# ДОДАТОК А

Графічний матеріал кваліфікаційної роботи


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ
Кваліфікаційна робота
на тему: «Програмний застосунок для автоматизації заповнення медичних документів»
Здобувач:
Анастасія СМОТРОВА
КІУКІу-22-1
Керівник:
Георгій ІВАЩЕНКО
доц. каф. ЕОМ
Харків - 2025


Актуальність проблеми
• Відсутність єдиної системи керування інформаційними ресурсами медичних закладів.
• Складність підготовки медичних документів вручну.
• Велика кількість часу на надання послуг при використанні традиційних картотек.
2

## Аналіз існуючих рішень

| Критерії | Helsi | Megapolis | DocuWare | OpenText |
|---|---|---|---|---|
| Збереження документів | + | + | + | + |
| Створення та редагування документів | + | - | + | + |
| Безкоштовність використання | + | - | - | - |
| Швидке навчання персоналу | + | - | - | - |
| Інтуїтивність застосунку | + | + | - | - |
| Можливість додавати внутрішні форми специфічної структури | - | - | + | + |
| Сумісність із різними типами документів | - | + | + | + |
| Підтримка створення резервних копій | - | + | + | + |

3

## Постановка задачі

Створення програмного продукту з простим та зрозумілим інтерфейсом для автоматизації заповнення медичних документів.
Основний функціонал продукту:

- Збір та зберігання інформації про пацієнтів
- Генерування необхідних документів (Виписки ОГП ОЦО, №066)
- Експорт та імпорт інформації про пацієнтів

4

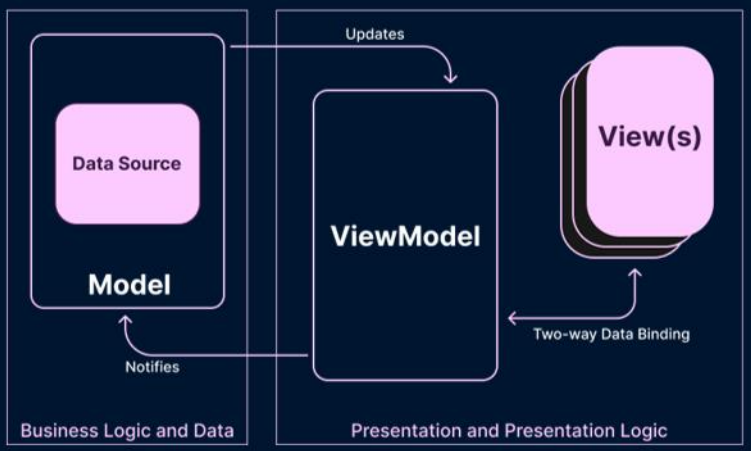
## Засоби розробки

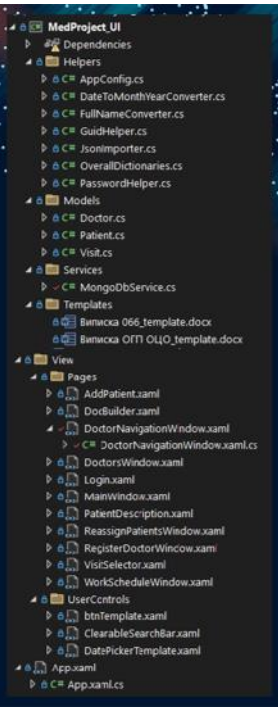
Згідно із поставленими завданнями були використані такі інструменти:

- IDE MS Visual Studio 2022;
- платформа .NET версії 8;
- мова програмування C# версії 10;
- Windows Presentation Foundation (WPF);
- бібліотека EasyDox;
- нереляційна СКБД MongoDB.

5

## Архітектура застосунку



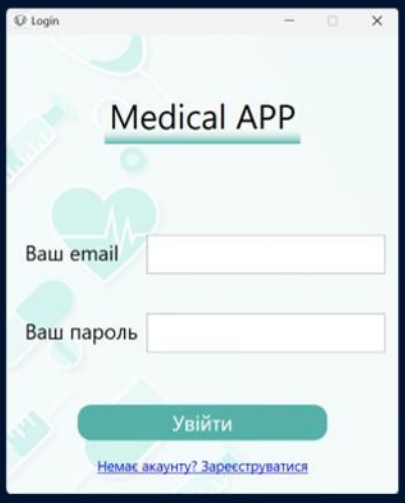

6

## Робота з даними


**Архітектура типу "товстий клієнт із локальним сховищем" (Thick Client with Local DB)** передбачає, що вся прикладна логіка та інтерфейс користувача виконуються на клієнтському пристрої, а доступ до централізованої бази даних відбувається безпосередньо через локальну мережу або на локально розгорнуту СКБД.

7

## Інтерфейс користувача





8

## Рівні доступу

| | Головний лікар | Лікар | Адміністратор |
|---|---|---|---|
| Перегляд списку лікарів | + | + | + |
| Зміна даних лікарів | + | Тільки своїх | + |
| Перегляд графіку роботи | + | + | + |
| Редагування графіку роботи | + | Тільки свій | - |
| Переглядати дані пацієнтів | + | Тільки своїх | + |
| Редагувати дані пацієнтів | + | Тільки своїх | - |
| Генерація документів | + | + | + |
| Експорт даних пацієнта | + | + | - |
| Імпорт даних пацієнтів | + | - | - |

9

## Перегляд даних пацієнта


10

## Процес генерації документу


11

## Висновки

- Проведено детальний аналіз можливостей автоматизації заповнення медичних документів та ведення документообігу.
- Розроблено десктопний застосунок для автоматизації заповнення медичних документів та керування даними пацієнтів.
- Реалізовано зручний та адаптивний інтерфейс.
- Реалізовано надійне зберігання даних користувачів, імпорт та експорт даних.
- Реалізовано можливість ведення графіку змін докторів для подальшого впровадження в інформаційну систему лікарні.
- За результатами роботи були опубліковані тези на п'ятнадцятій міжнародній науково-технічній конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління».

12

# ДОДАТОК Б

## Фрагменти вихідного коду програмного засобу

### Б.1 Простір імен «Helpers»

#### Б.1.1 Клас «AppConfig»

```
public class AppConfig
{
    public string MongoDbConnection { get; set; }
    public string DatabaseName { get; set; }
    public string PatientsCollection { get; set; }
    public string DoctorsCollection { get; set; }
    public static AppConfig Load(string path = "config.json")
    {
        var json = File.ReadAllText(path);
        return JsonConvert.DeserializeObject<AppConfig>(json);
    }
}
```

