Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ПСИХОЛОГІЧНОЇ ДОПОМОГИ РІЗНИХ РІВНІВ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-20-2
Самородов В.К.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Руденко Д.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Самородову Віталію Костянтиновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка застосунку для психологічної допомоги різних рівнів

затверджена наказом університету від 20 травня 2024 року № 464 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 27 травня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, фреймворк для створення серверних застосунків NestJS, бібліотека з відкритим кодом для фронтенду, призначена для створення користувацьких інтерфейсів React, інтегроване середовище розробки WebStorm, платформа контейнеизації, серверна платформа Node.js.

4. Перелік питань, що потрібно опрацювати в роботі

1. Аналіз предметної області, використання застосунків для психологічної допомоги.

2. Проєктування архітектури застосунку та моделювання бази даних.

3. Розробка застосунку з використанням NestJS, React та реалізація фізичного рівня бази даних з використанням PostgreSQL та PrismaORM.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Використання сучасних технологій для психологічної допомоги, постановка задачі, концептуальна діаграма бази даних, діаграма сутностей фізичної моделі бази даних, схематики файлових структур, UML-діаграми компонент серверної та клієнтської частин.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.04.2024	
2	Аналіз завдання, підбір літератури	08.04.24-13.04.24	
3	Аналіз літератури з досліджуваної проблеми	13.04.24-17.04.24	
4	Аналіз технічних засобів	17.04.24-25.04.24	
5	Проєктування застосунку	25.04.24-12.05.24	
6	Програмна реалізація	12.05.24-20.05.24	
7	Оформлення пояснювальної записки	20.05.24-24.05.24	
8	Перевірка на плагіат	28.05.24	
9	Рецензування	29.05.24	
10	Підготовка презентації та доповіді	30.05.24-02.06.24	
11	Занесення роботи в електронний архів	05.06.24	
12	Попередній захист кваліфікаційної роботи	05.06.24	

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ _____ доц. Руденко Д.О._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 92 с., 6 табл., 28 рис., 30 джерел.

ПСИХОЛОГІЧНА ДОПОМОГА, ВИКОРИСТАННЯ СУЧАСНИХ ТЕХНОЛОГІЙ ДЛЯ ПСИХОЛОГІЧНОЇ ДОПОМОГИ, ВЕБЗАСТОСУНОК, NESTJS, POSTGRESQL, PRISMAORM, TYPESCRIPT, WEBSTORM.

Об’єктом роботи є інструменти психологічної підтримки у вигляді вебзастосунків.

Метою роботи є розробка вебзастосунку для психологічної допомоги різних рівнів з використанням фреймворку NestJS для серверної частини та бази PostgreSQL для роботи з даними.

Досліджено та використано принципи проєктування вебзастосунків, їх автоматизації та чистого коду. Змодельовано базу даних на концептуальному та фізичному рівнях з використанням PostgreSQL, налагоджено взаємодію з даними та сервером через PrismaORM. Проведено аналіз принципів надання психологічної допомоги з використанням інформаційних технологій, а також їх впровадження у вебзастосунок.

У результаті роботи здійснена програмна реалізація сервісу для отримання психологічної допомоги та її надання онлайн.

PSYCHOLOGICAL ASSISTANCE, UTILIZATION OF MODERN TECHNOLOGIES FOR PSYCHOLOGICAL SUPPORT, WEB APPLICATION, NESTJS, POSTGRESQL, PRISMAORM, TYPESCRIPT, WEBSTORM.

The object of the work is psychological support tools in the form of mobile applications.

The purpose of the work is to develop a web application for psychological assistance of various levels using the NestJS framework for the server part and the PostgreSQL database for working with data.

The principles of designing web applications, their automation and clean code were studied and used. The database was modeled at the conceptual and physical levels using PostgreSQL and the interaction with the data and the server was established through PrismaORM. An analysis of the principles of providing psychological assistance using information technologies, as well as their implementation in web applications, was carried out.

As a result of the work, the software implementation of the service for receiving psychological help and providing it online was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

КПТ – когнітивно-поведінкова терапія

БД – база даних

SEO – Search Engine Optimization (оптимізація пошукових систем)

IDE – Integrated Development Environment (інтегроване середовище розробки)

API – Application Programming Interface (програмний інтерфейс застосунку)

DTO – Data Transfer Object (об’єкти передачі даних)

ВСТУП

Сучасний світ постійно змінюється, а разом з ним зростає навантаження на психіку людей. Стрес, тривога, депресія – це лише частина психологічних проблем, з якими зіштовхуються сучасні люди. У цьому контексті, питання психологічного благополуччя та підтримки стає надзвичайно актуальним для нашого суспільства.

