TransferUtility(s3Client);
            await     fileTransferUtility.UploadAsync(filePath,     bucketName,
s3FileName);
        }
```

```
        private void SaveCandidateCv(string fileName, string cvData, string cvPath)
        {
            using (MySqlConnection connection = new MySqlConnection(ConnectionString))
            {
                string query = "INSERT INTO candidate_cv (file_name, cv_data, cv_path, created_at) VALUES (@file_name, @cv_data, @cv_path, @created_at)";
                using (MySqlCommand command = new MySqlCommand(query, connection))
                {
                    command.Parameters.AddWithValue("@file_name", fileName);
                    command.Parameters.AddWithValue("@cv_data", cvData);
                    command.Parameters.AddWithValue("@cv_path", cvPath);
                    command.Parameters.AddWithValue("@created_at", DateTime.Now);
                    connection.Open();
                    command.ExecuteNonQuery();
                }
            }
        }
        public async Task<bool> AuthenticateUserAsync(string username, string password)
        {
            return await Task.Run(() => AuthenticateUser(username, password));
        }

        private bool AuthenticateUser(string username, string password)
        {
            string query = "SELECT COUNT(*) FROM users WHERE name = @username AND password = @password";

            try
            {
                using (MySqlConnection con = new MySqlConnection(ConnectionString))
                {
                    con.Open();
                    using (MySqlCommand cmd = new MySqlCommand(query, con))
                    {
                        cmd.Parameters.AddWithValue("@username", username);
                        cmd.Parameters.AddWithValue("@password", HashPassword(password));
```

```
                    int userCount = Convert.ToInt32(cmd.ExecuteScalar());
                    return userCount > 0;
                }
            }
        }
        catch (Exception)
        {
            return false;
        }
    }

    private string HashPassword(string password)
    {
        using (SHA256 sha256 = SHA256.Create())
        {
            byte[]                              bytes                              =
sha256.ComputeHash(Encoding.UTF8.GetBytes(password));
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < bytes.Length; i++)
            {
                builder.Append(bytes[i].ToString("x2"));
            }
            return builder.ToString();
        }
    }


private string ReadFileContent(string filePath)
    {
        string fileContent = "";
        string extension = Path.GetExtension(filePath).ToLowerInvariant();

        if (extension == ".pdf")
        {
            fileContent = ReadPdfContent(filePath);
        }
        else if (extension == ".docx")
        {
            fileContent = ReadDocContent(filePath);
        }

        return fileContent;
```

```
        }

        private string ReadPdfContent(string filePath)
        {
            using (PdfReader reader = new PdfReader(filePath))
            {
                using (PdfDocument pdfDoc = new PdfDocument(reader))
                {
                    StringBuilder text = new StringBuilder();
                    for (int page = 1; page <= pdfDoc.GetNumberOfPages();
page++)
                    {

text.Append(PdfTextExtractor.GetTextFromPage(pdfDoc.GetPage(page)));
                    }
                    return text.ToString();
                }
            }
        }

        private string ReadDocContent(string filePath)
        {
            using (WordprocessingDocument wordDoc =
WordprocessingDocument.Open(filePath, false))
            {
                Body body = wordDoc.MainDocumentPart.Document.Body;
                return body.InnerText;
            }
        }
    }
}
```

Файл CandidatesViewModel.cs

```
using System;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Input;
using Amazon;
using Diploma.Models;
using Diploma.Services;
using ReactiveUI;
using Avalonia.Controls;
```

```
namespace Diploma.ViewModels
{
    public class FilterItem : ReactiveObject
    {
        private bool _isChecked;
        private bool _isVisible;

        public string Name { get; set; }
        public bool IsChecked
        {
            get => _isChecked;
            set => this.RaiseAndSetIfChanged(ref _isChecked, value);
        }

        public bool IsVisible
        {
            get => _isVisible;
            set => this.RaiseAndSetIfChanged(ref _isVisible, value);
        }
    }

    public class CandidatesViewModel : ReactiveObject
    {
        private readonly DataService _dataService;
        private string _searchTerm;
        private string _newFilter;
        private bool _isResetFiltersButtonEnabled;
        private bool _isCreateFilterButtonEnabled;

        public ObservableCollection<Candidate> Candidates { get; set; } = new
ObservableCollection<Candidate>();
        public ObservableCollection<FilterItem> Filters { get; set; } = new
ObservableCollection<FilterItem>();
        public ICommand CreateFilterCommand { get; }
        public ICommand ResetFiltersCommand { get; }
        public ICommand ImportCommand { get; }
        public ICommand DownloadCommand { get; }
        public ICommand FilterCommand { get; }

        public event EventHandler<string> ShowNotificationRequested;

        public CandidatesViewModel()
        {
            _dataService = new DataService();
            CreateFilterCommand = ReactiveCommand.Create(OnCreateFilter);
            ResetFiltersCommand = ReactiveCommand.Create(OnResetFilters);
            ImportCommand = ReactiveCommand.Create<Window>(async (parentWindow)
=> await OnImport(parentWindow));
            DownloadCommand = ReactiveCommand.Create<string>(OnDownload);
            FilterCommand = ReactiveCommand.Create<FilterItem>(OnFilterChanged);
            LoadData();
        }

        public string NewFilter
        {
            get => _newFilter;
            set
            {
                this.RaiseAndSetIfChanged(ref _newFilter, value);
                IsCreateFilterButtonEnabled = !string.IsNullOrWhiteSpace(value);
            }
        }
```