#### Б.1.2 Клас «OverallDictionaries»

```
internal class OverallDictionaries
{
    public OverallDictionaries()
    {
        Dictionaries = new Dictionary<string, Dictionary<int,
string>>()
        {
            { "dictOverallItem1",  dictOverallItem1 },
            { "dictOverallItem2",  dictOverallItem2 },
            { "dictOverallItem3",  dictOverallItem3 },
            { "dictOverallItem4",  dictOverallItem4 },
            { "dictOverallItem5",  dictOverallItem5 },
            { "dictOverallItem6",  dictOverallItem6 },
            { "dictOverallItem8",  dictOverallItem8 },
            { "dictOverallItem10", dictOverallItem10 },
            { "dictOverallItem13", dictOverallItem13 },
            { "dictOverallItem14", dictOverallItem14 }
        };
    }
    public Dictionary<string, Dictionary<int, string>>
Dictionaries { get; set; }
```

```
public Dictionary<int, string> dictOverallItem1 = new()
{
    { 0, "Задовільний" },
    { 1, "Відносно задовільний" },
    { 2, "Середньої важкості" },
    { 3, "Важкий" }
};

public Dictionary<int, string> dictOverallItem2 = new()
{
    { 0, "Нормостенік" },
    { 1, "Астенік" },
    { 2, "Гиперстенік" }
};

public Dictionary<int, string> dictOverallItem3 = new()
{
    { 0, "Підвищене" },
    { 1, "Помірне" },
    { 2, "Знижене" },
    { 3, "Кахексія" }
};

public Dictionary<int, string> dictOverallItem4 = new()
{
    { 0, "Немає" },
    { 1, "При фізичному навантаженні" },
    { 2, "При розмові" },
    { 3, "В спокої" }
};

public Dictionary<int, string> dictOverallItem5 = new()
{
    { 0, "Звичайного кольору" },
    { 1, "Бліда" },
    { 2, "Акроцианоз" },
    { 3, "Центральний цианоз" }
};

public Dictionary<int, string> dictOverallItem6 = new()
{
    { 0, "Вологий, чистий" },
    { 1, "Обкладений" },
    { 2, "Сухий" }
};

public Dictionary<int, string> dictOverallItem8 = new()
{
    { 0, "В нормі" },
    { 1, "Закрепи" },
    { 2, "Проноси" }
};
```

```
    public Dictionary<int, string> dictOverallItem10 = new()
    {
        { 0, "Звичайної форми" },
        { 1, "Не збільшений у розмірах" },
        { 2, "Не роздутий" },
        { 3, "Бере участь у дихані" },
        { 4, "При пальпації м'який, безболісний" },
        { 5, "Перитонеальні симптоми негативні" }
    };

    public Dictionary<int, string> dictOverallItem13 = new()
    {
        { 0, "В нормі" },
        { 1, "Знижений" },
        { 2, "Дізурія" }
    };

    public Dictionary<int, string> dictOverallItem14 = new()
    {
        { 0, "Чисті" },
        { 1, "Приглушені" },
        { 2, "Ритмічні" },
        { 3, "Екстрасистолія" },
        { 4, "Мерцальна аритмія" }
    };
}
```

### Б.1.3 Клас «PasswordHelper»

```
internal static class PasswordHelper
{
    private const int SaltSize = 16; // 128 bit
    private const int KeySize = 32; // 256 bit
    private const int Iterations = 10000;

    public static string HashPassword(string password)
    {
        using (var rng = RandomNumberGenerator.Create())
        {
            var salt = new byte[SaltSize];
            rng.GetBytes(salt);

            using (var pbkdf2 = new Rfc2898DeriveBytes(password,
salt, Iterations, HashAlgorithmName.SHA256))
            {
                var key = pbkdf2.GetBytes(KeySize);
                var hashParts = new byte[SaltSize + KeySize];
                Buffer.BlockCopy(salt, 0, hashParts, 0,
SaltSize);
                Buffer.BlockCopy(key, 0, hashParts, SaltSize,
KeySize);
```

```
                return Convert.ToBase64String(hashParts);
            }}
     }
    public static bool VerifyPassword(string password, string
hashedPassword)
    {
        var hashBytes =
Convert.FromBase64String(hashedPassword);

        var salt = new byte[SaltSize];
        Buffer.BlockCopy(hashBytes, 0, salt, 0, SaltSize);

        using (var pbkdf2 = new Rfc2898DeriveBytes(password,
salt, Iterations, HashAlgorithmName.SHA256))
        {
            var key = pbkdf2.GetBytes(KeySize);

            for (var i = 0; i < KeySize; i++)
                if (hashBytes[SaltSize + i] != key[i])
                    return false;
            return true;
        }
    }
}
```

## Б.2 Простір імен «Models»

### Б.2.1 Клас «Doctor»

```
public class Doctor
{
    [BsonId]
    [BsonRepresentation(BsonType.ObjectId)]
    public string Id { get; set; }

    public string LastName { get; set; } // Прізвище
    public string FirstName { get; set; } // Ім'я
    public string MiddleName { get; set; } // По батькові

    [BsonDateTimeOptions(Kind = DateTimeKind.Local)]
    public DateTime BirthDate { get; set; } // Дата народження
    public string Phone { get; set; } // Номер телефону
    public string Email { get; set; } // Електронна пошта
    public string Address { get; set; } // Адреса проживання

    public string Position { get; set; } // Посада (e.g. Лікар-
уролог)
    [BsonDateTimeOptions(Kind = DateTimeKind.Local)]
    public DateTime StartDate { get; set; } // Дата початку
```

```
роботи
    public string Username { get; set; } // Логін
    public string PasswordHash { get; set; } // Пароль (у
вигляді хешу)
    public string AccessLevel { get; set; } // Рівень доступу:
"Admin", "Doctor", "Chief_Doctor" тощо

    public List<string> PatientIds { get; set; } = new(); // ID
пацієнтів
    public List<WorkPeriod> WorkSchedule { get; set; } = new();
// Робочий графік

    [BsonIgnore] // Щоб не зберігати в MongoDB
    public string FullName => $"{LastName} {FirstName}
{MiddleName}";

    [BsonIgnore] // Щоб не зберігати в MongoDB
    public string ShortName => $"{FirstName[0]}.{MiddleName[0]}.
{LastName}";
    [BsonIgnore] // Щоб не зберігати в MongoDB
    public string ShortNameWithSpec =>
$"{FirstName[0]}.{MiddleName[0]}. {LastName} | {Position}";
    [BsonIgnore]
    public string OnDutyStatus { get; set; }
}
```

### Б.2.2 Клас «Patient»

```
public class Patient
{
    [BsonId]
    [BsonRepresentation(BsonType.ObjectId)]
    public string Id { get; set; }

    public string CardNumber { get; set; } // _colCardNumber
    public string LastName { get; set; } // _colLastName
    public string FirstName { get; set; } // _colFirstName
    public string MiddleName { get; set; } // _colMiddleName
    [BsonDateTimeOptions(Kind = DateTimeKind.Local)]
    public DateTime BirthDate { get; set; } // _colBirthDay
    public int Age { get; set; } // _colAge
    public string Address { get; set; } // _colAddress
    public string Profession { get; set; } // _colProfession
    public string Phone { get; set; } = "0000000000";
    public string Email { get; set; } = "unknown@example.com";
    public string Gender { get; set; } = "unknown";

    public string Doctor { get; set; } // з _fieldDoctor
    public string DoctorId { get; set; } // зв'язок з Doctor.Id
    public List<Visit> Visits { get; set; } = new();
}
```