Сучасний розвиток технологій впливає майже на усі сфери життя людини, включаючи психологічну допомогу. Застосування сучасних методів розробки засобів для психологічної підтримки відкриває нові можливості і перспективи для покращення ефективності та доступності цієї важливої сфери. Насамперед, це стосується вебтехнологій, які забезпечують широкий доступ до інформації та інтерактивність у взаємодії з користувачами.

Метою кваліфікаційної роботи є розробка інтерактивного вебзастосунку для психологічної допомоги різних рівнів. Застосунок буде спрямований на підтримку психічного здоров’я, надання психологічних порад та консультацій, а також навчання ефективним стратегіям саморегуляції та покращення емоційного стану.

Актуальність даної роботи полягає в необхідності використання сучасних технологій для розвитку інструментів психологічної підтримки, зокрема у вигляді вебплатформи. З врахуванням росту популярності цифрових рішень у сфері здоров’я та благополуччя, створення інтерактивного застосунку для психологічної допомоги є кроком уперед у наданні якісної та доступної психологічної підтримки для широкого кола користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Вступ до психології: наука про людську психіку

1.1.1 Психологія як наука

Перш ніж поглиблюватись у аспекти психологічної допомоги, треба розглянути що з себе представляє взагалі психологія.

Психологія – це наука, що вивчає психічні процеси, стани та властивості людини, а також поведінку людей у різних ситуаціях. Вона досліджує, як людина сприймає світ, як вона думає, відчуває, навчається, запам’ятовує, розвивається та взаємодіє з іншими людьми.

Психологія є багатогранною наукою, яка охоплює широкий спектр тем, включаючи когнітивні процеси, емоції, мотивацію, особистість, соціальну психологію, розвиток та психопатологію [1].

Вивчення когнітивних процесів дозволяє розуміти, як людина сприймає інформацію, думає, вирішує проблеми та приймає рішення. Аналіз емоцій допомагає зрозуміти, як людина відчуває, виражає та регулює свої почуття. Мотивація вивчається для з'ясування, що спонукає людину до дій. Особистість досліджує, що робить кожну людину унікальним індивідуумом. Соціальна психологія вивчає взаємодію людей та вплив соціального середовища на них. Розвиток ж досліджує, як людина змінюється протягом життя. Психопатологія вивчає виникнення та лікування психічних розладів.

1.1.2 Предмет психології

Предметом психології як науки є психіка людини, її закономірності розвитку та функціонування [2, 3].

До складу психіки входять психічні процеси, такі як когнітивні (пізнавальні), емоційні та вольові, а також психічні стани, які можуть бути

короткочасними (емоції, відчуття) або стійкими (настрої, афекти). Крім того, до складу психіки входять психічні властивості, які можуть бути індивідуально-психологічними (темперамент, характер) або загальнолюдськими (інтелект, спрямованість особистості).

Психологія досліджує, як ці компоненти психіки взаємодіють між собою та з зовнішнім середовищем, формуючи нашу поведінку та визначаючи наш особистий досвід.

1.1.3 Методи психології

Психологи використовують різноманітні методи [3] для вивчення психіки людини, які можна поділити на три основні групи: природничо-наукові методи, емпіричні методи та методи впливу на психіку людини.

Природничо-наукові методи передбачають спостереження, тобто вивчення поведінки людей у природних умовах або в спеціально організованих експериментах, експеримент, що досліджує причинно-наслідкові зв’язки між змінними шляхом маніпулювання незалежною змінною та вимірювання залежної змінної та тестування, в якому використовуються стандартизовані завдання для вимірювання психологічних характеристик людей.

Емпіричні ж методи включають інтроспекцію, тобто самоспостереження людини та опис власних психічних процесів, інтерв’ювання людей про їхні думки, почуття та досвід, а також аналіз аналіз продуктів діяльності: вивчення творів мистецтва, листів, щоденників та інших продуктів людської діяльності для розуміння особистості та психічних процесів людей.

Методи впливу на психіку спрямовані на допомогу шляхом активного впливу на психіку. Це включає в себе психотерапію, психокорекцію, психотренінг та психологічне консультування.

1.2 Існуючі підходи до надання психологічної підтримки

З основ психологічної практики, які розглянуті у попередньому підрозділі, витікають саме вже підходи надання психологічної допомоги.

