МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«18» червня 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

«Вебзастосунок надання психотерапевтичних консультацій»

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Ярослав ПОПОВ**

«18»червня 2025 р.

Керівник роботи
старший викладач

_____ **Іван БУРЛАЧЕНКО**

«18»червня 2025 р.

Миколаїв – 2025

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп’ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«13» листопада 2024 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Попова Ярослава

1. Тема кваліфікаційної роботи «Вебзастосунок надання психотерапевтичних консультацій» затверджена наказом ректора ЧНУ ім. Петра Могили № 315 від «13» листопада 2024 р.
2. Строк представлення кваліфікаційної роботи «23» червня 2025 р.
3. Очікуваний результат роботи та початкові дані якщо такі потрібні.

Розроблений вебзастосунок надання психотерапевтичних консультацій

4. Перелік питань, що підлягають розробці:
   - аналіз предметної області;
   - розробка проєктних рішень застосунку;
   - моделювання та конструювання ПЗ;

- формування структури бази даних;
- розробка інтерфейсу користувача;
- реалізація серверної частини вебзастосунку;
- розробка клієнтської частини вебзастосунку.

5. Перелік графічних матеріалів: презентації
6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «13» листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **«Вебзастосунок надання психотерапевтичних консультацій»**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	24.10.2024	13.11.2024	Виконано
2.	Огляд літератури за темою роботи	14.11.2024	22.11.2024	Виконано
3.	Складання календарного плану КБР	25.11.2024	26.11.2024	Виконано
4.	Аналіз предметної області	27.11.2024	06.12.2024	Виконано
5.	Розробка проєктних рішень	09.12.2024	13.12.2024	Виконано
6.	Моделювання та конструювання ПЗ	16.12.2024	27.12.2024	Виконано
7.	Розробка структури бази даних та міграцій до неї	30.12.2024	10.01.2025	Виконано
8.	Кодування ПЗ для API	13.01.2025	28.02.2025	Виконано
9.	Кодування ПЗ для клієнтської частини	03.03.2025	04.04.2025	Виконано
10.	Тестування ПЗ	07.04.2025	25.04.2025	Виконано
11.	Відгук керівника КБР	28.04.2025	30.04.2025	Виконано
12.	Оформлення КБР та презентації	01.05.2025	26.05.2025	Виконано
13.	Попередній захист	27.05.2025	28.05.2025	Виконано
14.	Завершення оформлення КБР та презентації	29.05.2025	06.06.2025	Виконано
15.	Рецензування	07.06.2025	16.06.2025	Виконано
16.	Захист КБР	23.06.2025	24.06.2025	Виконано

Здобувач _____ **Ярослав ПОПОВ**

«26» листопада 2024 р

Керівник роботи

старший викладач

_____ **Іван БУРЛАЧЕНКО**

«26» листопада 2024р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«Вебзастосунок надання психотерапевтичних консультацій»

Здобувач 408 гр.: Ярослав Попов

Керівник: старший викладач, Іван Бурлаченко

Сучасна геополітична та загальна соціальна ситуація громадян різних держав, в особливості України, змушує замислюватися над більш вагомими питаннями та моральними виборами. Психічний стан людини є ключовою складовою власного ментального здоров'я, яке важливо підтримувати.

Такі фактори, як соціальна депривація, життя рідних, власний успіх або моральний тиск суспільства не є рідкістю у наші часи, а походи до психолога можуть вартувати великих фінансових потреб. В цьому випадку пізнання власного психологічного стану людини шляхом безкоштовного тестування стає актуальним.

Онлайн-сервіси цього напрямку у своїй більшості не є безкоштовні, натомість надають комплексну допомогу через психотерапевтів. Більшість мобільних застосунків, у свою чергу, доступні лише на мобільних пристроях, та не інтегровані у браузер.

Додатковим аспектом для відповідного сервісу є можливість впровадження штучного інтелекту із його подальшим розвитком. Цей застосунок має поєднувати у собі швидкість та якість надання рекомендацій без участі психотерапевтів на основі переважних відповідей користувача.

Об’єктом кваліфікаційної роботи є розробка вебзастосунку оцінки психологічного стану людини.

Предметом кваліфікаційної роботи виступає інструментарій для створення вебзастосунків на базі фрейморків Angular та Nest.js.

Мета кваліфікаційної роботи полягає в розробці вебзастосунку надання рекомендацій для покращення психічного та ментального стану людини, шляхом аналізу пройденого користувачем тестування на вебплатформі.

Застосовано теоретичні методи: аналіз предметної області та аналогів застосунку, аналіз потенціальних терапій та сумісність відповідно до психічного

стану користувача. Використання практичних методів дозволили наступне: реалізацію та налаштування архітектури, використання інструментів повнотекстового пошуку та кешування даних.

Кваліфікаційна робота містить вступ, чотири розділи, висновки, перелік джерел посилання та додатки. Вступ зазначає актуальність, мету, об’єкт та предмет роботи. Перший розділ описує предметну область та наявні аналоги застосунку. У другому розділі складено специфікацію вимог до програмного забезпечення, описана функціональність застосунку. Третій розділ містить опис проєктування системи із визначеним стеком технологій розробки, ER-діаграм, діаграм класів, архітектурних рішень. Четвертий розділ пов’язаний із процесом створення застосунку, безпосередньо кодуванням системи, реалізацією та тестуванням інтерфейсу користувача. Висновки підсумовують етапи розробки застосунку та описують результати виконаної роботи.

Кваліфікаційна бакалаврська робота викладена на 64 сторінки, вона містить 4 розділи, 38 ілюстрації, 5 таблиць, 19 джерел в переліку посилань, 6 додатків.

Ключові слова: індексація, психологічне консультування, стандарти авторизації, терапія, вебзастосунок, Angular, ElasticSearch, Nest.js, PostgreSQL, RabbitMQ, Redis, RPC, Websockets.

ABSTRACT

of the Bachelor’s Thesis

«WEB APPLICATION FOR PSYCHOTHERAPEUTIC COUNSELING»

Student of group 408: Yaroslav Popov

Supervisor: senior lecturer, Ivan Burlachenko

The current geopolitical and social situation of citizens from various countries, particularly Ukraine, prompts reflection on weightier questions and moral choices. A person's mental state is a crucial component of their mental health, which is important to maintain.

Factors such as social deprivation, family life, personal success, or societal moral pressure are common nowadays, and visits to psychologists can incur significant financial costs. In this context, understanding one's psychological state through free testing becomes relevant.

Most online services in this field are not free but offer comprehensive assistance through psychotherapists. However, many mobile applications are only available on mobile devices and are not integrated into browsers.

An additional aspect for an effective service is the implementation of artificial intelligence with its further development. This application aims to combine speed and quality in providing recommendations without the involvement of psychotherapists based on user responses.

The **object** of the thesis is the process of assessing a person's psychological state.

The **subject** of the thesis is software for creating web applications based on Angular and Nest.js frameworks.

The **purpose** of the thesis is to develop a web application for providing recommendations to improve the mental and psychological state of individuals through the analysis of user-test data on a web platform.

Theoretical methods applied include analysis of the subject area and application analogs, analysis of potential therapies and their compatibility with the user's mental state. Practical methods involved the implementation and configuration of architecture, use of full-text search tools, and data caching.

The thesis consists of an introduction, four chapters, conclusions, a list of references, and appendices. The introduction outlines the relevance, aim, object, and subject of the thesis. The first chapter describes the subject area, existing application analogs, and specifies requirements. The second chapter discusses the software requirements specification, describing the application's functionality. The third chapter details the system design with a defined technology stack, including ER diagrams, class diagrams, and architectural decisions. The fourth chapter covers the application development process, including system coding, implementation, and user interface testing. Conclusions summarize the stages of application development and describe the results of the completed work.

The bachelor's thesis consists of 64 pages, including 4 chapters, 38 illustrations, 5 tables, 19 references in the bibliography, and 6 appendices.

Keywords: Angular, ElasticSearch, Nest.js, PostgreSQL, RabbitMQ, RPC, Redis, authentication standards, indexing, psychological counseling, therapy, web application, Websockets.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

КПТ	–	Когнітивно-поведінкова терапія
ПЗ	–	Програмне забезпечення
AI	–	Artificial Intelligence
API	–	Application Programming Interface
CPU	–	Central Processing Unit
GB	–	Gigabyte
GDPR	–	General Data Protection Regulation
HDD	–	Hard Disk Drive
HTTP	–	HyperText Transfer Protocol
HTTPS	–	HyperText Transfer Protocol Secure
JWT	–	JSON Web Token
LSTM	–	Long Short-Term Memory
RAM	–	Random Access Memory
RBAC	–	Role-Based Access Control
RPC	–	Remote Procedure Call
SQL	–	Structured Query Language
TLS	–	Transport Layer Security
UI	–	User Interface

ВСТУП

У сучасному суспільстві, що динамічно розвивається під впливом технологічних інновацій та глобалізації, психічне здоров’я стає ключовим чинником якості життя. Проте на тлі стрімкого прогресу в галузі інформаційних технологій спостерігається паралельне зростання психологічного навантаження на людину. Такі глобальні виклики, як збройні конфлікти, економічні кризи, соціальна нерівність та індивідуальні стресові фактори, призводять до значного погіршення ментального стану населення. Особливо це актуально для українського суспільства, яке в умовах триваючої війни зіткнулося з безпрецедентним рівнем психоемоційного напруження, що проявляється в підвищеній тривожності, депресивних станах, емоційному вигоранні та посттравматичних розладах.

На жаль, доступ до психологічної допомоги залишається обмеженим для значної частини населення через низку об’єктивних бар’єрів. До них належать висока вартість послуг психотерапевтів, дефіцит фахівців у віддалених регіонах, а також соціальні стигми, пов’язані із зверненням за психологічною підтримкою. Існуючі онлайн-сервіси, хоча й частково вирішують цю проблему, мають ряд суттєвих недоліків: більшість із них є платними, орієнтовані виключно на мобільні платформи або не використовують сучасні технології для автоматизації діагностики та надання персоналізованих рекомендацій.

У даному контексті розробка вебзастосунку для надання психотерапевтичних консультацій стає особливо актуальною, оскільки він може поєднати доступність, зручність використання та інноваційні підходи до підтримки ментального здоров’я. Головною перевагою такого рішення є можливість швидкої та безкоштовної первинної діагностики психоемоційного стану користувача з подальшою генерацією індивідуальних рекомендацій на основі аналізу його відповідей.

Об’єктом роботи є розробка вебзастосунку оцінки психологічного стану людини.

Предметом роботи виступає інструментарій для створення вебзастосунків на базі фрейморків Angular та Nest.js.

Мета роботи полягає в розробці вебзастосунку надання рекомендацій для покращення психічного та ментального стану людини, шляхом аналізу пройденого користувачем тестування на вебплатформі.

Відповідно до мети визначено такі завдання:

– провести аналіз вебзастосунків для проведення психологічних консультацій;

– сформувати специфікацію вимог до ПЗ;

– спроєктувати архітектуру вебзастосунку;

– розробити структуру бази даних відповідно до вимог;

– реалізувати серверну частину вебзастосунку;

– розробити клієнтську частину вебзастосунку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Українське суспільство зіткнулося з безпрецедентним викликом у сфері психічного здоров'я через триваючу війну. За даними МОЗ України (2023):

– 75% населення потребують психологічної підтримки;

– лише 15% мають доступ до психологічної допомоги;

– середня вартість консультації психолога (500-1500 грн) недоступна для 60% населення.

Сьогодні українське суспільство переживає безпрецедентний психологічний тиск через триваючу війну. Згідно з останніми дослідженнями (4Service, 2025), абсолютна більшість населення (83%) відчуває сильний стрес, причому 78% пов'язують це безпосередньо з воєнними діями. Найбільш тривожними факторами стали: безпека близьких (74%), ризик життєвої небезпеки (54%) та фінансові труднощі (39%) [1].

Цікаво, що за останні три роки суттєво змінилася громадська свідомість щодо психологічної допомоги. Якщо у 2022 році лише 7% населення зверталися до фахівців, то у 2025 цей показник зріс до 17%. При цьому кількість людей, які категорично відмовляються від допомоги, скоротилася з 46% до 22%. Це свідчить про поступове подолання соціальних стигм у сфері ментального здоров'я.

Аналіз соціологічних даних виявляє чітку диференціацію потреб опитуваних:

– військовослужбовці (68%);

– люди, які втратили близьких (65%);

– цивільні у зоні бойових дій (42%).

Стратегії подолання стресу:

– соціальна підтримка (спілкування з близькими – 36%);

– пасивні методи релаксації (перегляд фільмів – 35%, музика – 30%);

– віртуальна комунікація (соцмережі – 40%).

Важливим аспектом є зростаюча солідарність суспільства – 86% українців підтримують необхідність відкритого обговорення проблем ментального здоров'я, а 74% готові надавати підтримку навіть незнайомим людям.

1.2 Існуючі аналогічні рішення

Сучасний ринок цифрової психологічної допомоги пропонує різноманітні рішення, кожне з яких має свої особливості.

BetterHelp (рис. 1.1, табл. 1.1) – один із найпопулярніших міжнародних сервісів онлайн-терапії, що надає доступ до ліцензованих психологів через вебінтерфейс та мобільні застосунки. Платформа пропонує індивідуальний підбір фахівців, різні формати спілкування (текст, аудіо, відео) та доступ до групових сесій [2].

Таблиця 1.1 – Характеристики BetterHelp

Назва	**BetterHelp**
Архітектура	Web, iOS, Android.
Мова реалізації	JavaScript, Java, Swift.
Основні функції	Підбір терапевта, мультиформатна терапія, групові сесії, особистий кабінет.
Переваги	Глобальна доступність, гнучкий графік, різноманітність терапевтів.
Недоліки	Висока вартість ($60-90/тиждень), відсутність української мови, немає одноразових консультацій.

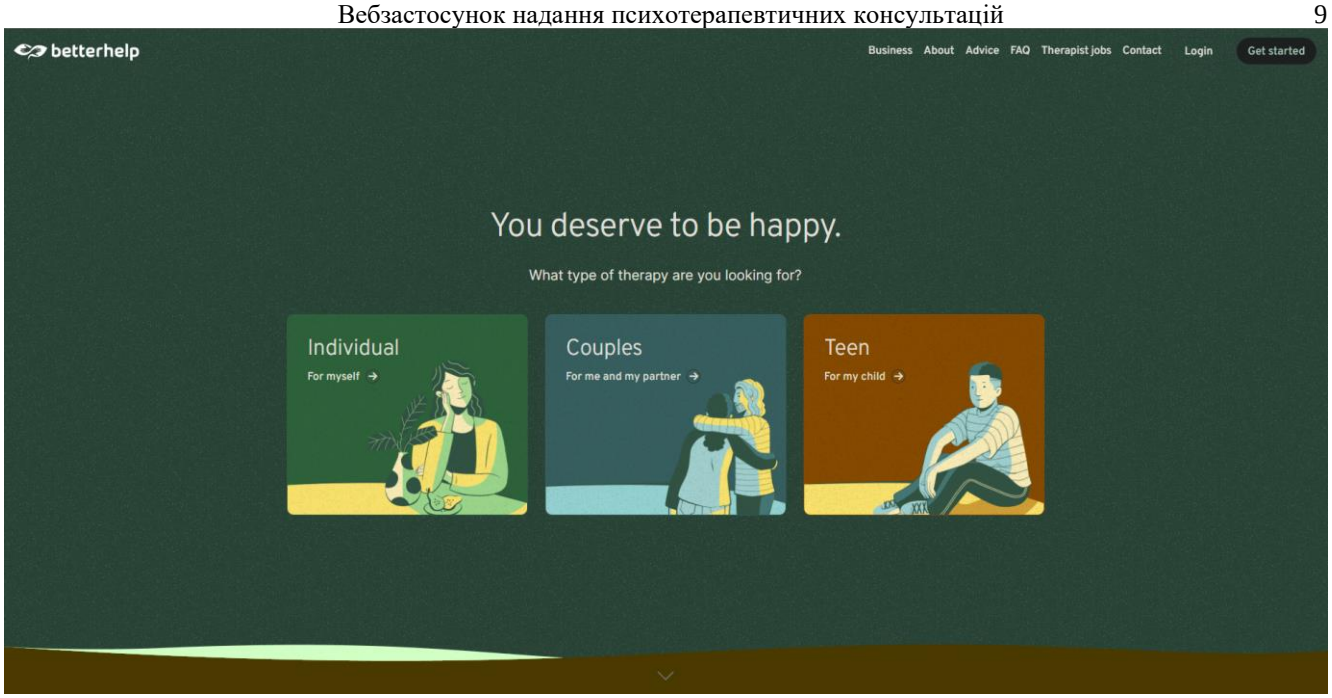


Рисунок 1.1 – Інтерфейс вебзастосунку BetterHelp

Talkspace (рис. 1.2, табл. 1.2) – американський сервіс онлайн-терапії, що спеціалізується на текстових, аудіо- та відеоконсультаціях. Особливістю є наявність спеціалізованих програм для роботи з ПТСР та іншими розладами [3].

Таблиця 1.2 – Характеристики Talkspace

Назва	**Talkspace**
Архітектура	Web, iOS, Android.
Мова реалізації	JavaScript, Java, Swift.
Основні функції	Текст/аудіо/відео консультації, спецпрограми для ПТСР, особистий терапевтичний чат.
Переваги	Ліцензовані терапевти, спеціалізовані програми, підтримка 24/7.
Недоліки	Орієнтація на ринок США, обмежені сесії в тарифах, висока вартість

Рисунок 1.2 – Інтерфейс вебзастосунку Talkspace

Mental Health (BetterMe) (рис. 1.3, табл. 1.3) – мобільний застосунок для психологічної самодопомоги, який пропонує медитації, вправи для зниження стресу та трекінг психоемоційного стану [4].

Таблиця 1.3 – Характеристики Mental Health

Назва	**Mental Health**
Архітектура	iOS, Android.
Мова реалізації	Swift, Java/Kotlin.
Основні функції	Бібліотека медитацій, трекінг настрою, персоналізовані рекомендації, аудіоуроки.
Переваги	Простий інтерфейс, щоденні мотиваційні повідомлення, доступність.
Недоліки	Відсутність професійної терапії., обмежений безкоштовний функціонал, немає вебверсії.

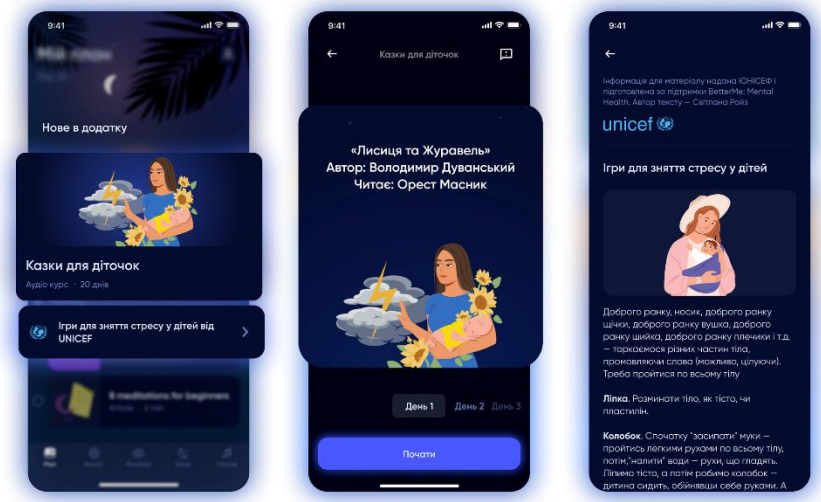


Рисунок 1.3 – Інтерфейс мобільного застосунку Mental Health

Woebot (рис. 1.4, табл. 1.4) – інноваційний чат-бот з елементами штучного інтелекту, що використовує методи когнітивно-поведінкової терапії. Застосунок працює на основі алгоритмів машинного навчання для надання персоналізованих рекомендацій [5].