```
        public bool IsResetFiltersButtonEnabled
        {
            get => _isResetFiltersButtonEnabled;
            set => this.RaiseAndSetIfChanged(ref _isResetFiltersButtonEnabled,
value);
        }

        public bool IsCreateFilterButtonEnabled
        {
            get => _isCreateFilterButtonEnabled;
            set => this.RaiseAndSetIfChanged(ref _isCreateFilterButtonEnabled,
value);
        }

        private async Task LoadData()
        {
            Candidates.Clear();
            var candidates = await _dataService.LoadCandidatesFromDatabase();
            foreach (var candidate in candidates)
            {
                Candidates.Add(candidate);
            }

            var filters = await _dataService.LoadFiltersFromDatabase();
            Filters.Clear();
            foreach (var filter in filters.Select(f => new FilterItem { Name =
f.FilterData, IsVisible = true }))
            {
                Filters.Add(filter);
            }
        }

        public string SearchTerm
        {
            get => _searchTerm;
            set
            {
                this.RaiseAndSetIfChanged(ref _searchTerm, value);
                FilterListBoxItems();
            }
        }

        private void FilterListBoxItems()
        {
            if (string.IsNullOrEmpty(SearchTerm))
            {
                foreach (var filter in Filters)
                {
                    filter.IsVisible = true;
                }
            }
            else
            {
                foreach (var filter in Filters)
                {
                    filter.IsVisible = filter.Name.Contains(SearchTerm,
StringComparison.OrdinalIgnoreCase);
                }
            }
        }

        private async void OnCreateFilter()
        {
            Debug.WriteLine("OnCreateFilter called");
```

```
            if (string.IsNullOrWhiteSpace(NewFilter)) return;

            bool isUnique = await _dataService.IsFilterUnique(NewFilter);

            if (!isUnique)
            {
                NewFilter = string.Empty;
                ShowNotificationRequested?.Invoke(this, "Даний фільтр вже 
існує");
                return;
            }

            await _dataService.SaveFilter(NewFilter);
            NewFilter = string.Empty;
            await LoadData();
        }


        private async void OnResetFilters()
        {
            foreach (var filter in Filters)
            {
                filter.IsChecked = false;
                filter.IsVisible = true;
            }
            IsResetFiltersButtonEnabled = false;
            await LoadData();
        }

        private async Task OnImport(Window parentWindow)
        {
            await _dataService.ExtractFilesFromZip(parentWindow);
            var candidates = await _dataService.LoadCandidatesFromDatabase();
            Candidates.Clear();
            foreach (var candidate in candidates)
            {
                Candidates.Add(candidate);
            }
            ShowNotificationRequested?.Invoke(this, "Імпорт документів успішно 
завершено");
        }

        private async void OnDownload(string cvPath)
        {
            if (string.IsNullOrWhiteSpace(cvPath)) return;

            try
            {
                var region = RegionEndpoint.EUNorth1;
                string bucketName = "ccvs";

                string localFilePath = await 
_dataService.DownloadFileFromS3(cvPath, bucketName, region);

                if (File.Exists(localFilePath))
                {
                    using Process fileOpener = new Process();
                    fileOpener.StartInfo.FileName = "explorer";
                    fileOpener.StartInfo.Arguments = "\"" + localFilePath + 
"\"";
                    fileOpener.Start();
                }
                else
```

```
                {
                    ShowNotificationRequested?.Invoke(this, "Файл не
знайдено.");
                }
            }
            catch (Exception ex)
            {
                ShowNotificationRequested?.Invoke(this, $"Помилка при імпорті
файлу: {ex.Message}");
            }
        }

        private void OnFilterChanged(FilterItem filterItem)
        {
            ApplyFilters();
        }

        private async void ApplyFilters()
        {
            var selectedFilters = Filters.Where(f => f.IsChecked).Select(f =>
f.Name).ToList();
            IsResetFiltersButtonEnabled = selectedFilters.Any();
            if (selectedFilters.Any())
            {
                var filteredCandidates = await
_dataService.FilterCandidates(selectedFilters);
                Candidates.Clear();
                foreach (var candidate in filteredCandidates)
                {
                    Candidates.Add(candidate);
                }
            }
            else
            {
                await LoadData();
            }
        }
    }
}
```

ДОДАТОК Б

Демонстраційний матеріал

№ доку-мента	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК. 758710.006 ПЗ	Пояснювальна записка	А4, 53 с.
		Додаткові матеріали	
2		Лістинг коду	А4, 14 с.
3		Демонстраційний матеріал	А4, 14 с.

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮИК. 758710.006 ПЗ		
Розроб.		Маковєєва В.К.			Розробка програмного застосунку для автоматизації роботи HR – відділів	Літера	Аркуш	Аркушів
Перевір.		Чала О.О.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Демська Н.П.						
Затв.		Невлюдов І.Ш.						