**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки**

**Катедра інформаційних систем та технологій**

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____2025 р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**
**за освітньо-професійною програмою «Інформаційні управляючі системи та технології»**
**спеціальності 126 «Інформаційні системи та технології»**
**на тему: «Вебзастосунок підбору психолога на основі індивідуальних потреб користувачів»**

Виконала:
студентка IV курсу, групи ІС-11
Щербініна Олександра Володимирівна _____

Керівник:
доцент катедри ІСТ, к.ф.-м.н., доцент
Рибачук Людмила Віталіївна _____

Рецензент:
доцент кафедри ІПІ, к.т.н.
Олійник Юрій Олександрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2025 рік

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Факультет інформатики та обчислювальної техніки**
**Катедра інформаційних систем та технологій**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2025 р.

# ЗАВДАННЯ
**на дипломний проєкт студентці**
**Щербініній Олександрі Володимирівні**

1. Тема проєкту «Вебзастосунок підбору психолога на основі індивідуальних потреб користувачів», керівник проєкту Рибачук Людмила Віталіївна, к.ф.-м.н., доцент, затверджені наказом по університету від «23» травня 2025 р. № 1705-с

2. Термін подання студентом проєкту: «09» червня 2025 р

3. Вихідні дані до проєкту: мови програмування C#, TypeScript, фреймворки ASP.NET, Angular.

4. Зміст пояснювальної записки: опис предметної області, аналіз існуючих рішень, формування вимог до системи, вибір технологій розроблення, розроблення вебзастосунку, математичне забезпечення, тестування інформаційної системи.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): діаграма процесів, діаграма варіантів використань, діаграма комунікації клієнта з сервером, діаграма структури бази даних.

6. Дата видачі завдання: «7» березня 2025р.

Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
|---|---|---|---|
| | Аналіз предметної області | | |
| | Огляд існуючих аналогів | | |
| | Розробка функціональної моделі | | |
| | Формування вимог до системи | | |
| | Вибір технології розробки | | |
| | Розробка програмного забезпечення | | |
| | Опис технічного розв’язку | | |
| | Дослідження і розробка математичного забезпечення | | |
| | Тестування системи | | |

Студентка Олександра ЩЕРБІНІНА

Керівник Людмила РИБАЧУК

# АНОТАЦІЯ

Вебзастосунок підбору психолога на основі індивідуальних потреб користувачів.

Проєкт містить 63 с. тексту, 17 рисунків, 12 таблиць, посилання на 15 літературних джерел, додаток та 4 конструкторських документи.

ВЕБРОЗРОБЛЕННЯ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ПІДБІР ПСИХОЛОГА, ПРИЙНЯТТЯ РІШЕНЬ, TOPSIS.

Об’єктом розроблення є процес підбору психолога на основі індивідуальних потреб користувача.

Мета розроблення – автоматизація процесу підбору психолога відповідно до визначених індивідуальних критеріїв користувача.

У дипломному проєкті розроблено вебзастосунок, який забезпечує підбір психолога, запис на консультацію, панель адміністратора та спеціаліста. Проведено аналіз існуючих рішень, на основі якого будувалась стратегія розробки.

Отримані результати можуть бути корисними для людей, що мають на меті отримати психологічну допомогу і обрати психолога за важливими для себе критеріями.

# SUMMARY

Web application for selecting a psychologist based on users' individual needs.

The project contains 63 pages of text, 17 figures, 12 tables, references to 15 sources, appendices and 4 design documents.

Keywords: client-server architecture, decision making, psychologist selection, TOPSIS, web development.

The object of development is the process of selecting a psychologist based on individual user needs.

The goal of development is to automate the process of psychologist selection according to defined individual user criteria.

In this diploma project, a web application has been developed that provides psychologist selection, appointment booking, and administrator and specialist panels. An analysis of existing solutions was conducted, which formed the basis for the development strategy.

The obtained results can be useful for people who aim to receive psychological help and choose a psychologist based on criteria important to them.

| № рядка | Формат | Позначення | Найменування | Кільк. ст. | № екз. | Прим. |
|---|---|---|---|---|---|---|
| 1 | | | Документація загальна | | | |
| 2 | | | | | | |
| 3 | | | Знову розроблена | | | |
| 4 | | | | | | |
| 5 | А4 | ІС11.290БАК.005 ПЗ | Вебзастосунок підбору психолога | 63 | | |
| 6 | | | на основі індивідуальних потреб | | | |
| 7 | | | користувачів. | | | |
| 8 | | | Пояснювальна записка | | | |
| 9 | А3 | ІС11.290БАК.005 Д1 | Вебзастосунок підбору психолога | 1 | | |
| 10 | | | на основі індивідуальних потреб | | | |
| 11 | | | користувачів. | | | |
| 12 | | | Діаграма процесів | | | |
| 13 | А3 | ІС11.290БАК.005 Д2 | Вебзастосунок підбору психолога | 1 | | |
| 14 | | | на основі індивідуальних потреб | | | |
| 15 | | | користувачів. | | | |
| 16 | | | Діаграма варіантів використань | | | |
| 17 | А3 | ІС11.290БАК.005 Д3 | Вебзастосунок підбору психолога | 1 | | |
| 18 | | | на основі індивідуальних потреб | | | |
| 19 | | | користувачів. | | | |
| 20 | | | Діаграма комунікації клієнта | | | |
| 21 | | | з сервером | | | |
| 22 | А3 | ІС11.290БАК.005 Д4 | Вебзастосунок підбору психолога | 1 | | |
| 23 | | | на основі індивідуальних потреб | | | |
| 24 | | | користувачів. | | | |
| 25 | | | ER-діаграма | | | |
| 26 | | | | | | |
| 27 | | | | | | |
| 28 | | | | | | |

| Зм. | Аркуш | № докум. | Підпис | Дата | ІС11.290БАК.005 ТП | | |
|---|---|---|---|---|---|---|---|
| Розроб. | | Щербініна О. В. | | | Вебзастосунок підбору психолога на основі індивідуальних потреб користувачів. Відомість дипломного проєкту | Літ. | Аркуш 1 / Аркушів 1 |
| Керівн. | | Рибачук Л. В. | | | | | |
| Реценз. | | | | | | КПІ ім. Ігоря Сікорського Група ІС-11 | |
| Н. Контр. | | | | | | | |
| Затверд. | | | | | | | |

# Пояснювальна записка
# до дипломного проєкту
# на тему: «Вебзастосунок підбору психолога на основі індивідуальних потреб користувачів»

Київ – 2025 рік

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних.

ПЗ – програмне забезпечення.

СУБД – система управління базами даних.

РСУБД – реляційна система управління базами даних.

API – Application Programming Interface.

BLL – Business Logic Layer.

DAL – Data Access Layer.

E2E testing – End to End testing.

HTTP – HyperText Transfer Protocol.

IDE – Integrated Development Environment.

REST – REpresentional State Transfer.

SOAP – Simple Object Access Protocol.

TOPSIS – The Technique for Order of Preference by Similarity to Ideal Solution.

UI – User Interface.

UX – User Experience.

# ВСТУП

У сучасному світі, коли базові потреби людей здебільшого задоволені, з'являється можливість потурбуватись і про вищі потреби, такі як емоційні та психологічні. Задля природного розвитку особистості потрібно мати можливість пошуку гармонії з собою та суспільством. Тому, дедалі більше людей звертаються за підтримкою до спеціалістів, що готові допомогти віднайти баланс у ментальному та фізичному.

Однак, на початку цього шляху може виникнути низка запитань: де приймає цей спеціаліст, чи є можливість працювати дистанційно, який тип комунікації використовується. Також виникають і важливіші питання, що напряму впливають на користь терапії: з якими запитами працює терапевт, як знайти «свою» людину, наскільки безпечно і комфортно буде під час сеансів.

Неоднозначність в цих питаннях часто заганяє в кут потенційних клієнтів, що обумовлюється страхом обрати непідходящого спеціаліста, що, можливо, не тільки не покращить ситуацію, а й зробить ще гірше. У висновку, така людина лишається наодинці зі своїми проблемами та переживаннями, блокуючи можливість дати собі шанс на краще розуміння себе.

Підбір психолога – це перший, але дуже важливий етап у майбутній терапії. Це вибір, що задає курс подальшому лікуванню та психологічному добробуту у майбутньому. Правильно обраний спеціаліст може допомогти глибше пізнати себе, навчити обробляти свої емоції, прийняти особливості характеру або здолати внутрішні бар'єри.

Метою розробки є автоматизація процесу знаходження психолога, за визначеними критеріями.

Задачею дипломного проєкту є розробка вебзастосунку, що забезпечує підтримку підбору психолога засобами клієнтської та серверної частини. Для оптимального алгоритму має бути здійснений аналіз наявних рішень. Також, значна увага

має бути приділена розробці програмного забезпечення та реалізації архітектури системи.

Для досягнення мети наступні задачі мають бути виконаними:

− аналіз наявних систем зі схожим функціоналом;

− виокремлення переваг майбутньої системи з існуючими;

− вирізнення критеріїв підбору та алгоритму прийняття рішення;

− проєктування та розробка системи підбору психолога на основі індивідуальних потреб користувачів.

Результати дипломного проєкту мають потенціал у використанні як окремим сервісом так і частиною великої платформи, у яку реалізований програмний продукт матиме можливість інтегруватись. Вебзастосунок буде швидким і зручним розв’язанням проблеми пошуку психолога, оскільки надаватиме можливість оптимального вибору спеціаліста майбутньому користувачеві.

Дипломний проєкт складається з наступних розділів: вступ, опис предметної області, аналіз існуючих рішень, формування вимог до системи, вибір технологій розроблення, розроблення вебзастосунку, математичне забезпечення, тестування системи, висновки, список використаних джерел із 15 ресурсами, 1 додаток.

Графічна частина містить 4 кресленики формату А3. Загальний обсяг 63 сторінки.

# 1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Опис процесу діяльності

Головною метою дипломного проєкту є створення вебзастосунку, що дає можливість користувачу легко і швидко підібрати психолога по власним потребам. Користувачі даного застосунку мають доступ до таких функцій: проходження тесту на визначення запитів до майбутнього терапевта, вибір спеціаліста на основі результатів опитування, перегляд анкет рекомендованих психологів, запис до терапевта, відміна або перенесення запланованої зустрічі.

Окрім цього, має бути передбачено панель адміністратора сайту для контролю робочого процесу: додавання та видалення спеціаліста, редагування інформації, перегляд та управління записами.

Для самих спеціалістів має бути реалізовано персональну панель, яка дозволяє обробляти запити користувачів, оновлювати інформацію у своєму профілі та керувати власним робочим графіком.

## 1.2 Постановка задачі

З розвитком сучасного суспільства все більше уваги приділяється психологічному здоров’ю. Проте, через насиченість підходів до терапії та спеціалістів, що їх використовують, людині, що проходить шлях терапії вперше, легко розгубитись та втратити самоконтроль. Звичайно, майбутній клієнт має можливість скористатись мережею Інтернет і отримати інформацію про всі види терапії, запити, з якими працюють фахівці тощо. Але в реаліях сьогодення, коли ритм життя з кожною миттю пришвидшується, далеко не кожен має час і бажання заглиблюватись у цю інформацію задля того аби зробити усвідомлений вибір.

Саме тому, важливо мати зручний та надійний інструмент, що дасть можливість полегшити та пришвидшити процес пошуку свого терапевта. Користуючись розробленою системою, можливо буде легко розпочати свій шлях у терапії та почати роботу над своїм емоційним благополуччям.

### 1.2.1 Призначення системи

Метою роботи є розроблення вебзастосунку підбору психолога на основі індивідуальних потреб користувачів.

Майбутня система призначена для людей, що прагнуть почати свій шлях у терапії, проте стикаються з браком часу та недостатнім обсягом знань у сфері психології для самостійного вибору фахівця. Мета системи – спростити пошук психолога, зробити його більш швидким, зручним і відповідним запитам користувача.

### 1.2.2 Цілі та задачі розробки

Ціллю даної роботи є розроблення вебзастосунку, що здійснює підбір психолога на основі результатів опитувальника, використовуючи алгоритм, який оптимізує процес вибору фахівця за заданими критеріями.

Майбутня розробка у комплексі має мати повністю реалізований функціонал для користувача, адміністратора та спеціаліста для ефективної, зручної та швидкої роботи застосунку.

## Висновки до розділу 1

Отже, областю дослідження є розробка вебзастосунку підбору психолога за індивідуальними потребами користувачів, яка допоможе автоматизувати процес вибору спеціаліста для потенційних клієнтів.

Було окреслено проблематику даної сфери, на основі яких було сформовано цілі дипломного проєкту.

# 2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Майбутній вебзастосунок націлений на вільне перебування у онлайн-просторі, щоб потенційні користувачі могли легко знайти та скористатись розробленою системою.