Таблиця 1.4 – Характеристики Woebot

Назва	**Woebot**
Архітектура	iOS, Android.
Мова реалізації	Python, JavaScript.
Основні функції	Щоденні перевірки настрою, інтерактивні вправи КПТ, освітні матеріали, аналіз шаблонів мислення.
Переваги	Цілодобова доступність, науково обґрунтовані методи, зручний інтерфейс спілкування.
Недоліки	Обмежена глибина аналізу, відсутність людського фактора, англомовний інтерфейс.

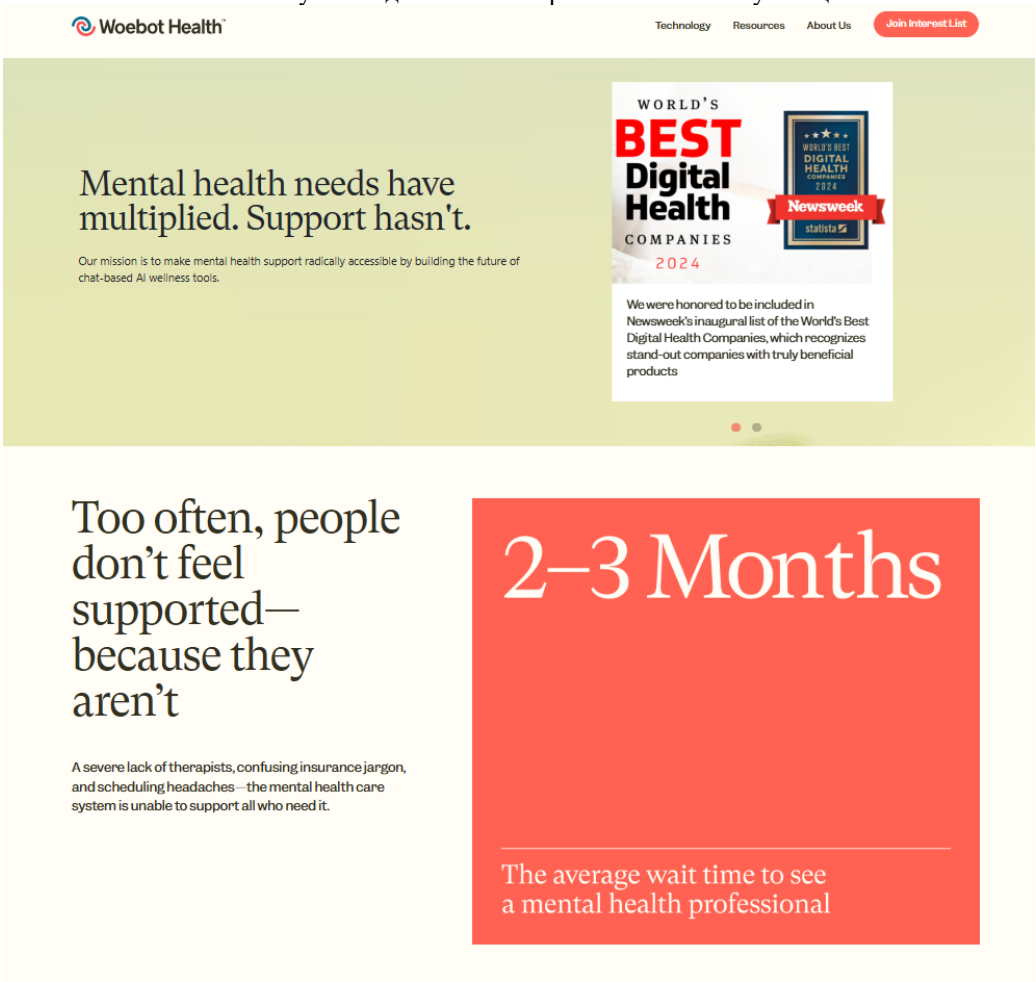


Рисунок 1.4 – Інтерфейс вебзастосунку WoeBot

Проаналізувавши окремі аналоги, можна скласти загальну порівняльну таблицю (табл. 1.5). Ця таблиця виражає слабкі та сильні сторони розглянутих застосунків.

Таблиця 1.5 – Порівняння сервісів надання психологічної допомоги

Критерій	**BetterHelp**	**Talkspace**	**Mental Health (BetterMe)**	**Woebot**
Тип сервісу	Професійна онлайн-терапія.	Професійна онлайн-терапія	Самодопомога, медитації.	AI-чатбот (КПТ).
Формати	Чат, аудіо, відео.	Чат, аудіо, відео.	Медитації, вправи.	Текстовий чат.
Терапевти	Ліцензовані спеціалісти.	Ліцензовані спеціалісти.	Відсутні.	Відсутні (AI).

Кінець таблиці 1.5

Вартість	$60-90/тиждень.	$65-99/тиждень.	$9.99-14.99/місяць.	Безкоштовно.
Мови	Англійська.	Англійська.	Англійська.	Англійська.
Платформи	Web, iOS, Android.	Web, iOS, Android.	iOS, Android.	iOS, Android.
Для кого/чого	Хто потребує регулярної терапії.	Хто потребує спеціалізованої допомоги.	Для щоденного самопочуття.	Для миттєвої підтримки.

З цього випливає, що основні переваги аналогів діляться на декілька категорій: що мають повноцінну підтримку співробітників у якості ліцензованих психотерапевтів, що містять базу знань у вигляді терапій та що виступають у якості чатбота для спілкування.

1.3 Особливості арт-терапії

Підбір оптимального виду арт-терапії вимагає комплексного підходу, що враховує індивідуальні особливості клієнта, специфіку його стану та терапевтичні цілі [6]. Основний принцип полягає у співвідношенні між характером емоційного напруження та терапевтичним потенціалом кожного методу.

Для клієнтів з підвищеною тривожністю (Anxiety) найбільш показаними є структуровані методи, такі як мандалотерапія. Геометрична чіткість форм мандал (рис. 1.5) створює відчуття стабільності та порядку, що допомагає знизити хаотичність думок. Процес розфарбовування активує парасимпатичну нервову систему, сприяючи фізіологічному розслабленню. Альтернативою може виступати монотонна робота з глиною або піском, яка має схожий ефект заспокоєння.

Рисунок 1.5 – Мандалотерапія

При депресивних станах (Depression) особливу ефективність демонструють методи, що стимулюють емоційну виразність. Кольоротерапія в живописі дозволяє виразити пригнічені почуття через підсвідомий вибір кольорів. Вільний малюнок без оцінки технічної майстерності створює безпечний простір для емоційного вираження. Для клієнтів з апатією (Apathy) може бути корисним використання яскравих, насичених кольорів, які активізують емоційну сферу [7].

Для роботи з травматичними переживаннями піскотерапія (рис. 1.6) пропонує унікальні можливості. Можливість фізично впливати на пісочну композицію, змінювати її та «переписувати» сценарії надає відчуття контролю. Цей метод особливо цінний при роботі з почуттям провини (Guilt) або сорому (Shame), оскільки дозволяє символічно трансформувати важкі переживання.

Рисунок 1.6 – Піскотерапія

При агресії (Anger) або фрустрації ефективними виявляються методи з елементами фізичної активності. Робота з глиною, де можна м'яти, розривати або формувати матеріал, надає безпечний канал для вираження гніву. Драматерапія з можливістю імпровізації також дозволяє виплеснути напругу через дію.

Для клієнтів із почуттям самотності (Loneliness) або ізоляції (Isolation) особливо корисними можуть бути групові форми арт-терапії. Спільне музикування або створення колективних арт-об’єктів формує відчуття приналежності. Музикотерапія в групі створює ефект синхронізації емоційних станів через ритм і мелодію.

При страхах (Fear) або панічних станах варто звернути увагу на методи, що дозволяють матеріалізувати об’єкт тривоги. Створення тривимірних об'єктів (наприклад, з паперу або глини) дає можливість «відсторонити» страх, перетворивши його на конкретний, осяжний предмет, з яким можна працювати.

Для клієнтів із почуттям безнадії (Hopelessness) може бути корисним використання наративних методів у поєднанні з арт-терапією. Створення серії малюнків, що відображають зміну стану, або візуалізація «дерева життя» допомагає відновити відчуття перспективи.

Важливо враховувати, що вибір методу має ґрунтуватися не лише на діагностичних критеріях, але й на особистих уподобаннях клієнта. Людина, яка відчуває антипатію до малювання, може знайти терапевтичний ефект у роботі з

музикою або рухом. Терапевту варто пропонувати різноманітні варіанти та спостерігати за реакцією клієнта.

Ключовим аспектом є поступовий перехід від структурованих завдань до більш вільних форм творчості у міру покращення стану клієнта. Початкові обмежені завдання (наприклад, заповнення контурів мандали) дають відчуття безпеки, тоді як на пізніших етапах вільний вираз дозволяє глибше працювати з проблемами.

Інтеграція різних методів арт-терапії часто виявляється найбільш ефективною. Наприклад, поєднання малюнку з подальшим описом створеного образу або використання музики як фону для піскотерапії може посилити терапевтичний ефект. Гнучкість у підборі методів дозволяє створити індивідуальну програму, що максимально відповідає потребам конкретного клієнта.

Висновки до розділу 1

У першому розділі роботи проведено комплексний аналіз сучасних підходів до надання психологічної допомоги, зокрема через вебзастосунки та мобільні платформи. Досліджено предметну область, що включає актуальні проблеми психічного здоров’я в Україні, потреби користувачів та існуючі технологічні рішення.

Проведено порівняльний аналіз провідних аналогів (BetterHelp, Talkspace, Mental Health, Woebot), з урахуванням їх архітектури, функціональності, цільової аудиторії та ефективності. Встановлено, що більшість існуючих сервісів орієнтовані на платну терапію з ліцензованими фахівцями або обмежені англомовним контентом, що робить їх недоступними для значної частини українських користувачів.

2 МОДЕЛЮВАННЯ ОБ’ЄКТУ ТА ПРЕДМЕТУ

2.1 Інструментарій

Розробка вебзастосунку Healthy вимагає ретельного аналізу сучасних технологій для забезпечення оптимальної архітектури, що поєднує високу продуктивність, масштабованість та безпеку. Вибір інструментарію ґрунтується на аналізі специфіки проекту, зокрема необхідності обробки чутливих психологічних даних, забезпечення стабільної роботи під час пікових навантажень та створення інтуїтивного інтерфейсу для користувачів з різним рівнем технічної підготовки.

PostgreSQL (рис. 2.1) основна система керування базами даних через свою надійність та широку функціональність [8]. Ця реляційна база даних забезпечує повну підтримку принципів ACID (Atomicity, Consistency, Isolation, Durability), що є критично важливим при роботі з конфіденційною інформацією про стан психічного здоров'я користувачів. Можливість зберігати структуровані дані у форматі JSON дозволяє гнучко керувати складними структурами, такими як ієрархія питань у тестах або динамічні профілі користувачів. Інтеграція з ElasticSearch через розширення pg_trgm забезпечує ефективний повнотекстовий пошук у базі терапевтичних рекомендацій, що особливо важливо для швидкого доступу до релевантного контенту.

Рисунок 2.1 – PostgreSQL

Angular (рис. 2.2) фреймворк для клієнтської частини був обраний через його архітектурні переваги. Використання TypeScript забезпечує строгу типізацію, що

значно зменшує кількість потенційних помилок на етапі розробки. Модульна структура застосунку дозволяє чітко розділити функціональні блоки (тестування, рекомендації, особистий кабінет), що спрощує подальшу підтримку та масштабування. Вбудовані механізми Dependency Injection та RxJS для роботи з асинхронними потоками даних є особливо корисними для реалізації динамічних інтерфейсів тестування, де необхідно обробляти велику кількість взаємодій користувача в реальному часі.

Рисунок 2.2 – Angular v19

Nest.js (рис. 2.4) серверний фреймворк пропонує ідеальний баланс між продуктивністю та структурованістю коду [9]. Його модульна архітектура, натхненна Angular, дозволяє створювати чітко організований бекенд з підтримкою різних транспортних протоколів (REST, WebSockets). Вбудована підтримка TypeScript забезпечує типобезпеку на всіх рівнях застосунку, від контролерів до сервісів. Підтримка мікросервісної архітектури через вбудовані засоби створення модулів дозволяє легко інтегрувати спеціалізовані сервіси, такі як AI-аналізатор результатів тестування або система рекомендацій.

Рисунок 2.3 – Nest.js

Поєднання цих технологій дозволяє створити масштабовану систему, здатну ефективно обробляти великі обсяги даних про психоемоційний стан користувачів. PostgreSQL забезпечує надійне зберігання та цілісність даних, Angular надає потужний інструментарій для створення динамічного інтерфейсу, а Nest.js виступає ідеальною проміжною ланкою між клієнтом та базою даних, забезпечуючи високу продуктивність API та безпеку передачі даних.

Крім основних технологій, архітектура системи передбачає використання додаткових інструментів для забезпечення повноцінного функціоналу. Redis (рис. 2.4) використовується для кешування сесій та проміжних результатів тестування, що значно знижує навантаження на основну базу даних [10].

Рисунок 2.4 – Redis

RabbitMQ (рис. 2.5) застосовується для організації асинхронної обробки складних завдань, таких як генерація персоналізованих звітів або аналіз тривалих тестів [11].

Рисунок 2.5 – RabbitMQ

ElasticSearch (рис. 2.6) інтегрований для реалізації потужного пошуку по базі терапевтичних рекомендацій, що дозволяє швидко знаходити найбільш релевантні методи втручання для конкретного психоемоційного стану [12].

Рисунок 2.6 – ElasticSearch

Сучасні технологічний інструментарій для створення вебзастосунку враховує продуктивність, супроводжуваність та можливості до майбутнього розширення. На відміну від традиційних підходів, де використовувалися модулі, Angular 19 обраний з його новою парадигмою Standalone Components. Це дозволяє створювати більш автономні та гнучкі компоненти, де кожен елемент інтерфейсу містить власну логіку та залежності, що значно спрощує процес розробки та подальшого супроводу.

На серверній стороні Nest.js залишається ідеальним вибором завдяки своїй модульній архітектурі, яка дозволяє чітко структурувати код на рівні бекенду. Хоча клієнтська частина відійшла від модульного підходу, серверна реалізація зберігає модульну організацію, що забезпечує логічне розділення функціоналу (автентифікація, тестування, рекомендації тощо).

Для забезпечення масштабованості обрано рішення на базі Docker контейнерів з можливістю горизонтального масштабування через балансувальник навантаження Nginx (рис. 2.7). Такий підхід дозволяє ефективно розподіляти навантаження між кількома екземплярами застосунку, забезпечуючи стабільну роботу навіть при значному зростанні кількості користувачів.

Рисунок 2.7 – Nginx

Поєднання Angular 19 (зі Standalone Components) і Nest.js (з модульною архітектурою) створює потужний симбіоз, де клієнтська частина отримує максимальну гнучкість у розробці інтерфейсів, а серверна – чітку структурованість бізнес-логіки. Використання Docker (рис. 2.8) контейнерів спрощує інфраструктуру розгортання, зберігаючи при цьому всі необхідні можливості для масштабування системи [13, 14].

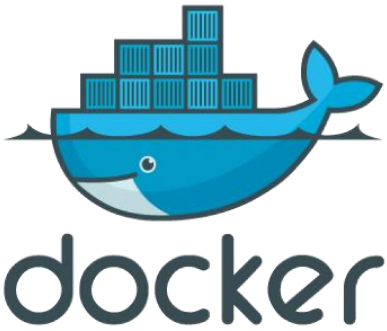


Рисунок 2.8 – Docker

Для забезпечення персоналізованих рекомендацій та аналізу психоемоційного стану користувачів через арт-терапевтичні практики система інтегрує Ollama (рис. 2.9) – потужний інструмент для локального запуску та управління AI-моделями [15]. Ollama дозволяє розгортати легковагові LLM без залежності від хмарних API, що критично важливо для обробки конфіденційних даних у межах захищеної інфраструктури.

Рисунок 2.9 – Ollama

Безпека даних є одним з ключових аспектів системи, що працює з конфіденційною інформацією про стан здоров'я. Обраний стек технологій дозволяє реалізувати всі необхідні заходи захисту: шифрування даних при передачі (HTTPS/TLS), хешування паролів (bcrypt), двофакторна аутентифікація, контроль доступу на рівні ролей (RBAC). PostgreSQL забезпечує надійний захист даних на рівні бази, включаючи можливість налаштування деталізованих прав доступу до таблиць і рядків.

2.2 Специфікація вимог до програмного забезпечення

1) **Призначення системи:**

1.1) вебзастосунок Healthy створений для надання автоматизованої психологічної підтримки через інтерактивну систему тестування. Система дозволяє користувачам отримувати миттєву оцінку свого психоемоційного стану та персоналізовані рекомендації щодо його покращення. Головна мета – забезпечити доступ до базової психологічної допомоги без необхідності прямої участі фахівця;

1.2) погодження та стандарти: відповідність вимогам GDPR для захисту персональних даних, підтримка української мови інтерфейсу, застосування сучасних вебтехнологій для забезпечення кросплатформної роботи;

1.3) межі проекту: розробка вебверсії з повною адаптацією під мобільні пристрої, впровадження основних тестів на стартовому етапі, відсутність функціональності живих консультацій з психологами, не включає розробку окремого мобільного застосунку.

2) **Сфера застосування:**

Система розрахована на користувачів віком від 18 років, які:

– відчувають потребу у швидкій оцінці свого психічного стану;

– шукають доступні методи самодопомоги;

– бажають зберігати анонімність при отриманні рекомендацій.

Характеристики користувачів:

– основні користувачі: дорослі віком від 18 років, особи з базовими навичками роботи з вебінтерфейсами, користувачі, які потребують анонімної психологічної підтримки, не вимагають спеціальних медичних чи технічних знань;

– адміністратори/модератори: працівники психологічних служб, технічні спеціалісти з правами управління контентом, мають доступ до аналітики та редагування бази знань.

3) **Архітектура системи** (рис. 2.10):

3.1) клієнтська частина: Angular 19 з використанням Standalone Components;

3.2) серверна частина: Nest.js фреймворк;

3.3) база даних: PostgreSQL для структурованого зберігання;

3.4) додаткові сервіси: Redis для кешування, ElasticSearch для пошуку, ollama – оркестратор моделей AI.

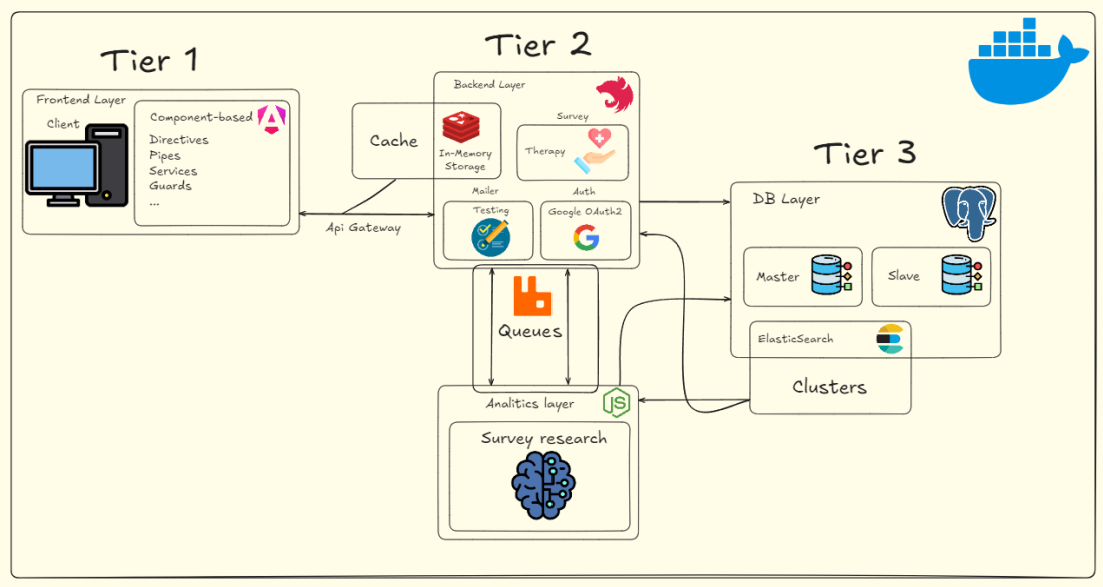


Рисунок 2.10 – Архітектура системи

4) **Загальні обмеження:**

4.1) технічні обмеження: необхідність стабільного інтернет-з'єднання, підтримка лише браузерів (Chrome, Firefox, Safari, Edge останніх версій), обмеження на кількість тестів;

4.2) функціональні обмеження: не надає діагностику серйозних психічних розладів, не замінює офлайн-консультації з психологом, обмежений набір тестів на стартовому етапі.