## Б.3 Простір імен «Services»

### Б.3.1 Клас «MongoDbService»

```
internal class MongoDbService : IDisposable
{
    private readonly IMongoCollection<Patient> _patients;
    private readonly IMongoCollection<Doctor> _doctors;
    private readonly IMongoDatabase _database;
    private readonly MongoClient _client;
    private bool _disposed = false;

    public MongoDbService(string connectionString, string
dbName)
    {
        var config = AppConfig.Load();

        _client = new MongoClient(connectionString);
        _database = _client.GetDatabase(dbName);

        _patients =
_database.GetCollection<Patient>(config.PatientsCollection);
        _doctors =
_database.GetCollection<Doctor>(config.DoctorsCollection);
    }

    public async Task<List<Patient>> GetAllPatientsAsync()
    {
        return await _patients.Find(_ => true).ToListAsync();
    }

    public async Task<List<Patient>>
GetPatientsByDoctorIdAsync(string doctorId)
    {
        var filter = Builders<Patient>.Filter.Eq(p =>
p.DoctorId, doctorId);
        return await _patients.Find(filter).ToListAsync(); }
    public async Task<Patient?> GetPatientByIdAsync(string id)
    {
        return await _patients.Find(p => p.Id ==
id).FirstOrDefaultAsync();
    }

    public async Task AddPatientAsync(Patient patient)
    {
        await _patients.InsertOneAsync(patient);
    }

    public async Task InsertPatientAsync(Patient patient)
    {
```

```
        await _patients.InsertOneAsync(patient);
    }

    public async Task UpdatePatientAsync(Patient patient)
    {
        await _patients.ReplaceOneAsync(p => p.Id == patient.Id,
patient);
    }

    public async Task InsertOrUpdatePatientAsync(Patient
patient)
    {
        if (string.IsNullOrEmpty(patient.Id))
            await InsertPatientAsync(patient);
        else
            await UpdatePatientAsync(patient);
    }

    public async Task DeletePatientAsync(string id)
    {
        await _patients.DeleteOneAsync(p => p.Id == id);
    }

    public async Task<List<Doctor>> GetAllDoctorsAsync()
    {
        return await _doctors.Find(_ => true).ToListAsync();
    }

    public async Task<Doctor?> GetDoctorByIdAsync(string id)
    {
        return await _doctors.Find(d => d.Id ==
id).FirstOrDefaultAsync();
    }

    public async Task<Doctor?> GetDoctorByUsernameAsync(string
username)
    {
        return await _doctors.Find(d => d.Username ==
username).FirstOrDefaultAsync();
    }
    public async Task<Doctor?> GetDoctorByEmailAsync(string
email)
    {
        return await _doctors.Find(d => d.Email ==
email).FirstOrDefaultAsync();
    }

    public async Task<bool> CheckDoctorExistsByEmailAsync(string
email)
    {
        var filter = Builders<Doctor>.Filter.Eq(d => d.Email,
email);
        var result = await
```

```
_doctors.Find(filter).FirstOrDefaultAsync();
        return result != null;
    }

    public async Task AddDoctorAsync(Doctor doctor)
    {
        await _doctors.InsertOneAsync(doctor);
    }

    public async Task UpdateDoctorAsync(Doctor doctor)
    {
        await _doctors.ReplaceOneAsync(d => d.Id == doctor.Id,
doctor);
    }

    public async Task<bool> UpdateDoctorScheduleAsync(string
doctorId, List<WorkPeriod> schedule)
    {
        var filter = Builders<Doctor>.Filter.Eq(d => d.Id,
doctorId);
        var update = Builders<Doctor>.Update.Set(d =>
d.WorkSchedule, schedule);

        var result = await _doctors.UpdateOneAsync(filter,
update);
        return result.ModifiedCount > 0;
    }

    public async Task DeleteDoctorAsync(string id)
    {
        await _doctors.DeleteOneAsync(d => d.Id == id);
    }

    public async Task EnsureDatabaseSetupAsync()
    {
        try
        {
            var dbList = await
_database.Client.ListDatabaseNamesAsync();
            var dbExists =
dbList.ToList().Contains(_database.DatabaseNamespace.DatabaseNam
e);
            if (!dbExists)
            {
                await
_database.CreateCollectionAsync("Doctors");
                await
_database.CreateCollectionAsync("Patients");
                return;
            }

            var collections = await
_database.ListCollectionNames().ToListAsync();
```

```
            if (!collections.Contains("Doctors"))
                await
_database.CreateCollectionAsync("Doctors");

            if (!collections.Contains("Patients"))
                await
_database.CreateCollectionAsync("Patients");
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Помилка підключення до бази даних
MongoDB:\n{ex.Message}",
                "Помилка ініціалізації", MessageBoxButton.OK,
MessageBoxImage.Error);
            Environment.Exit(1);
        }
    }

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    protected virtual void Dispose(bool disposing)
    {
        if (!_disposed)
        {
            if (disposing)
            {
                _client?.Cluster.Dispose();
            }
            _disposed = true;
        }
    }
}
```

## Б.4 Простір імен «View.Pages»

### Б.4.1 Клас «Login»

```
public partial class Login : Window
{
    public Login()
    {
        InitializeComponent();
    }

    private async void BTN_Login_Click(object sender,
RoutedEventArgs e)
```

```
    {
        var email = tbEmail.Text.Trim();
        var password = tbPass.Password;

        if (string.IsNullOrWhiteSpace(email) ||
string.IsNullOrWhiteSpace(password))
        {
            MessageBox.Show("Будь ласка, заповніть обидва поля
для входу.", "Помилка", MessageBoxButton.OK,
MessageBoxImage.Warning);
            return;
        }

        try
        {
            var config = AppConfig.Load();
            var mongoService = new
MongoDbService(config.MongoDbConnection, config.DatabaseName);

            var doctor = await
mongoService.GetDoctorByEmailAsync(email);
            if (doctor == null)
            {
                MessageBox.Show("Користувача з таким email не
знайдено.", "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
                return;
            }

            if (!PasswordHelper.VerifyPassword(password,
doctor.PasswordHash))
            {
                MessageBox.Show("Неправильний пароль.",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
                return;
            }

            if (doctor.AccessLevel == "doctor" ||
doctor.AccessLevel == "chief_doctor" || doctor.AccessLevel ==
"admin")
            {
                App.CurrentUser = doctor;

                var doctorNav = new
DoctorNavigationWindow(doctor);
                Close();
                doctorNav.ShowDialog();
            }
            else
            {
                var main = new MainWindow();
                Hide();
                main.ShowDialog();
```

```
                tbPass.Password = "";
                Show();
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Сталася помилка при вході: " +
ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        }
    }

    private void RegisterLink_Click(object sender,
MouseButtonEventArgs e)
    {
        var registerWindow = new RegisterDoctorWindow();
        registerWindow.ShowDialog();
    }
}
```