1.2.1 Психодинамічний підхід

Психодинамічна терапія зазвичай проводиться одним терапевтом і одним клієнтом. Терапевт використовує різні техніки, такі як інтерпретація, вільний асоціації та аналіз перенесення, щоб допомогти клієнту зрозуміти несвідомі процеси, які впливають на його поведінку.

Використання психодинамічного підходу [4] передбачає створення безпечного та конфіденційного простору для клієнта, де він може відчувати себе комфортно для висловлення своїх думок та почуттів. Також важливо активно слухати клієнта з увагою до його слів, невербальних сигналів та емоцій, і допомагати йому у розумінні зв'язку між минулим досвідом та теперішньою поведінкою. Крім того, психодинамічний підхід включає заохочення клієнта до дослідження своїх несвідомих думок та почуттів, а також інтерпретацію несвідомих процесів на основі знань психоаналітичної теорії.

1.2.2 Когнітивно-поведінкова терапія (КПТ)

КПТ передбачає бути короткостроковою та орієнтована на вирішення конкретних проблем [5]. Терапевт співпрацює з клієнтом, щоб ідентифікувати негативні думки та поведінку, а також розробити стратегії для їх зміни.

КПТ включає створення чіткого терапевтичного контракту з клієнтом для визначення цілей терапії та очікувань обох сторін. Також цей підхід

передбачає навчання клієнта моделі ABC, яка пояснює, як негативні думки (A) призводять до негативних емоцій (B) та неадаптивної поведінки (C). Допомога клієнту в ідентифікації його негативних думок та перевірці їх на об'єктивність також важлива. Крім того, проводиться навчання клієнта технікам корекції негативних думок, таких як когнітивне переструктурування та декатастрофізація, і допомога у розвитку більш адаптивної поведінки.

1.2.3 Гуманістичний підхід

Гуманістична терапія проводиться в атмосфері неупередженості та поваги. Терапевт створює середовище прийняття та підтримки, де клієнт відчуває себе цінним та зрозумілим.

В рамках гуманістичного підходу [6] важливо створення терапевтичних стосунків, що базуються на безумовному позитивному ставленні, емпатії та конгруентності. Також до нього входить заохочення клієнта розповідати про свою історію та виражати свої почуття. Гуманістичний підхід передбачає допомогу клієнту у розвитку самоповаги та самоприйняття, а також заохочення його до зосередження на власному потенціалі та прагненні до самоактуалізації. Крім того, використання віддзеркалення виступає як техніка, що допомагає клієнту краще зрозуміти свої думки та почуття.

1.2.4 Системний підхід

Системна терапія націлена на роботу з сім’ями, парами або іншими групами людей, які взаємодіють між собою. Терапевт допомагає членам системи зрозуміти, як їхні взаємодії впливають один на одного, і розробити більш здорові моделі спілкування.

Для надання психологічної допомоги в рамках системного підходу [7] потрібно провести збір інформації про історію системи та взаємини між її членами, ідентифікувати патерни взаємодії, що підтримують проблему, і допомагати членам системи у розвитку більш ефективних навичок спілкування. Під час терапевтичної сесії важливе спостереження за взаємодією членів системи, а також заохочування членів системи до взяття на себе відповідальності за власну поведінку.

Отже, існує багато різних підходів до надання психологічної допомоги, кожен з яких має свої переваги та недоліки. Вибір найкращого підходу залежить від специфічних потреб клієнта. Важливо пам’ятати, що найголовніше – створити безпечну та підтримуючу атмосферу, де людина відчуває себе комфортно, щоб ділитися своїми думками та почуттями.

Сучасна психологічна допомога пропонує широкий спектр підходів до надання підтримки людям з різними проблемами психічного здоров’я. Ці підходи також мають своє місце в світі сучасних реалій, та надання допомоги стає більш розвинутим з використанням розвинутих технологій, зокрема вебзастосунків, які не просто доповнюють, а й розширюють можливості психологічної допомоги, роблячи її більш доступною та гнучкою.

1.3 Використання сучасних технологій для психологічної допомоги

Сучасні технології, такі як штучний інтелект, машинне навчання та інтернет, відкривають нові можливості для надання психологічної допомоги [8, 9]. Вебзастосунки, вебзастосунки та інші онлайн-платформи стають все більш доступними та популярними інструментами для психологів та клієнтів [10].

Отже, далі буде розглянуто різні типи вебзастосунків, їх переваги та недоліки, а також етичні питання, які постають у зв’язку з їх використанням.