5) **Функціональні можливості:**

5.1) опис функції «Особистий кабінет та автентифікація»: реєстрація, авторизація та управління профілем користувача. Вхідна інформація: дані Google акаунта (OAuth2). Вихідна інформація: токен доступу, персональна сторінка, доступ до сервісів. Функціональні вимоги: підтримка OAuth2 через Google;

5.2) опис функції «Тестування та аналіз»: проходження психологічних тестів з автоматичним аналізом результатів. Вхідна інформація: відповіді на тестові

питання. Вихідна інформація: результати у балах за шкалою, статистика. Функціональні вимоги: візуалізація результатів, підбір питань з урахуванням попередніх відповідей;

5.3) опис функції «Генерація персоналізованих рекомендацій»: генерація персоналізованих рекомендацій на основі тестів. Вхідна інформація: результати тестування, попередня історія рекомендацій. Вихідна інформація: список терапевтичних практик. Функціональні вимоги: сортування рекомендацій за ефективністю;

5.4) опис функції «Управління контентом (для адміністраторів)»: створення тестів, питань і бази знань. Вхідна інформація: нова версія тесту або опису терапії, коригування варіантів відповідей. Вихідна інформація: оновлений контент у системі, лог змін. Функціональні вимоги: інтерфейс для завантаження матеріалів, модерація контенту;

5.5) опис функції «Аналітика та звіти»: моніторинг статистики та ефективності рекомендацій. Вхідна інформація: результати тестувань. Вихідна інформація: статистика користувача. Функціональні вимоги: розрахунок статистики на основі пройденого тестування;

5.6) опис функції «Системні налаштування»: керування інтерфейсом і параметрами безпеки. Вхідна інформація: вибір кольорової теми. Вихідна інформація: оновлений інтерфейс. Функціональні вимоги: темний/світлий режим.

6) **Технічні вимоги:**

6.1) серверна частина: мінімум 4 ядра CPU, 16 GB RAM, 100 GB HDD;

6.2) клієнтська частина: підтримка останніх версій Chrome, Firefox, Safari;

6.3) мережа: стабільне інтернет-з'єднання з підтримкою HTTPS;

6.4) рекомендоване середовище розгортання: Docker контейнери.

7) **Перспективи розвитку:**

Розширення бази тестів і терапевтичних практик, додавання нових мов інтерфейсу, можливість інтеграції з мобільним застосунком, впровадження додаткових інструментів аналітики.

8) **Властивості програмного забезпечення:**

8.1) **гнучкість.** Можливість адаптації під різні сценарії використання: налаштування рівнів доступу, кастомізація тестових методик, можливість локалізації для нових ринків;

8.2) **сумісність**. Система демонструє високу ступінь інтеграції: підтримка сучасних вебстандартів, сумісність з основними браузерами останніх версій, API для потенційної інтеграції з іншими сервісами;

8.3) **масштабованість.** Архітектура дозволяє легко розширювати функціонал: горизонтальне масштабування серверної частини, модульна структура коду, гнучкі механізми інтеграції з зовнішніми системами;

8.4) **продуктивність.** Середній час відгуку API: 300-500 мс, обробка одного тестування: до 2 секунд, підтримує до 1000 одночасних користувачів, використовує кешування для часткових запитів;

8.5) **безпека.** Анонімізація медичних даних, рольовий доступ (RBAC);

8.6) **переносимість.** Підтримка Docker-контейнерів, легка міграція між серверами, незалежність від ОС;

8.7) **супроводжуваність.** Детальна документація API, система логування подій, модульна архітектура;

8.8) **надійність.** Автоматичне резервне копіювання даних, розподілене навантаження;

8.9) **доступність.** Адаптивний інтерфейс, клавіатурна навігація.

9) **Додаткові вимоги:**

Логування всіх дій користувачів для аналітики, детальна документація API для розробників, можливість масштабування системи при зростанні навантаження, регулярні оновлення контенту та функціональності.

2.3 Інформаційна та функціональна моделі програмного забезпечення

Інформаційна модель системи базується на сучасній реляційній структурі даних, реалізованій за допомогою PostgreSQL. Основу моделі складають такі ключові сутності:

– користувачі (User) з різними ролями (Role) формують ядро системи, де кожен користувач має унікальні ідентифікатори та атрибути для персоналізації;

– тести (Test) складаються з питань (Question), які в свою чергу містять варіанти відповідей (Answer) з визначеним впливом (influence) на психічний стан;

– результати тестувань (Result) зберігають зв'язки між користувачем, питанням та обраною відповіддю, що дозволяє формувати індивідуальні висновки (Conclusion). Терапії (Therapy) пропонуються на основі аналізу цих даних.

Функціональна модель описує ключові процеси:

– користувацький функціонал: реєстрація та авторизація через OAuth2, проходження адаптивних тестів з динамічним підбором питань, отримання результатів у формі рекомендацій арт-терапій, доступ до історії тестувань та рекомендацій;

– адміністративний функціонал: управління тестами та питаннями, редагування бази терапій, аналіз статистики використання системи, модерація контенту;

– системні процеси: аналіз відповідей та оцінка психічного стану, генерація персоналізованих рекомендацій, формування звітів та аналітики, налаштування інтерфейсу (кольорові теми тощо).

Модель передбачає обробку даних у реальному часі з автоматичним оновленням результатів. Система забезпечує конфіденційність через довіреність сервісу Google (OAuth2) та контроль доступу. Архітектура дозволяє масштабування функціоналу та обробку великої кількості запитів.

2.4 Застосування штучного інтелекту в психології

Одним із ключових напрямків застосування AI у психології є автоматизоване виявлення психічних розладів на основі аналізу текстових, аудіальних та візуальних даних. Сучасні підходи включають:

– аналіз текстів (NLP). Використання трансформерних моделей для оцінки емоційного стану через соціальні мережі, чати або відкриті відповіді в тестах. Наприклад, алгоритми виявляють лінгвістичні маркери депресії (частоту займенників першої особи, негативну лексику, низьку лексичну різноманітність). Дослідження показують точність до 92% у виявленні депресії серед студентів [16];

– обробка мовлення. AI-моделі аналізують парамовленнєві ознаки (темп, інтонацію, паузи), що дозволяє виявляти тривогу, стрес або когнітивні порушення. Наприклад, системи на основі LSTM досягають 85% точності у діагностиці деменції за голосовими записами;

– комп’ютерний зір. Аналіз міміки обличчя (наприклад, за допомогою OpenFace) дозволяє виявляти емоційні стани (депресію, агресію) через відеозаписи консультацій або селфі-відео.

AI активно інтегрується у психотерапію, пропонуючи адаптивні інструменти допомоги:

– чат-боти терапевтичної підтримки. Такі системи, як Woebot або Replika, використовують когнітивно-поведінкові техніки (КПТ) для допомоги при легкій депресії чи тривозі. Вони аналізують відповіді користувачів і пропонують персоналізовані вправи. Дослідження показують, що такі боти можуть знизити симптоми депресії на 20-30%;

– рекомендаційні системи. AI аналізує історію тестувань, відповіді та фізіологічні дані (наприклад, з гаджетів) для підбору оптимальних терапевтичних методик. Наприклад: для пацієнтів з тривожними розладами – медитації та дихальні вправи, для депресивних станів – арт-терапія або когнітивні тренінги;

– VR-терапія. Штучний інтелект керує віртуальними середовищами для експозиційної терапії (наприклад, при фобіях) або тренування соціальних навичок у людей з аутизмом.

Сучасні системи на основі штучного інтелекту відкривають нові можливості для проактивної психологічної допомоги. Завдяки аналізу великих масивів даних, алгоритми машинного навчання здатні виявляти тонкі закономірності в поведінці та мовленні, які можуть свідчити про ризик розвитку психічних розладів. Наприклад, аналіз активності в соціальних мережах дозволяє ідентифікувати ранні ознаки депресії або тривожних станів ще до появи клінічних симптомів.

Особливу цінність становлять системи моніторингу емоційного стану в реальному часі. Вони аналізують текст повідомлень, тон голосу, навіть ритм набору тексту на клавіатурі, щоб виявити зміни в психічному стані користувача. Такі технології вже демонструють високу ефективність у виявленні суїцидальних тенденцій, де точність прогнозування сягає 89%.

Профілактичний потенціал AI проявляється і в освітніх програмах. Адаптивні системи можуть пропонувати персоналізовані психологічні вправи або рекомендації щодо зміни способу життя, базуючись на індивідуальному профілі користувача. Це дозволяє не лише лікувати вже наявні проблеми, але й запобігати їх виникненню.

Обрана модель samantha:mistral – це версія Mistral 7B, спеціалізована на емпатичній комунікації та підтримці. Вона ідеально підходить для арт-терапевтичного застосунку завдяки своїй психологічній чутливості та здатності адаптуватись до індивідуальних потреб користувачів.

Важливим аспектом є інтеграція моделі з арт-терапевтичним процесом. Вона генерує тематичні підказки для творчості, а також аналізує символи та метафори у роботах користувача, формуючи корисні інсайти для терапевта.

Для забезпечення безпеки модель оснащена вбудованими фільтрами, які запобігають шкідливим або потенційно тригерним відповідям, що робить її надійним інструментом у роботі з конфіденційною психологічною інформацією.

Висновки до розділу 2

У розділі 2 було проведено комплексний аналіз основних аспектів розробки ПЗ Healthy. На основі дослідження інформаційної та функціональної моделей було визначено ключові сутності системи та їх взаємозв'язки, що дозволяє сформувати чітку структуру бази даних. Функціональна модель продемонструвала логіку роботи системи, зокрема процеси тестування, аналізу результатів та генерації рекомендацій.

Вибір інструментарію розробки ґрунтувався на аналізі сучасних технологічних рішень, що дозволило обрати оптимальний стек технологій для реалізації проекту. Специфікація вимог до системи включала детальний опис вимог та властивостей програмного забезпечення, що стало основою для подальшої реалізації проєкту.

Інтеграція штучного інтелекту в систему Healthy може суттєво підвищити точність діагностики та персоналізацію рекомендацій шляхом автоматизованого аналізу текстових, мовленнєвих та поведінкових даних користувачів. Використання AI-алгоритмів дозволить не лише ефективніше виявляти психоемоційні стани, але й запропонувати адаптивні інструменти психологічної підтримки, що робить систему більш інтелектуальною та корисною для кінцевих користувачів.

3 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ НАДАННЯ ПСИХОТЕРАПЕВТИЧНИХ КОНСУЛЬТАЦІЙ

3.1 Архітектура програмного забезпечення

Завдання обумовлює використання трьохланкової структури застосунку. Враховуючи описану раніше специфікацію вимог та функціональну і інформаційну моделі, доцільною є реалізація наступної архітектури ПЗ (рис. 3.1).

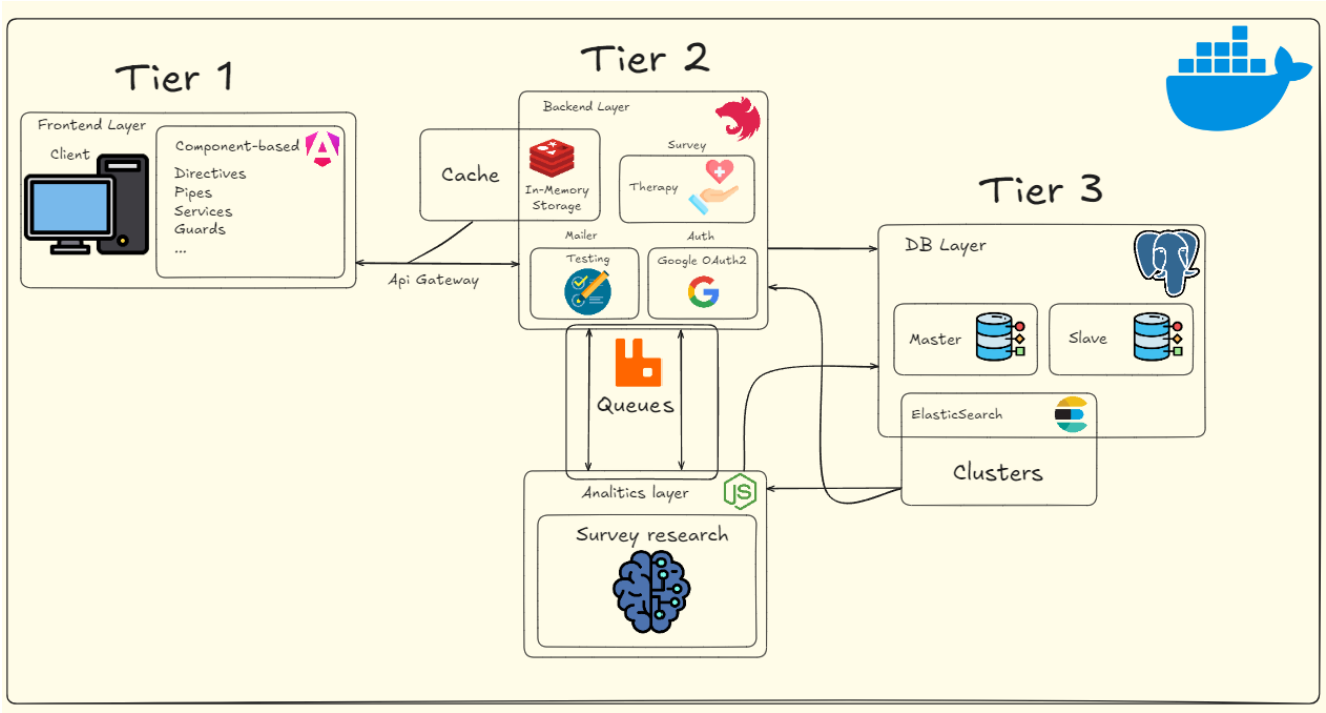


Рисунок 3.1 – Концепція архітектури ПЗ

Щоб реалізувати архітектуру застосунку, необхідно побудувати три основні рівні: клієнтський (Tier 1), серверний (Tier 2) та рівень баз даних і зберігання (Tier 3).

На першому рівні використовується фронтенд-інтерфейс за допомогою Angular. Система повинна бути компонентно-орієнтованою, з чітким поділом логіки на компоненти, сервіси, пайпи, гварди та директиви. Інтерфейс повинен забезпечувати зручну взаємодію користувача із системою через API Gateway. Всі клієнтські запити повинні бути спрямовані до шлюзу, що обробляє маршрутизацію до відповідних сервісів другого рівня. Для підвищення швидкодії частина відповідей має кешуватись у Redis.

На другому рівні реалізується бекенд на основі Nest.js. Розробка окремих модулів для різних функцій системи, зокрема: модуль опитувань та оцінювання (Evaluation), модуль терапевтичної підтримки (Therapy), авторизаційний модуль користувачів (User) з використанням Google OAuth2. Redis використовується як сховище в пам’яті для кешування даних і тимчасових об'єктів, що підвищує продуктивність.

Необхідно інтегрувати RabbitMQ як механізм асинхронної обробки задач. Через нього мають передаватись задачі, пов’язані з аналітикою, перекладом та іншими фоновими процесами. Бекенд публікує повідомлення в черги, а інші частини системи, зокрема аналітичний шар, підписуються на них.

Аналітична частина реалізується окремим мікросервісом на Node.js. Цей сервіс обробляє дані з черг RabbitMQ, агрегує статистику, аналізує результати опитувань. Для глибшого аналізу необхідно підключити локальну мовну модель через Ollama (samantha:mistral), яка допомагає формувати інтерпретації або висновки на основі зібраних відповідей. Отримані результати можуть зберігатися або індексуватись для подальшого пошуку.

Третій рівень відповідає за зберігання даних. Основна база даних реалізується на PostgreSQL у конфігурації master-slave. Усі операції запису виконуються на master-інстансі, а читання – на slave, що забезпечує навантажувальне розділення та відмовостійкість. Для забезпечення швидкого пошуку по великих масивах текстових даних впроваджується Elasticsearch. Його можна використовувати для індексації даних опитувань, результатів AI-аналізу, текстових полів тощо.

Уся інфраструктура застосунку організована за допомогою Docker та оркеструється через docker-compose. Кожен компонент архітектури реалізовано у вигляді окремого сервісу, що дозволяє ізолювати логіку, спростити розгортання та масштабування системи. Усі сервіси підключені до однієї мережі за замовчуванням з типом bridge, що дозволяє їм бачити один одного через імена контейнерів. Таке контейнеризоване середовище забезпечує модульність, контроль над ресурсами,

легке тестування, розгортання і масштабування, а також гнучкість у розвитку окремих модулів системи без втручання в інші частини.

Користувач взаємодіє з Angular-фронтендом, який надсилає запити через API Gateway до Nest.js. Частина запитів обробляється з кешу Redis. Далі бізнес-логіка виконується у відповідних модулях. За потреби дані записуються в PostgreSQL або надсилаються в RabbitMQ. Звідти задачі потрапляють до аналітичного шару, де обробляються і за потреби аналізуються мовною моделлю. Оброблені дані індексуються в Elasticsearch для швидкого пошуку. Результати повертаються на фронтенд через той самий API Gateway.

Така архітектура дозволяє створити масштабовану, розширювану та продуктивну систему, яка може обробляти великі обсяги даних у реальному часі та надавати якісний користувацький досвід.

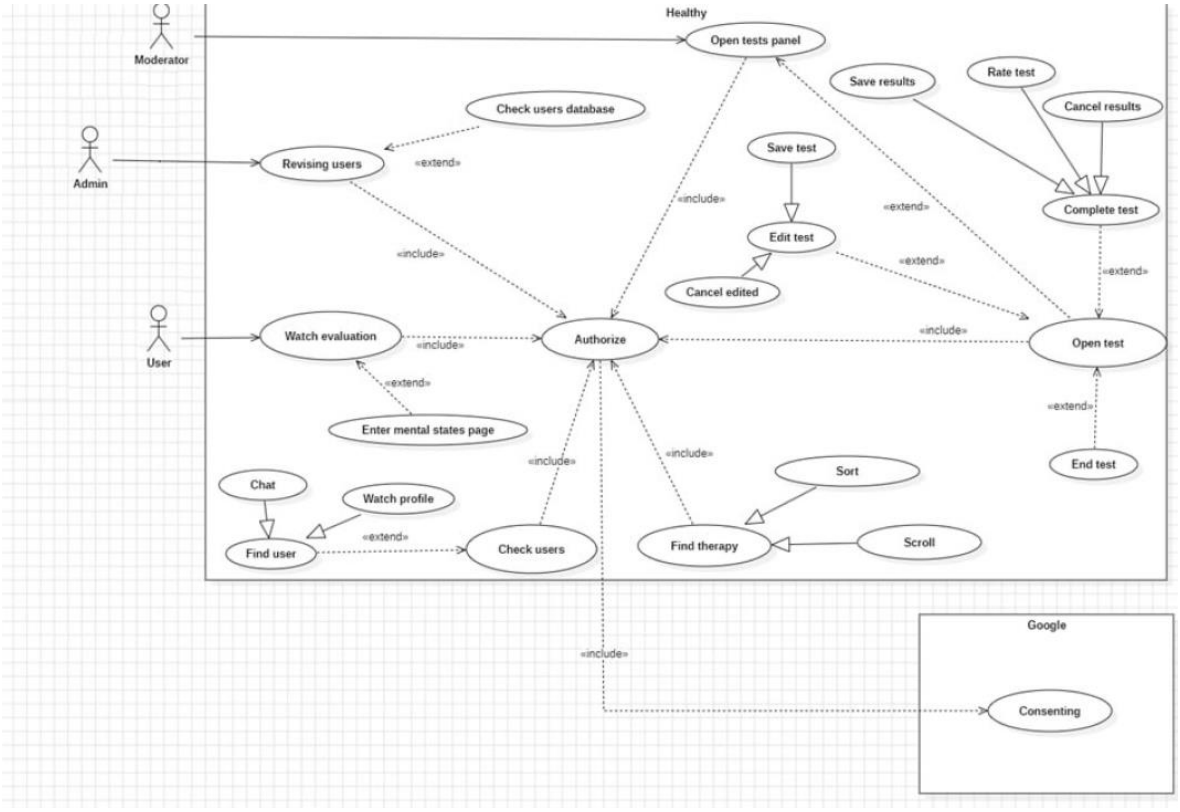


Рисунок 3.2 – Уніфікований сценарій використання

Загалом кінцевий користувач (актор) повинен мати приблизні опції використання, що відображені вище (рис. 3.2). Базове використання передбачає авторизацію, автентифікацію, перегляд списку тестів, проходження тестів,

призупинення тесту, перегляд бази знань арт-терапій, отримання рекомендацій, перегляд та редагування власного профілю і пошук користувачів, перегляд їх статистики, створення тестів для адміністраторів та модераторів.