Наразі, існує багато схожих сервісів, що дають можливість підібрати спеціаліста та ініціювати співпрацю. Безсумнівно, такі сервіси мають багато переваг, але також вони мають і недоліки на рівні логіки формування підбору та user experience.

Для огляду існуючих рішень можна виділити три сервіси:

а) Rozmova [1];

б) Qui.help [2];

в) Pleso [3].

## 2.1 Огляд існуючих рішень

На сьогодні існує велика кількість різноманітних платформ по пошуку психолога. Кожна з них різниться своїм способом підбору спеціаліста, підходами до комунікації з кінцевим користувачем, додатковими функціями, що можуть стати у нагоді. Також вони мають різне наповнення сторінки – деякі сервіси опираються на підбір психолога за допомогою опитувальників, в той час як інші реалізують тільки пошук по фільтрам. Ці невеликі на перший погляд деталі неабияк впливають на досвід користувача та ймовірність того, що він скористається цим застосунком знову чи порадить його до користування знайомим та друзям.

У цьому підрозділі буде оглянуто деякі з наявних рішень на ринку, що повністю функціонують та надають послуги з метою виявлення їх сильних та слабких сторін. На основі цього дослідження будуть сформовані висновки, що матимуть змогу окреслити орієнтири майбутньої розробки.

### 2.1.1 Огляд сервісу Rozmova

Rozmova – український продукт. Це вебзастосунок, функціонал якого представлений українською та російською мовами. Головне меню застосунку показано на рисунку 2.1.

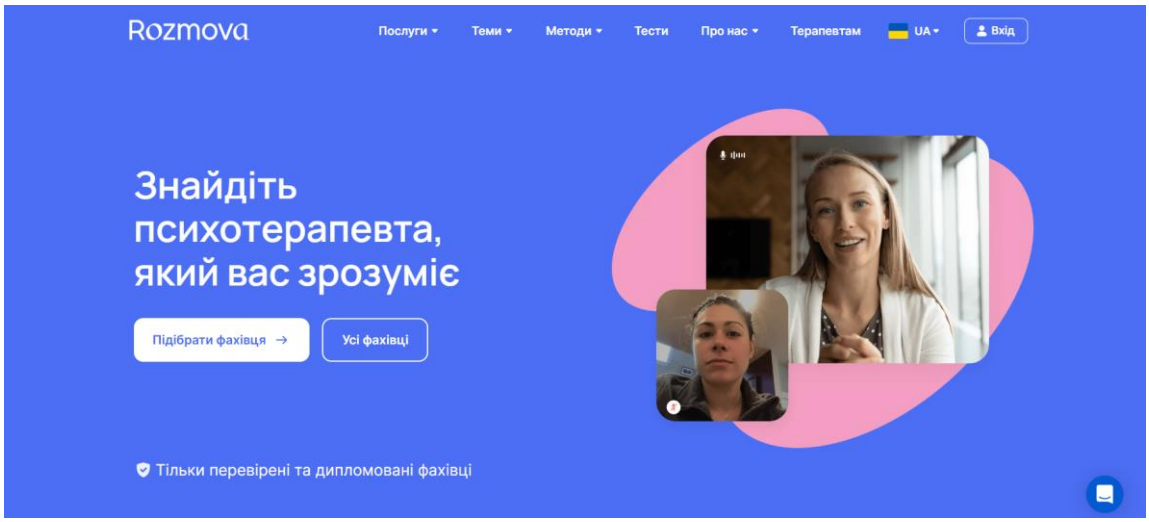


Рисунок 2.1 – Головне меню Rozmova [1]

На головному екрані розміщена кнопка переходу на опитувальник, що визначає запити та потреби клієнта. Пройти опитувальник можна без реєстрації, що збільшує ймовірність проходженння тестування. Проте для перегляду рекомендованих терапевтів потрібно зареєструватись, що може знизити залученість нових користувачів після завершення тесту. Сам опитувальник складається з невеликої кількості простих питань, які дозволяють швидко та ефективно визначити потреби користувача.

Після проходження опитувальника алгоритм застосунку обробляє отримані відповіді та підбирає фахівців, що найбільше відповідають потребам користувача, але для їх перегляду потрібно зареєструватись на платформі.

Переваги:

- мультиплатформеність;
- наявний функціонал для клієнта та спеціаліста.

Недоліки:

− для перегляду рекомендованих фахівців потрібна реєстрація;

− наявність російської мови.

### 2.1.2 Огляд сервісу Qui.help

Наступний сервіс для підбору спеціаліста це qui.help. Тут, на відміну від Rozmova, немає можливості пройти опитувальник, але для цього користувач може використати фільтри у пошуку. Головна сторінка, зображена на рисунку 2.2, дає можливість швидко переглянути доступних спеціалістів та перейти на їхній профіль.

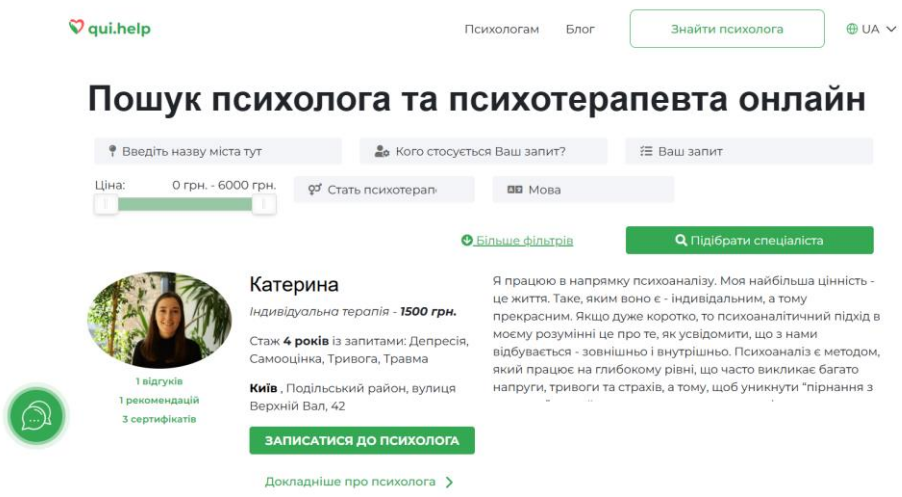


Рисунок 2.2 – Головна сторінка сервісу qui.help [2]

Для запису до спеціаліста використовується коротка форма, де користувач вказує свої контактні дані, обирає зручний час, за бажанням може озвучити свої запити та очікування від сесії з терапевтом. Загалом, функціонал застосунку є необширним та зрозумілим для користувача, що сприяє комфортному використанню та формуванню позитивного враження про даний сервіс.

Переваги:

− зручний та зрозумілий інтерфейс;

– можливість фільтрації;

– не потрібна реєстрація для запису;

Недоліки:

– недопрацьована система фільтрації по запиту (є можливість обрати лише один);

– наявність російської мови в інтерфейсі.

### 2.1.3 Огляд сервісу Pleso

Останнім на огляді є Pleso. На рисунку 2.3 зображена головна сторінка цього вебсайту.

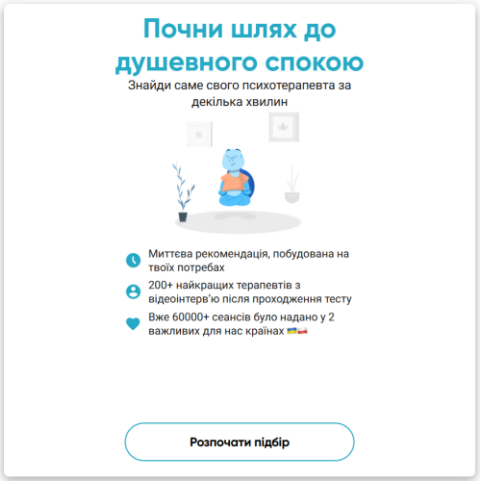


Рисунок 2.3 – Головна сторінка сервісу Pleso [3]

Тут користувачеві одразу пропонується пройти опитувальник, що підбере потрібного спеціаліста за кілька хвилин. Опитувальник містить велику кількість коротких питань, що дозволяють оцінити стан і очікування від терапії опитуваного.

Кількість питань в опитувальнику може бути як перевагою, так і недоліком. З одного боку це допомагає дізнатись системі краще про людину та надати більш

чіткий результат, але з іншого це може відштовхувати потенційного клієнта, оскільки процес проходження тестування буде куди довше, ніж на інших платформах на ринку.

Аналогічно до першого сервісу результати опитувальника можна переглянути лише після реєстрації, що також може бути як перевагою так і недоліком, оскільки такий підхід забезпечує легку ідентифікацію зареєстрованого клієнта, але з іншого боку може знизити залученість нових користувачів через додаткові кроки, обов’язковість яких не є обґрунтованою.

Після реєстрації застосунок дає можливість переглянути терапевтів, що відгукуються по запитам з опитувальника, прочитати їх короткий опис та подивитись відео. Крім того, користувач може записатись на сеанс та переглядати відгуки на спеціалістів.

Переваги:

- обширний опитувальник;
- зручний та зрозумілий інтерфейс.

Недоліки:

- обширний опитувальник;
- для перегляду рекомендованих фахівців потрібна реєстрація;
- неможливість перегляду всіх спеціалістів на платформі.

## 2.2 Аналіз існуючих рішень

Під час дослідження було оглянуто і проаналізовано три сервіси, що надають можливість підібрати психолога. Об’єктами досліджень були вебзастосунки Rozmova, Qui.help та Pleso.

Дані застосунки користуються популярністю серед користувачів, що обрали звернутись за підтримкою до спеціаліста. Вони мають велику кількість переваг, через які їх обирають: зручний інтерфейс, можливість швидко задовольнити свою потребу, полегшення вибору психолога завдяки фільтрації або алгоритмам підбору.

На противагу цьому, при аналізі обраних систем, було виділено низку недоліків, через яку потенційні користувачі можуть припинити користування платформами. Основними недоліками є примус реєстрації для перегляду результатів опитування та запису до спеціаліста, неможливість переглянути спеціалістів, якщо людина вирішила не проходити опитувальник, нелогічна фільтрація та наявність російської мови в інтерфейсі застосунку.

По результатам аналізу було вирішено врахувати переваги конкурентів та усунути виявлені недоліки задля конкурентоспроможності. Створена система матиме наступні переваги над іншими платформами:

- отримання результатів опитування без реєстрації;
- можливість переглянути всіх спеціалістів на платформі;
- можливість зручної фільтрації по спеціалістам;
- повністю україномовний інтерфейс.

Висновки до розділу 2

У даному розділі було розглянуто та проаналізовано декілька застосунків з суміжним функціоналом, що дало змогу сформувати вичерпні вимоги до системи для досягнення конкурентоспроможності. Аналіз існуючих рішень дозволив виявити їх сильні та слабкі сторони, які стали основою для розробки майбутнього сервісу. Отримані висновки також дають змогу уникнути типових помилок на етапах проєктування та реалізації системи, що сприятиме підвищенню її ефективності, зручності та відповідності потребам цільової аудиторії.

# 3 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

Для формування вимог до системи необхідно мати чітке уявлення про процеси, які будуть реалізовані в межах майбутньої розробки. Тому на цьому етапі доцільно розглянути основні функції та операції, що ляжуть у основу створюваного вебзастосунку. Їх опис сприятиме задоволенню цілей, поставлених у першому розділі та кращого формування бізнес-логіки.

У таблиці 3.1 наведено опис майбутніх процесів.

Таблиця 3.1 – Опис майбутніх процесів

| Назва | Опис |
|---|---|
| Проходження опитувальника | Потенційний клієнт обирає пройти опитувальник, відповідає на всі поставлені запитання, вказує свої побажання щодо майбутньої терапії |
| Підбір спеціаліста | Після проходження опитувальника система за допомогою математичного алгоритму формує запит користувача та надає йому список рекомендованих спеціалістів |
| | Користувач обирає не проходити опитувальник, і для підбору спеціалістів оглядає список наявних на платформі або користується пошуком за фільтрацією |
| Запис на консультацію | Після перегляду рекомендованих спеціалістів користувач обирає найкращого для себе та записується на консультацію |

Опис процесів з використанням діаграми BPMN (Business Process Model and Notation) наведено на кресленику ІС11.290БАК.005 Д1.

## 3.1 Вимоги до системи в цілому

Для формування розуміння функціональності майбутньої системи, варто сформувати вимоги до неї і притримуватись їх під час розроблення.

Майбутня система не має бути підв’язаною під конкретну операційну систему або технічні характеристики пристрою, повинна функціонувати у різних браузерах та на різному забезпеченні.

Система має бути стійкою до аварійних ситуацій: проблеми з мережею, помилки серверів, неправильне введення даних користувачем.