1.3.1 Типи вебзастосунків для психологічної допомоги

Існує багато різних типів вебзастосунків, які можна використовувати для надання психологічної допомоги [11]. Найпоширенішими є інтернет-терапія, самодопомога, психоосвітні програми, групи підтримки онлайн.

Інтернет-терапія – вебзастосунки та програми дозволяють клієнтам спілкуватися з ліцензованими психологами або консультантами через текстові чати, відеочати або електронну пошту.

Самодопомога – вебсайти та програми надають клієнтам інформацію та ресурси, які можуть допомогти їм впоратися з психічними проблемами.

Психоосвітні програми – вебсайти та програми навчають клієнтів про психічне здоров'я та психічні розлади.

Групи підтримки онлайн – вебсайти та форуми дозволяють людям, які переживають подібні проблеми, зв'язатися один з одним для підтримки та спілкування.

1.3.2 Переваги використання вебзастосунків

Вебзастосунки мають ряд переваг [11] для надання психологічної допомоги. Основною перевагою є доступність: вебзастосунки доступні з будь-якого місця, де є підключення до інтернету. Це може зробити психологічну допомогу більш доступною для людей, які живуть у сільській місцевості або мають обмежену мобільність.

Крім того, вебзастосунки можна використовувати в зручний для клієнта час. Це може бути корисно для людей, які зайняті або мають щільний графік.

Вебзастосунки можуть забезпечити анонімність та конфіденційність [12, 13], що може бути важливим для людей, які не хочуть, щоб їхні друзі чи родина знали про те, що вони шукають психологічну допомогу.

Використання вебзастосунків може бути більш доступним з точки зору матеріальних витрат, ніж традиційна терапія. Це може зробити психологічну допомогу більш доступною для людей з обмеженим бюджетом.

Онлайн-застосунки та вебплатформи можуть допомогти людям самостійно працювати над своїм психічним здоров'ям, пропонуючи вправи, техніки релаксації та інструменти для відстеження прогресу. Наприклад, застосунок може пропонувати щоденні вправи з усвідомленості або техніки глибокого дихання для зняття стресу.

1.3.3 Недоліки використання вебзастосунків

Незважаючи на багато переваг, використання вебзастосунків для надання психологічної допомоги має деякі недоліки [11]. Тут можна виділити відсутність особистого контакту – користувачам може бути важче розвинути довірливі стосунки.

Технічні проблеми, такі як проблеми з підключенням до Інтернету або проблеми з програмним забезпеченням, можуть перешкоджати терапевтичному процесу.

Крім того, існує ризик того, що конфіденційність клієнта може бути порушена, якщо його особисті дані не захищені належним чином.

Не всі психологічні питання підходять для вирішення в онлайн-форматі – деякі психічні проблеми можуть вимагати більш інтенсивного лікування, яке неможливо надати через вебзастосунок.

На жаль, не всі онлайн-платформи та вебзастосунки розроблені кваліфікованими фахівцями. Важливо ретельно обирати джерела інформації, щоб уникнути небезпечних або неефективних методів. Деякі застосунки можуть містити помилкову інформацію або пропонувати неетичні методи лікування.

Варто зазначити, що не всі люди мають необхідні навички для використання технологій для отримання психологічної допомоги. Наприклад, люди похилого віку можуть мати труднощі з використанням сучасних застосунків або онлайн-платформ.

1.3.4 Етичні питання використання вебзастосунків

Використання вебзастосунків для надання психологічної допомоги ставить ряд етичних питань [14]. Сюди варто віднести:

– компетентність терапевта: важливо переконатися, що терапевти, які надають допомогу через вебзастосунки, мають належну кваліфікацію та досвід;

– скринінг клієнтів: важливо, щоб вебзастосунки проводили належний скринінг клієнтів, щоб переконатися, що вони підходять для онлайн-терапії;

– конфіденційність клієнта: вебзастосунки повинні вживати заходів для захисту конфіденційності клієнта;

– інформована згода: клієнти повинні отримати всю необхідну інформацію про вебзастосунок та послуги, що надаються, перш ніж погодитися на їх використання.

1.3.5 Дослідження ефективності вебзастосунків

Існує зростаючий обсяг досліджень, що досліджують ефективність вебзастосунків для надання психологічної допомоги. Дослідження показують, що вебзастосунки можуть бути ефективними для лікування різних психічних розладів, включаючи тривогу, депресію та розлади харчової поведінки [15].

Наприклад, огляд досліджень, проведений у 2017 році, виявив, що когнітивно-поведінкова терапія, що надається через Інтернет, була такою ж ефективною, як і традиційна терапія, для лікування тривоги та депресії [16].