3.2 UML-конструювання

Конструювання програмного забезпечення включає в себе деталізацію логіки реалізації компонентів системи, їх взаємодії, структури та поведінки. На основі трирівневої архітектури, описаної у попередньому підрозділі, створено відповідний набір UML-діаграм, що ілюструють структуру й динаміку ПЗ.

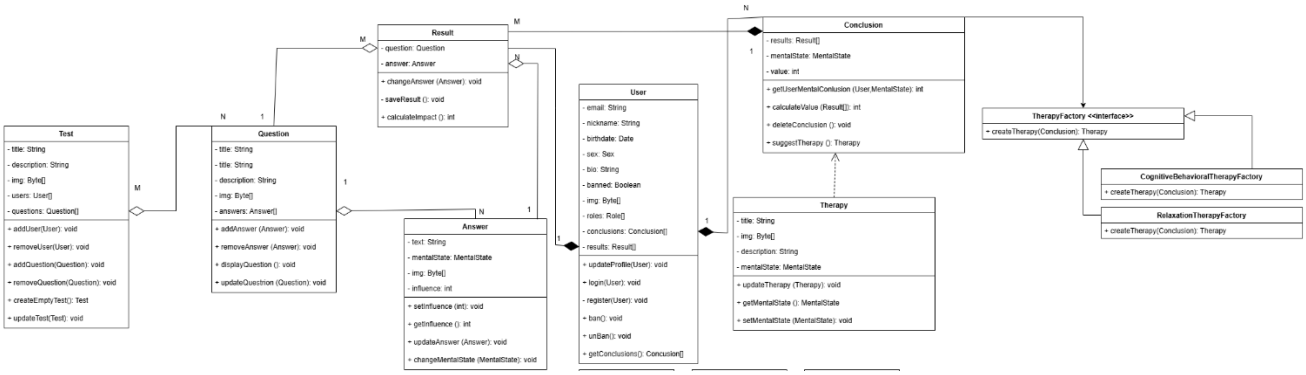


Рисунок 3.3 – Діаграма класів

На рисунку 3.3 представлена діаграма класів основних сутностей системи, які реалізують логіку створення, проходження та аналізу психологічних тестів, а також збереження й відображення результатів користувачам. Діаграма демонструє структуру класів, їх атрибути, методи, а також зв’язки між об’єктами. Класи та їх опис:

– Test – клас, що відповідає за тест як загальну одиницю. Містить назву, опис та зв’язок із множиною питань (Question). Забезпечує доступ до методів отримання, збереження та видалення питань, що входять до нього;

– Question – визначає окреме питання у межах тесту. Має текст питання, і список можливих відповідей (Answer). Пов'язаний із тестом через агрегацію;

– Answer – модель варіантів відповіді. Кожна відповідь має числове значення впливовості на ментальний стан та відноситься до конкретного питання;

– Result – клас, що описує результат проходження тесту. Містить зв’язок із конкретним користувачем (User) та набором відповідей на питання (Answer). Результат є основою для формування висновків (Conclusion);

– User – модель користувача системи. Містить основні ідентифікаційні атрибути, такі як email, ім’я, роль. Має зв’язки з результатами, висновками, рекомендованими терапіями. Також реалізує методи автентифікації;

– Conclusion – клас, що моделює висновки, отримані після обробки результатів. Містить висновки результатів, а також пропонує терапії;

– Therapy – модель терапевтичного методу, що включає назву, опис і категорію. Пов'язана з висновками, які її рекомендують;

– TherapyFactory – абстрактна фабрика, яка породжує конкретні типи терапій.

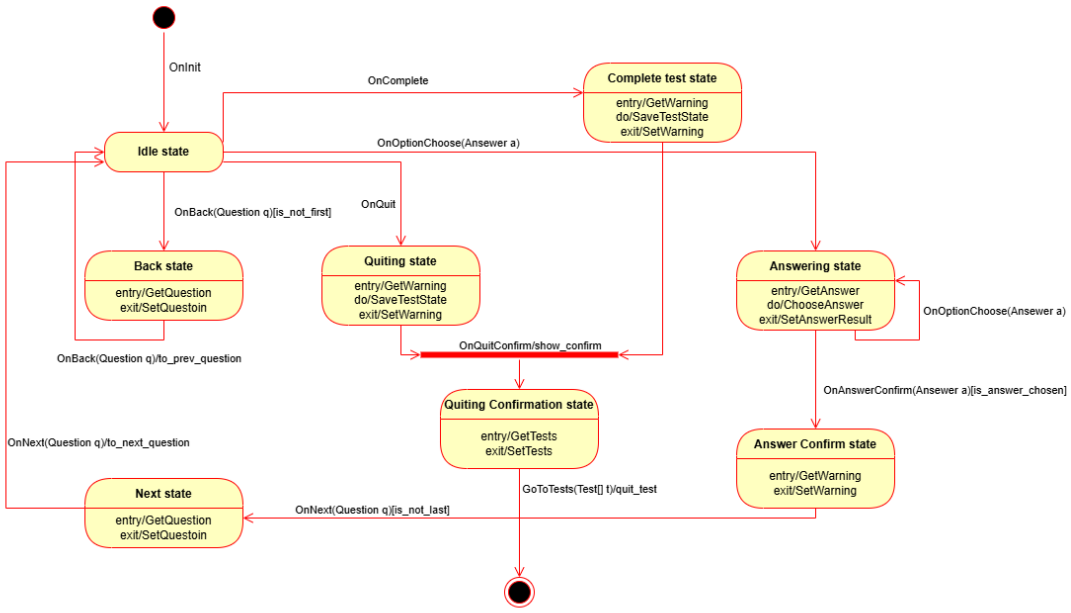


Рисунок 3.4 – Діаграма станів проходження тестів

Основним джерелом інформації про ментальний стан користувача у застосунку є тести. На рисунку 3.4 представлено діаграму станів, яка відображає поведінку користувача під час проходження психологічного тесту в системі. Діаграма моделює можливі стани, в яких може перебувати об’єкт під час процесу

проходження тесту, а також переходи між цими станами залежно від дій користувача.

Користувач може почати тестування, відповідати на запитання, підтверджувати вибрані відповіді, переходити до наступного або попереднього запитання, завершити тест або вийти з нього достроково. Система реагує на кожну дію, змінюючи стан: наприклад, при виборі відповіді переходить у стан підтвердження, після чого або продовжує тест, або завершує його. У разі спроби вийти з тесту система запитує підтвердження і, залежно від вибору, або завершує роботу, або повертається до тестування.

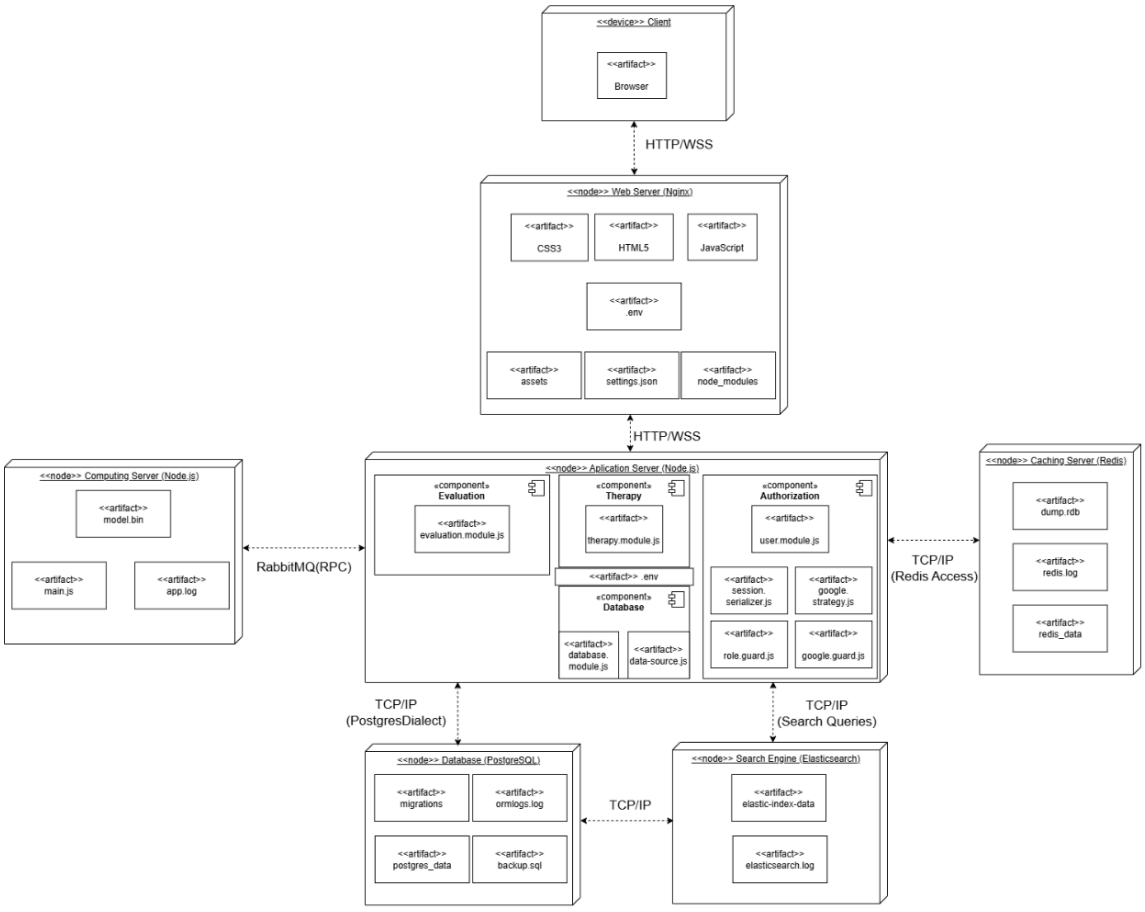


Рисунок 3.5 – Діаграма розгортання

Діаграма розгортання (рис. 3.5) ілюструє архітектуру вебзастосунку, який побудований за принципами мікросервісної, з використанням контейнеризації Docker. На верхньому рівні знаходиться клієнтська частина – веббраузер, через

який користувачі взаємодіють із системою. Уся взаємодія відбувається по HTTP або WebSocket-протоколу.

Клієнтська частина збирається у вигляді статичних ресурсів (HTML, CSS, JavaScript) і подається через NGINX, який виконує роль вебсервера [17]. NGINX розгорнутий в окремому контейнері, і забезпечує доступ до інтерфейсу користувача.

Основна бізнес-логіка розміщена у контейнері з NestJS. Контейнер Redis використовується як кеш-сервер для тимчасового збереження даних, таких як токени, сесії або проміжні результати, що дозволяє зменшити навантаження на основну базу даних. Окремий контейнер виділений під AI-компонент – це сервер на Node.js, який обробляє запити, надіслані з основного застосунку через RabbitMQ. Цей сервіс може виконувати складні обчислення, такі як переклад або генерація відповідей. Усі сервіси з'єднані в єдину Docker-мережу, що дозволяє їм взаємодіяти через вказані порти.

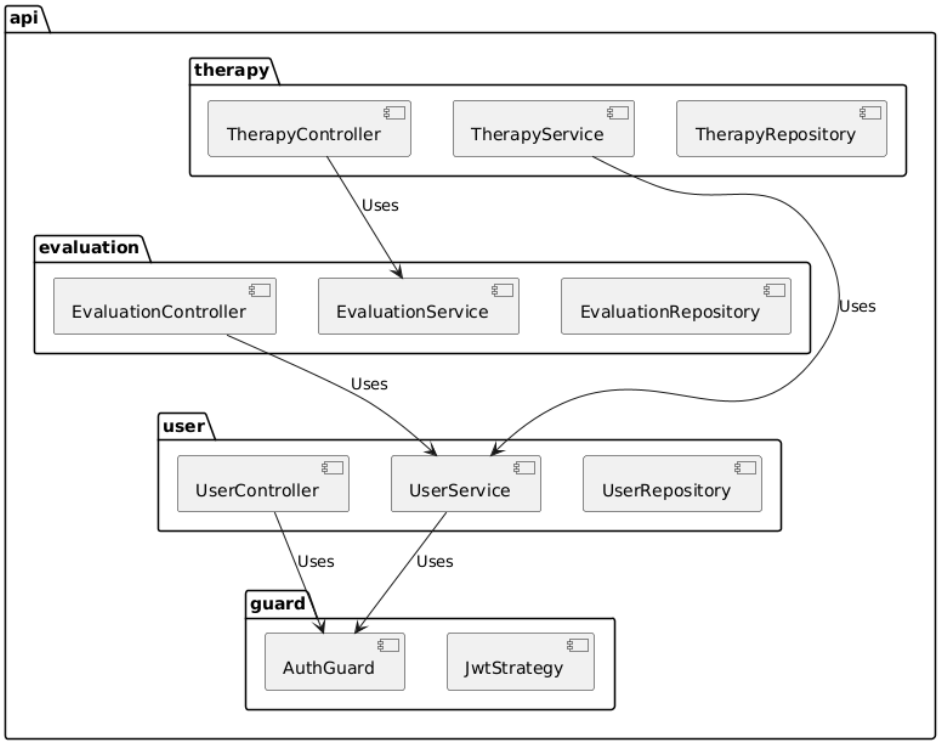


Рисунок 3.6 – Діаграма пакетів

Діаграма на рисунку 3.6 демонструє модульну архітектуру бекенд-застосунку на NestJS, де система поділена на чітко ізольовані доменні області: therapy, evaluation, user і guard.

Модуль therapy реалізує обробку логіки, пов’язаної з терапіями. Запити надходять до TherapyController, далі передаються в TherapyService, який взаємодіє з TherapyRepository для доступу до бази даних. Така ж структура використана і в evaluation – там є EvaluationController, EvaluationService та EvaluationRepository, що обробляють інформацію, пов’язану з оцінюванням.

Доцільним для цих модулів є використання наступних пакетів:

– @nestjs/common, @nestjs/core – основа фреймворку NestJS для структурування модулів, контролерів і сервісів;

– @nestjs/passport, passport, passport-google-oauth20 – для інтеграції з Passport.js та реалізації стратегій авторизації;

– @nestjs/typeorm, typeorm, pg – для роботи з базою даних PostgreSQL через ORM;

– @nestjs/config – для зчитування конфігурації з .env;

– @nestjs/swagger – для генерації OpenAPI-документації;

– class-validator, reflect-metadata – для валідації та типізації DTO.

3.3 Математична модель емоційного стану

У межах функціонування інтелектуального компонента системи, що надає рекомендації з арт-терапії на основі психологічного профілю користувача, реалізовано формалізований підхід до оцінювання емоційного стану. Ця модель дозволяє кількісно представити суб’єктивні емоційні показники у вигляді вектора параметрів, які змінюються в результаті проходження користувачем психологічних тестів. Кожен користувач має початковий стан, представлений множиною емоційних характеристик:

$$E = \{e_1, e_2, e_3, \dots, e_n\}, e_i \in [0{,}100] \quad (1)$$

$n = 15$, як кількість параметрів, таких як тривожність, апатія, страх, гнів, депресія тощо. На початку кожного сеансу кожен з цих параметрів має значення:

$$e_i^0 = 50, \forall_i \quad (2)$$

Під час проходження тестів користувач відповідає на питання, кожне з яких впливає на певні емоційні стани. Кожна відповідь q_j має вектор впливу $\Delta e_{i,j}$, який обмежений діапазоном:

$$\Delta e_{i,j} \in [-10, +10] \quad (3)$$

Сумарний вплив усіх відповідей на певний емоційний стан обчислюється як:

$$e_i = \min(100, \max(0, e_i^0 + \sum_{j=1}^{m} \Delta e_{i,j})), \quad (4)$$

де m – кількість питань у тесті. Результатом обчислень є вектор психологічного стану користувача:

$$E = [e_1, e_2, e_3, \dots, e_n] \in [0,100]^n \quad (5)$$

Цей вектор перетворюється у текстовий опис, який слугує частиною запита для LLM моделі samantha:mistral, реалізованої через локальний Ollama-сервіс [18]. Запит до моделі включає перелік доступних видів арт-терапій та актуальні значення емоційних показників.

3.4 Мокапи застосунку

Відповідно до вимог, інтерфейс застосунку повинен мети щонайменше дві кольорові теми та бути зрозуміли і легкодоступним для користувача.

Рисунок 3.7 – Домашня сторінка

Домашня сторінка застосунку має яскравий фон з геометричним візерунком. У верхній частині розташоване меню навігації. Основний контент сторінки містить заголовок "Арт-терапія для психічного здоров'я", короткий опис платформи, яка

пропонує психологічну допомогу через мистецтво, та кнопку "Розпочати" для переходу до авторизації, за її відсутності у користувача в поточній сесії.

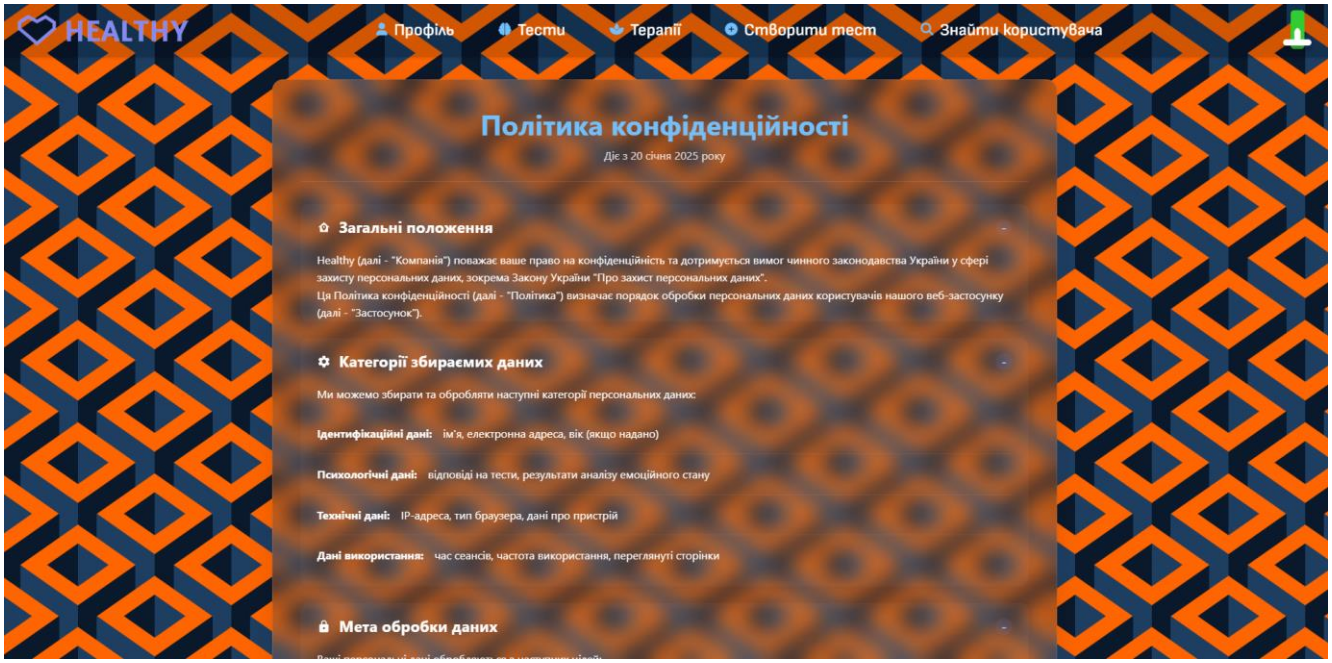


Рисунок 3.8 – Політики користування

Політика конфіденційності. Текст подано у зручному для читання форматі з маркованими списками й підзаголовками, що полегшує сприйняття юридичної інформації.