Також система забезпечення збереження даних повинна гарантувати стійке і чітке відпрацювання, незалежно від обставин та мати можливість відновлювати втрачені дані.

## 3.2 Вимоги до функціональних характеристик

Розроблена система повинна підтримувати три ролі:

- клієнт;
- спеціаліст;
- адміністратор.

Розглянемо функціональні вимоги для кожної ролі:

а) клієнт:

роходження опитувальника;

ормування запису;

б) загальні для спеціаліста та адміністратора:

вторизація;

едагування профіля спеціаліста;

едагування інформації про записи;

в) адміністратор:

одавання спеціаліста;

идалення спеціаліста.

Діаграма варіантів використань, побудована на основі описаних вимог, наведена на кресленику ІС11.290БАК.005 Д2.

## 3.3 Вимоги до видів забезпечення

Розроблений алгоритм підбору спеціаліста повинен ґрунтуватися на аналізі даних та використанні математичних методів для досягнення оптимального розв’язку поставленої задачі.

Процеси збереження та управління даними мають бути надійними, працювати стабільно незалежно від зовнішніх чинників. Для забезпечення зручної роботи з базою даних буде застосована система управління базами даних (СУБД), що забезпечує надійність та високу продуктивність.

Для зручного використання вебзастосунком користувач повинен використовувати один з наступних браузерів:

- Google Chrome версії 120 або новіше;
- Firefox версії 81 або новіше;
- Microsoft Edge версії 91 або новіше.

## Висновки до розділу 3

У даному розділі було описано функціональні процеси, які має забезпечувати розроблена система, та вимоги, яким вона має відповідати. Було сформульовано вимоги до системи в цілому, зокрема до надійності в умовах аварійних ситуацій, до функціональних характеристик та до видів забезпечення.

# 4 ВИБІР ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ

## 4.1 Вибір стеку технологій

Метою даної роботи є створення програмного забезпечення, що містить у собі реалізацію підбору психолога за індивідуальними потребами користувачів. Було вирішено розробити вебзастосунок, оскільки такий вибір зумовлений низкою переваг, у порівнянні з десктопними або мобільними застосунками:

− кросплатформенність: розгортання вебзастосунку для різних браузерів є значно простішим, ніж створення застосунків для окремих платформ, оскільки його можна використовувати незалежно від пристрою та його операційної системи;

− швидкість оновлення та ціна помилки: оскільки вебзастосунок не націлений на встановлення на пристрій користувача, а функціонує на сервері, будь-які зміни мають змогу бути закоміченими миттєво, і це оновлення одразу діятиме для всіх користувачів, що значно зменшує ризики при оновленні та витрати на обслуговування системи;

− масштабованість: вебзастосунок дозволяє легко і гнучко масштабуватись без різких змін у архітектурі застосунку;

− ціна розробки: вебзастосунки створюються без прив’язки до системи, на якій він функціонує, натомість для звичайного застосунку потрібно інтегруватись для різних операційних систем, їх версій та типів пристроїв, що використовуються;

− інтеграція з зовнішніми сервісами: наразі існує достатньо багато сервісів, що дають змогу розгортати вебзастосунки швидше і легше, розширюючи можливості не створювати деякі процеси з нуля, а використати існуючий, стабільний розв’язок.

Для розробки вебзастосунку будуть використовуватись такі компоненти технічного стеку:

− C# як мова програмування;

− ASP.NET фреймворк для бекенд частини;

− Angular фреймворк для фронтенд частини;

− PostgreSQL для зберігання даних;

− Entity Framework як компонента взаємодії з базою даних;

− Visual Studio як середовище розробки для серверної частини застосунку;

− Visual Studio Code як середовище розробки для клієнтської частини застосунку.

## 4.2 Мова програмування C#

C# – це кросплатформна, об’єктно-орієнтована мова програмування, розроблена компанією Microsoft у 2000 році як частина платформи .NET. Ця мова була створена на основі напрацювань з інших C-подібних мов розробки: Java, C++, C, JavaScript, через що користувачі виділяють її як одну з найсильніших мов програмування сучасності.

Основними особливостями цієї мови програмування є:

− об'єктно-орієнтований підхід: C# реалізує в собі всі принципи об'єктно-орієнтованого програмування: поліморфізм, абстракція, наслідування та інкапсуляція;

− інтеграція з платформою .NET: C# була створена як частина платформи .NET, яка містить велику кількість функцій та фреймворків, яких немає для інших мов програмування, тому їх взаємна робота є потужною і продуктивною, маючи змогу розробляти безліч типів застосунків не витрачаючи зайвих ресурсів;

− висока продуктивність розробки: з останніми оновленнями, C# має можливість роботи з різними платформами та дає змогу розробляти програмне забезпечення ще швидше та ефективніше;

− строга типізація: C# є строго типізованою мовою, що дає змогу безпроблемно заглиблюватись в код, без витрат часу на з'ясування контексту у окремих частинах застосунку;

− управління пам'яттю: у C# реалізовано автоматичну систему збирання сміття (garbage collector), що вивільняє пам'ять, використану об'єктами, які більше не потрібні [4].

## 4.3 Фреймворк ASP.NET

ASP.NET Framework (Active Server Pages Network Enabled Technologies) – фреймворк, розроблений компанією Microsoft для створення бекенд-логіки застосунків, є частиною платформи .NET. Зазвичай, даний фреймворк використовується для розробки різноманітних вебзастосунків, хмарних сервісів або для API частини деяких мобільних застосунків.

ASP.NET часто обирають для розробки застосунків через низку переваг:

− швидкість: ASP.NET вважається високопродуктивним фреймворком, через мінімальні затримки, швидку обробку запитів та безпомилковість у роботі, що обумовлюється асинхронним та паралельним виконанням, що оптимізує життєвий цикл виконання програмного забезпечення;

− легкомасштабованість: ASP.NET підтримує комплексну архітектуру та гнучку інфраструктуру, через що масштабувати проєкт стає швидше і ефективніше;

− інтеграції: дана платформа дає можливість інтегрування з безліччю сервісів, має готові шаблони роботи, що спрощують роботу з базами даних, автентифікацією користувачів, платіжними системами тощо, що дозволяє розгортати застосунки легше через відсутність необхідності розробляти деякий функціонал з нуля;

− велика кількість бібліотек: через приналежність до платформи .NET при розробці на ASP.NET розробник має змогу використовувати неймовірно велику кількість бібліотек, що полегшують і покращують роботу;

− вичерпна документація: компанія-розробник ASP.NET Microsoft потурбувалась про майбутніх користувачів цього фреймворку, і тому на їхньому офіційному сайті [5] можна знайти базові гайди та відповіді на популярні питання.

## 4.4 Фреймворк Angular

Angular – один з найпопулярніших фреймворків для фронтенд-розробки, розроблений Google на основі мови програмування TypeScript.

Завдяки своїм можливостям: компонентна архітектура, підтримка односторінкових застосунків, двостороннє зв’язування даних, цей фреймворк гарантує високу продуктивність та гнучкість користувацького інтерфейсу (UI).

Angular часто обирають у парі з ASP.NET, що дає змогу реалізувати як клієнтську, так і серверну частину застосунку. Разом вони утворюють потужний інструмент для створення зручних у підтримці та масштабованих вебзастосунків. Angular відповідає за взаємодію з користувачем та підтягування вмісту сторінки без оновлень, що зменшує навантаження на серверну частину аплікації.

Дане поєднання технологій дозволяє ефективно використовувати ресурси системи, збільшуючи продуктивність застосунку та покращуючи користувацький досвід (user experience).

Також, варто відмітити додаткові плюси у співпраці Angular і ASP.NET:

− модульність: цей підхід використовує Angular, працюючи за схемою конструктора, де деякі частини коду можна перевикористовувати у різних частинах програми;

− зручність інтеграції: Microsoft потурбувався про взаємодію цих двох фреймворків разом, тому існує купа шаблонів проєктів, де Angular та ASP.NET вже налаштовані для спільної роботи;

− схожість мов програмування: як вже було зазначено раніше, Angular використовує мову програмування TypeScript, а ASP.NET використовує C#, що робить їх поєднання досить зручним, оскільки обидві мови є об’єктно-орієнтованими та C-подібними.

У виборі такого стеку для розробки однією з ключових переваг є спрощення найму персоналу у майбутньому. Наразі такий стек є досить поширеним, що робить пошук фахівця, що розуміється у Angular та ASP.NET набагато простішим, порівнюючи з менш популярними технологіями. Це, у свою чергу, пришвидшує пошук підбору кадрів, навчання та результативність роботи розробника, оскільки він має змогу орієнтуватись у всіх технологіях, що використовуються на проєкті, чим також пришвидшує time to market розробок [6].

## 4.5 РСУБД PostgreSQL

PostgreSQL – одна з найбільш використовуваних РСУБД (реляційна система управління базами даних) на ринку. Вона є легкою у використанні, активно підтримується та має можливість інтеграції з проєктами розробки. Ключовими характеристиками PostgreSQL є:

− інтеграція: .NET реалізує можливість підключення до PostgreSQL та має змогу працювати з базами даних через код застосунку;

− відкритий код: PostgreSQL є абсолютно безкоштовною, що дозволяє використовувати всі її функції без обмежень;

− підтримка ACID (Atomicity, Consistency, Isolation, and Durability) для транзакцій;

− надійсність: PostgreSQL містить функціонал для реплікації, резервації даних та їх відновлення;

− відновлювальність: дана СУБД має вбудовані інструменти для моніторингу збоїв та відновлення після них;

− широка спільнота: PostgreSQL має невичерпний спектр використання, і тому має широку, активну спільноту, що створює документацію, або дає відповіді на запитання по напрямку на різноманітних форумах, що полегшує і пришвидшує роботу [7].

## 4.6 Фреймворк Entity Framework

Для взаємодії між серверною частиною застосунку та базою даних буде використано бібліотеку Entity Framework. Ця бібліотека є ORM (Object-relational Mapper), що є з’єднувальною частиною між програмуванням та реляційними базами даних. Дякуючи Entity Framework можна реалізувати зв’язок між базою даних і застосунком за допомогою класів та об’єктів, що відображають структуру БД. Це полегшує роботу розробнику, який може легко почати використовувати інтеграцію

з джерелом даних у себе в проєкті без зайвих дій та критичної зміни коду застосунку. Основними перевагами Entity Framework є:

− використання LINQ: EF інтегрується з LINQ (Language Integrated Query), що дозволяє розробникам використовувати запити LINQ для взаємодії з даними бази даних, замість використання SQL запитів;

− об’єктно-реляційне відображення (ORM): EF має змогу автоматично перетворювати об’єкти в таблиці бази даних та навпаки, завдяки чому розробник може працювати з даними як з об’єктами, а не рядками у таблицях;

− можливість створення моделі даних: EF містить функціонал для генерації БД по наявній моделі з коду (Code First), або для генерації класів по наяних таблицях у базі даних (Database First), що дозволяє пришвидшити процес інтеграції зберігання даних на початку розробки;

− сумісність: EF забезпечує стабільну роботу з широким спектром СУБД, зокрема Microsoft SQL Server, Oracle, MySQL, PostgreSQL тощо.

Отже, Entity Framework є незамінною технологією у застосунку, що використовує бази даних, оскільки на сьогодні це найзручніша бібліотека для C#, яка забезпечує стабільну роботу та інтеграцію БД з програмним кодом.

## 4.7 Середовище розробки Visual Studio

Оскільки застосунок буде розроблено за допомогою фреймворків ASP.NET та Angular, доцільно використовувати Visual Studio для розробки серверної частини та Visual Studio Code для розробки клієнтської частини.

Visual Studio – це середовище розробки (IDE), що було створено компанією Microsoft для мов програмування C, C++, C#, JavaScript, TypeScript та інших. Наразі, Visual Studio – потужний інструмент для створення різноманітного програмного забезпечення – від вебзастосунків та десктопних застосунків до хмарних сервісів.

Visual Studio підтримує широкий спектр технологій, що можна застосовувати у розробці, наприклад .NET, ASP.NET, Xamarin, Unity тощо. Також ця IDE підтримує інтеграцію з Azure, що значно спрощує процес розробки, розгортання у хмарі та керування хмарними застосунками. Важливою є підтримка систем контролю версій GitHub та GitLab, що дає змогу легко керувати версійністю коду безпосередньо з середовища розробки.

Visual Studio надає можливість зручного відлагодження коду, що реалізована в функціоналі IDE. Користувач має можливість легко налаштувати точки зупинки (breakpoints) та переглядати результати виконання через дебаг. Для покрокового виконання реалізовано кроки step over, step into, step out для керування процесом відлагодження. Реалізована змога переглянути стан змінних, які є активними в поточному контексті, також можна переглянути значення змінної навівши на неї курсором. Важливо підмітити, що під час процесу відлагодження також є можливість змінювати код, не зупиняючи при цьому роботу програми, що може бути особливо корисним у випадках коли потрібно терміново змінити логіку, виправити баг, або перевірити альтернативні сценарії виконання.