Інше дослідження, проведене у 2019 році, виявило, що інтернет-терапія була ефективною для лікування розладів харчової поведінки [17].

Дослідження також показують, що вебзастосунки можуть бути корисними для запобігання психічних розладів та сприяння психічному благополуччю. Наприклад, дослідження, проведене у 2018 році, виявило, що програма самодопомоги на основі уважності, що реалізована через Інтернет, була ефективною для зниження симптомів стресу та підвищення рівня зосередженості [15].

1.4 Постановка задачі

Таким чином, вебзастосунки є перспективним інструментом для надання психологічної допомоги. Вони можуть зробити психологічну допомогу більш швидкою, зручною та доступною для людей з обмеженим бюджетом. Однак, необхідно зауважити їх обмеження, а відповідно, вжити заходів для захисту конфіденційності клієнтів та забезпечити використання вебзастосунків кваліфікованими фахівцями зі сторони сервісу. Тому ставиться завдання розробки такого застосунку для психологічної допомоги, з використанням сучасних фреймворків проєктування вебресурсів, яке буде відповідати як нормам надання психологічної допомоги, так й принципам якісної розробки програмного забезпечення.

Об’єктом роботи є інструменти психологічної підтримки у вигляді вебзастосунків.

Метою роботи є розробка вебзастосунку для психологічної допомоги різних рівнів з використанням фреймворку NestJS для серверної частини та бази PostgreSQL для роботи з даними. Для досягнення мети необхідно вирішити такі завдання:

- провести аналіз відповідних рішень;
- визначити функціональні та нефункціональні вимоги;

– спроєктувати архітектуру застосунку;

– змоделювати на концептуальну рівні БД;

– реалізувати фізичний рівень БД;

– реалізувати серверну та клієнтську сторони застосунку;

– реалізувати взаємозв’язок та комунікацію між клієнтською, серверною сторонами та рівнем даних.

2 ПРОЄКТУВАННЯ ФУНКЦІОНАЛЬНОЇ ЧАСТИНИ ВЕБЗАСТОСУНКУ

2.1 Аналіз вимог до вебзастосунку

2.1.1 Цільова аудиторія та їх потреби

Розробка вебзастосунку для психологічної допомоги вимагає чіткого розуміння цільової аудиторії та їх особливих потреб. Цільова аудиторія такого застосунку включає не лише людей, які страждають на психологічні розлади, а й тих, хто шукає способи подолання повсякденного стресу чи підвищення особистісного зросту. Особливу увагу слід звернути на доступність послуг в будь-який час та з будь-якого місця, що особливо важливо для осіб, які можуть відчувати психологічний дискомфорт або навіть кризу. Застосунок повинен надавати приватне середовище, де користувачі могли б вільно виражати свої емоції та думки без страху бути засудженими.

Аналізуючи дослідження ринку та відгуки потенційних користувачів, стає зрозуміло, що доступ до психологічних послуг через Інтернет може значно знизити бар’єри [18], такі як стигма та фізичні обмеження. Таким чином, користувачі можуть шукати швидку підтримку без потреби довгого очікування або планування візитів до психолога, що є ключовим аспектом для високої адаптації та ефективності вебзастосунку.

2.1.2 Функціональні вимоги

Для ефективної роботи вебзастосунку для психологічної допомоги необхідно врахувати ряд ключових функціональних вимог, які відіграють критичну роль у забезпеченні доступності, безпеки та користувацької зручності.

Реєстрація та авторизація користувачів: основою безпеки вебзастосунку є забезпечення контрольованого доступу до особистих кабінетів. Реєстрація має бути інтуїтивно зрозумілою та зручною, з одночасним дотриманням високих стандартів безпеки, включаючи використання хешування паролів та валідації ведених даних. Авторизація має забезпечувати надійний захист від несанкціонованого доступу, особливо при обробці конфіденційних даних користувачів.

Система профілів: кожен користувач має можливість створювати та редагувати свій профіль, додавати особисту інформацію та відомості, які можуть впливати на види та методи психологічної підтримки. Ця інформація є ключовою для індивідуалізації допомоги та підбору відповідних ресурсів.

Модуль статей-запитів: цей модуль дозволяє користувачам створювати запити для психологічної допомоги, що структуруються та категоризуються для зручності обробки психологами. Це сприяє ефективній взаємодії та швидкому знаходженню необхідної допомоги.