### Б.4.2 Клас «DocBuilder»

```
public partial class DocBuilder : Window
{
    private readonly Patient _patient;
    private readonly Visit _visit;

    public DocBuilder()
    {
        InitializeComponent();
    }

    public DocBuilder(Patient patient, Visit visit = null)
    {
        InitializeComponent();
        _patient = patient;
        _visit = visit ?? new Visit();

        btnCreateF1.btnClick += BtnCreateF1_Click;
        btnCreateF2.btnClick += BtnCreateF2_Click;
    }

    private string GetSymptom(string key, string fallback = "---
-----")
    {
        var visit = _visit;
        if (visit.Symptoms == null ||
!visit.Symptoms.TryGetValue(key, out var value))
            return fallback;

        try
```

```
        {
            var list =
JsonConvert.DeserializeObject<List<string>>(value);
            if (list != null)
                return string.Join("\n ", list.Prepend(""));
        }
        catch
        {
            if (key == "_fieldClaims")
                MessageBox.Show(
                    "Скарги не будуть виведені коректно,
оскільки збережені в неправильному форматі.",
                    "Попередження",
                    MessageBoxButton.OK,
                    MessageBoxImage.Warning
                );
        }

        return value;
    }

    private void BtnCreateF1_Click(object sender,
RoutedEventArgs e)
    {
        var claims = FormatList(GetSymptom("_fieldClaims"));
        var anamnesisItems = new[]
        {
            "_fieldAnamnesisItem1", "_fieldAnamnesisItem2",
"_fieldAnamnesisItem3",
            "_fieldAnamnesisItem4", "_fieldAnamnesisItem5",
"_fieldAnamnesisItem6",
            "_fieldAnamnesisItem7"
        };
        var anamnesisBuilder = string.Join(", ", anamnesisItems
            .Select(key => GetSymptom(key))
            .Where(val => !string.IsNullOrWhiteSpace(val)));

        var fieldValues = new Dictionary<string, string>
        {
            { "_docNumber", "456" },
            { "_docNumberAdditional", "123" },
            { "_docLastFirstMiddleName", $"{_patient.LastName}
{_patient.FirstName} {_patient.MiddleName}" },
            { "_docBirthDay",
_patient.BirthDate.ToString("dd.MM.yyyy") },
            { "_docLivingAddress", _patient.Address },
            { "_docProfession", _patient.Profession },
            { "_docHospitalStartDate",
_visit.StartDate.ToString("dd.MM.yyyy") ?? "" },
            { "_docHospitalEndDate",
_visit.EndDate.ToString("dd.MM.yyyy") ?? "" },
            { "_docDiagnosisMain",
GetSymptom("_fieldFinalDiagnosis") },
```

```
            { "_docComplication",
GetSymptom("_fieldComplication") },
            { "_docAdditionalDiagnosis",
GetSymptom("_fieldAdditionalDiagnosis") },
            { "_docClaims", claims },
            { "_docAnamnesis", anamnesisBuilder },
            { "_docAnamnesisItem8",
GetSymptom("_fieldAnamnesisItem8") },
            { "_docAnamnesisItem9",
GetSymptom("_fieldAnamnesisItem9") },
            { "_docAnamnesisItem10",
GetSymptom("_fieldAnamnesisItem10") },
            { "_docAnamnesisItem11",
GetSymptom("_fieldAnamnesisItem11") },
            { "_docHistology", GetSymptom("_fieldHistology") },
            { "_docChemotherapyDate",
GetSymptom("_fieldChemotherapyDate") ?? "" },
            { "_docChemotherapy",
GetSymptom("_fieldChemotherapy") },
            { "_docCreationDate",
DateTime.Now.ToString("dd.MM.yyyy") },
            { "_docDoctor", _patient.Doctor }
        };
        GenerateDocument("Выписка ОГП ОЦО_template.docx",
fieldValues,

$"Выписка_ОГП_ОЦО_{_patient.LastName}_{_patient.FirstName}_{_pat
ient.MiddleName}");
    }

    private void BtnCreateF2_Click(object sender,
RoutedEventArgs e)
    {
        var fieldValues = new Dictionary<string, string>
        {
            { "_docCardNumber", _patient.CardNumber },
            { "_docHospitalStartDate",
_visit.StartDate.ToString("dd.MM.yyyy") ?? "" },
            { "_docLastFirstMiddleName", $"{_patient.LastName}
{_patient.FirstName} {_patient.MiddleName}" },
            { "_docBirthDay",
_patient.BirthDate.ToString("dd.MM.yyyy") },
            { "_docAge", _patient.Age.ToString() },
            { "_docLivingAddress", _patient.Address },
            { "_docProfession", _patient.Profession },
            { "_docHospitalEndDate",
_visit.EndDate.ToString("dd.MM.yyyy") ?? "" },
            { "_docDiagnosisMain",
GetSymptom("_fieldFinalDiagnosis") },
            { "_docMKX", GetSymptom("_fieldMKX") },
            { "_docFirstOpertaionDate",
GetSymptom("_fieldOperationDate") ?? "" },
            { "_docFirstOpertaionName",
```

```
GetSymptom("_fieldOperationName") },
            { "_docSecOpertaionDate", "" },
            { "_docSecOpertaionName", "" },
            { "_docDoctor", _patient.Doctor },
            { "_docCreationDate",
DateTime.Now.ToString("dd.MM.yyyy") }
        };
        GenerateDocument("Виписка 066_template.docx",
fieldValues,$"Виписка_066_{_patient.LastName}_{_patient.FirstNam
e}_{_patient.MiddleName}");
    }
    private void GenerateDocument(string templateFileName,
Dictionary<string, string> values, string outputName)
    {
        var engine = new Engine();
        try
        {
            var dialog = new SaveFileDialog
            {
                FileName = outputName,
                DefaultExt = ".docx",
                Filter = "Word Documents (.docx)|*.docx"
            };
            if (dialog.ShowDialog() == true)
            {
                var templatePath =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
"Templates", templateFileName);
                engine.Merge(templatePath, values,
dialog.FileName);
                MessageBox.Show("Документ було створено!",
"Документ", MessageBoxButton.OK,
                    MessageBoxImage.Information);
                new Process
                {
                    StartInfo = new
ProcessStartInfo(dialog.FileName) { UseShellExecute = true }
                }.Start();
            }
        }
        catch (Exception ex)
        {
            var msg = ex.Message.Contains("being used")
                ?
                "Документ вже створений та відкритий! Перевірте
будь-ласка запущені процеси."
                : ex.Message.Contains("Could not find file")
                    ? "Шаблон документу не знайдено. Будь-ласка
зверніться до підтримки!"
                    :
                    "Неочікувана помилка. Будь-ласка зверніться
до підтримки!";
            MessageBox.Show(msg, "Помилка", MessageBoxButton.OK,
```

```
MessageBoxImage.Error);
        }
        Close();
    }

    private string FormatList(string raw)
    {
        try
        {
            var list =
JsonConvert.DeserializeObject<List<string>>(raw);
            if (list != null)
                return string.Join(", ", list);
        }
        catch
        {
            // do nothing
        }
        return raw ?? "";
    }
}
```