Також, дана IDE має обширні ресурси для автоматизованого тестування, як для написання unit тестів, так і для створення автоматизаційних фреймворків з UI та API тестуванням. Visual Studio автоматично знаходить тести у програмі та реалізує їх групування, паралелізацію та фільтрування.

Для підтримки написання коду Visual Studio має вбудований інструмент IntelliSense, що автоматично доповнює код, показує опис та призначення параметрів, підсвічує синтаксис, має змогу перевіряти помилки у реальному часі, дає вичерпні підказки у розробці. Прискорює розробку програмного забезпечення, робить код більш чистим та чітким та допомагає уникнути синтаксичних помилок.

## 4.8 Середовище розробки Visual Studio Code

Visual Studio Code – це редактор коду, створений Microsoft.

На відміну від Visual Studio, VS Code є саме редактором коду і має невеликий функціонал, що можна збільшити встановленням розширень.

Завдяки такому підходу, VSC працює з безліччю мов програмування, такими як C#, Java, JavaScript, TypeScript, Python тощо. Базовий функціонал включає перевірку синтаксису, автодоповнення, відлагодження коду, інтеграцію з системами контролю версій та IntelliSense, проте завдяки гнучкій системі розширень з VSC можна розбудувати повноцінне середовище розробки, що відповідатиме потребам конкретного проєкту. Також, за рахунок extensions у Visual Studio Code користувач має можливість інтегруватись з багатьма сервісами, що значно спрощує роботу з кодом.

Описані вище характеристики роблять Visual Studio Code повноцінним та потужним середовищем розробки, який дозволяє легко та швидко втілити бажання розробника у реальність, не завдаючи зайвого клопоту з створенням ефективного та гнучкого робочого простору. Завдяки зрозумілому інтерфейсу, високій продуктивності та широким спектром налаштувань цей редактор коду може стати в нагоді будь-якому спеціалісту – від junior до senior-рівня.

Visual Studio Code обрано використовувати для розроблення клієнтської частини застосунку. Це одне з найефективніших середовищ розробки, через свою невелику функціональність при встановленні та можливість гнучкого і легкого розширення. За допомогою незліченної кількості бібліотек та фреймворків, які підключаються до проєкту створення застосунку стає зручним та швидким, і робить фінальне програмне забезпечення якісним та стійким.

## 4.9 Підходи до розробки

У розробці програмного забезпечення найголовніше зберігати баланс кількості та якості коду. Код має бути першочергово зрозумілий для інших людей, оскільки при роботі в команді його читають, змінюють або розширюють всі співробіт-

ники, а робити його важким та навантаженим лише збільшить витрату часу на поставлені задачі. Також варто дотримуватись написання чистого коду: логічної структури, читабельності, легкій підтримці та розширенню.

Для цього у спільноті розробників були створені ключові принципи програмування: SOLID, DRY, KISS, YAGNI. Все це абревіатури, що чітко і швидко описують правила створення програм:

– принципи SOLID містять у собі п’ять важливих вказівок, які вдосконалюють програмне забезпечення, вони включають у собі Single Responsibility Principle (принцип єдиної відповідальності), Open/Closed Principle (принцип відкритості для розширення, але закритості для модифікації), Liskov's Substitution Principle, Interface Segregation Principle (принцип сегрегації інтерфейсів), Dependency Inversion Principle (принцип інверсії залежностей);

– DRY (don’t repeat yourself) – принцип, що визначає зменшення повторів у коді, заміняючи їх на абстракції;

– KISS (keep it simple, stupid) – принцип, що вимагає створення коду максимально простим та зрозумілим, задля кращої продуктивності програми та легкого розширення у майбутньому;

– YAGNI (you ain’t gonna need it) – принцип, який пропонує не реалізовувати ті функції, що не потрібні на даний момент розробки, аргументуючи тим, що, можливо, у майбутньому, вони і не знадобляться.

Для створення програмного забезпечення у рамках даної роботи було вирішено підпорядковуватись описаним вище принципам задля спрощення коду, підвищення продуктивності застосунку та кращої розширюваності продукту у майбутній дорозробці.

Висновки до розділу 4

В даному розділі було обрано технології, що будуть використовуватись для розроблення майбутнього вебзастосунку.

Також описано критерії підбору для кожної з частин стеку розробки та підходи до розробки загалом.

Для розроблення вебзастосунку потрібно реалізувати серверну та клієнтську частини, також вимагається використання БД для зберігання даних, що генеруватимуться у застосунку. Отже, для написання серверної частини було обрано мову програмування C# та фреймворк ASP.NET. Для написання клієнтської частини було обрано фреймворк Angular з мовою програмування TypeScript. У якості СУБД буде використано PostgreSQL з Entity Framework для її інтеграції у проєкт.

Всі обрані технології є актуальними та важливими у контексті створюваної системи, завдяки чому дають можливість реалізувати потрібний функціонал максимально точно, швидко та зручно.

# 5 РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКУ

## 5.1 Структура системи

Як зазначалось у розділі 4, для реалізації серверної частини застосунку було обрано мову програмування C# та фреймворк ASP.NET. Backend створеної програми був написаний у розробницькому середовищі Visual Studio. На рисунку 5.1 зображено структуру системи відповідно до обраного шаблону тришарової архітектури [8].

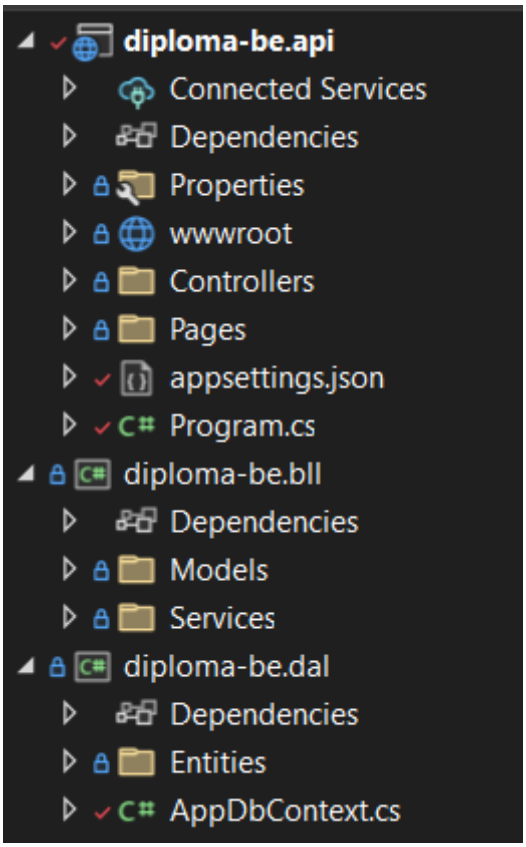


Рисунок 5.1 – Структура серверної частини

Тут реалізовано три шари взаємодії застосунку з користувачем: DAL, BLL та API, що дали змогу ефективно розділити логіку для більш зручного розширення проєкту та його підтримки. Саме на цьому рівні застосунку також реалізована інтеграція з БД, оскільки саме тут є необхідність використовувати та створювати дані для стабільної роботи застосунку.

## 5.2 Архітектура системи

### 5.2.1 Архітектура серверної частини

У випадку створюваного вебзастосунку було обрано керуватись підходом тришарової архітектури. Таким чином, застосунок розділяється на три частини: Business Logic Layer (BLL), Data Access Layer (DAL), Application Programming Interface (API). Розгляньмо кожен з цих шарів детальніше:

– DAL – це шар доступу до даних, він відповідає за їх цілісність, безпечне збереження та зміну, також, у цьому рівні ізолюється взаємодія з базою даних, що дозволяє за потреби змінювати СУБД, не змінюючи при цьому інші шари застосунку;

– BLL – це шар бізнес-логіки, який відповідає за обробку даних, отриманих з шару доступу до даних (DAL), тут реалізуються правила та алгоритми, що керують роботою програми відповідно до вимог: на цьому рівні відбуваються всі важливі перевірки даних, виконуються обчислення, прийняття рішень згідно до умов системи, після чого передаються на наступний шар застосунку (API).

– API – це шар взаємодії клієнта з системою, цей шар спілкується з іншими сервісами, приймає запити через користувацький інтерфейс, і за допомогою інших шарів обробляє їх та повертає результат, що робить API це інструмент для швидкої та зрозумілої комунікації всіх частин системи, забезпечує безпеку бази даних та стандартизацію даних, якими обмінюється застосунок.

Цей підхід було обрано для розробки через легкість імплементації та розмежування різних процесів у застосунку, що робить створення програми швидшим і зручнішим розуміння. Також, завдяки наведеним вище перевагам, розширення функціоналу в майбутньому не потребує значних змін у реалізовану коді, що суттєво спрощує підтримку системи.

Структура розробленої тришарової архітектури зображена на рисунку 5.1.

### 5.2.2 Комунікація системи

Для комунікації з сервером використовують API, що дає можливість створювати та відправляти запити, отримувати відповідь на них. Для будування application programming interface застосунку існують різні архітектурні стилі та протоколи, що мають своє застосування у різних випадках. Найбільш популярним є REST API (Representational State Transfer), оскільки є легким в імплементації, безпечним та швидким у підтримці стилем будування застосунків, що дає змогу створити API застосунку точним та простим для розуміння. Також на ринку представлені і інші види архітектурних стилів та протоколів, такі як:

− webhooks (вебперехоплювач) – механізм, у якому одна система надсилає повідомлення іншій, якщо спрацьовує якийсь тригер, цей підхід часто знаходить застосування у системах, що потребують автоматичної відповіді серверу: платіжні системи, git-платформи, інтеграції з зовнішніми сервісами, системи моніторингу тощо;

− SOAP – протокол, що є заснованим на текстовому форматі XML, комунікація SOAP з сервісами відбувається структурою XML, що включає в собі envelope (конверт), header (заголовок), body (тіло запиту), fault (інформація про помилку, має місце лише при невдачі у виконанні запиту), через високу безпеку передачі даних використовується найчастіше у банківських сервісах або в суміжних критичних процесах;

− GraphQL – мова запитів до API, що дозволяє отримати потрібні дані через один запит, на відміну від REST, у якому в деяких випадках потрібно робити кілька запитів для отримання бажаного результату, через що знаходить своє використання у сервісах, де потрібно мінімізувати кількість запитів;

− WebSocket – протокол, що гарантує безперервне з’єднання між клієнтом та сервером, має низьку затримку в надсиланні та отриманні запитів та підтримує постійний двосторонній обмін даними, застосовується у месенджерах, онлайн-іграх, системах, де потрібно стабільне з’єднання між кількома користувачами.

Для розробки програми у рамках дипломної роботи було обрано використовувати саме REST-архітектуру, оскільки вона є простою у реалізації, зручною у використанні та безпечною для передачі даних.

REST використовує звичайні HTTP-методи, такі як GET, POST, PUT, PATCH, DELETE. Ці методи є стандартизованими, що забезпечує уніфікований підхід до розробки та взаємодії між сервісами.

Також, REST-архітектура базується на концепції ресурсів, де кожен має свою окрему URL-адресу. Таким чином кожен ендпоінт забезпечує доступ до окремих ресурсів. Розгляньмо кілька прикладів такої взаємодії для кращого розуміння цих процесів:

- GET app-link.com/products – повертає список продуктів;
- GET app-link.com/products/1 – повертає окремий продукт;
- POST app-link.com/products – створює сутність продукту (для цього методу також необхідно мати тіло запиту (body) для конфігурації даних створюваної сутності).

Найчастіше у REST-архітектурі використовують п’ять методів: GET (отримати дані про сутності), POST (створити сутність), PUT (оновити сутність повністю), PATCH (повністю оновити сутність), DELETE (видалити сутність). Також там є вичерпна кількість інших запитів, але вони не набули такого широкого застосування, як ці п’ять [9].

Для розроблюваної системи було створено шість контролерів. Кожен з них відповідає за роботу з HTTP-запитами, пов’язаних з окремим типом даних. Контролери працюють з функціями адміністратора, спеціаліста, клієнта, також з функціоналом авторизації, пошуку та мають допоміжного контролера. Контролери забезпечують чітке відпрацювання запитів та передачу отриманих даних на інші рівні системи. Їхнє успішне відпрацювання є критичним для налагодженої роботи застосунку, комунікаціями з БД та сервісами. На рисунку 5.2 зображено вміст проєкту API.