Модуль категорій-тегів: введення системи тегів і категорій дозволяє користувачам організувати запити за типами проблематики, що спрощує навігацію та пошук відповідних запитів і ресурсів для психологів і користувачів. Це забезпечує більш точне та спрямоване надання допомоги.

Модуль консультацій: можливість психологів проглядати запити та надавати консультації є фундаментом для роботи платформи. Важливо забезпечити наявність зручних інструментів для ведення діалогу між користувачем і психологом, забезпечуючи конфіденційність та безпеку комунікації.

Зв’язок з психологом або користувачем: в профілі кожного користувача та психолога повинно бути посилання для зв’язку, що дозволяє ініціювати комунікацію в разі необхідності. Це забезпечує ефективність комунікаційних процесів і сприяє своєчасному обміну інформацією.

Освітні ресурси: наявність бібліотеки статей та інших освітніх ресурсів, створених користувачами чи психологами, є важливою для підтримки

самоосвіти користувачів. Ці ресурси повинні бути легкодоступними та категоризованими для зручності пошуку.

Можливість вподобання та збереження потрібних ресурсів: функція вподобань та збереження дозволяє користувачам формувати персональну колекцію ресурсів, що сприяє їхньому особистісному зростанню та самодопомозі.

Модуль коментарів: важливою є можливість користувачів обговорювати статті та ресурси, залишаючи коментарі. Це створює спільноту підтримки та сприяє обміну досвідом між користувачами.

Безпека та приватність даних користувачів: оскільки застосунок зберігає та обробляє конфіденційну інформацію, критично важливо забезпечити, що користувачі та психологи мають доступ тільки до тих ресурсів, які відповідають їх правам доступу. Використання найсучасніших засобів шифрування та аутентифікації є обов’язковим для захисту цих даних.

2.1.3 Нефункціональні вимоги

Нефункціональні вимоги забезпечують основу для створення надійного, ефективного та доступного вебзастосунку. Ці вимоги охоплюють аспекти, які не стосуються прямо функціональності, але критично важливі для забезпечення високої якості обслуговування, підвищення юзабіліті сайту [19] та користувацького досвіду.

Безпека: основоположним аспектом вебзастосунку для психологічної допомоги є безпека, зокрема забезпечення конфіденційності та цілісності даних користувачів. Захист даних вимагає комплексного підходу, включно з шифруванням даних, як у передачі, так і в зберіганні. Використання протоколу HTTPS для шифрування даних, що передаються, та застосування алгоритмів шифрування для даних на серверах забезпечує захист від несанкціонованого

доступу. Важливо також регулярно проводити аудит безпеки та пентестування для виявлення та усунення потенційних вразливостей.

Продуктивність: швидкість відгуку вебзастосунку має критичне значення, особливо коли користувачі шукають допомогу в стані стресу або кризи. Оптимізація часу відгуку, забезпечення плавності роботи інтерфейсу та швидке завантаження контенту є обов'язковими аспектами, що вимагають уваги при проєктуванні архітектури та виборі технологій. Кешування даних, ефективне використання ресурсів сервера та розподілення навантаження можуть значно покращити продуктивність застосунку.

Масштабованість: вебзастосунок має бути спроєктований з урахуванням можливості легкого масштабування для обслуговування зростаючої кількості користувачів та обробки даних. Використання мікросервісної архітектури або контейнерізації може допомогти у досягненні гнучкості в управлінні ресурсами та швидкому розширенні системи без втрати продуктивності. Автоматичне горизонтальне масштабування, залежно від навантаження, дозволить застосунку ефективно адаптуватися до змін у вимогах до обробки.

Доступність: забезпечення стабільності та постійного доступу до вебзастосунку є необхідністю, щоб користувачі могли покладатися на послуги в будь-який час. Використання надійних хостингових рішень, резервне копіювання даних та впровадження стратегій відновлення після збоїв є критичними компонентами для забезпечення високого рівня доступності. Важливо також мати план відновлення роботи системи для мінімізації часу простою.

Міжнародна підтримка: оскільки вебзастосунок може використовуватися користувачами з різних країн, підтримка багатьох мов є важливою. Локалізація інтерфейсу та контенту, забезпечення підтримки різних часових поясів та культурних особливостей зробить застосунок доступнішим та зручнішим для міжнародної аудиторії.