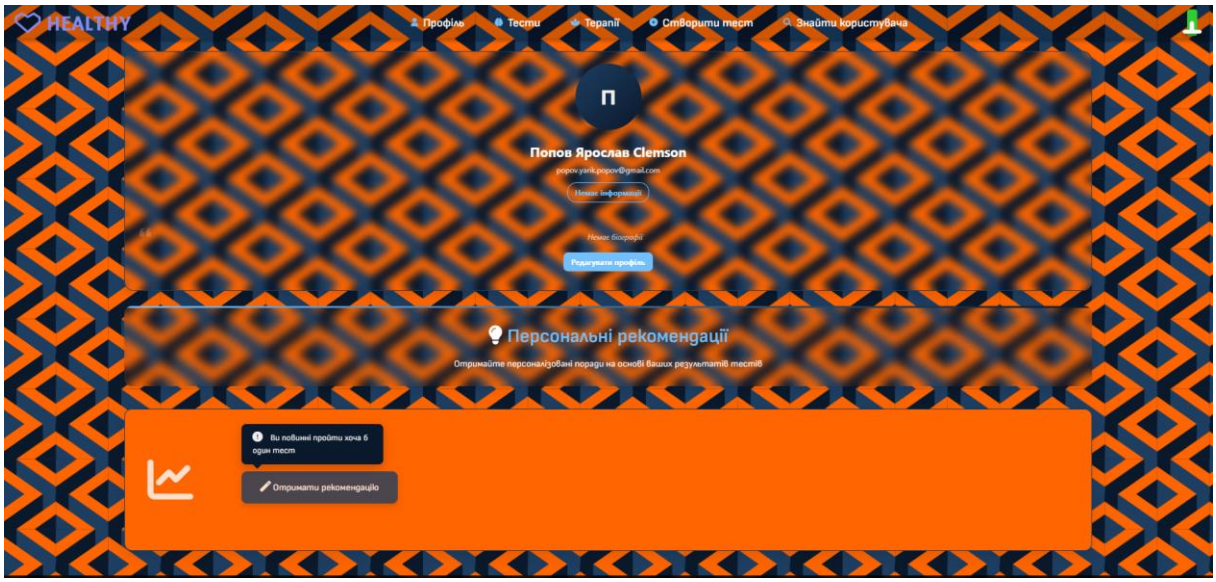


Рисунок 3.9 – Профіль користувача

Інтерфейс профілю містить ім’я користувача, аватар з ініціалом, кнопки для редагування профілю та перегляду налаштувань. У верхній частині є навігаційне меню з основними розділами застосунку. Нижче розташований блок персональних рекомендацій, де можна отримати поради на основі результатів.

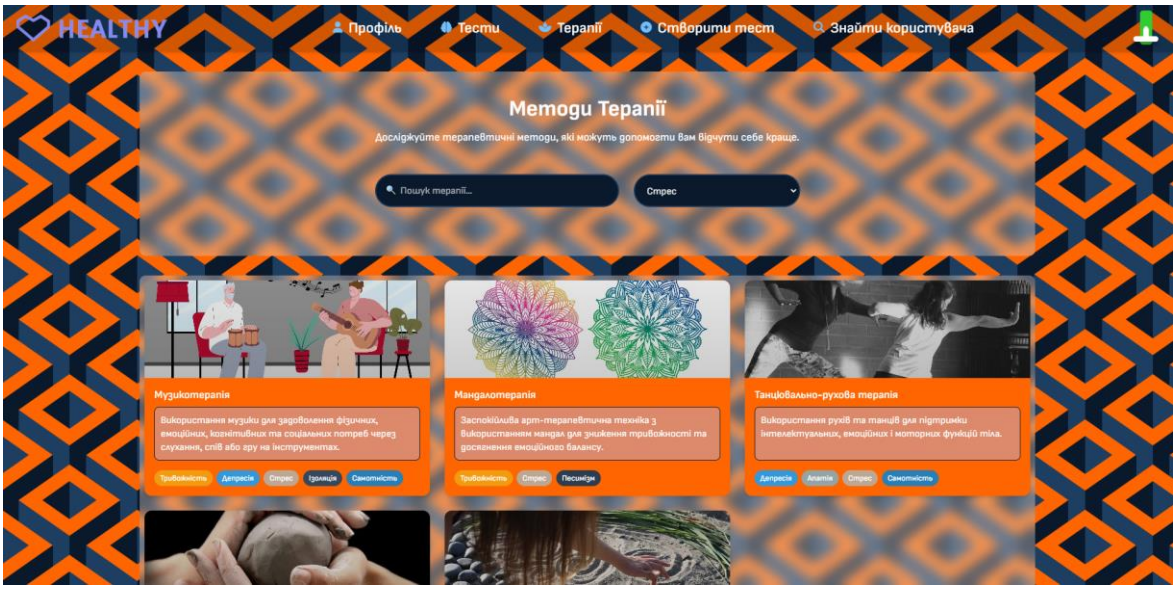


Рисунок 3.10 – База знань терапій

Методи терапії. Ця сторінка дозволяє користувачам ознайомитися з різними видами арт-терапевтичних методик. Вгорі розміщено заголовок «Методи Терапії» та підзаголовок з мотиваційним текстом. Нижче – поле для пошуку з фільтрацією за ключовим словом та типом.

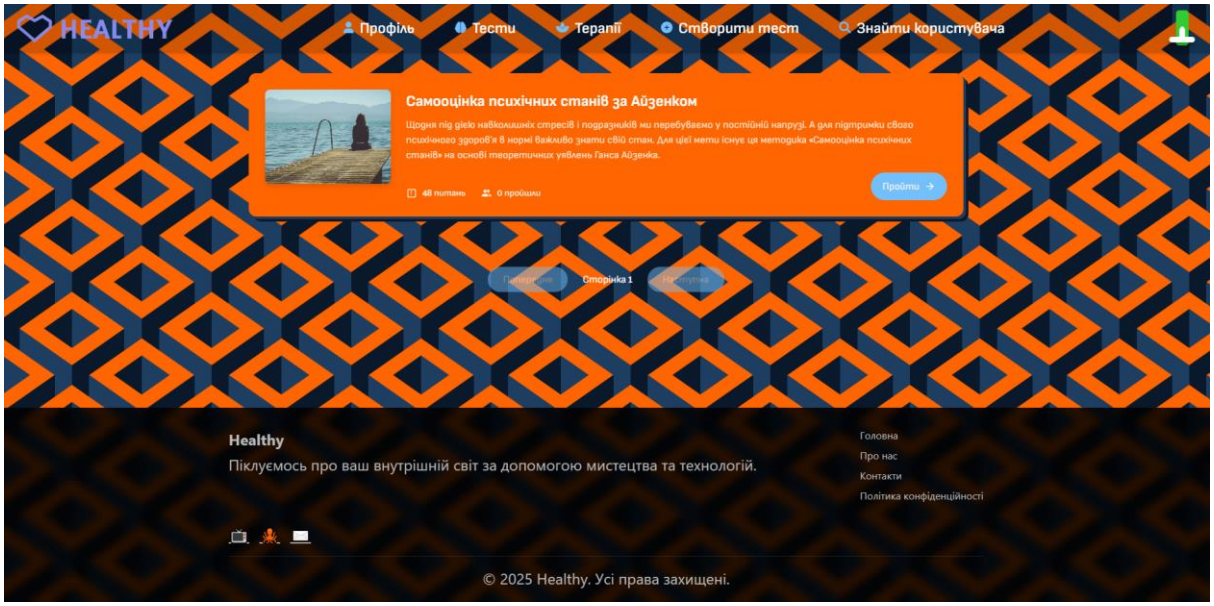


Рисунок 3.11 – Список тестів

Інтерфейс списку тестів показує окремі картки з описами психологічних тестів. У кожній картці є зображення, короткий опис тесту, кількість питань, кількість користувачів, що пройшли тест, а також кнопка для проходження. Наявна пагінація.

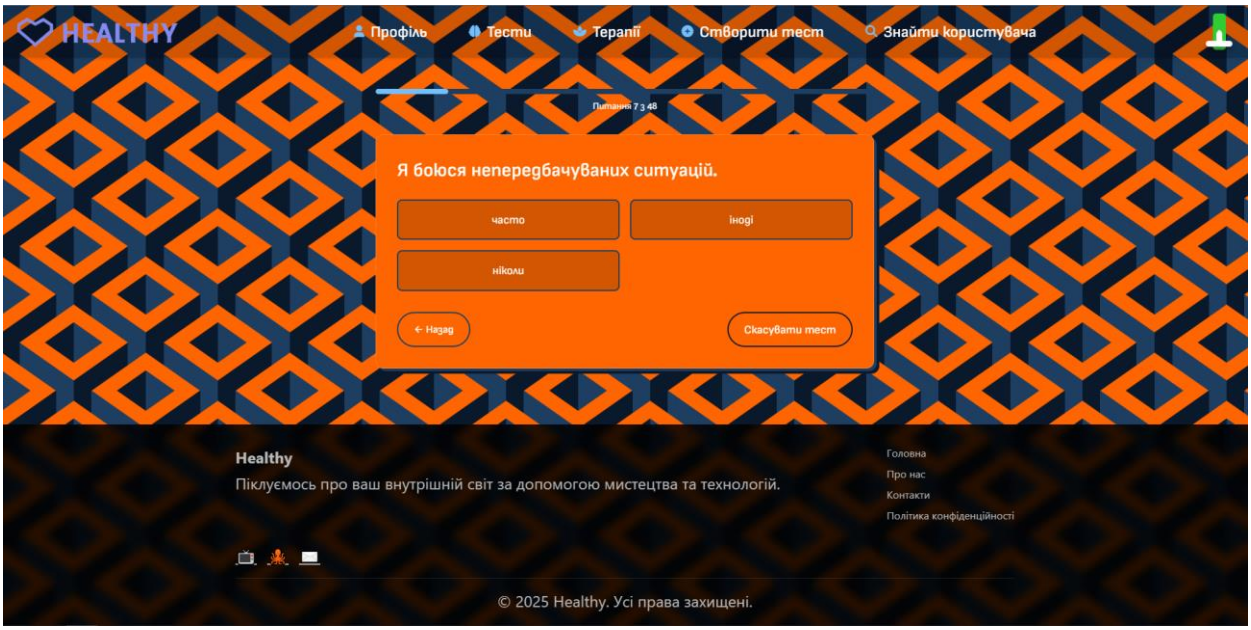


Рисунок 3.12 – Проходження тесту

Інтерфейс проходження тесту містить запитання з варіантами відповіді у вигляді кнопок. Угорі вказано номер питання з загальної кількості. Нижче є кнопки для переходу назад та завершення тесту.

Рисунок 3.13 – Ментальна карта (статистика) користувача

Інтерфейс статистики користувача відображає результати за різними психологічними шкалами у вигляді кольорових індикаторів прогресу. Кожна шкала має підпис і оцінку з максимуму. Візуально дані розташовані в сітці, що робить інформацію зручною для перегляду

Висновки до розділу 3

У розділі 3 всебічно розглянуто процес побудови програмного забезпечення, що реалізує оцінювання емоційного стану користувача.

Описано архітектуру ПЗ, що побудована на принципах мікросервісної з чітким поділом на фронтенд і бекенд, продемонстровано як компоненти взаємодіють між собою, описані головні класи застосунку, діаграми пакетів, діяльності, розгортання.

Продемонстровано, як реалізуються модулі, сервіси та контролери відповідно до архітектурних шаблонів. Наведено структуру проєкту, використані пакетні залежності та описано логіку роботи системи з елементами авторизації, обробки запитів та збереження даних. Розроблено підхід до аналізу вхідних даних, використання оціночних шкал.

Представлені мокапи інтерфейсу користувача, що дозволяють візуалізувати основні сценарії взаємодії з застосунком, структуру сторінок та взаємозв’язки між елементами інтерфейсу.

4 КОДУВАННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Формування бази даних

Підрозділ описує побудову та використання моделі бази даних, використання міграцій, формування ER-діаграми («сутність-зв’язок»). Також наявний опис використання технології повнотекстового пошуку та індексації.

Представлена ER-діаграма (рис. 4.1) описує структуру реляційної бази даних, яка лежить в основі застосунку для аналізу емоційного стану. Вона охоплює кілька функціональних зон: управління користувачами, питаннями, відповідями, результатами оцінювання та зв’язками між цими сутностями.

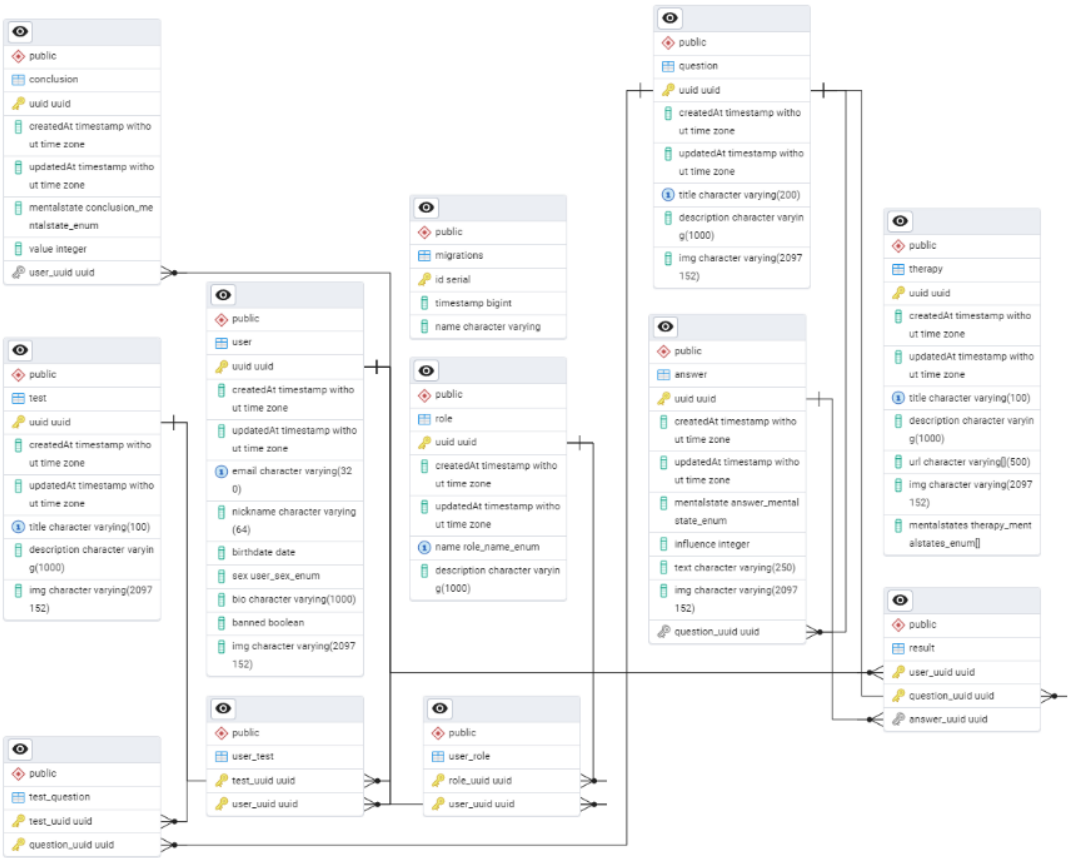


Рисунок 4.1 – ER-діаграма

У центрі логіки знаходиться таблиця users, яка зберігає облікову інформацію користувачів, включно з email, логіном, паролем, описом, а також типом доступу. З нею напряму пов’язані результати тестувань, оцінки, відповіді та запити.

Таблиці question, answer та test та реалізують модель тестових запитань. question агрегує групу питань, що зберігається в test, і призначається користувачеві через зв’язки.

Таблиця result реєструє метаінформацію щодо оцінювання. Поєднання зовнішніх ключів користувача, відповіді та питання у єдиний кластерний ідентифікатор надає змогу ефективно поєднувати інформацію із різних таблиць.

Таблиця conclusion слугує допоміжною у формування запиту до генерації відповіді ШІ на основі відповідних атрибутів mentalstate та influence.

Для налаштування та автоматичного розгортання структури бази даних використано механізм міграцій TypeORM. Це дозволяє централізовано створювати, оновлювати та відкочувати зміни до схеми БД без втручання вручну.

У файлі entity.ts описується сутність для використання з TypeORM, яка відображає структуру таблиці в базі даних. Кожне поле класу відповідає колонці таблиці, а декоратори @Entity(), @PrimaryGeneratedColumn(), @Column() вказують TypeORM, як саме цю сутність треба зберігати. Генерація міграцій з використанням CLI TypeORM (typeorm migration:generate) автоматично фіксує зміни в таких сутностях для створення або оновлення відповідних таблиць у базі даних.

Лістинг коду прикладу entity.ts:

```
@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;
  @Column()
  name: string;
  @Column({ unique: true })
  email: string;
  @Column({ default: true })
  isActive: boolean;
}
```

Цей клас описує таблицю User з полями id, name, email, isActive. Після створення або оновлення такої сутності, викликається команда typeorm migration:generate, яка генерує SQL-інструкції для оновлення структури бази.

Redis застосовується як інструмент кешування для пришвидшення доступу до часто використовуваних даних, зменшуючи навантаження на основну базу

даних. Завдяки збереженню результатів запитів у пам’яті, Redis дозволяє значно скоротити час відповіді при повторному зверненні.

Лістинг коду використання кешування:

```
export async function cacheSet(key: string, value: any, ttlSeconds = 300) {
  await redis.set(key, JSON.stringify(value), "EX", ttlSeconds);
}
export async function cacheGet(key: string): Promise<any | null> {
  const data = await redis.get(key);
  return data ? JSON.parse(data) : null;
}
```

Elasticsearch використовується для реалізації повнотекстового пошуку, що дає змогу ефективно обробляти запити користувача по великих обсягах текстової інформації. Його можливості з індексації документів та гнучке ранжування результатів дозволяють забезпечити швидкий і релевантний пошук у системі.