### Б.4.3 Клас «DoctorNavigationWindow»

```
public partial class DoctorNavigationWindow : Window
{
    private readonly Doctor _doctor;

    public DoctorNavigationWindow(Doctor doctor)
    {
        InitializeComponent();
        _doctor = doctor;
        txtGreeting.Text = $"Вітаю, {_doctor.FirstName}
{_doctor.LastName}!";

        RenderButtonsBasedOnRole();
    }

    private void RenderButtonsBasedOnRole()
    {
        // Створюємо кнопки
        var btnDoctors = CreateButton("Переглянути список
докторів", BtnViewDoctors_Click);
        var btnPatients = CreateButton("Переглянути список
пацієнтів", BtnViewPatients_Click);
        var btnSelfEdit = CreateButton("Редагувати власні дані",
BtnEditSelf_Click);
        var btnExit = CreateButton("Вихід", BtnExit_Click);

        mainPanel.Children.Add(btnDoctors);
        mainPanel.Children.Add(btnPatients);
```

```
        mainPanel.Children.Add(btnSelfEdit);

        // Графік: назва змінюється в залежності від ролі
        if (_doctor.AccessLevel == "doctor" ||
_doctor.AccessLevel == "chief_doctor")
        {
            var btnExport = CreateButton("Експортувати
пацієнтів", BtnExportPatients_Click);
            mainPanel.Children.Add(btnExport);
        }

        if (_doctor.AccessLevel == "chief_doctor")
        {
            var btnImport = CreateButton("Імпортувати
пацієнтів", BtnImportPatients_Click);
            mainPanel.Children.Add(btnImport);
        }

        // Додаємо кнопки до панелі
        if (_doctor.AccessLevel.Contains("doctor"))
        {
            Button btnSchedule;
            if (_doctor.AccessLevel == "chief_doctor")
                btnSchedule = CreateButton("Редагувати графік",
BtnEditSchedule_Click);
            else
                btnSchedule = CreateButton("Переглянути графік",
BtnViewSchedule_Click);
            mainPanel.Children.Add(btnSchedule);
        }
        mainPanel.Children.Add(btnExit);
    }
    private Button CreateButton(string content,
RoutedEventHandler clickHandler)
    {
        var button = new Button
        {
            Content = content,
            Height = 45,
            Style = FindResource("BootstrapButtonPrimary") as
Style
        };
        button.Click += clickHandler;
        return button;
    }

    // Обробники
    private void BtnViewDoctors_Click(object sender,
RoutedEventArgs e)
    {
        var doctorListWindow = new DoctorsWindow(); // назва
вікна перегляду докторів
        Hide();
```

```
        doctorListWindow.ShowDialog();
        Show();
    }

    private void BtnViewPatients_Click(object sender,
RoutedEventArgs e)
    {
        var patientsWindow = new MainWindow();
        Hide();
        patientsWindow.ShowDialog();
        Show();
    }

    private void BtnEditSelf_Click(object sender,
RoutedEventArgs e)
    {
        var editWindow = new RegisterDoctorWindow(_doctor,
isSelfEdit: true);
        Hide();
        if (editWindow.ShowDialog() == true)
        {
            txtGreeting.Text = $"Вітаю, {_doctor.FirstName}
{_doctor.LastName}!";
        }
        Show();
    }

    private void BtnViewSchedule_Click(object sender,
RoutedEventArgs e)
    {
        var doctorOnlyList = new List<Doctor> { _doctor };
        var scheduleWindow = new
WorkScheduleWindow(doctorOnlyList, isReadOnly: true);
        Hide();
        scheduleWindow.ShowDialog();
        Show();
    }

    private async void BtnEditSchedule_Click(object sender,
RoutedEventArgs e)
    {
        try
        {
            var editableList = await GetEditableDoctors() ?? new
List<Doctor>();
            if (editableList.Count > 0)
            {
                var scheduleWindow = new
WorkScheduleWindow(editableList);
                Hide();
                scheduleWindow.ShowDialog();
                Show();
            }
```

```
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Помилка при завантаженні графіка:
{ex.Message}", "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        }
    }

    private async void BtnExportPatients_Click(object sender,
RoutedEventArgs e)
    {
        try
        {
            ExportData();
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Помилка експорту: {ex.Message}",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }

    private async void BtnImportPatients_Click(object sender,
RoutedEventArgs e)
    {
        var openFileDialog = new OpenFileDialog
        {
            Title = "Оберіть ZIP-файл для імпорту",
            Filter = "ZIP files (*.zip)|*.zip"
        };
        if (openFileDialog.ShowDialog() != true)
            return;

        try
        {
            ImportData(openFileDialog.FileName);
        }
        catch(Exception ex)
        {
            MessageBox.Show($"Помилка при імпорті:
{ex.Message}", "Помилка", MessageBoxButton.OK,
                MessageBoxImage.Error);
        }

    }

    private void BtnExit_Click(object sender, RoutedEventArgs e)
    {
        var loginWindowBack = new Login();
        Close();
        loginWindowBack.ShowDialog();
    }
```

```
    private async void ExportData()
    {
        try
        {
            using (var _mongoService = new
MongoDbService(AppConfig.Load().MongoDbConnection,
AppConfig.Load().DatabaseName)) {
                // Отримання пацієнтів згідно з роллю
                var patients = _doctor.AccessLevel ==
"chief_doctor"
                    ? await _mongoService.GetAllPatientsAsync()
                    : await
_mongoService.GetPatientsByDoctorIdAsync(_doctor.Id);

                if (patients.Count == 0)
                {
                    MessageBox.Show("Немає пацієнтів для
експорту.", "Інформація", MessageBoxButton.OK,
                        MessageBoxImage.Information);
                    return;
                }

                // Мінімізований формат: зберігаємо лише
потрібні поля
                var minimalPatients = patients.Select(p => new
                {
                    p.Id,
                    p.CardNumber,
                    p.FirstName,
                    p.LastName,
                    p.MiddleName,
                    p.BirthDate,
                    p.Age,
                    p.Address,
                    p.Profession,
                    p.Phone,
                    p.Email,
                    p.Gender,
                    p.Doctor,
                    p.DoctorId,
                    p.Visits
                });

                var json =
JsonConvert.SerializeObject(minimalPatients, Formatting.None);

                // Діалог збереження
                var saveDialog = new SaveFileDialog
                {
                    Title = "Зберегти архів пацієнтів",
                    Filter = "ZIP files (*.zip)|*.zip",
                    FileName = "exported_patients.zip"
```

```
                };

                if (saveDialog.ShowDialog() == true)
                {
                    using (var zipStream = new
FileStream(saveDialog.FileName, FileMode.Create))
                    using (var archive = new
ZipArchive(zipStream, ZipArchiveMode.Create))
                    {
                        var entry =
archive.CreateEntry("patients.json", CompressionLevel.Optimal);
                        using var entryStream = entry.Open();
                        using var writer = new
StreamWriter(entryStream);
                        writer.Write(json);
                    }

                    MessageBox.Show("Пацієнтів експортовано у
архів успішно.", "Успіх", MessageBoxButton.OK,
                        MessageBoxImage.Information);
                }
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Помилка експорту: {ex.Message}",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }

    private async void ImportData(string zipFileName)
    {
        try
        {
            using (var _mongoService = new
MongoDbService(AppConfig.Load().MongoDbConnection,
AppConfig.Load().DatabaseName))
            {
                using var zip = ZipFile.OpenRead(zipFileName);
                // Перевірка, що всередині лише один файл і це
.json
                if (zip.Entries.Count != 1 ||
!zip.Entries[0].Name.EndsWith(".json"))
                {
                    MessageBox.Show("Архів має містити один
JSON-файл.", "Помилка структури", MessageBoxButton.OK,
                        MessageBoxImage.Error);
                    return;
                }

                // Зчитування JSON
                string jsonContent;
                using (var stream = zip.Entries[0].Open())
```

```
                using (var reader = new StreamReader(stream))
                {
                    jsonContent = reader.ReadToEnd();
                }

                // Десеріалізація
                var importedPatients =
JsonConvert.DeserializeObject<List<Patient>>(jsonContent);

                if (importedPatients == null ||
importedPatients.Count == 0)
                {
                    MessageBox.Show("Файл не містить даних
пацієнтів.", "Помилка", MessageBoxButton.OK,
                        MessageBoxImage.Warning);
                    return;
                }

                // Перевірити скільки нових/оновлюваних
                var existingPatients = await
_mongoService.GetAllPatientsAsync();
                var existingIds = new
HashSet<string>(existingPatients.Select(p => p.Id));

                var toUpdate = importedPatients.Where(p =>
existingIds.Contains(p.Id)).ToList();
                var toInsert = importedPatients.Where(p =>
!existingIds.Contains(p.Id)).ToList();

                // Запит підтвердження
                var msg = $"Буде оновлено {toUpdate.Count}
пацієнтів та додано {toInsert.Count}. Продовжити?";
                var result = MessageBox.Show(msg, "Підтвердження
імпорту", MessageBoxButton.YesNo,
                    MessageBoxImage.Question);

                if (result != MessageBoxResult.Yes)
                    return;

                // Вставлення та оновлення
                foreach (var patient in toInsert)
                    await
_mongoService.AddPatientAsync(patient);

                foreach (var patient in toUpdate)
                    await
_mongoService.UpdatePatientAsync(patient);

                MessageBox.Show("Імпорт завершено успішно.",
"Успіх", MessageBoxButton.OK, MessageBoxImage.Information);
            }
        }
        catch (Exception ex)
```

```
        {
            MessageBox.Show($"Помилка при імпорті:
{ex.Message}", "Помилка", MessageBoxButton.OK,
                MessageBoxImage.Error);
        }
    }

    private async Task<List<Doctor>?> GetEditableDoctors()
    {
        try
        {
            using var mongoService = new
MongoDbService(AppConfig.Load().MongoDbConnection,
AppConfig.Load().DatabaseName);
            var allDoctors = await
mongoService.GetAllDoctorsAsync();

            // Отримати список лікарів, яких можна відображати
(без admin, chief_doctor, visitor)
            var editableDoctors = allDoctors
                .Where(d => d.AccessLevel == "doctor")
                .ToList();

            if (editableDoctors.Count == 0)
            {
                MessageBox.Show("Немає лікарів для
відображення.", "Інформація", MessageBoxButton.OK,
MessageBoxImage.Information);
                return null;
            }
            return editableDoctors;
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Помилка при завантаженні графіка:
{ex.Message}", "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
            return null;
        }
    }
}
```