Технічне обслуговування: ефективне технічне обслуговування вебзастосунку є ключовим для забезпечення його сталої та безперебійної

роботи. Регулярне оновлення програмного забезпечення, моніторинг системних ресурсів [20] і виявлення можливих збоїв в роботі системи є критичними завданнями, які допомагають попередити більшість технічних проблем, перш ніж вони вплинуть на користувачів. Впровадження процесів автоматизованого тестування та неперервної інтеграції може значно підвищити якість оновлень та знизити ризик виникнення помилок під час впровадження нових функцій. Архітектура застосунки має відповідати вимогам, мати чітку архітектуру, а також не бути перевантаженою для подальшої підтримки платформи. Також, важливо забезпечити наявність підтримки користувачів, щоб вони могли швидко отримувати допомогу у вирішенні технічних питань, що забезпечує високий рівень задоволеності користувачів і сприяє їхній лояльності.

Забезпечення виконання цих нефункціональних вимог є фундаментальним для створення надійного, ефективного, та зручного вебзастосунку, який може ефективно відповідати на потреби користувачів у психологічній підтримці.

2.2 Проєктування архітектури застосунку

2.2.1 Архітектурні шаблони та моделі

При розробці вебзастосунку для психологічної допомоги, виходячи з аналізу функціональних вимог, критично важливо вибрати правильні архітектурні підходи та патерни, які забезпечать оптимальну працездатність, безпеку, масштабованість та легкість управління додатком [21].

На основі функціональних вимог можна виділити ключові сутності, які є необхідними для вебзастосунку. Важливість кожної сутності визначається її роллю в загальній структурі системи і взаємодії з користувачами. Ці сутності включають:

– користувач з основними даними та персоналізованими налаштуваннями;

– психолог як розширення користувача з додатковими правами та обов’язками;

– стаття для забезпечення освітнього контенту;

– теги для ефективного упорядкування та пошуку контенту;

– підписки що дозволяють користувачам слідкувати за оновленнями;

– коментарі для забезпечення зворотного зв’язку та обговорення матеріалів.

Розуміння та чітке визначення цих сутностей сприяє кращому дизайну та реалізації системи, оскільки кожна сутність буде містити відповідні поля та взаємодіяти з іншими компонентами системи відповідно до визначених бізнес-логік.

Застосування сучасних патернів та принципів програмування є важливим для створення масштабованої, ефективної та легко підтримуваної системи. В контексті розробки вебзастосунків, особливо з використанням функціональних поділів, можна застосувати наступні підходи:

Патерн проєктування «Feature-based structure» дозволяє групувати весь функціонал, пов’язаний з конкретною функцією системи, що спрощує знаходження та оновлення коду, що якраз відповідає меті проєктування стосовно сутностей нашої програми. Такий підхід забезпечує високу згуртованість і низьку зв’язаність, сприяючи легшому тестуванню та розширенню системи.

Також, важливим є використання принципів програмування, таких як SOLID [22]. Вони надають методологічну основу для розробки більш чистого, зрозумілішого коду, що легко піддається модифікації та масштабуванню без втрати функціональності. Принцип SOLID – це ключовий набір методологій у об’єктно-орієнтованому програмуванні, кожна літера якого відповідає певному принципу:

– Single Responsibility Principle (принцип єдиної відповідальності): цей принцип стверджує, що клас повинен мати лише одну причину для зміни, тобто він повинен бути відповідальний лише за одну частину функціональності, яку надає програма. Наприклад, у нашому застосунку клас користувача повинен відповідати лише за зберігання та управління даними користувача, а не за логіку аутентифікації чи інтерфейс користувача;

– Open/Closed Principle (принцип відкритості/закритості): об’єкти або сутності повинні бути відкритими для розширення, але закритими для модифікації. Це означає, можуть бути додані нові функціональності без зміни існуючого коду. В контексті нашого застосунку можна розробити систему обробки запитів, яка дозволить додавати нові типи запитів без зміни існуючих класів;

– Liskov Substitution Principle (принцип підстановки Лісков): функції, що використовують базовий тип, мають мати змогу використовувати підтипи базового типу, не знаючи про це. Це означає, що об’єкти класу «Користувач» можна замінити на об’єкти класу «Психолог» без впливу на очікувані характеристики програми;

– Interface Segregation Principle (принцип розділення інтерфейсу): клієнти не повинні бути змушені імплементувати інтерфейси, які вони не використовують. Це означає, що інтерфейси повинні бути специфічними для клієнтів, які їх використовують, не навантажуючи їх непотрібними методами. У нашому вебзастосунку, інтерфейс користувача повинен відрізнятися від інтерфейсу психолога, оскільки їхні ролі та потреби різні;

– Dependency Inversion Principle (принцип інверсії залежностей): високорівневі модулі не повинні залежати від низькорівневих модулів. Обидва типи модулів повинні залежати від абстракцій. Це допомагає зменшити залежність між виконанням програми та конкретними платформами або важкими компонентами, покращуючи можливість перевикористання коду. Наприклад, модуль авторизації може залежати від абстрактного інтерфейсу

замість конкретної реалізації, що дозволить легше змінювати механізми аутентифікації в майбутньому.