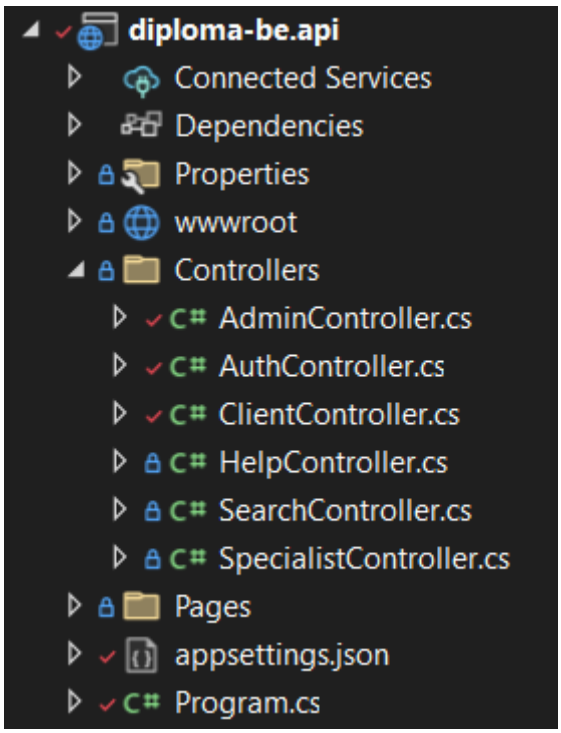


Рисунок 5.2 – Вміст проєкту API

### 5.2.3 Загальна архітектура

Розроблена система побудована за моделлю клієнт-сервер. Головною суттю цієї моделі є чітке розділення задач між клієнтською частиною застосунку та серверною.

Клієнтська частина є інтерфейсом користувача, що перетворює складні машинні процеси на простий візуал, з яким може працювати пересічна людина [10]. Сервер у свою чергу, є прихованим для звичайних користувачів, реалізує логіку системи, спілкується з базою даних, оброблює дані та повертає результат. Як правило, у системах з клієнт-серверною архітектурою вся бізнес-логіка знаходиться на серверній частині застосунку, а на клієнтській частині реалізується лише взаємодія користувача з сервером та візуальні елементи.

На кресленику ІС11.290.005 Д3 зображено діаграму комунікації клієнта з сервером, яка є реалізованою в створеній системі та забезпечує якісну та безперебійну комунікацію між всіма частинами застосунку.

5.3 Функціональна модель системи

Створена система містить реалізацію для декількох типів користувачів: для клієнта, спеціаліста та адміністратора. Ці ролі розглянуті детальніше в таблиці 5.1.

Таблиця 5.1 – Ролі у створюваній системі

| № | Роль | Опис | Права |
|---|---|---|---|
| 1 | Клієнт | Неавторизований користувач | Перегляд усіх спеціалістів на платформі, пошук по-фільтрації, проходження опитувальника, перегляд рекомендованих спеціалістів, запис до спеціаліста. |
| 2 | Спеціаліст | Авторизований користувач, обліковий запис створюється адміністратором | Авторизація, перегляд запитів на запис на сеанс, їх редагування, редагування свого профілю |
| 3 | Адміністратор | Авторизований користувач з максимальними правами | Авторизація, перегляд запитів на запис на сеанс, їх редагування, редагування профілю спеціаліста, додавання спеціаліста, видалення спеціаліста |

Було вирішено лишити три ролі користувачів через те, що вони найкраще відповідають вимогам розробленого вебзастосунку та реалізовують всі поставлені цілі. При аналізі потенційних ролей для більшого розділення прав було виявлено, що у контексті створеної системи це не буде доцільним. У майбутньому, у разі розширення функціональності, конфігурація ролей може бути розглянута знову та

удосконалена. Однак, для поточного стану системи, цього є цілком достатньо для повного та коректного функціонування.

Таким чином всі ролі в системі є виправданими, оскільки відображають реальні сценарії взаємодії користувачів з програмним забезпеченням та дозволяють зберегти функції застосунку ефективними та зручними у використанні [11].

## 5.4 Модель бази даних

Для коректного функціонування системи важливим фактором стало проєктування стабільної та простої для покращення бази даних. Саме на ній лежить основне навантаження, оскільки використання даних є критичним флоу застосунків такого типу. Правильне формування бази даних дає можливість програмному забезпеченню функціонувати швидко та узгоджено, з мінімальною кількістю проблемних моментів.

Створена база даних містить чотири таблиці: clients, users, specialists та appointments. Їх невелика кількість, завдяки чому робота з БД є швидкою та доступною. У таблиці 5.2 описані поля по кожній таблиці бази даних.

Таблиця 5.2 – Структура бази даних, що інтегрується з створюваною системою

| № | Назва таблиці | Поля |
|---|---|---|
| 1 | clients | ідентифікатор клієнта, ідентифікатор користувача (з таблиці users), список запитів клієнта та поля для опитувальника: бюджет, онлайн/офлайн робота, стать спеціаліста, мова, якою має бути обслуговування, час створення |
| 2 | users | ідентифікатор користувача, ім’я, прізвище, email користувача, номер телефону користувача, пароль, роль, час створення |

Продовження таблиці 5.2

| № | Назва таблиці | Поля |
|---|---|---|
| 3 | specialists | ідентифікатор спеціаліста, ідентифікатор користувача (з таблиці users), освіта, досвід, спеціалізація, ціна за сесію, онлайн/офлайн робота, стать, мова обслуговування, доступність, час створення |
| 4 | appointments | ідентифікатор запису, ідентифікатор клієнта (з таблиці clients), сутність клієнта (з таблиці clients), ідентифікатор спеціаліста (з таблиці specialists), сутність спеціаліста (з таблиці specialists), дата запису, статус запису, додаткові уточнення по запису, час створення |

На кресленику ІС11.290БАК.005 Д4 зображена ER-діаграма.

Для правильної комунікації бази даних та застосунку програмно було реалізовано шар доступу до даних (DAL) [12]. На рисунку 5.3 зображена структура проєкту DAL.

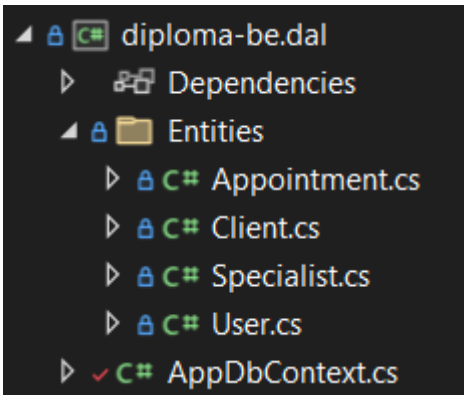


Рисунок 5.3 – Структура проєкту DAL

У цьому проєкті реалізовані моделі даних, які застосовуються у розробленому застосунку. Через те, що Entity Framework взаємодіє з базою даних напряму

досягається безпека даних, зменшується кількість помилок та автоматизується процес роботи з даними. Також, у майбутньому це може позитивно вплинути на подальше розширення та підтримку продукту.

У DAL в сутності User було реалізовано поля, які користувач має заповнити для підбору спеціаліста у ході проходження опитувальника. Ці дані у майбутньому оброблятимуться математичним алгоритмом. Детальніше про них йтиметься мова у наступному розділі.

Структура цих моделей даних є класифікованою. Класи репрезентують окремі таблиці у базі даних, і кожне поле реалізує стовпець цієї таблиці. На рисунку 5.4 наведено модель User, що відповідає за основну інформацію користувача для створення профілю, його перегляду та авторизації та містить поля Id, FirstName, LastName, Email, Phone, PasswordHash, Role, CreatedAt.

```
24 references
public class User
{
    18 references
    public Guid Id { get; set; } = Guid.NewGuid();

    [Required]
    [MaxLength(100)]
    38 references
    public string FirstName { get; set; } = string.Empty;

    [Required]
    [MaxLength(100)]
    38 references
    public string LastName { get; set; } = string.Empty;

    [Required]
    [EmailAddress]
    35 references
    public string Email { get; set; } = string.Empty;

    21 references
    public string Phone { get; set; } = string.Empty;

    [Required]
    20 references
    public string PasswordHash { get; set; } = string.Empty;

    [Required]
    22 references
    public string Role { get; set; } = string.Empty;

    11 references
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
}
```

Рисунок 5.4 – Структура класу User

## 5.5 Передавання та обробка даних

У розробленому застосунку велика увага наділяється даним, їх отриманню, створенню та обробці. Як було описано раніше, для цього використовується REST API з базовими HTTP-запитами. Всі перевірки даних застосовуються на серверній частині застосунку.

Вхідними даними вважаються ті, що користувач створив сам, тим самим направив запит до серверу і очікує відповіді від нього. У створеній системі вхідними даними є:

- дані для авторизації (логін та пароль користувача);
- дані, що вводяться клієнтом під час проходження опитувальника;
- дані, що вводяться клієнтом при записі на сеанс;
- дані, які створює про себе спеціаліст при формуванні профілю;
- дані про спеціаліста, які редагуються спеціалістом або адміністратором;
- дані про спеціаліста, які формує адміністратор при створенні профілю спеціаліста.

У свою чергу, після введення даних, система має перевірити їх задля безпеки застосунку та БД. Процес обробки даних включає в себе валідацію полів, у які користувач має можливість ввести якісь значення (наприклад, ім’я, номер телефону або електронна адреса), обробка пошуку по фільтрації, обробка алгоритмів, що застосовуються у програмному забезпеченні, передача даних до місця їх збереження. Також в цей момент формується результат створеного користувачем запиту задля поверненню відповіді на проведені дії.

Вихідними даними у даному випадку є:

- результат запиту (успіх або невдача);
- повернення оброблених даних (у розрізі створюваної роботи – результат проходження опитувальника, список рекомендованих спеціалістів);
- повернення на доробку (коли якась з валідацій не пройшла, і користувач має можливість переробити ввід інформації без втрати попередніх записів).

Дані, що передаються та обробляються, постійно рухаються між різними рівнями застосунку. Найперше вони потрапляють на клієнтську частину, де користувач створює дані. Далі вони переходять до бекенду, а саме до API. Там, дані з запиту оброблюються BLL-шаром, де міститься вся бізнес-логіка програмного забезпечення, методи та алгоритми, за допомогою яких система відпрацьовує швидко та точно. Business Logic Layer у своїй реалізації використовує шар DAL для доступу до інформації бази даних, можливості створення нових записів або зміни існуючих. Після обробки даних та комунікації з БД, бізнес-логіка (BLL-рівень) формує результат запиту. Цей результат повертається назад до контролерів на API-рівень. Наприкінці користувач отримує результат запиту на користувацькому інтерфейсі.

У цьому процесі важливу роль відіграє BLL-рівень застосунку, оскільки саме він обробляє отримані дані, виконує різноманітні обчислення, фільтрації та перевірки та бере відповідальність за позитивне проходження всіх сценаріїв використання програмного забезпечення. У BLL реалізується вся логіка застосунку: відбуваються математичні підрахунки, виконуються основні перевірки даних, приймаються рішення відповідно до результатів виконання алгоритмів. BLL є посередником між API та DAL рівнями, що робить його найвідповідальнішою частиною застосунку.

На рисунку 5.5 зображена структура проєкту BLL розробленої системи.

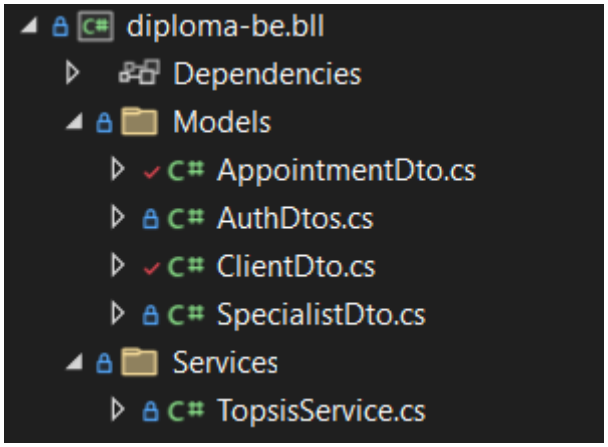


Рисунок 5.5 – Структура проєкту BLL

У проєкті бізнес-логіки було реалізовано один сервіс – TopsisService, оскільки він несе основну відповідальність за функціональність застосунку. Інша логіка застосунку реалізована у контролерах у шарі API. Така структура менше обтяжує створений код, дає можливість пришвидшити роботу запитів та збільшити точність їх відпрацювання.

Для доступу до даних було реалізовано моделі даних (DTO). Не дивлячись на те, що в рівні доступу до даних вже були реалізовані моделі даних, було вирішено створити DTO на рівні BLL через кращу оптимізацію обміну даними та їх більшу безпеку. Також DTO допомагають приховати реалізацію внутрішньої структури бази даних від зовнішніх шарів системи. Створені моделі даних відрізняються від створених раніше у рівні DAL задля оптимізованої роботи та ефективного користування даними.