### Б.4.4 Клас «ReassignPatientsWindow»

```
public partial class ReassignPatientsWindow : Window
{
    private readonly List<Patient> _patients;
    private readonly List<Doctor> _availableDoctors;
    private readonly Dictionary<string, ComboBox>
_patientDoctorSelectors = new();
```

```
    public Dictionary<string, string> ReassignedMap { get;
private set; } = new(); // PatientId => NewDoctorId

    public ReassignPatientsWindow(List<Patient> patients,
List<Doctor> availableDoctors)
    {
        InitializeComponent();
        _patients = patients;
        _availableDoctors = availableDoctors;
        PopulatePatients();
    }

    private void PopulatePatients()
    {
        foreach (var patient in _patients)
        {
            var grid = new Grid
            {
                Margin = new Thickness(0)
            };

            grid.ColumnDefinitions.Add(new ColumnDefinition {
Width = new GridLength(2, GridUnitType.Star) });  // Номер
картки
            grid.ColumnDefinitions.Add(new ColumnDefinition {
Width = new GridLength(1, GridUnitType.Star) });  // ПІБ
            grid.ColumnDefinitions.Add(new ColumnDefinition {
Width = new GridLength(2, GridUnitType.Star) });  // КомбоБокс

            var fullNameBlock = new TextBlock
            {
                Text = $"{patient.LastName} {patient.FirstName}
{patient.MiddleName}",
                FontSize = 16,
                FontFamily = new FontFamily("Roboto"),
                VerticalAlignment = VerticalAlignment.Center,
                Margin = new Thickness(10, 0, 0, 0)
            };
            Grid.SetColumn(fullNameBlock, 0);

            var dbTextBlock = new TextBlock
            {
                Text = patient.BirthDate.ToString("dd/MM/yyyy"),
                FontSize = 16,
                FontFamily = new FontFamily("Roboto"),
                VerticalAlignment = VerticalAlignment.Center,
                Margin = new Thickness(10, 0, 0, 0)
            };
            Grid.SetColumn(dbTextBlock, 1);

            var comboBox = new ComboBox
            {
                FontSize = 16,
```

```
                FontFamily = new FontFamily("Roboto"),
                Margin = new Thickness(10, 5, 10, 5),
                DisplayMemberPath = "ShortNameWithSpec",
                SelectedValuePath = "Id",
                ItemsSource = _availableDoctors,
                VerticalAlignment = VerticalAlignment.Center
            };
            Grid.SetColumn(comboBox, 2);

            grid.Children.Add(dbTextBlock);
            grid.Children.Add(fullNameBlock);
            grid.Children.Add(comboBox);

            var border = new Border
            {
                Background = Brushes.White,
                CornerRadius = new CornerRadius(5),
                Padding = new Thickness(5),
                Margin = new Thickness(0, 5, 0, 5),
                Child = grid,
                Effect = new DropShadowEffect
                {
                    Color = Colors.Black,
                    Direction = 270,
                    ShadowDepth = 2,
                    Opacity = 0.2,
                    BlurRadius = 6
                }
            };

            PatientsPanel.Children.Add(border);

            _patientDoctorSelectors[patient.Id] = comboBox;
        }
    }

    private void BtnTakeAll_Click(object sender, RoutedEventArgs
e)
    {
        var currentUserId = App.CurrentUser?.Id;
        if (string.IsNullOrEmpty(currentUserId)) return;

        foreach (var selector in _patientDoctorSelectors.Values)
        {
            selector.SelectedValue = currentUserId;
        }
    }

    private void BtnSave_Click(object sender, RoutedEventArgs e)
    {
        ReassignedMap.Clear();
        foreach (var kvp in _patientDoctorSelectors)
        {
```

```
            var selectedDoctorId = kvp.Value.SelectedValue as
string;
            if (string.IsNullOrEmpty(selectedDoctorId))
            {
                MessageBox.Show("Усі пацієнти повинні бути
переназначені.", "Помилка", MessageBoxButton.OK,
MessageBoxImage.Warning);
                return;
            }
            ReassignedMap[kvp.Key] = selectedDoctorId;
        }

        DialogResult = true;
        Close();
    }

    private void BtnBack_Click(object sender, RoutedEventArgs e)
    {
        DialogResult = false;
        Close();
    }

    public List<(Patient, Doctor)> GetReassignments()
    {
        var result = new List<(Patient, Doctor)>();

        foreach (var patient in _patients)
        {
            if (ReassignedMap.TryGetValue(patient.Id, out var
newDoctorId))
            {
                var doctor = _availableDoctors.Find(d => d.Id ==
newDoctorId);
                if (doctor != null)
                {
                    result.Add((patient, doctor));
                }
            }
        }

        return result;
    }
}
```

### Б.4.5 Клас «DoctorsWindow»

```
public partial class DoctorsWindow : Window
{
    private List<Doctor> allDoctors = new();

    public DoctorsWindow()
```

```
    {
        InitializeComponent();
    }

    private void tbSearch_TextChanged(object sender,
TextChangedEventArgs e)
    {
        btnClearSearch.Visibility =
string.IsNullOrEmpty(tbSearch.Text)
            ? Visibility.Collapsed
            : Visibility.Visible;

        tbSearchPlaceholder.Visibility =
string.IsNullOrWhiteSpace(tbSearch.Text) && !tbSearch.IsFocused
            ? Visibility.Visible
            : Visibility.Collapsed;

        ApplyFilters();
    }


    private void tbSearch_LostFocus(object sender,
RoutedEventArgs e)
    {
        if (string.IsNullOrWhiteSpace(tbSearch.Text))
            tbSearchPlaceholder.Visibility = Visibility.Visible;
    }
    private void btnClearSearch_Click(object sender,
RoutedEventArgs e)
    {
        tbSearch.Clear();
        tbSearch.Focus();
    }

    private async void LoadDoctors()
    {
        using var mongoService = new
MongoDbService(AppConfig.Load().MongoDbConnection,
AppConfig.Load().DatabaseName);
        allDoctors = await mongoService.GetAllDoctorsAsync();

        foreach (var doctor in allDoctors) doctor.OnDutyStatus =
GetOnDutyStatus(doctor);

        ApplyFilters();
    }

    private void ApplyFilters()
    {
        var filtered = allDoctors;

        // Фільтр по тексту
        if (!string.IsNullOrWhiteSpace(tbSearch.Text))
```

```
        {
            var search = tbSearch.Text.Trim().ToLower();
            filtered = filtered.Where(d =>
                $"{d.LastName} {d.FirstName}
{d.MiddleName}".ToLower().Contains(search) ||
                $"{d.FirstName} {d.MiddleName}
{d.LastName}".ToLower().Contains(search) ||
                $"{d.LastName}
{d.FirstName}".ToLower().Contains(search)
            ).ToList();
        }

        // Фільтр по "на зміні"
        if (cbOnDutyOnly.IsChecked == true)
            filtered = filtered.Where(d =>
IsDoctorOnDuty(d)).ToList();

        DoctorsGrid.ItemsSource = filtered;
    }

    private bool IsDoctorOnDuty(Doctor doctor)
    {
        var today = DateTime.Today;
        return doctor.WorkSchedule.Any(p =>
            p.Status == WorkStatus.Work &&
            p.From.Date <= today &&
            p.To.Date >= today); }
    private void BtnEdit_Click(object sender, RoutedEventArgs e)
    {
        if (sender is Button btn && btn.DataContext is Doctor
selectedDoctor)
        {
            var editWindow = new
RegisterDoctorWindow(selectedDoctor);
            editWindow.ShowDialog();

            // Оновити список після редагування
            LoadDoctors();
        }
    }

    private async void BtnDelete_Click(object sender,
RoutedEventArgs e)
    {
        if (sender is Button btn && btn.DataContext is Doctor
selectedDoctor)
        {
            DeleteDoctorWithReassignment(selectedDoctor);
        }
    }

    private void tbSearch_GotFocus(object sender,
RoutedEventArgs e)
```

```
    {
        tbSearchPlaceholder.Visibility = Visibility.Collapsed;
    }

    private void btnBack_Click(object sender, RoutedEventArgs e)
    {
        Close();
    }

    private void CbOnDutyOnly_Checked(object sender,
RoutedEventArgs e)
    {
        ApplyFilters();
    }

    private void cbOnDutyOnly_Unchecked(object sender,
RoutedEventArgs e)
    {
        ApplyFilters();
    }

    private void Window_Loaded(object sender, RoutedEventArgs e)
    {
        // Показати кнопки тільки для головного лікаря або
адміністратора
        if (App.CurrentUser != null &&
            (App.CurrentUser.AccessLevel.ToLower() == "admin" ||
             App.CurrentUser.AccessLevel.ToLower() ==
"chief_doctor"))
        {
            colEdit.Visibility = Visibility.Visible;
            colDelete.Visibility = Visibility.Visible;
        }
        else
        {
            colEdit.Visibility = Visibility.Collapsed;
            colDelete.Visibility = Visibility.Collapsed;
        }

        LoadDoctors();
    }

    private string GetOnDutyStatus(Doctor doctor)
    {
        var today = DateTime.Today;

        // Якщо на зміні сьогодні
        var isOnShiftToday = doctor.WorkSchedule.Any(p =>
            p.Status == WorkStatus.Work &&
            p.From.Date <= today &&
            p.To.Date >= today);

        if (isOnShiftToday)
```

```
            return "На зміні";

        // Знайти найближчу майбутню зміну
        var future = doctor.WorkSchedule
            .Where(p => p.Status == WorkStatus.Work &&
p.From.Date > today)
            .OrderBy(p => p.From)
            .FirstOrDefault();

        return future != null ?
future.From.ToString("dd.MM.yyyy") : "—";
    }

    private async void DeleteDoctorWithReassignment(Doctor
_doctor) {

        using var mongoService = new
MongoDbService(AppConfig.Load().MongoDbConnection,
AppConfig.Load().DatabaseName);
        var patients = await
mongoService.GetPatientsByDoctorIdAsync(_doctor.Id);

        if (_doctor.AccessLevel.ToLower() == "viewer" ||
patients.Count == 0)
        {
            var confirm = MessageBox.Show(
                $"Ви впевнені, що хочете видалити лікаря
{_doctor.LastName}?",
                "Підтвердження видалення",
                MessageBoxButton.YesNo,
                MessageBoxImage.Warning);

            if (confirm == MessageBoxResult.Yes)
            {
                await
mongoService.DeleteDoctorAsync(_doctor.Id);
            }

            LoadDoctors();
            return;
        }

        // Якщо є пацієнти, відкриваємо вікно переназначення
        var allDoctors = await
mongoService.GetAllDoctorsAsync();

        var reassignableDoctors = allDoctors
            .Where(d =>
                d.Id != _doctor.Id &&
                d.AccessLevel.ToLower() != "visitor"
                && d.AccessLevel.ToLower() != "admin")
            .ToList();
```

```
        var reassignWindow = new
ReassignPatientsWindow(patients, reassignableDoctors);

        if (reassignWindow.ShowDialog() == true)
        {
            // Зберігаємо переназначення
            foreach (var (patient, newDoctor) in
reassignWindow.GetReassignments())
            {
                patient.DoctorId = newDoctor.Id;
                patient.Doctor = ${newDoctor.FullName};
                await mongoService.UpdatePatientAsync(patient);
            }

            // Видаляємо лікаря
            await mongoService.DeleteDoctorAsync(_doctor.Id);

            MessageBox.Show("Лікаря видалено, а пацієнтів
переназначено.", "Успіх",
                MessageBoxButton.OK,
MessageBoxImage.Information);

            LoadDoctors();
        }
    }
}
```