Лістинг коду індексації атрибута:

```
export async function indexDocument(index: string, id: string, body: any) {
  await client.index({ index, id, body });
  await client.indices.refresh({ index });
}
```

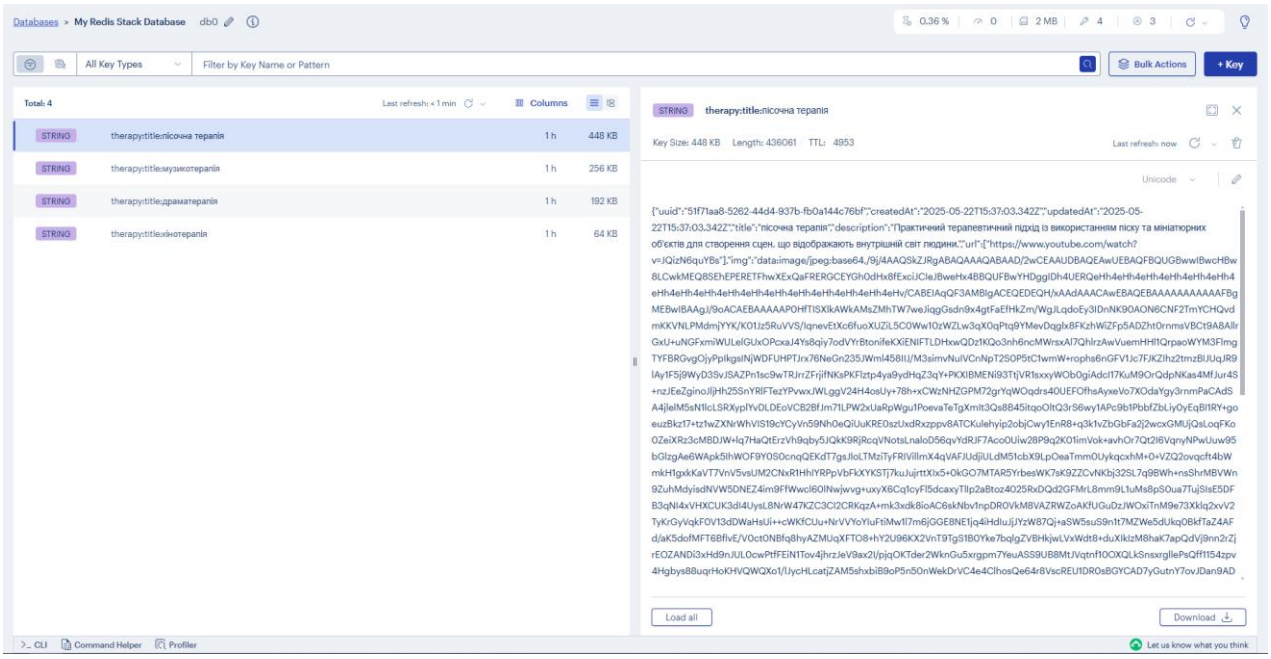


Рисунок 4.2 – Redis Cache Manager

Redis Cache Manager (рис. 4.2) – це менеджер управління кешем у Redis, зазвичай використовується у вебзастосунках для зменшення навантаження на базу

даних і пришвидшення відповіді системи. Він дозволяє легко додавати, оновлювати, видаляти та автоматично протерміновувати кешовані значення.

4.2 Реалізація серверної частини застосунку

Наступний крок після формування структури бази даних – реалізація серверної частини застосунку, яка дозолить відмовостійко звертатися до власного API і обробляти запити швидко і коректно.

Серверна частина застосунку побудована на базі фреймворку Nest.js, який реалізує модульний підхід до організації коду, спираючись на архітектурні принципи Domain-Driven Design та Inversion of Control. Такий підхід дозволяє ефективно розділяти відповідальність між різними функціональними частинами системи, сприяє масштабованості та спрощує підтримку коду.

Основною одиницею структурування в Nest.js є модуль (module), який інкапсулює логіку певного домену. У межах кожного модуля виділяються такі складові:

– контролери (controllers) – відповідають за обробку HTTP-запитів і маршрутизацію;

– сервіси (services) – містять бізнес-логіку, більш детальну обробку даних, які надходять із запиту;

– DTO (Data Transfer Object) – структури даних для валідації та передачі інформації;

– інтерфейси та сутності (interfaces, entities) – визначають структуру даних, що використовуються в межах модуля;

– мідлвари, пайпи, гварди, інтерсептори – додаткові елементи для обробки запитів, авторизації, трансформації даних тощо. Містять фільтри пропуску та зазвичай перехоплюють метадані запиту.

Лістинг коду DTO отримання результатів:

```
export class SubmitResultDto {
  @IsUUID()
  question_uuid: string;

  @IsUUID()
```

```
  answer_uuid: string;
}

export class SubmitResultsDto {
  @IsUUID()
  test_uuid: string;

  @ValidateNested({ each: true })
  @Type(() => SubmitResultDto)
  results: SubmitResultDto[];
}
```

Модулі мають чіткі залежності і можуть повторно використовуватись у різних частинах системи. Наприклад, модуль User відповідає за автентифікацію та керування користувачами, модуль Evaluation – за проходження тестів, Therapy – за рекомендації тощо. Деякі модулі можуть використовуватися для з’єднання із зовнішніми сервісами, такими як кешування, база даних, брокери повідомлень або мікросервіси. Визначення модуля використовує декоратор, який приймає відповідні аргументи.

Лістинг коду модуля бази даних:

```
@Module({
  imports: [
    TypeOrmModule.forRootAsync({
      useFactory: (configService: ConfigService) => ({
        type: "postgres",
        host: configService.get("DATABASE_HOST"),
        port: configService.get<number>("DATABASE_PORT"),
        username: configService.get("DATABASE_USER"),
        password: configService.get("DATABASE_PASSWORD"),
        database: configService.get("DATABASE_NAME"),
        entities: [__dirname + "/../**/*.entity.{ts,js}"],
        synchronize: false,
        autoLoadEntities: true,
        migrations: [__dirname + "/migrations/**/*.{ts,js}"],
      }),
      inject: [ConfigService],
    }),
  ],
})
export class DatabaseModule {}
```

Для розмежування API між версіями реалізовано механізм версіонування. Кожна версія API інкапсулюється у відповідну папку або namespace, наприклад: /v0.1/user, /v0.2/user. Це дозволяє підтримувати кілька версій API одночасно, не ламаючи поточну функціональність. Версіонування реалізується як через URL-

префікси, так і за допомогою декларативного контролю версій у контролерах (@Version('0.1')), що підтримується Nest.js з коробки.

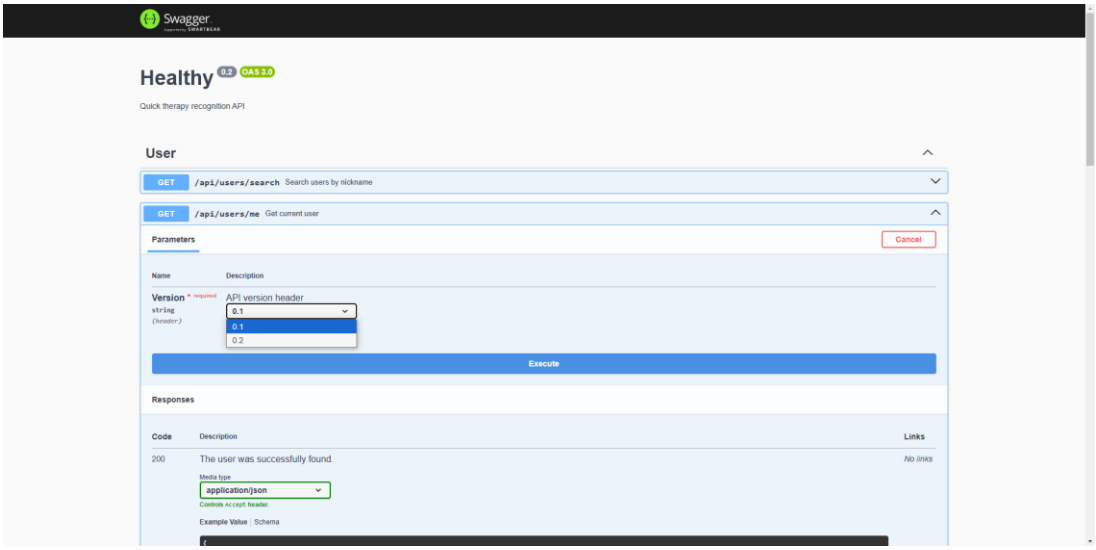


Рисунок 4.3 – Версіонування та документація, Swagger

Крім того, Nest.js підтримує автоматичну генерацію документації за допомогою Swagger (рис. 4.3), що значно полегшує комунікацію з фронтендом та сторонніми розробниками (Додаток А).

Контролери є ключовим елементом, відповідальним за прийом вхідних HTTP-запитів, обробку маршрутів та делегування виконання відповідним сервісам. Вони виступають посередниками між зовнішнім інтерфейсом (API) і внутрішньою бізнес-логікою системи. Кожен контролер відповідає за певну доменну область, наприклад:

– UserController обробляє запити на авторизацію, автентифікацію, авторизацію користувачів, роботу з профілем;

– EvaluationController керує запитами, пов’язаними з тестами, їх створення, проходження, перегляд результатів;

– TherapyController опрацьовує запити на отримання рекомендацій арт-терапій.

Контролери визначають відповідні маршрути за допомогою декораторів @Controller(), @Get(), @Post(), @Put(), @Delete() та інших. Вони приймають параметри маршруту, запити з тіла, заголовки, query-параметри тощо. Nest.js

дозволяє використовувати декоратори @Param(), @Body(), @Query() тощо для інжекції необхідних даних до методів контролерів (Додаток Б).

У проєкті реалізовано механізм автентифікації та авторизації на основі OAuth 2.0 з інтеграцією Google, що забезпечує безпечний та зручний спосіб входу користувачів до системи. Для цього використовується модуль @nestjs/passport разом зі стратегією passport-google-oauth20.

Основна автентифікаційна стратегія реалізована у класі GoogleStrategy, який наслідується від PassportStrategy. У конструкторі задаються необхідні параметри OAuth – clientID, clientSecret, callbackURL та області доступу (scope). Після проходження Google-автентифікації, у методі validate відбувається перевірка користувача в локальній системі за допомогою сервісу UserService. Якщо користувач існує, він повертається, і його сесія буде збережена. Якщо ні – система може створити нового користувача або відмовити у доступі.

Лістинг коду:

```
@Injectable()
export class GoogleStrategy extends PassportStrategy(Strategy) {
  constructor(private readonly userService: UserService) {
    super({
      clientID: process.env.CLIENT_ID,
      clientSecret: process.env.CLIENT_SECRET,
      callbackURL: process.env.CALLBACK_URL,
      scope: ["profile", "email"],
      passReqToCallback: null,
    });
  }
  async validate(_a: string, _r: string, profile: Profile) {
    const user = await this.userService.validateUser({
      email: profile.emails[0].value,
      nickname: profile.displayName,
    });
    return user || null;
  }
}
```

Для підтримки сесій використовується SessionSerializer, який реалізує PassportSerializer. У методі serializeUser дані користувача зберігаються в сесії, тоді як deserializeUser дозволяє отримати повну інформацію про користувача на основі його ідентифікатора (uuid), включаючи ролі, що є необхідним для подальшої авторизації.

Лістинг коду:

```
@Injectable()
export class SessionSerializer extends PassportSerializer {
  constructor(private readonly userService: UserService) {
    super();
  }
  serializeUser(user: User, done: Function) {
    done(null, user);
  }
  deserializeUser(payload: User, done: Function) {
    const user = firstValueFrom(
      this.userService.findById(payload.uuid, ["roles"])
    );
    return user ? done(null, user) : done(null, null);
  }
}
```

Система підтримує рольову модель доступу, що реалізується через Nest.js Guards. Для обмеження доступу до окремих маршрутів використовуються гварди (RolesGuard), які перевіряють наявність відповідної ролі у користувача (наприклад, admin, moderator, user). Це дає змогу реалізувати багаторівневу авторизацію для окремих частин API.

Крім того, конфігурація сесійної авторизації забезпечує підтримку стійкої ідентифікації користувача протягом усього періоду взаємодії із системою, навіть при зміні сторінки чи оновленні.

Взаємодія з мовною моделлю Ollama в системі реалізована через окремий мікросервіс, який обробляє запити до AI у фоновому режимі (Додаток В). Бекенд Nest.js надсилає сформований запит у чергу RabbitMQ, яку слухає AI-сервіс. Отримавши повідомлення, сервіс передає запит до локального API Ollama, де розгорнута модель samantha-mistral, отримує згенеровану відповідь і повертає її назад у відповідну чергу. Завдяки використанню черг та ідентифікаторів кореляції забезпечується асинхронна, надійна та масштабована взаємодія, що дозволяє системі працювати без блокувань і затримок при генерації аналітичних або рекомендаційних відповідей на основі емоційного стану користувача (рис. 4.4). За потреби відповідь LLM перекладається для україномовних користувачів.

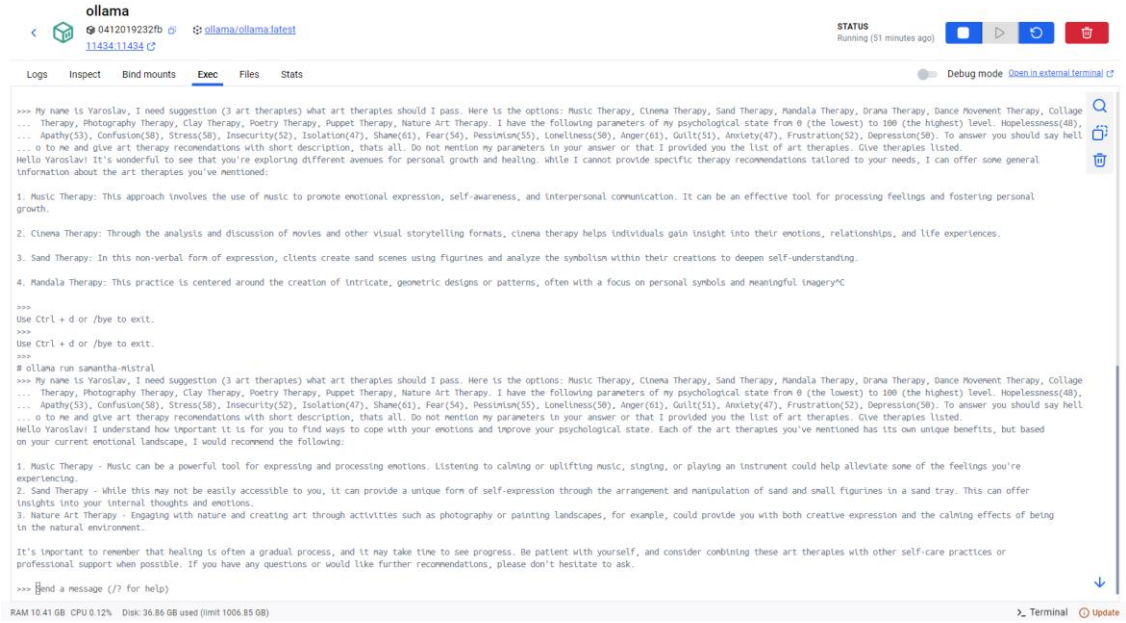


Рисунок 4.4 – Відповідь LLM на запит

У системі реалізовано окремий модуль для автоматичного перекладу текстових відповідей, отриманих від мовної моделі (Додаток Д). Цей сервіс приймає POST-запити з текстом та цільовою мовою перекладу, після чого використовує бібліотеку gtranslate для виконання перекладу з автоматичним визначенням мови джерела. Після успішного перекладу текст повертається у форматі JSON у відповідь на запит. Це дозволяє адаптувати відповіді AI до мови користувача, зберігаючи зручність і зрозумілість взаємодії незалежно від мовних бар’єрів.

4.3 Розробка клієнтської частина застосунку

Після розробки серверної частини приділяється увага реалізації інтерфейсу, розробці клієнтських сторінок. Вони використовують всю функціональність реалізованого бекенду.

Клієнтська частина застосунку реалізована на Angular 17+ з використанням сучасного підходу – standalone компонентів, які не потребують оголошення в NgModule. Кожен компонент має власну структуру з шаблоном, стилями та логікою, а також самостійно імпортує необхідні модулі (наприклад, CommonModule, FormsModule, HttpClientModule, UI-бібліотеки). Це підвищує

ізольованість компонентів, спрощує повторне використання і пришвидшує ініціалізацію застосунку.

Лістинг коду standalone компонента:

```
@Component({
  selector: 'app-footer',
  templateUrl: './footer.component.html',
  styleUrls: ['./footer.component.css'],
  imports: [RouterLink],
})
export class FooterComponent {
  year: number = new Date().getFullYear();
}
```

Проєкт має функціональну структуру, в якій кожен домен або функціональний розділ (наприклад, error, tests, profile) має власну директорію з окремими компонентами, сервісами, типами, утилітами. Це забезпечує високу модульність та легкість підтримки коду.

Лістинг коду сервісу обробки питань:

```
@Injectable({
  providedIn: 'root',
})
export class QuestionService {
  private apiUrl = `http://localhost:${environment.API_PORT}/${environment.A
PI_PREFIX}/questions`;
  constructor(private http: HttpClient) {}

  getQuestionById(id: string): Observable<Question> {
    return this.http.get<Question>(`${this.apiUrl}/${id}`);
  }
  submitResults(results: SubmitResult[], test_uuid: string): Observable<any>
{
    return this.http.post(`${this.apiUrl}/submit-results`, {
      test_uuid,
      results,
    });
  }
}
```

Сервіси використовують providedIn: 'root' для глобального впровадження через Angular Dependency Injection. Це дозволяє використовувати їх у будь-якому компоненті без необхідності повторного імпорту. Сервіси відповідають за логіку роботи з API, локальне збереження, стан користувача тощо.

Проєкт повністю типізований за допомогою TypeScript. Використовуються інтерфейси для опису структури даних (наприклад, тестів, результатів, сесій), а

також утилітні типи для генерації динамічних структур, таких як словники результатів чи форм.

Маршрутизація в застосунку реалізована на основі вбудованого роутера Angular. Основна конфігурація маршрутів зберігається у вигляді масиву об'єктів типу Routes, які прив’язують URL-шляхи до відповідних компонентів. Оскільки в проєкті використовуються standalone компоненти, кожен з них напряму імпортується у маршрути, без необхідності модульної обгортки.

До основних сторінок відносяться головна (home), профіль користувача (me), список тестів, перегляд терапій та окремі сторінки для створення тестів і перегляду користувацької статистики. Для глибшої навігації використовуються вкладені маршрути (наприклад, therapies/:name або tests/:id), які дозволяють деталізувати функціональність без дублювання логіки.

Для захисту приватного функціоналу реалізовано кілька guard-функцій:

– authGuard – перевіряє, чи автентифікований користувач;

– roleGuard – перевіряє, чи належить користувач до певної ролі (наприклад, admin або moderator), що дозволяє обмежувати доступ до адміністративних функцій;

– unsavedTestGuard – перехоплює вихід зі сторінки тестування, якщо є незбережені зміни, та запитує підтвердження дій користувача.

Сторінка помилок (error) підтримує як динамічну передачу параметрів (код помилки, повідомлення, іконка тощо), так і статичне відображення універсальної помилки. Для невалідних шляхів застосунок перенаправляє користувача на домашню сторінку завдяки маршруту **.

Лістинг коду вкладеного маршруту:

```
        {
          path: 'therapies',
          children: [
            { path: '', component: TherapyListComponent },
            { path: ':name', component: TherapyComponent, canActivate: [authGuard]
},
          ],
        }
```

Це дозволяє відобразити список усіх терапій на шляху /therapies та окрему терапію за іменем – наприклад, /therapies/music.

У застосунку використовується бібліотека NgRx – реактивне сховище стану для Angular, побудоване за принципами Redux. NgRx дозволяє централізовано керувати станом застосунку, зберігаючи його у сховищі (store), та реагувати на дії (actions), що викликають зміну стану через редуктори (reducers) та побічні ефекти (effects).

NgRx використовує уніфіковану модель: дані зберігаються в єдиному об'єкті стану, змінюються виключно через actions, а логіка зміни стану описана в reducers. У застосунку реалізовано окремий slice стану для користувача.

Лістинг коду user.actions.ts:

```
export const loadUser = createAction('[User] Load Me');

export const loadUserSuccess = createAction(
  '[User] Load Me Success',
  props<{ user: User }>()
);

export const loadUserFailure = createAction(
  '[User] Load Me Failure',
  props<{ error: any }>()
);
```

Файл user.effects.ts обробляє асинхронні операції – звернення до API сервісу користувача. За допомогою createEffect() NgRx відстежує відповідні actions, виконує запити через UserService та диспатчить дії залежно від результату.