Висновки до розділу 5

В цьому розділі було розглянуто загальну архітектуру створеного програмного забезпечення. Обґрунтовано структуру проєкту, коротко описано загальні процеси у застосунку. Також, було розглянуто існуючі архітектурні стилі та було описано обраний стиль, його переваги в контексті розробленої системи.

Окрему увагу приділено реалізації багаторівневої архітектури, а саме трьох шарів DAL, BLL, API та обґрунтовано їх використання для ефективної роботи та відповідності поставленим вимогам. Також було описано структуру збереження даних, а саме використані таблиці у базі даних та відповідних до них сутності в проєкті.

Всі описані процеси формують стабільне програмне забезпечення, що відповідає поставленим вимогам і створює надійну основу для подальшого розширення та підтримки функціональності.

# 6 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

## 6.1 Змістовна постановка задачі

Для вирішення проблеми вибору спеціаліста потрібно підібрати критерії підбору. Було виділено наступні критерії:

- ціна;
- кількість запитів, що збігаються у спеціаліста з потенційним клієнтом;
- доступність працювати онлайн/офлайн;
- стать спеціаліста;
- мова, якою надаються послуги.

Визначивши ці критерії, потрібно визначити їх ваги та реалізувати алгоритм, що забезпечить оптимальний підбір терапевта. Для цього використовуватиметься матриця попарних рішень, оскільки вона є простим та зручним способом визначення пріоритетів підбору та оцінок критеріїв.

Найважливішим критерієм підбору є кількість запитів, що збігаються у спеціаліста та клієнта – користувач обирає запити, які він очікує вирішити за допомогою терапії і алгоритм порівнює їх з запитами, які вирішує терапевт. Також не мало важливими є критерії ціни, способу роботи онлайн або офлайн, статі спеціаліста та мови, якою надаються послуги.

## 6.2 Математична постановка задачі

Дано:

– $U = \{u_y\}, y = 1, \dots l$ – множина клієнтів системи;

$= \{p_k\}, k = 1, \dots n$ – множина спеціалістів;

$= \{c_i\}, i = 1, \dots m$ – множина критеріїв, за якими здійснюється оцінка спеціалістів;

$= \{v_{iky}\}, i = 1, \dots m, k = 1, \dots n,\ y = 1, .. l$ – множина оцінок, що відображають значення критеріїв C для спеціалістів P;

– матриця попарних порівнянь критеріїв.

Метою є знайти для кожного користувача U спеціаліста P, що найбільше відповідає поставленим критеріям.

### 6.3 Обґрунтування методів розв’язання

#### 6.3.1 Оцінка ваг критеріїв

У створеній системі важливим етапом є визначення ваг критеріїв, оскільки ця частина є фундаментальною для коректного відпрацювання алгоритму та подальшого підбору спеціаліста. Для цього варто скласти матрицю попарних порівнянь, що містить оцінки кожного з критеріїв відносно загального скоупу поставленої задачі. Для її формування потрібно:

− визначити критерії підбору, які у подальшому будуть оцінюватись, у випадку системи підбору психолога це ціна, кількість збігів у запитах клієнта та спеціаліста, мова, формат роботи (дистанційно/офлайн), стать;

− формування початкової матриці розмірності $n \times n$, у якій порівнюються критерії, у випадку створеної системи матриця буде мати розмірність $5 \times 5$ з одиницями на головній діагоналі, оскільки це порівняння критерія з самим собою;

− заповнення матриці за бізнес-вимогами;

− обрахунок ваг критеріїв за допомогою нормалізації та обрахунку вектору ваг критеріїв.

Ці обрахунки мають місце лише при професійній оцінці критеріїв за біснес-вимогами та потребами користувачів задля забезпечення об’єктивного та аргументованого відпрацювання алгоритму. При правильній розстановці ваг цих критеріїв система здатна забезпечити максимально релевантний результат для кінцевих користувачів. Таким чином розроблений застосунок матиме змогу відповідати потребам та очікуванням клієнтів та забезпечити оптимальну відповідність між потребою і пропозицією.

### 6.3.1 Підбір спеціаліста

Для формування ваг критеріїв було обрано використовувати метод аналізу ієрархій (МАІ, англ. AHP – analytic hierarchy process). Цей інструмент дає змогу системно підійти до проблеми прийняття рішень, зрозумілим і раціональним чином структурувати складну проблему у вигляді ієрархії, порівняти і виконати кількісну оцінку альтернативних варіантів вибору [13].

Основною ідеєю цього методу є побудова ієрархічної структури, що представляє проблему прийняття рішення у вигляді дерева. Це дерево має визначену структуру: вище всіх знаходиться мета, далі критерії, що впливають на досягнення мети, ще нижче базуються підкритерії (якщо вони мають зміст у досліджуваному контексті), а найнижче знаходяться альтернативи, тобто можливі варіанти рішень. Метод аналізу ієрархій є унікальним через свій спектр застосування: він може знайти своє місце у багатьох галузях, де потрібно прийняти рішення – від програмування до медицини. Це досягається завдяки використанню як об’єктивних так і суб’єктивних чинників впливу на поставлену задачу.

Для оцінки ваг критеріїв методом аналізу ієрархій потрібно виконати кілька кроків, серед яких: створення матриці попарних порівнянь, базуючись на шкалі Сааті, обрахувати геометричне середнє для кожного критерія, після чого нормалізувати матрицю і даний результат буде шуканою оцінкою ваг критеріїв, після чого можна приступати до підбору спеціаліста.

Для розв’язання поставленої задачі підбору спеціаліста було обрано використовувати багатокритеріальний метод прийняття рішень (TOPSIS), що дає змогу обрати найкращу альтернативу для поставленої задачі. TOPSIS базується на фундаментальній передумові, що найкращий вибір має найменшу відстань від позитивно-ідеального розв’язку та найбільшу відстань від негативно-ідеального [14].

TOPSIS був обраний для використання у контексті розробленої системи через низку причин:

− простота реалізації: TOPSIS не вимагає використовувати складні математичні обрахунки та через це легко інтегрується у будь-який застосунок;

− урахування ваги критеріїв: як було описано раніше, для підбору спеціаліста ваги критеріїв не є рівномірними, оскільки деякі з них є важливішими, а деякі можна упустити, в цьому випадку обраний метод повністю відповідає поставленим вимогам, оскільки дає змогу враховувати ці ваги, що визначаються за допомогою матриці попарних рішень;

− ранжування альтернатив: за допомогою TOPSIS можливо отримати не лише найкращий результат підбору, а й визначити повне впорядкування спеціалістів за шуканими критеріями.

## 6.4 Опис методу розв’язання

### 6.4.1 Оцінка ваг критеріїв

Для оцінки ваг критеріїв було проаналізовано обрані критерії, розставлено їх пріоритет та створено матрицю попарних порівнянь (МПП). Для оцінки використовувалась шкала Сааті, значення якої описані на рисунку 6.1.

| Ступінь значимості | Визначення | Пояснення |
|---|---|---|
| 1 | Однакова значимість | Дві дії мають однаковий внесок у досягнення мети |
| 3 | Слабка значимість | Існують не достатньо переконливі міркування на користь переваги однієї з дій |
| 5 | Істотна значимість | Маються надійні дані для того, щоб показати перевагу однієї з дій |
| 7 | Очевидна значимість | Переконливе свідчення на користь однієї дії перед іншою |
| 9 | Абсолютна значимість | Незаперечні переконливі свідчення на користь переваги однієї дії перед іншою |
| 2,4,6,8 | Проміжні значення між сусідніми судженнями | Ситуація, коли необхідне компромісне рішення |

Рисунок 6.1 – Шкала Сааті [13]

У матриці використовувались такі критерії:

− кількість збігів у запитах – K1, є найважливішим критерієм, тому має найвище значення відносно інших;

− ціна – К2, є другим по важливості критерієм, тому йому надають перевагу лише після найважливішого критерія;

− формат співпраці (онлайн/офлайн) – К3;

− стать спеціаліста – К4;

− мова обслуговування – К5.

К3, К4, К5 є найменш важливими критеріями, тому мають найнижчий пріоритет у підборі.

У таблиці 6.1 зображена створена МПП за правилами створення таких матриць та базуючись на обраних критеріях.

Таблиця 6.1 – МПП

| | К1 | К2 | К3 | К4 | К5 |
|---|---|---|---|---|---|
| К1 | 1 | 5 | 4 | 6 | 5 |
| К2 | 1/5 | 1 | 2 | 3 | 2 |
| К3 | 1/4 | 1/2 | 1 | 2 | 2 |
| К4 | 1/6 | 1/3 | 1/2 | 1 | 1 |
| К5 | 1/5 | 1/2 | 1/2 | 1 | 1 |

Після заповнення матриці попарних порівнянь потрібно обрахувати геометричне середнє $v_i$ для кожного критерію ($i = 1 \dots n, n -$ кількість критеріїв) за формулою

$$v_i = \sqrt[n]{\prod_{j=1}^{n} a_{ij}} \tag{6.1}$$

де $a_{ij}$ – вхідне значення матриці з індексом [i,j]

Обраховане геометричне середнє описане у таблиці 6.2.

Таблиця 6.2 – Геометричне середнє для кожного критерія

| | |
|---|---|
| K1 | 3.245 |
| K2 | 1.189 |
| K3 | 0.871 |
| K4 | 0.505 |
| K5 | 0.549 |

Після обрахування геометричного середнього обраховується нормалізація за формулою

$$w_i = \frac{v_i}{\sum_{j=1}^{n} v_j} \tag{6.2}$$

Результатом даного обрахунку є оцінка ваг критеріїв, що відображає важливість кожного з критеріїв у процесі прийняття рішення. Надалі, отримавши ці дані можна використовувати їх у алгоритмі прийняття рішення, зокрема у обраному методі TOPSIS.

Чіткий підрахунок на кожному етапі пошуку ваг критеріїв формують якісний пошук оптимального результату за визначеними критеріями у майбутніх розрахунках.

У таблиці 6.3 зображено результат нормалізації матриці.

Таблиця 6.3 – Оцінка ваг критеріїв

| | |
|---|---|
| K1 | 0.511 |
| K2 | 0.187 |
| K3 | 0.137 |
| K4 | 0.079 |
| K5 | 0.086 |

6.4.2 Підбір спеціаліста

Для наглядності алгоритму було створено тестові дані, за допомогою яких буде шукатись оптимальний спеціаліст для клієнта по його запитам. У таблиці 6.4 описані створені тестові дані.

Таблиця 6.4 – Тестові дані

| Ім’я | User1 (клієнт) | User2 (спеціаліст) | User3 (спеціаліст) | User4 (спеціаліст) |
|---|---|---|---|---|
| Запити | Тривожність, низька самооцінка | Депресія, тривожність, фобії | Депресія, нав’язливі думки, панічні атаки | Тривожність, низька самооцінка, криза ідентичності |
| Онлайн/ офлайн робота | Онлайн | Онлайн + офлайн | Офлайн | Онлайн + офлайн |
| Стать | Жіноча | Жіноча | Жіноча | Чоловіча |
| Мова обслуговування | Українська | Українська, англійська | Українська, польська | Українська |

На основі тестових даних було створено матрицю значень, за допомогою якої будуть проводитись обрахунки. У таблиці 6.5 описано результат формування матриці значень.

Таблиця 6.5 – Сформована матриця значень

| Кількість збігів у запитах | Ціна | Онлайн/ офлайн робота | Стать | Мова обслуговування |
|---|---|---|---|---|
| 1 | 1300 | 1 | 1 | 1 |
| 0 | 900 | 0 | 1 | 1 |
| 2 | 2000 | 1 | 0 | 1 |

У попередньому підрозділі було обраховано ваги критеріїв, тож для цього обрахунку скористаємось ними. Як було вказано раніше, для обрахунку пріоритетності спеціалістів обрано алгоритм TOPSIS. Даний алгоритм складається з кількох етапів.

Першим етапом є обчислення нормалізованих оцінок альтернатив за формулою

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}}, \text{де } i = 1 \dots m, j = 1 \dots n \quad (6.3)$$

Результат цього обчислення зображено у таблиці 6.6.

Таблиця 6.6 – Результат нормалізації оцінок

| | Збіги | Ціна | Формат | Стать | Мова |
|---|---|---|---|---|---|
| U2 | 0.447 | 0.51 | 0.707 | 0.707 | 0.577 |
| U3 | 0 | 0 | 0 | 0.707 | 0.577 |
| U4 | 0.894 | 0.707 | 0.707 | 0 | 0.577 |

Наступним кроком йде оцінка зважених нормалізованих оцінок альтернатив за формулою

$$v_{ij} = w_j \cdot r_{ij}, \text{де } i = 1 \dots m, j = 1 \dots n \quad (6.4)$$

У таблиці 6.7 зображено результат другого обчислення.