### Б.4.6 Клас «PatientDescription»

```
public partial class PatientDescription : Window
{
    private readonly Patient _patient;
    private Visit _visit;
    private readonly OverallDictionaries _overallDictionaries =
new();
    private readonly MongoDbService _mongoDbService;

    public PatientDescription(Patient patient)
    {
        InitializeComponent();

        var config = AppConfig.Load();
        _mongoDbService = new MongoDbService(
            config.MongoDbConnection,
            config.DatabaseName);

        _patient = patient;
        LoadData(_patient);

        if (App.CurrentUser == null)
        {
```

```
            btnChangeDesc.Visibility = Visibility.Hidden;
            btnAddVisit.Visibility = Visibility.Hidden;
            btnGenerateDoc.Visibility = Visibility.Hidden;
            btnChangeVisit.Visibility = Visibility.Hidden;
        }

        if (App.CurrentUser.AccessLevel == "admin")
        {
            btnChangeDesc.Visibility = Visibility.Hidden;
            btnAddVisit.Visibility = Visibility.Hidden;
        }

        btnGenerateDoc.btnClick += btnGenerateDocument;
        btnChangeDesc.btnClick += btnChangeDescription;
        btnAddVisit.btnClick += btnAddNewVisit;
        btnChangeVisit.btnClick += btnChangeVisitTo;
    }

    private Visit GetLatestVisit()
    {
        return _patient.Visits?.LastOrDefault() ?? new Visit();
    }

    private void LoadData(Patient patient, Visit? selectedVisit
= null)
    {
        var visit = selectedVisit ?? GetLatestVisit(); //
використати обраний або останній
        _visit = visit;

        string GetSymptom(string key, string fallback = "-------
-")
        {
            if (visit.Symptoms == null ||
!visit.Symptoms.TryGetValue(key, out var value))
                return fallback;

            try
            {
                var list =
JsonConvert.DeserializeObject<List<string>>(value);
                if (list != null)
                    return string.Join("\n• ",
list.Prepend("")).TrimStart('\n').Trim();
            }
            catch
            {
                if (key == "_fieldClaims")
                    MessageBox.Show(
                        "Скарги не будуть виведені коректно,
оскільки збережені в неправильному форматі.",
                        "Попередження",
                        MessageBoxButton.OK,
```

```
                    MessageBoxImage.Warning
                );
            }

            return value;
        }

        lbCardNumber.Content = $"Картка #{patient.CardNumber}";
        lbTypDescriptionItem1.Content = "Прізвище: " +
patient.LastName;
        lbTypDescriptionItem2.Content = "Ім'я: " +
patient.FirstName;
        lbTypDescriptionItem3.Content = "По-батькові: " +
patient.MiddleName;
        lbTypDescriptionItem4.Content = "Дата народження: " +
patient.BirthDate.ToString("dd.MM.yyyy");
        lbTypDescriptionItem5.Content = "Вік: " + patient.Age;

        lbCurrentVisit.Content = $"Візит від
{visit.StartDate.ToString("d")}";

        tbLastName.Text = patient.LastName;
        tbFirstName.Text = patient.FirstName;
        tbMiddleName.Text = patient.MiddleName;
        tbBirthday.Text =
patient.BirthDate.ToString("dd.MM.yyyy");
        tbLivingAddress.Text = patient.Address;
        tbWork.Text = patient.Profession;
        tbHospitalStart.Text =
visit.StartDate.ToString("dd.MM.yyyy") ?? "--.--.----";
        tbHospitalEnd.Text = visit.EndDate.Year == 1 ? "--.--.--
--" : visit.EndDate.ToString("dd.MM.yyyy");

        try
        {
            var claimsRaw = GetSymptom("_fieldClaims");
            var claimsList =
JsonConvert.DeserializeObject<string[]>(claimsRaw ?? "[]");
            tbClims.Text = claimsList != null ? string.Join(",
", claimsList) : "--------";
        }
        catch
        {
            tbClims.Text = GetSymptom("_fieldClaims") ??
                           "Скарги не будуть виведені коректно,
оскільки збережені в неправильному форматі";
        }

        tbEntrDiagnosis.Text =
GetSymptom("_fieldEntrDiagnosis");
        tbFinalDiagnosis.Text =
GetSymptom("_fieldFinalDiagnosis");
        tbComplications.Text = GetSymptom("_fieldComplication");
```

```
        tbAdditionDiagnosis.Text =
GetSymptom("_fieldAdditionalDiagnosis");
        tbMKX.Text = GetSymptom("_fieldMKX");
        tbOperationName.Text =
GetSymptom("_fieldOperationName");
        dateOperation.Text = GetSymptom("_fieldOperationDate",
"--.--.----");
        tbChemotherapyName.Text =
GetSymptom("_fieldChemotherapy");
        dateChemotherapy.Text =
GetSymptom("_fieldChemotherapyDate", "--.--.----");
        tbHistology.Text = GetSymptom("_fieldHistology");
        tbDoctorName.Text = patient.Doctor;
        tbDepartmentHead.Text =
GetSymptom("_fieldDepartmentHead");
        tbDepartHeadAssistant.Text =
GetSymptom("_fieldDepartHeadAssistant");

        // Extended TabItems fields
        //tbOverallItem1.Text =
_overallDictionaries.Dictionaries["dictOverallItem1"][GetSymptom
("_fieldOverallItem1")];
        tbOverallItem1.Text = GetDictValue("dictOverallItem1",
GetSymptom("_fieldOverallItem1"));
        tbOverallItem2.Text = GetDictValue("dictOverallItem2",
GetSymptom("_fieldOverallItem2"));
        tbOverallItem3.Text = GetDictValue("dictOverallItem3",
GetSymptom("_fieldOverallItem3"));
        tbOverallItem4.Text = GetDictValue("dictOverallItem4",
GetSymptom("_fieldOverallItem4"));
        tbOverallItem5.Text = GetDictValue("dictOverallItem5",