Лістинг коду user.effects.ts:

```
return actions$.pipe(
      ofType(loadUser),
      switchMap(() =>
        userService.getMe().pipe(
          map((user) => loadUserSuccess({ user })),
          catchError((error) =>
            of(loadUserFailure({ error: error?.message || 'Unknown error' }))
          )
        )
      )
    );
```

user.reducer.ts визначає, як змінюється стан UserState при виконанні певних дій. Наприклад, коли надходить дія loadUser, вмикається індикатор завантаження.

Лістинг коду user.reducer.ts:

```
  on(loadUser, (state) => ({ ...state, loading: true, error: null })),
  on(loadUserSuccess, (state, { user }) => ({
    ...state,
    user,
    loading: false,
  })),
  on(loadUserFailure, (state, { error }) => ({
    ...state,
    error,
    loading: false,
  }))
```

Селектори дозволяють отримувати конкретні частини стану з глобального сховища, не прив’язуючись до структури стану в компонентах.

Лістинг коду user.selectors.ts:

```
export const selectUserState = createFeatureSelector<UserState>('user');


export const selectUser = createSelector(
  selectUserState,
  (state) => state.user
);


export const selectUserLoading = createSelector(
  selectUserState,
  (state) => state.loading
);

export const selectUserError = createSelector(
  selectUserState,
  (state) => state.error
);
```

Клієнтський інтерфейс (рис. 4.5) представляє зручне використання застосунку у декілька кроків, що визначаються наступною послідовністю:

– вхід до застосунку (домашня сторінка);

– авторизація та автентифікація (кнопка розпочати);

– перехід до тестів (навігаційна панель);

– проходження тестів (інтуїтивно зрозумілі відповіді на питання);

– перехід до профілю та отримання рекомендацій;

– перехід до терапій за бажанням користувача.

Рисунок 4.5 – Навігаційна панель та рекомендації

Основні сторінки користування наведені раніше, у якості мокапів, адже вони наближені до результатів виконаної розробки клієнтських інтерфейсів.

4.4 Тестування якості програмного забезпечення

Після розробки застосунку потрібно надати оцінку якості, яка свідчить про якість написаного коду. Це дає змогу приблизно визначити які модулі або функції менш відмовостійкі, які треба покращити.

Тестування з використанням Lighthouse дозволяє отримати оцінку швидкості завантаження сторінки, визначити проблеми доступності для користувачів з обмеженими можливостями, перевірити, чи відповідає сайт стандартам прогресивних вебзастосунків (PWA), ідентифікувати SEO-проблеми та рекомендації щодо покращень у загальній продуктивності (рис. 4.6).

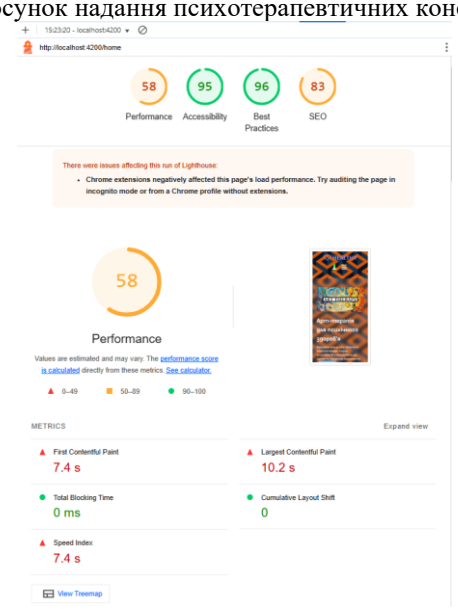


Рисунок 4.6 – Lighthouse

На рисунку 4.7 показано звіт про покриття тестами (coverage) для застосунку, імовірно написаного з використанням фреймворку Nest.js. Звіт охоплює покриття коду тестами за такими метриками: Statements, Branches, Functions, Lines.

Рисунок 4.7 – Покриття тестами серверної частини

Клієнтська частина протестована на адаптивність інтерфейсів до різних пристроїв. Це допомагаєм упевнитися у тому, що браузер правильно передає вигляд застосунка мобільним користувачам. Основні сторінки, підпорядковані мануальному тестуванню інтерфейсу застосунку продемонстровані на рисунку 4.8.

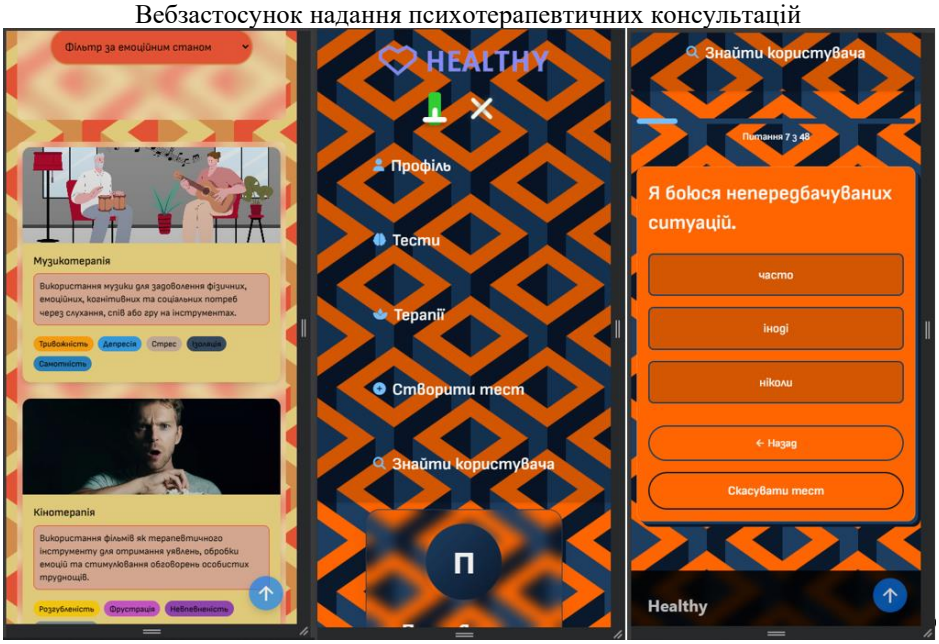


Рисунок 4.8 – Мануальне тестування

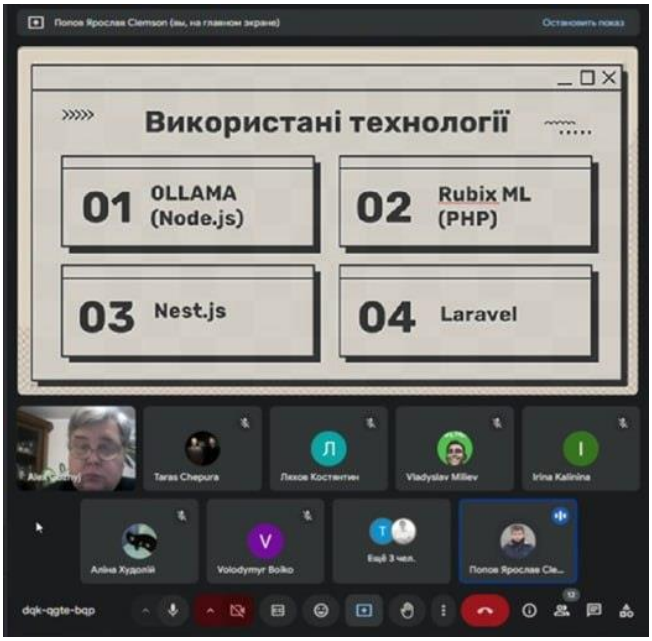


Рисунок 4.9 – Апробація

Апробація результатів (рис. 4.9). Вебзастосунок надання психотерапевтичних консультацій представлено на науково-практичній конференції молодих вчених, аспірантів і студентів (2-4 грудня 2024р., ЧНУ ім. Петра Могили) (Додаток Е) [19].

4.5 Контейнеризація застосунку

Контейнеризація дозволяє ізолювати сервіси застосунку, забезпечити портативність середовища розробки, тестування та розгортання. У цьому проєкті для контейнеризації використано Docker та систему оркестрації docker-compose, яка визначає сервіси, мережі та точки збереження даних (volumes) в одному конфігураційному файлі.

Файл docker-compose.yaml (Додаток Ж) описує запуск усіх необхідних сервісів для застосунку. Усі сервіси працюють у спільній мережі з типом bridge, що забезпечує їхню взаємодію за DNS-іменами контейнерів. Основні сервіси:

– postgres, postgres_slave – основна та реплікована база даних PostgreSQL;

– pgadmin – вебінтерфейс для керування PostgreSQL;

– redis – in-memory сховище даних, також із RedisStack UI;

– rabbitmq – брокер повідомлень з вебінтерфейсом;

– elasticsearch – пошукова система, використовується для повнотекстового пошуку.

– api – бекенд, написаний на NestJS;

– client – фронтенд на Angular, зібраний і поданий через NGINX вебсервер;

– ai – мікросервіс, що виконує AI-обчислення, взаємодіє з RabbitMQ та Ollama;

– ollama – модельний inference-сервер;

– translator – окремий сервіс для перекладу тексту.

У складі системи використано декілька індивідуальних сервісів, кожен із яких має власний Dockerfile. Образ базується на node:22-alpine – це легковажна версія Node.js. Виконується встановлення залежностей із package.json. Копіюється весь вихідний код і запускається команда npm run build, яка створює фінальну збірку-дистрибутив.

Лістинг коду Dockerfile для API та клієнта:

```
FROM node:22-alpine as build
WORKDIR /usr/src/app
COPY ./client/package*.json ./
RUN npm install
```

```
COPY ./client .
RUN npm run build
FROM nginx:alpine
COPY --from=build /usr/src/app/dist/client/browser /usr/share/nginx/html
COPY ./client/nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE ${CLIENT_PORT:-4200}
CMD ["nginx", "-g", "daemon off;"]

FROM node:22-alpine as build
WORKDIR /usr/src/app
COPY ./api/package*.json ./
RUN npm install
COPY ./api .
RUN npm run build
FROM node:22-alpine
WORKDIR /usr/src/app
COPY --from=build /usr/src/app/dist ./dist
COPY --from=build /usr/src/app/node_modules ./node_modules
COPY ./api/package*.json ./
EXPOSE ${API_PORT:-3000}
CMD ["node", "dist/src/main.js"]
```

Ці файли описують процес створення Docker-образу для кожного сервісу – API (NestJS), клієнтського застосунку (Angular), перекладача тощо.

Висновки до розділу 4

У розділі 4 реалізовано всі основні компоненти застосунку. База даних побудована з урахуванням індексації, кешування через Redis і повнотекстового пошуку за допомогою ElasticSearch. Серверна частина реалізована на NestJS із підтримкою тестування та масштабування. Клієнтська частина створена з використанням Angular для забезпечення зручного інтерфейсу. Якість програмного забезпечення перевірено як автоматизованим, так і мануальним тестуванням. Завершальним етапом стала контейнеризація, яка забезпечує просте розгортання і портативність застосунку. Застосунок апробовано на науково-практичній конференції молодих вчених, аспірантів і студентів.

У процесі розробки особлива увага приділялася відповідності сучасним вимогам до продуктивності, надійності та безпеки. Використання актуальних технологій дозволило створити масштабований застосунок, готовий до подальшого розвитку та інтеграції з іншими сервісами. Це закладає надійну основу для майбутнього комерційного або наукового використання системи.

ВИСНОВКИ

У кваліфікаційній бакалаврській роботі здійснено повний цикл створення програмного забезпечення для оцінки психологічного стану користувачів із застосуванням методів арт-терапії та сучасних технологій. На першому етапі проведено глибокий аналіз предметної області, досліджено існуючі рішення, особливості використання арт-терапії та її ефективність у психологічній діагностиці. Далі сформульовано вимоги до системи, побудовано функціональні та інформаційні моделі, а також розглянуто можливості застосування штучного інтелекту у сфері психології.

У проєктній частині описано архітектуру програмного забезпечення, розроблено наближені інтерфейси користувача, здійснено математичну інтерпретацію оцінки емоційного стану та створено прототип інтерфейсу. Розділ кодування охопив створення бази даних з урахуванням індексації й кешування, реалізацію серверної логіки на NestJS із підтримкою авторизації та інтеграції зі сторонніми сервісами, а також клієнтської частини на Angular із керуванням станом за допомогою NgRx. Якість програмного забезпечення підтверджена автоматизованим і мануальним тестуванням, а фінальна контейнеризація забезпечила зручне розгортання.

Результати проєкту були апробовані в межах науково-практичної конференції, що підтвердило практичну цінність і готовність застосунку до використання в психологічній практиці. Робота демонструє успішне поєднання теоретичних знань, інженерних підходів та інноваційних технологій у вирішенні актуальних задач сучасної психології.

У рамках досягнення поставленої мети – розробки вебзастосунку для надання рекомендацій щодо покращення психічного та ментального стану людини – послідовно виконано низку ключових завдань:

– проведено аналіз наявних вебрішень у сфері психологічного консультування, що дозволило виявити актуальні потреби користувачів та визначити шляхи вдосконалення;

– сформовано чітку специфікацію вимог до програмного забезпечення, на основі якої спроєктовано архітектуру системи, що охоплює як клієнтську, так і серверну частини;

– спроєктувано архітектуру вебзастосунку з урахуванням специфікацій;

– розроблено структуру бази даних із урахуванням подальшої масштабованості, індексації та кешування;

– реалізовано повнофункціональну серверну логіку з підтримкою автентифікації, авторизації та інтеграцією з модулем штучного інтелекту для аналізу тестування;

– створено клієнтський інтерфейс, який забезпечує зручну взаємодію користувача з системою та представлення результатів аналізу у зрозумілій формі.

Усі етапи виконання завдань спрямовані на досягнення головної мети – створення вебзастосунку для психологічної підтримки користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Village – Скільки українців не задоволені своїм психічним здоров'ям. Результати дослідження. URL: https://www.village.com.ua/village/life/edu-news/359651-skilki-vidsotkiv-ukrayintsiv-ne-zadovoleni-svoyim-psihichnim-zdorov-yam-rezultati-doslidzhennya, (last accessed: 25.05.2025).

2. BetterHelp – Professional Therapy With A Licensed Therapist. URL: https://www.betterhelp.com/ (last accessed: 25.05.2025).

3. Talkspace – #1 Rated Online Therapy, 1 Million+ Users. URL: https://www.talkspace.com/ (last accessed: 25.05.2025).

4. BetterMe Mental Health. URL: https://betterme.world/ua/product/meditation (last accessed: 25.05.2025).

5. Woebot: The Mental Health Ally. URL: https://woebothealth.com/ (last accessed: 25.05.2025).

6. Strunk D. R., Adler A. D., Hollon S. D. Cognitive therapy of depression. *The Oxford Handbook of Mood Disorders*. 2015. P. 132–137.

7. Nakao M., Shirotsuki K., Sugaya N. Cognitive–behavioral therapy for management of mental health and stress-related disorders: Recent advances in techniques and technologies. 2021, 4 p.

8. Makris A., Tserpes K., Spiliopoulos G., Zissis D., Anagnostopoulos D. Correction to: MongoDB Vs PostgreSQL: a comparative study on performance aspects (GeoInformatica, (2020), 10.1007/s10707-020-00407-w). 2021, 268 p.

9. Documentation NestJS – A progressive Node.js framework. URL: https://docs.nestjs.com/, (last accessed: 28.04.2025).

10. Sanka A. I., Chowdhury M. H., Cheung R. C. C. Efficient High-Performance FPGA-Redis Hybrid NoSQL Caching System for Blockchain Scalability. *Computer Communications*. 2021. Vol. 169. P. 89–90.

11. Johansson L., Dossot D. RabbitMQ Essentials Build distributed and scalable applications with message queuing using RabbitMQ. 2020. 948–952 p.

12. Walter-Tscharf F. F. W. V. Indexing, Clustering, and Search Engine for Documents Utilizing Elasticsearch and Kibana. *Lecture Notes on Data Engineering and Communications Technologies*. 2022. P. 902–910.

13. Acharya J. N., Suthar A. C. Docker Container Orchestration Management: A Review. 2022. P. 152–153.

14. М. М. Котенко, Т. А. Вакалюк. Впровадження мікросервісної архітектури: технічні виклики та рішення. *Інформаційні технології*. 2024. № 4(91). с. 299-305. doi : 10.35546/kntu2078-4481.2024.4.39.

15. Ollama – Get up and running with large language models. URL: https://ollama.com/, (last accessed: 25.05.2025).

16. О. В. Мазурець, І. А. Тимофієв, В. І. Кліменко, О. О. Тищенко. Метод виявлення депресивного стану пов'язаного із навчанням у закладах освіти із використанням нейромережі дуальної архітектури. *Інформаційні технології.* 2024. № 4(91). с. 311-318. doi : 10.35546/kntu2078-4481.2024.4.41.

17. Chyrvon A., Lisovskyi K., Kyryndas N. THE MAIN METHODS OF LOAD BALANCING ON THE NGINX WEB SERVER. 2023, 151p.

18. Pun F. W., Ozerov I. V., Zhavoronkov A. AI-powered therapeutic target discovery. 2023, 572 p.

19. Інтелектуальні інформаційні системи: Всеукр. наук.-практ. конф. молодих вчених, аспірантів, студентів 2-4 грудня 2024 р., м. Миколаїв. Вид-во ЧНУ ім. Петра Могили, 2025. – 124 с. URL: https://dspace.chmnu.edu.ua/jspui/handle/123456789/2631 (дата звернення: 24.05.2025).