Таблиця 6.7 – Оцінка зважених нормалізованих альтернатив

| | Збіги | Ціна | Формат | Стать | Мова |
|---|---|---|---|---|---|
| U2 | 0.2286 | 0.0954 | 0.0969 | 0.0559 | 0.0496 |
| U3 | 0 | 0.066 | 0 | 0.0559 | 0. 0496 |
| U4 | 0.4569 | 0.1466 | 0. 0969 | 0 | 0. 0496 |

Далі потрібно обчислити ідеальний і неідеальний розв’язки за формулами

$$PIS = A^{+} = \{(max_j v_{ij}(x) |\ i \in C^{+}), (min_j v_{ij}(x) | i \in C^{-}) |\ j = 1, \dots, n\} \quad (6.5)$$

$$NIS = A^{-} = \{(min_j v_{ij}(x) |\ i \in C^{+}), (max_j v_{ij}(x) | i \in C^{-}) |\ j = 1, \dots, n\} \quad (6.6)$$

У таблиці 6.8 зображено результат третього кроку для заданих значень.

Таблиця 6.8 – Результат обчислення PIS та NIS відповідно

| | | | | | |
|---|---|---|---|---|---|
| $A^{+}$ | 0.4569 | 0.066 | 0.0969 | 0.0559 | 0.0496 |
| $A^{-}$ | 0 | 0.1466 | 0 | 0 | 0.0496 |

Далі, після отримання ідеального та неідеального розв’язку, потрібно знайти відстані до $A^{+}$ та $A^{-}$ за формулами

$$D_i^{+} = \sqrt{\sum_{j=1}^{n} \left(v_{ij} - v_j^{+}\right)^2}, i = 1, \dots, m \quad (6.7)$$

$$D_i^{-} = \sqrt{\sum_{j=1}^{n} \left(v_{ij} - v_j^{-}\right)^2}, i = 1, \dots, m \quad (6.8)$$

У таблиці 6.9 зображено результат обчислень відстаней до $A^{+}$ та $A^{-}$.

Таблиця 6.9 – Обчислені значення відстаней

| | $D^{+}$ | $D^{-}$ |
|---|---|---|
| U2 | 0.23 | 0.26 |
| U3 | 0.467 | 0.098 |
| U4 | 0.081 | 0.467 |

Далі потрібно обрахувати близькість до ідеального розв’язку за формулою

$$C_i^* = \frac{D_i^-}{D_i^+ + D_i^-}, i = 1, \dots, m \quad (6.9)$$

У таблиці 6.10 показано результат обчислень близькості до ідеального розв’язку.

Таблиця 6.10 – Наближеність кожної з альтернатив до ідеального розв’язку

| | |
|---|---|
| U2 | 0.53 |
| U3 | 0.174 |
| U4 | 0.852 |

На основі результатів останніх обрахунків можна зробити висновок, що ідеальною альтернативою для клієнта User1 буде спеціаліст User4, оскільки він є найближчим до ідеалу. Не дивлячись на те, що вони не підходять одне одному за ціновою характеристикою, інші критерії в них збігаються, що робить цей розв’язок оптимальним. Далі, за спаданням, наступною альтернативою є User2, оскільки вони з User1 мають достатню кількість збігів, хоч і не таку велику як з User4. Найгіршою альтернативою виявився спеціаліст User3, оскільки у них з User1 є мінімальна кількість збігів по критеріям.

Висновки до розділу 6

У даному розділі було описано математичне забезпечення задачі з підбору спеціаліста. Було описано постановку задачі, обрано алгоритми, за допомогою яких проводились обрахунки. Ними стали метод аналізу ієрархій та TOPSIS (Technique for Order Preference by Similarity to Ideal Solution). Також було проведено тестовий обрахунок підбору спеціаліста користувачеві.

Отримані результати було проаналізовано та обґрунтовано.

# 7 ТЕСТУВАННЯ СИСТЕМИ

## 7.1 Мета випробувань

Метою випробувань є тестування розробленого програмного забезпечення, задля перевірки працездатності, стабільності та відповідності поставленим вимогам.

Тестування завжди є важливою частиною розробки, оскільки воно дає можливість виявити невідповідності у роботі, знайти та виправити помилки до встановлення застосунку у продуктив. Це економить значну кількість часу та коштів у майбутньому.

Як правило при комерційній розробці функції розроблення ПЗ та тестування розділяються на різні посади, аби скоординувати дані процеси по фаху. Людина, що тестує програмне забезпечення – інженер з забезпечення якості, що може реалізовувати тестування як мануально (Manual Quality Assurance Engineer) так і автоматизовано (Automation Quality Assurance Engineer). Незалежно від спеціалізації, головна місія QA-інженера це забезпечення якості ПЗ на всіх етапах розробки. Також, основними задачами є виявлення багів, перевірка функціоналу на працездатність та відповідність вимогам.

Таким чином тестування є невідокремною частиною створення застосунків будь-якого формату, оскільки саме цей процес дає можливість створювати надійні, якісні та зручні продукти.

## 7.2 Загальні положення

Процес тестування проводився базуючись на принципах тестування, задля оптимізації ресурсів та результату.

Для тестування було обрано використовувати автоматизовані unit та integration тести і мануальні UI-тести, що слідує піраміді тестування. Згідно з під-

ходом піраміди тестування рекомендується спершу автоматизовувати менші та легші у підтримці типи тестів. Дякуючи цьому підходу гарантується швидке і надійне покриття критичних сценаріїв використання системи.

Структура піраміди тестування [15]:

- unit;
- integration;
- e2e.

Дана структура описує в якій послідовності і коли потрібно описувати тести задля забезпечення максимальної ефективності тестування.

### 7.2.1 Модульне тестування

Модульне тестування є основою піраміди тестування і за ієрархією знаходиться знизу і займає найбільший скоуп у піраміді. Це обґрунтовується тим, що модульне тестування охоплює найменші частинки коду, перевіряючи дію окремих логік та функцій застосунку. Такі тести є невеликими по об’єму, але є досить важливою складовою гарної та якісної перевірки розробленого функціоналу. Вони також є досить швидкими у виконанні, що дозволяє робити ці перевірки безпосередньо під час розробки. Також їх часто запускають після доопрацювань задля того, щоб переконатись у тому, що новий функціонал не «зламав» зміни до цього.

Для функціоналу застосунку модульні тести було створено після розробки. Такий підхід називають post-development testing (тестування після реалізації). На відміну від test driven development (розробка, що керується тестами), де тести пишуться до розробки і загально визначають її напрям, тут реалізація функціоналу передувала тестуванню. Це допомогло пришвидшити процес розробки та охопити наявні сценарії використання системи, не спираючись на тести.

На рисунку 7.1 зображена структура проєкту з unit тестами, що включає в собі покриття розробленого функціоналу.

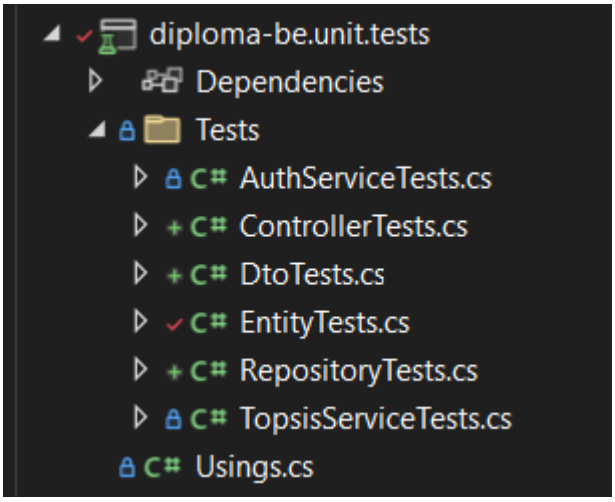


Рисунок 7.1 – Структура проєкту unit тестування

Даний проєкт складається з директорії Tests що тестує код з шару BLL. Також всі тести логічно розділені на тестові класи, аналогічно до сервісів у програмному забезпеченні. Кожен тест тестує маленькі частинки розробленого коду (методи) використовуючи моки для підміни даних та їх ізоляції. Unit-тести зазвичай пишуться через принцип AAA – Arrange, Act, Assert (підготуй, роби, перевіряй), тому в даному випадку було обрано слугуватись цією практикою. На рисунку 7.2 зображено приклад тесту з проєкту unit.tests.

```
[Fact]
0 references
public async Task UserCreation_WithUniqueEmail_ShouldSucceed()
{
    // Arrange
    var user = new User
    {
        Id = Guid.NewGuid(),
        FirstName = "Test",
        LastName = "User",
        Email = "unique@test.com",
        Phone = "+380501234567",
        PasswordHash = BCrypt.Net.BCrypt.HashPassword("password123"),
        Role = "Client",
        CreatedAt = DateTime.UtcNow
    };

    // Act
    _context.Users.Add(user);
    await _context.SaveChangesAsync();

    // Assert
    var savedUser = await _context.Users.FirstOrDefaultAsync(u => u.Email == "unique@test.com");
    savedUser.Should().NotBeNull();
    savedUser!.FirstName.Should().Be("Test");
    savedUser.Role.Should().Be("Client");
}
```

Рисунок 7.2 – Приклад unit-тесту

### 7.2.2 Інтеграційне тестування

Інтеграційне тестування знаходиться посередині піраміди тестування, оскільки є більш складним за модульне, займає більше часу та сил на розробку. Воно включає в собі перевірку взаємодії окремих модулів застосунку, що є життєво важливим для гарантування якості створеного функціоналу. Зазвичай, написанням інтеграційних тестів займаються тестувальники, але у окремих випадках на їх роль можуть стати і розробники. Цей тип тестування проводять як автоматизовано, застосовуючи різноманітні мови програмування та фреймворки, так і мануально, застосовуючи спеціалізовані сервіси для таких перевірок.

Для створеного застосунку для проведення інтеграційного тестування буде застосована мова програмування C# та бібліотека для автоматизованого тестування API частини застосунку RestSharp. Ця бібліотека повністю реалізує функціонал для API-запитів, полегшуючи і пришвидшуючи процес написання автоматизованих тестів. На рисунку 7.3 зображена структура проєкту з інтеграційними тестами.

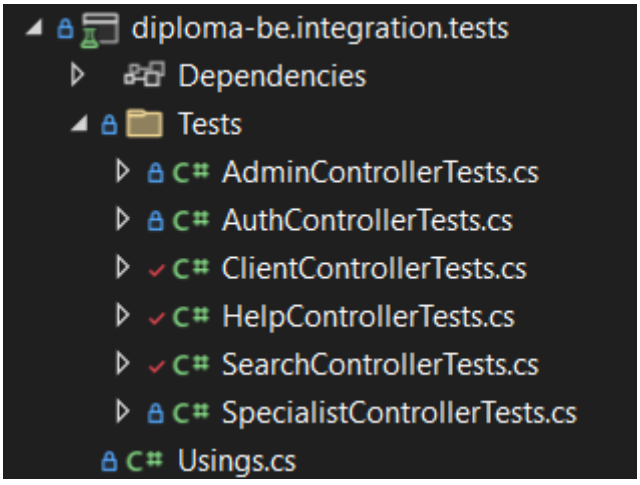


Рисунок 7.3 – Структура проєкту з інтеграційними тестами

У даному проєкті тестується API частина застосунку і тести перевіряють інтеграцію різних частин програмного забезпечення між собою. Тут також викорис-

товується практика AAA, описана у попередньому підрозділі. На рисунку 7.4 зображений приклад інтеграційного тесту з проєкту integration.tests. У цьому тесті відбувається логін користувача з правами адміністратора. На рисунку зображений позитивний тест, але також у проєкті реалізовані негативні сценарії задля максимального покриття перевірок функціоналу.

```
[Fact]
0 references
public async Task Login_WithValidAdminCredentials_ShouldReturnSuccessWithToken()
{
    // Arrange
    var loginRequest = new LoginRequest
    {
        Email = "admin@test.com",
        Password = "admin123"
    };

    // Act
    var result = await _controller.Login(loginRequest);

    // Assert
    result.Should().NotBeNull();
    var okResult = result.Result as OkObjectResult;
    okResult.Should().NotBeNull();

    var loginResponse = okResult!.Value as LoginResponse;
    loginResponse.Should().NotBeNull();
    loginResponse!.Token.Should().NotBeEmpty();
    loginResponse.Role.Should().Be("Admin");
    loginResponse.Name.Should().Be("Admin User");
}
```

Рисунок 7.4 – Приклад інтеграційного тесту

### 7.2.3 Наскрізне тестування

E2E (наскрізне) тестування знаходиться на найвищому рівні в піраміді тестування і займає в ній найменше місця. На це є обґрунтовані причини: ці тести є найбільшими по розміру та найдовшими по часу виконання. У звичайному розумінні цей тип тестування є наближеним до досвіду користувача, оскільки перевіряє UI (user interface) частину застосунку. Для такої перевірки використовують напряму розроблений застосунок, виконують потрібні сценарії, перевіряють результати. В звичайній практиці E2E тестування виконується тестувальником, але у деяких випадках до цього процесу можуть підключатись розробники або бізнес-замовники доопрацювань.