GetSymptom("_fieldOverallItem5"));
        tbOverallItem6.Text = GetDictValue("dictOverallItem6",
GetSymptom("_fieldOverallItem6"));
        tbOverallItem7.Text = GetSymptom("_fieldOverallItem7")
?? "--------";
        tbOverallItem8.Text = GetDictValue("dictOverallItem8",
GetSymptom("_fieldOverallItem8"));

        tbOverallItem9_1.Text =
GetSymptom("_fieldOverallItem9_1") == "true" ? "Позитивний" :
"Негативний";
        tbOverallItem9_2.Text =
GetSymptom("_fieldOverallItem9_1") == "true"
            ? GetSymptom("_fieldOverallItem9_2") == "true" ?
"Зліва" : "Зправа"
            : "---------";

        tbOverallItem10.Text =
GetMultiDictValue("dictOverallItem10",
GetSymptom("_fieldOverallItem10"));
        tbOverallItem11.Text = GetSymptom("_fieldOverallItem11")
?? "--------";
```

```
        tbOverallItem12.Text = GetSymptom("_fieldOverallItem12")
?? "--------";
        tbOverallItem13.Text = GetDictValue("dictOverallItem13",
GetSymptom("_fieldOverallItem13"));
        tbOverallItem14.Text =
GetMultiDictValue("dictOverallItem14",
GetSymptom("_fieldOverallItem14"));
        tbOverallItem15.Text = GetSymptom("_fieldOverallItem15")
?? "--------";

        tbLifeAnamnesisItem1.Text =
GetSymptom("_fieldLifeAnamnesisItem1") ?? "--------";
        tbLifeAnamnesisItem2.Text =
GetSymptom("_fieldLifeAnamnesisItem2") ?? "--------";
        tbLifeAnamnesisItem3.Text =
GetSymptom("_fieldLifeAnamnesisItem3") ?? "--------";
        tbLifeAnamnesisItem4.Text =
GetSymptom("_fieldLifeAnamnesisItem4") ?? "--------";
        tbLifeAnamnesisItem5.Text =
GetSymptom("_fieldLifeAnamnesisItem5") ?? "--------";
        tbLifeAnamnesisItem6.Text =
GetSymptom("_fieldLifeAnamnesisItem6") ?? "--------";
        tbLifeAnamnesisItem7.Text =
GetSymptom("_fieldLifeAnamnesisItem7") ?? "--------";
        tbLifeAnamnesisItem8.Text =
GetSymptom("_fieldLifeAnamnesisItem8") ?? "--------";
        tbLifeAnamnesisItem9.Text =
GetSymptom("_fieldLifeAnamnesisItem9") ?? "--------";
        tbLifeAnamnesisItem10.Text =
GetSymptom("_fieldLifeAnamnesisItem10") ?? "--------";

        tbLocusMorbiItem1.Text =
GetSymptom("_fieldLocusMorbiItem1") ?? "--------";
        tbLocusMorbiItem2.Text =
GetSymptom("_fieldLocusMorbiItem2") ?? "--------";
        tbLocusMorbiItem3.Text =
GetSymptom("_fieldLocusMorbiItem3") == "true" ? "Везикулярне" :
"Жорстке";
        tbLocusMorbiItem4.Text =
GetSymptom("_fieldLocusMorbiItem4") ?? "--------";
        tbLocusMorbiItem5.Text =
GetSymptom("_fieldLocusMorbiItem5") == "true" ? "Легеневий звук"
: "Коробчатий звук";
        tbLocusMorbiItem6.Text =
GetSymptom("_fieldLocusMorbiItem6") ?? "--------";
        tbLocusMorbiItem7.Text =
GetSymptom("_fieldLocusMorbiItem7") ?? "--------";
    }

    private void btnGenerateDocument(object sender,
RoutedEventArgs e)
    {
        var doc = new DocBuilder(_patient, _visit);
```

```
        doc.Show();
    }

    private async void btnChangeDescription(object sender,
RoutedEventArgs e)
    {
        var infoEditor = new AddPatient(_patient, false,
_visit);
        infoEditor.ShowDialog();

        var updatedPatient = await
_mongoDbService.GetPatientByIdAsync(_patient.Id);

        if (updatedPatient != null)
            LoadData(updatedPatient);
        else
            MessageBox.Show("Не вдалося оновити дані пацієнта.",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
    }

    private void btnAddNewVisit(object sender, RoutedEventArgs
e)
    {
        var infoEditor = new AddPatient(_patient, true);
        infoEditor.ShowDialog();
    }

    private void btnChangeVisitTo(object sender, RoutedEventArgs
e)
    {
        if (_patient.Visits == null || _patient.Visits.Count ==
0)
        {
            MessageBox.Show("Немає доступних візитів для
вибору.", "Інформація", MessageBoxButton.OK,
MessageBoxImage.Information);
            return;
        }

        var visitSelector = new VisitSelector(_patient.Visits,
$"{_patient.LastName} {_patient.FirstName}
{_patient.MiddleName}");
        if (visitSelector.ShowDialog() == true)
        {
            var selectedVisit = visitSelector.SelectedVisit;
            if (selectedVisit != null)
            {
                LoadData(_patient, selectedVisit); // оновлений
метод з параметром
            }
        }
    }
```

```
    private void typicalDescription_Loaded(object sender,
RoutedEventArgs e)
    {
    }

    private void Window_Activated(object sender, EventArgs e)
    {
    }

    private string GetDictValue(string dictName, string
indexStr)
    {
        if (int.TryParse(indexStr, out var index))
        {
            var dict =
_overallDictionaries.Dictionaries[dictName][index];
            if (dict != null && index >= 0)
                return dict;
        }

        return "--------";
    }

    private string GetMultiDictValue(string dictName, string
indicesStr)
    {
        try
        {
            var result = "";

            indicesStr.Where(char.IsDigit).ToList()
                .Select(c => char.GetNumericValue(c)).ToList()
                .ForEach(c => { result +=
$"{_overallDictionaries.Dictionaries[dictName][Convert.ToInt32(c
)]}, "; });

            return string.Join("\n• ", result.Trim()
                    .TrimEnd(',')
                    .Split(", ")
                    .Prepend("")
                )
                .TrimStart('\n').Trim();
        }
        catch
        {
        }

        return "--------";
    }
}
```