ДОДАТОК А

Лістинг коду запуску сервера

```
import { NestFactory } from "@nestjs/core";
import { AppModule } from "./app.module";
import { ConfigService } from "@nestjs/config";
import { DocumentBuilder, SwaggerModule } from "@nestjs/swagger";
import { VersioningType } from "@nestjs/common";
import * as seesion from "express-session";
import * as passport from "passport";
import * as bodyParser from "body-parser";
import { createConnection } from "net";

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.use(bodyParser.json({ limit: "2mb" }));

  const configService = app.get(ConfigService);
  const PORT = configService.get<number>("port") || 3000;
  const CLIENT_PORT = configService.get<number>("CLIENT_PORT") || 4200;
  const API_PREFIX = configService.get<string>("API_PREFIX") || "api";
  const VERSION = configService.get<string>("API_VERSION") || "1";
  const SWAGGER_PREFIX = configService.get<string>("SWAGGER_PREFIX") || "1";
  const SECRET = configService.get<string>("SECRET") || "1";

  app.enableCors({
    origin: `http://localhost:${CLIENT_PORT}`,
    methods: "GET,HEAD,PUT,PATCH,POST,DELETE",
    allowedHeaders: "Content-Type, Accept, Authorization",
    credentials: true,
  });

  app.setGlobalPrefix(API_PREFIX);
  app.use(
    seesion({
      secret: SECRET,
      saveUninitialized: false,
      resave: false,
      cookie: {
        maxAge: 2592000000,
        sameSite: "strict",
        httpOnly: false,
      },
    })
  );
  app.use(passport.initialize());
  app.use(passport.session());
  app.enableVersioning({
    type: VersioningType.HEADER,
    header: "Version",
    defaultVersion: VERSION,
  });

  const options = new DocumentBuilder()
```

```
        .setTitle("Healthy")
        .setDescription("Quick therapy recognition API")
        .setVersion(VERSION)
        .build();

      const document = SwaggerModule.createDocument(app, options);
      SwaggerModule.setup(`${SWAGGER_PREFIX}`, app, document);

      await app.listen(PORT, () => {
        console.log(`API running at: http://localhost:${PORT}/${API_PREFIX}`);
                              console.log(`DOCS            running            at:
http://localhost:${PORT}/${SWAGGER_PREFIX}`);
      });
    }

    bootstrap();
```

ДОДАТОК Б

Лістинг коду контролера користувача

```
import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  UseGuards,
  Req,
  Query,
} from "@nestjs/common";
import { UserService } from "./user.service";
import { CreateUserDto } from "./dto/create-user.dto";
import { UpdateUserDto } from "./dto/update-user.dto";
import {
  ApiHeader,
  ApiOperation,
  ApiResponse,
  ApiParam,
  ApiQuery,
} from "@nestjs/swagger";
import { ApiVersion } from "src/modules/versions";
import { from, Observable, switchMap } from "rxjs";
import { User } from "src/database/entities/user.entity";
import { DeleteResult, UpdateResult } from "typeorm";
import { RoleName, Sex } from "../../../../shared/enums/user.enum";
import { Roles } from "../decorator/role.decorator";
import { RolesGuard } from "../guard/role.guard";
import { Request } from "express";
import { faker } from "@faker-js/faker";
import { ElasticsearchService } from "@nestjs/elasticsearch";

@Controller({ path: "users", version: ApiVersion.Version02 })
@ApiHeader({
  name: "Version",
  enum: Object.values(ApiVersion),
  required: true,
  description: "API version header",
})
@UseGuards(RolesGuard)
export class UserController {
  constructor(
    private readonly userService: UserService,
    private readonly elasticsearchService: ElasticsearchService
  ) {}

  @Roles(RoleName.User, RoleName.Moderator, RoleName.Admin)
  @Get("search")
  @ApiOperation({ summary: "Search users by nickname" })
  @ApiQuery({
```

```
  name: "nickname",
  required: true,
  type: String,
  description: "Nickname to search for (fuzzy match supported)",
})
@ApiResponse({
  status: 200,
  description: "List of users found",
  type: [User],
})
handleSearchByNickname(
  @Query("nickname") nickname: string
): Observable<User[]> {
  return this.userService.findByNickname(nickname);
}

@Roles(RoleName.User)
@ApiOperation({ summary: "Get current user" })
@ApiResponse({
  status: 200,
  description: "The user was successfully found.",
  type: User,
})
@ApiResponse({
  status: 404,
  description: "User not found",
})
@Get("me")
async handleMe(@Req() req: Request) {
  return req.user;
}

@Roles(RoleName.User)
@Patch("me")
@ApiOperation({ summary: "Update current user information" })
@ApiResponse({
  status: 200,
  description: "User updated successfully.",
  type: UpdateResult,
})
@ApiResponse({
  status: 404,
  description: "User not found",
})
handleMeEdit(
  @Req() req: Request,
  @Body() userData: UpdateUserDto
): Observable<UpdateResult> {
  return from(req.user as Promise<User>).pipe(
    switchMap((user: User) => {
      if (!user || !user.uuid) {
        throw new Error("User not found");
      }
      return this.userService.updateUserProfile(user.uuid, userData);
    })
  );
```

```
}

@Roles(RoleName.User, RoleName.Moderator, RoleName.Admin)
@Get(":id")
@ApiOperation({ summary: "Get a user by ID" })
@ApiParam({ name: "id", description: "The unique identifier of the user" })
@ApiResponse({
  status: 200,
  description: "The user was successfully found.",
  type: User,
})
@ApiResponse({
  status: 404,
  description: "User not found",
})
handleFindOne(@Param("id") id: string): Observable<User> {
  return this.userService.findById(id);
}

@Roles(RoleName.Admin)
@Post("mock")
@ApiOperation({ summary: "Generate and save mock users for testing" })
@ApiQuery({
  name: "mockUsersCount",
  required: false,
  type: Number,
  example: 1000,
})
@ApiResponse({
  status: 201,
  description: "Mock users created",
  schema: {
    example: { count: 1000 },
  },
})
async generateMockUsers(
  @Query("mockUsersCount") mockUsersCount = 1000
): Promise<{ count: number }> {
  const mockUsers: CreateUserDto[] = [];

  await this.elasticsearchService.indices.create(
    {
      index: "users",
      body: {
        mappings: {
          properties: {
            nickname: {
              type: "text",
              analyzer: "standard",
            },
            email: { type: "keyword" },
            bio: { type: "text" },
            sex: { type: "keyword" },
            birthdate: { type: "date" },
          },
        },
```

```
        },
      },
      { ignore: [400] }
    );

    function mutateNickname(name: string): string {
      const arr = name.split("");
      const index = Math.floor(Math.random() * arr.length);
      arr[index] = faker.string.alpha(1);
      return arr.join("");
    }

    for (let i = 0; i < mockUsersCount; i++) {
      if (i % 10 === 0) {
        var baseNickname = faker.internet.userName();
      }

      const nickname = mutateNickname(baseNickname);
      const uniqueEmail = `user${i}_${faker.internet.email()}`;

      mockUsers.push({
        email: uniqueEmail,
        nickname,
        bio: faker.lorem.sentence(),
        sex: faker.helpers.arrayElement([Sex.Female, Sex.Male]),
        birthdate: faker.date.birthdate({ min: 18, max: 65, mode: "age" }),
      });
    }

    for (const user of mockUsers) {
      const created = await this.userService.create(user).toPromise();
      await this.elasticsearchService.index({
        index: "users",
        id: created.uuid,
        document: created,
      });
    }

    return { count: mockUsers.length };
  }
}
```

ДОДАТОК В

Лістинг коду мікросервісу обробки запиту штучного інтелекту

```
require("dotenv").config();
const express = require("express");
const amqp = require("amqplib");
const axios = require("axios");

const app = express();
const port = process.env.AI_PORT || 3050;
const RABBITMQ_URL =
  process.env.RABBITMQ_URL || "amqp://guest:guest@localhost:5672";
const AI_API_URL =
  process.env.AI_API_URL ||
  `http://${process.env.AI_HOST}:${process.env.OLLAMA_PORT}/api/generate`;
const AI_RETURN_QUEUE = process.env.AI_RETURN_QUEUE || "ai-response";
const AI_CONSUME_QUEUE = process.env.AI_CONSUME_QUEUE || "ai";
const AI_MODEL = process.env.AI_MODEL || "samantha-mistral";

app.use(express.json());

app.listen(port, () => {
  console.log(`AI microservice running on http://localhost:${port}`);
  connectToRabbit();
});

async function connectToRabbit() {
  const maxRetries = 5;
  const retryDelay = 5000; // 5 секунд

  for (let attempt = 1; attempt <= maxRetries; attempt++) {
    try {
      console.log(`[RabbitMQ] Attempt ${attempt} to connect...`);

      const conn = await amqp.connect(RABBITMQ_URL);
      const channel = await conn.createChannel();

      await channel.assertQueue(AI_CONSUME_QUEUE, { durable: true });
      await channel.assertQueue(AI_RETURN_QUEUE, { durable: true });

         console.log(`[*] Connected. Listening on "${AI_CONSUME_QUEUE}"
queue...`);

      channel.consume(AI_CONSUME_QUEUE, async (msg) => {
        if (msg !== null) {
          try {
            const { prompt } = JSON.parse(msg.content.toString());
            const { replyTo, correlationId } = msg.properties;

            console.log(`[x] Received prompt: ${prompt}`);
            const aiResponse = await callAIModel({ prompt });

            const responsePayload = {
              aiResponse,
```

```
            model: AI_MODEL,
          };

          if (replyTo && correlationId) {
            channel.sendToQueue(
              replyTo,
              Buffer.from(JSON.stringify(responsePayload)),
              { correlationId }
            );
          }

          channel.ack(msg);
        } catch (err) {
          console.error("[!] Failed to process message:", err.message);
          channel.nack(msg);
        }
      }
    });

    break;
  } catch (err) {
    console.error(
      `[!] Connection failed (attempt ${attempt}): ${err.message}`
    );

    if (attempt < maxRetries) {
      console.log(`[!] Retrying in ${retryDelay / 1000} seconds...`);
      await new Promise((resolve) => setTimeout(resolve, retryDelay));
    } else {
      console.error(
        "[!] Max retries reached. Could not connect to RabbitMQ."
      );
      process.exit(1);
    }
  }
 }
}

async function callAIModel({ prompt }) {
  try {
    const res = await axios.post(AI_API_URL, {
      model: AI_MODEL,
      prompt,
      stream: false,
    });
    return res.data.response || res.data;
  } catch (err) {
    console.error("[!] AI API call failed:", err.message);
    return "AI model failed to respond.";
  }
}
```

ДОДАТОК Д

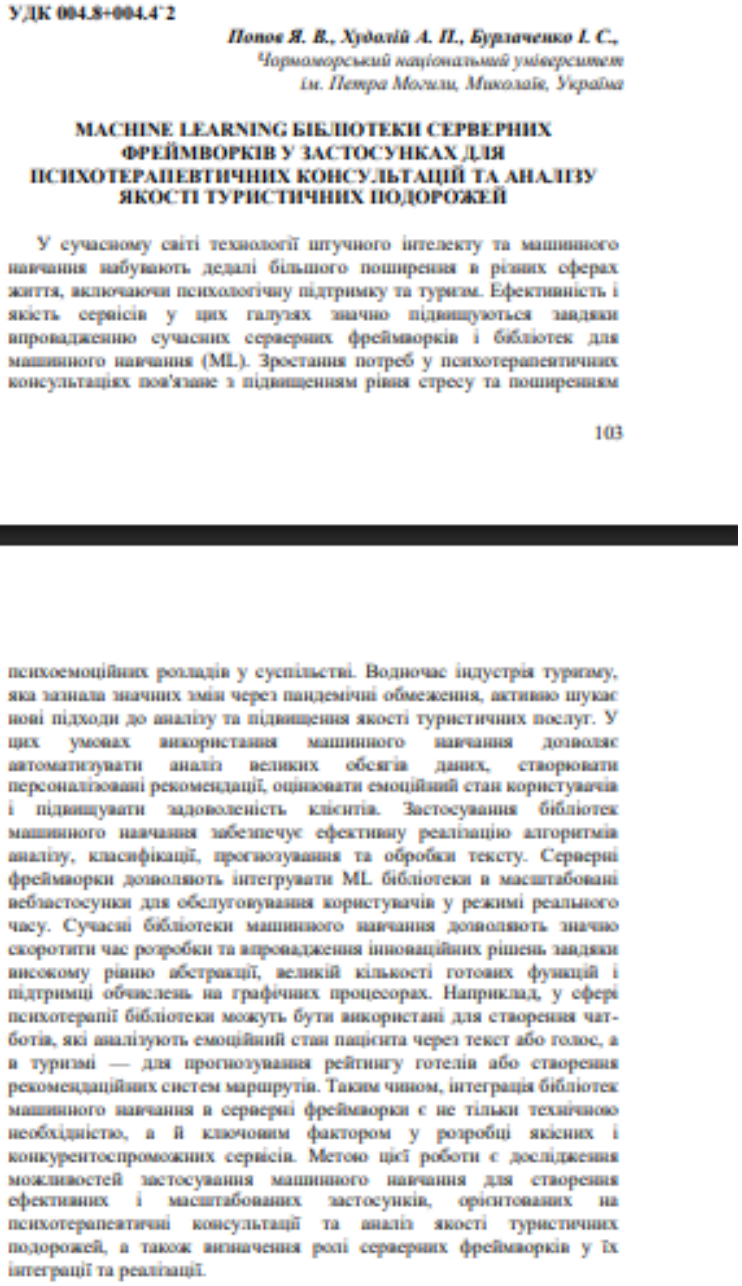
Лістинг коду модуля перекладача

```
package main

import (
    "encoding/json"
    "fmt"
    "log"
    "net/http"

    "github.com/bregydoc/gtranslate"
    "github.com/rs/cors"
)

type TranslateRequest struct {
    Text string `json:"text"`
    To   string `json:"to"`
}

type TranslateResponse struct {
    TranslatedText string `json:"translatedText,omitempty"`
    Status         bool   `json:"status"`
    Message        string `json:"message"`
}

func TranslateHandler(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
        return
    }

    var request TranslateRequest
    err := json.NewDecoder(r.Body).Decode(&request)
    if err != nil {
       sendErrorResponse(w, "Invalid request payload", http.StatusBadRequest)
        return
    }

        translated, err := gtranslate.TranslateWithParams(request.Text,
gtranslate.TranslationParams{
            From: "auto",
            To:   request.To,
        })
        if err != nil {
                            sendErrorResponse(w,     "Translation     failed",
http.StatusInternalServerError)
            return
        }

        response := TranslateResponse{
            TranslatedText: translated,
            Status:         true,
            Message:        "",
```

```
        }

        sendJSONResponse(w, response, http.StatusOK)
    }

    func sendErrorResponse(w http.ResponseWriter, message string, statusCode int)
{
        response := TranslateResponse{
            Status:  false,
            Message: message,
        }
        sendJSONResponse(w, response, statusCode)
    }

    func sendJSONResponse(w http.ResponseWriter, data interface{}, statusCode
int) {
        w.Header().Set("Content-Type", "application/json")
        w.WriteHeader(statusCode)
        err := json.NewEncoder(w).Encode(data)
        if err != nil {
            log.Printf("Failed to encode JSON response: %v", err)
        }
    }

    func main() {
        mux := http.NewServeMux()
        mux.HandleFunc("/translate", TranslateHandler)

        c := cors.Default().Handler(mux)

        fmt.Println("Starting server on http://localhost:8000")
        log.Fatal(http.ListenAndServe(":8000", c))
    }
```

ДОДАТОК Е

Апробація результатів



УДК 004.8+004.4'2

Попов Я. В., Худолій А. П., Будзяченко І. С.,
Чорноморський національний університет
ім. Петра Могили, Миколаїв, Україна

MACHINE LEARNING БІБЛІОТЕКИ СЕРВЕРНИХ ФРЕЙМВОРКІВ У ЗАСТОСУНКАХ ДЛЯ ПСИХОТЕРАПЕВТИЧНИХ КОНСУЛЬТАЦІЙ ТА АНАЛІЗУ ЯКОСТІ ТУРИСТИЧНИХ ПОДОРОЖЕЙ

У сучасному світі технології штучного інтелекту та машинного навчання набувають дедалі більшого поширення в різних сферах життя, включаючи психологічну підтримку та туризм. Ефективність і якість сервісів у цих галузях значно підвищуються завдяки впровадженню сучасних серверних фреймворків і бібліотек для машинного навчання (ML). Зростання потреб у психотерапевтичних консультаціях пов'язане з підвищенням рівня стресу та поширенням

психоемоційних розладів у суспільстві. Водночас індустрія туризму, яка зазнала значних змін через пандемічні обмеження, активно шукає нові підходи до аналізу та підвищення якості туристичних послуг. У цих умовах використання машинного навчання дозволяє автоматизувати аналіз великих обсягів даних, створювати персоналізовані рекомендації, оцінювати емоційний стан користувачів і підвищувати задоволеність клієнтів. Застосування бібліотек машинного навчання забезпечує ефективну реалізацію алгоритмів аналізу, класифікації, прогнозування та обробки тексту. Серверні фреймворки дозволяють інтегрувати ML бібліотеки в масштабовані вебзастосунки для обслуговування користувачів у режимі реального часу. Сучасні бібліотеки машинного навчання дозволяють значно скоротити час розробки та впровадження інноваційних рішень завдяки високому рівню абстракції, великій кількості готових функцій і підтримці обчислень на графічних процесорах. Наприклад, у сфері психотерапії бібліотеки можуть бути використані для створення чат-ботів, які аналізують емоційний стан пацієнта через текст або голос, а в туризмі — для прогнозування рейтингу готелів або створення рекомендаційних систем маршрутів. Таким чином, інтеграція бібліотек машинного навчання в серверні фреймворки є не тільки технічною необхідністю, а й ключовим фактором у розробці якісних і конкурентоспроможних сервісів. Метою цієї роботи є дослідження можливостей застосування машинного навчання для створення ефективних і масштабованих застосунків, орієнтованих на психотерапевтичні консультації та аналіз якості туристичних подорожей, а також визначення ролі серверних фреймворків у їх інтеграції та реалізації.

Рисунок Е – Тези «Матеріали всеукраїнської науково-практичної конференції молодих вчених, аспірантів і студентів»

ДОДАТОК Ж

Лістинг коду docker-compose.yml

```
services:
  postgres:
    image: postgres:latest
    container_name: postgres
    env_file:
      - .env
    environment:
      POSTGRES_HOST: ${DATABASE_HOST}
      POSTGRES_USER: ${DATABASE_USER}
      POSTGRES_PASSWORD: ${DATABASE_PASSWORD}
      POSTGRES_DB: ${DATABASE_NAME}
    ports:
      - "${DATABASE_PORT:-5432}:${DATABASE_PORT:-5432}"
    volumes:
      - postgres_data:/var/lib/postgresql/data
  pgadmin:
    image: dpage/pgadmin4:latest
    container_name: pgadmin
    env_file:
      - .env
    environment:
      PGADMIN_DEFAULT_EMAIL: ${PGADMIN_DEFAULT_EMAIL}
      PGADMIN_DEFAULT_PASSWORD: ${PGADMIN_DEFAULT_PASSWORD}
    ports:
      - "${PGADMIN_PORT:-8080}:80"
    depends_on:
      - postgres
    volumes:
      - pgadmin_data:/var/lib/pgadmin
  rabbitmq:
    image: rabbitmq:3-management
    container_name: rabbitmq
    ports:
      - "${RABBITMQ_PORT:-5672}:${RABBITMQ_PORT:-5672}"
        -  "${RABBITMQ_MANAGEMENT_PORT:-15672}:${RABBITMQ_MANAGEMENT_PORT:-
15672}"
    env_file:
      - .env
    volumes:
      - rabbitmq_data:/var/lib/rabbitmq
  api:
    build:
      context: .
      dockerfile: ./api/Dockerfile
    container_name: api
    env_file:
      - .env
    ports:
      - "${API_PORT:-3000}:${API_PORT:-3000}"
```

```
  depends_on:
    - postgres
    - rabbitmq
    - redis
    - elasticsearch
client:
  build:
    context: .
    dockerfile: ./client/Dockerfile
  container_name: client
  ports:
    - "${CLIENT_PORT:-4000}:80"
  depends_on:
    - api
  env_file:
    - .env
ai:
  build:
    context: ./ai
    dockerfile: Dockerfile
  container_name: ai
  ports:
    - "${AI_PORT:-3030}:${AI_PORT:-3030}"
  depends_on:
    - rabbitmq
    - ollama
  env_file:
    - .env
ollama:
  image: ollama/ollama
  container_name: ollama
  env_file:
    - .env
  ports:
    - "${OLLAMA_PORT:-11434}:${OLLAMA_PORT:-11434}"
  volumes:
    - ollama_data:/root/.ollama
  deploy:
    resources:
      reservations:
        devices:
          - capabilities: [gpu]
  runtime: nvidia
translator:
  build:
    context: ./translator
    dockerfile: Dockerfile
  container_name: translator
  env_file:
    - .env
  ports:
    - "${GO_SERVICE_PORT:-8000}:8000"
  depends_on:
    - rabbitmq
redis:
  image: redis/redis-stack:latest
```

```
    container_name: redis
    env_file:
      - .env
    ports:
      - "${REDIS_PORT:-6379}:6379"
      - "8001:8001"
    volumes:
      - redis_data:/data
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.11.3
    container_name: elasticsearch
    env_file:
      - .env
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
      - ELASTIC_PASSWORD=elastic
    ports:
      - "${ELASTICSEARCH_PORT:-9200}:${ELASTICSEARCH_PORT:-9200}"
                                  -         "${ELASTICSEARCH_TRANSPORT_PORT:-
9300}:${ELASTICSEARCH_TRANSPORT_PORT:-9300}"
    volumes:
      - esdata:/usr/share/elasticsearch/data
  postgres_slave:
    build:
      context: ./postgres-slave
    container_name: postgres_slave
    env_file:
      - .env
    environment:
      POSTGRES_USER: ${DATABASE_USER}
      POSTGRES_PASSWORD: ${DATABASE_PASSWORD}
      POSTGRES_DB: ${DATABASE_NAME}
      DATABASE_USER: ${DATABASE_USER}
      DATABASE_PASSWORD: ${DATABASE_PASSWORD}
      DATABASE_NAME: ${DATABASE_NAME}
    ports:
      - "5433:5432"
    volumes:
      - postgres_slave_data:/var/lib/postgresql/data
volumes:
  postgres_data:
  pgadmin_data:
  rabbitmq_data:
  ollama_data:
  redis_data:
  esdata:
  postgres_slave_data:

networks:
  default:
    driver: bridge
```