Мануальна частина E2E тестування включає в собі написання тестової документації (тест-кейси, чеклисти тощо), проходження по описаним сценаріям, оформлення результатів проведеної роботи (у разі успіху занесення результатів у систему, де фіксуються ці дані, у разі невдачі, формування звіту про помилку (bug report)). Автоматизована частина роботи заключається у покритті створених тестових сценаріїв автотестами. Для цього існує велика кількість фрейморків та бібліотек, що значно спрощують цей процес для автотестувальника.

У межах даної роботи було вирішено провести мануальне E2E тестування, оскільки на поточному етапі розробки достатньо автоматизованих unit та integration тестів для перевірки основної логіки застосунку. Автоматизація даного типу тестування є доцільною на пізніших етапах, коли всі вимоги виконано і не планується великих доробок системи.

## 7.3 Результати тестування

Як було визначено раніше, для перевірки компонентів та інтеграцій було обрано автоматизоване тестування, що допомогло швидко та точно отримати результати пророблені розробки та виправити наявні помилки. У процесі написання та прогону автоматизованих тестів було виявлено невідповідності у системі, завдяки чому з’явилась можливість додатково переглянути функціонал програмного забезпечення та змінити його відповідно до поставлених раніше вимогах. На рисунках 7.5 та 7.6 зображено результати запуску цих тестів.

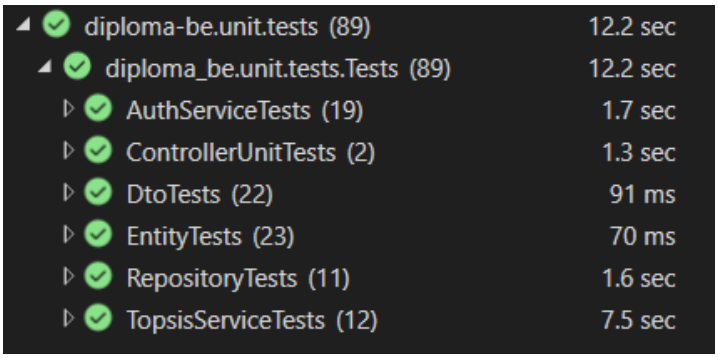


Рисунок 7.5 – Результат запуску unit-тестів

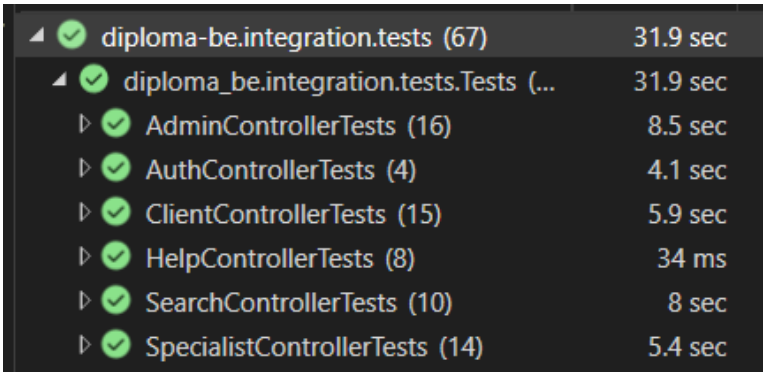


Рисунок 7.6 – Результат запуску інтеграційних тестів

Для перевірки E2E сценаріїв було обрано користуватись мануальним тестуванням. Для цього зазвичай використовують чек-листи або тест-кейси. Оскільки створена система ще не є достатньо об’ємною та не містить багато різноманітних функцій, доцільніше було користуватись саме чек-листами через швидкість у написанні, легкість у підтримці та скороченні часу для тестування. При розширенні застосунку та збільшенні його функціоналу варто переглянути обраний шлях для тестування користувацького інтерфейсу, та, можливо, перейти на написання тест-кейсів, оскільки вони надають більш зрозумілу та структуровану картинку сценарію, який тестується.

У таблиці 7.1 описані тести у форматі чек-листу та зазначені результати їх відтворення.

Таблиця 7.1 – Чек-лист для тестування системи

| Назва | Статус |
|---|---|
| Перевірити проходження опитувальника та отримання рекомендацій | Passed |
| Перевірити поля на валідації | Passed |
| Перевірити фільтрацію по спеціалістам | Passed |
| Перевірити запис на сеанс з пошуку | Passed |
| Перевірити запис на сеанс з проходження анкети | Passed |

Продовження таблиці 7.1

| Назва | Статус |
|---|---|
| Перевірити вхід для спеціаліста | Passed |
| Перевірити негативні сценарії входу – неправильний логін | Passed |
| Перевірити негативні сценарії входу – неправильний пароль | Passed |
| Перевірити негативні сценарії входу – пусті поля | Passed |
| Перевірити вхід для адміністратора | Passed |
| Перевірити редагування профілю спеціаліста спеціалістом | Passed |
| Перевірити редагування профілю спеціаліста адміністратором | Passed |
| Перевірити видалення спеціаліста | Passed |
| Перевірити створення спеціаліста | Passed |
| Перевірити вихід з системи | Passed |

Висновки до розділу 7

В даному розділі було визначено що таке тестування, його мету та важливість впровадження для стабільної роботи процесів у розроблених системах. Було обрано типи тестування, за допомогою яких було реалізовано тестування створеного веб-застосунку. Додатково, було обґрунтовано вибір типів тестування, оскільки саме цей вибір був основою успішної перевірки програмного забезпечення.

Процес тестування було оптимізовано завдяки використанню автоматизованого та мануального видів тестування, що допомогло швидко побачити недоліки

після розробки функціоналу застосунку та легко перевірити зміну цих невідповідностей після виправлення. Всі недоліки, що були виявлені у процесі проведення тестування створеної системи були повторно переглянуті та виправлені, після чого повторно протестовані.

Проведення тестування допомогло упевнитись, що створена система відповідає першочергово поставленим вимогам і є готовою до зустрічі з потенційними користувачами. Створені автоматизовані тести та чеклист можуть бути використані у майбутньому, при дорозробці програмного забезпечення, його розширення або інтеграцію з іншими системами.

# ВИСНОВКИ

Результатом дипломного проєкту є вебзастосунок підбору психолога за індивідуальними потребами користувачів.

У розділі «Опис предметної області» було описано цілі створюваної системи, її призначення та проблеми, які вона має вирішити. Завдяки цьому було визначено мету та пріоритети розробки програмного забезпечення.

У розділі «Аналіз існуючих рішень» було оглянуто деякі присутні аналогічні сервіси на ринку, проаналізовано їх функціонал, та виділено переваги та недоліки. Ця інформація допомогла створити порівняльну характеристику оглянутих сервісів та виокремити сценарії роботи, що зроблять створений застосунок конкурентоспроможним та максимально зручним у використанні. Завдяки проведеному дослідженню було відмічено низку переваг які надалі були реалізовані у вебзастосунку.

У розділі «Формування вимог до системи» було визначено різноманітні вимоги до системи: умови роботи, функціональні характеристики, види програмного забезпечення. Також у цьому розділі було сформовано опис майбутніх процесів, що є основними та критичними для застосунку і без яких функціонування застосунку не має сенсу. На основі описаних вимог було розроблено вебзастосунок, що максимально відповідає їм, містить описані переваги та блокує наведені недоліки.

У розділі «Вибір технологій розробки» було обрано стек технологій для розробки програмного забезпечення, обґрунтовано їх переваги використання у рамках поточного завдання та описано їх профіль використання. У результаті цього розділу було обрано такі технології: мова програмування C#, фреймворк ASP.NET, фреймворк Entity Framework, мова програмування TypeScript, фреймворк Angular, СУБД PostgreSQL, IDE Visual Studio та Visual Studio Code. Для формалізації стилю розробки було обрано слугуватись ключовими принципами програмування SOLID, DRY, KISS та YAGNI. Завдяки цьому код застосунку є чистим, простим та легкопідтримуваним, що дає змогу розширюватись у майбутньому без зайвих турбот та у короткий проміжок часу.

У розділі «Розроблення вебзастосунку» було окреслено структуру реалізованого застосунку, оглянуто архітектуру системи, обґрунтовано вибір комунікації системи та коротко описано наявні розв’язки. Також було описано функціональну модель, ролі у системі та їх права доступу. Важливою частиною розробленого є база даних та її інтеграція з проєктом. У даному розділі було описано її структуру та поєднання зі створеним програмним забезпеченням, виділено важливість використання бази даних та критичність налаштування правильного зв’язку між кодом та базою даних.

У ході написання розділу «Математичне забезпечення» було обрано алгоритм MAI для формування матриці попарних порівнянь та алгоритм багатокритеріального аналізу TOPSIS для отримання виважених результатів. Головною метою даного розділу було ознайомитись з обраним алгоритмом підбору, провести тестові обчислення та інтегрувати їх у розробку. Завдяки цьому розділу було окреслено важливість інтеграції математичного алгоритму до застосунку та вплив на отримання результатів під час користування системи потенційним клієнтом.

В останньому розділі «Тестування системи» було визначено мету випробувань, окреслено їх важливість та невідокремність від процесу розробки. Було описано спектр проведених досліджень, посилаючись на піраміду тестування як основу вичерпної перевірки створеного функціоналу. Також було обґрунтовано спосіб проведення тестування – автоматизовано та мануально, що дало змогу ефективно та точно відтворити всі потрібні перевірки. У процесі написання розділу та проведення тестування було знайдено деякі недоліки у програмному забезпеченні, які вдалось вчасно полагодити завдяки вичерпному тестуванню. У кінці розділу було наведено результати досліджень, які показали, що система функціонує відповідно до поставлених вимог та не містить критичних блокерів у роботі.

Розроблена система орієнтована на використання як самостійної платформи, якою може скористатись будь-хто, хто має доступ до мережі Інтернет. Однак, вона також може інтегруватись в інші системи, ставши основою розробки більш обширного за функціоналом застосунку або стати частинкою внутрішніх систем, таких

як корпоративні платформи, медичні застосунки тощо. Така гнучкість у використанні та інтеграції дозволяє створеному вебзастосунку адаптуватись до різноманітних потреб користувачів – від індивідуальних запитів до інтеграцій у великі корпорації, забезпечуючи стабільність та ефективність.

Створений вебзастосунок є достатнім для використання, але також має можливість розширюватись. Вектором на майбутній розвиток системи є впровадження штучного інтелекту для точнішого формування запиту клієнта, реєстрація з боку клієнта для швидшого управління своїми записами та інформацією, формування швидких джерел інформації для підвищення рівня обізнаності користувачів у даному напрямку.

Отже, створена система у рамках дипломного проєкту досягла заявленої мети та виконує всі поставлені задачі. Реалізований алгоритм відпрацьовує точно, показуючи виправдані результати та вирішує виокремлені проблеми.

## ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Rozmova. URL: https://www.rozmova.me
2. Pleso. URL: https://pleso.me/ua
3. Qui.help. URL: https://www.qui.help/Psychologists/Search
4. A tour of the C# language. URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview
5. ASP.NET documentation. URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-9.0
6. Angular documentation. URL: https://angular.dev/overview
7. PostgreSQL. URL: https://en.wikipedia.org/wiki/PostgreSQL\
8. Three-Layer Architecture Used in Software Development. URL: https://dev.to/sardarmudassaralikhan/three-layer-architecture-used-in-software-development-57ji
9. REST Endpoint Best Practices Every Developer Should Know. URL: https://levelup.gitconnected.com/rest-endpoint-best-practices-every-developer-should-know-24b0c1d0088f
10. Розуміння Клієнт-Серверної Архітектури на прикладах. URL: https://dou.ua/forums/topic/44636/
11. Role-Based Access Control. URL: https://auth0.com/docs/manage-users/access-control/rbac
12. EF Core Migrations Overview. URL: https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli
13. Метод аналізу ієрархій. URL: https://dss.tg.ck.ua/ahp-help
14. TOPSIS and Modified TOPSIS: A comparative analysis. URL: https://www.sciencedirect.com/science/article/pii/S277266222100014X#:~:text=TOPSIS%20is%20based%20on%20the,from%20the%20negative-ideal%20one
15. The testing pyramid: Strategic software testing for Agile teams. URL: https://circleci.com/blog/testing-pyramid