Ці підходи допомагають забезпечити, що архітектура вебзастосунку буде відповідати сучасним вимогам до якості програмного забезпечення, що є важливим для підтримки та розвитку комплексних систем.

Вибір архітектури програми є ключовим рішенням, яке впливає на весь процес розробки та підтримки продукту. Існують різні підходи, такі як монолітична архітектура, мікросервісна архітектура, монорепозиторій. Кожен з цих підходів має свої переваги та недоліки:

– мікросервісна архітектура забезпечує високу гнучкість та масштабованість, але може бути складною у підтримці та координації;

– монолітична архітектура простіша у підтримці, але не так ефективно масштабується;

– багатошарова архітектура розподіляє компоненти програми на три шари:

1) презентаційний шар (Presentation Layer): клієнтська частина, яка відповідає за взаємодію з користувачем. Це може бути вебінтерфейс, мобільний застосунок тощо;

2) логічний шар (Business Logic Layer): містить бізнес-логіку та обробку даних. Це серверна частина, яка виконує операції, керує транзакціями та інші обчислення;

3) шар даних (Data Layer): це база даних та її механізми управління. Цей шар відповідає за збереження, витягування та управління даними.

Для проєкту вибір монорепозиторія є оптимальним рішенням, оскільки він дозволяє зберігати всі компоненти вебзастосунку в одному репозиторії, спрощуючи управління залежностями та інтеграцію. Це особливо ефективно для проєктів середнього та малого розміру, де швидкість розробки та здатність швидко впроваджувати зміни є критично важливими. Хоча це може призвести до певних труднощів зі масштабуванням у майбутньому, переваги у термінах

спрощення розробки та підтримки переважають потенційні недоліки на даному етапі проєкту. Також, система буде організовано за відповідними модулями, де кожний відповідальний за свою частину, а отже буде використано багатошарову архітектуру для клієнтської, серверної сторони та даних. Таким чином найоптимальнішим рішенням буде використання поєднання монолітної структури організації застосунку з багатошаровою архітектурою.

2.2.2 Компоненти системи та їх взаємодії

Клієнтська частина (фронт-енд) вебзастосунку є інтерфейсом, через який користувачі взаємодіють зі службами застосунку. Організація фронт-енду зосереджується на створенні інтуїтивно зрозумілого та відгукового інтерфейсу. Це досягається за допомогою компонентного підходу, де кожен компонент управляє певною частиною UI та має визначені зв’язки з іншими компонентами. Такий підхід дозволяє локалізувати стани та логіку, забезпечуючи високий рівень повторного використання коду та легкість у тестуванні. Розподіл компонентів слід також робити з урахуванням динамічності вмісту, що вимагає асинхронної взаємодії з сервером для оновлення даних без перезавантаження сторінки.

Серверна частина (бек-енд) вебзастосунку відповідає за обробку бізнес-логіки, автентифікацію користувачів, обробку даних і їх зберігання. Організація бек-енду зазвичай ґрунтується на патерні «модель-вид-контролер» (MVC), де модель визначає структуру даних, вид відповідає за вивід результатів, а контролер забезпечує логіку зв’язку між користувачем та системою. Застосування цього патерну сприяє розділенню відповідальності між різними частинами аплікації, що полегшує розробку та підтримку коду. Також, бек-енд повинен забезпечувати API, який дозволяє фронт-енду взаємодіяти з сервером через HTTP запити.

База даних є центральним сховищем усієї інформації в застосунку та повинна бути структурована таким чином, щоб забезпечувати швидкий доступ до даних, їх цілісність та безпеку [23]. Проєктування бази даних часто включає нормалізацію для зниження дублювання даних і забезпечення їхньої консистенції. Однак, в деяких випадках денормалізація може бути застосована для оптимізації швидкодії запитів. Інтеграція ORM (Object-Relational Mapping) [24] дозволяє абстрагувати взаємодію з базою даних у бек-енд-аплікації, спрощуючи роботу з даними як з об’єктами вищого рівня.

Важливим аспектом архітектури вебзастосунку є забезпечення ефективної взаємодії між різними компонентами системи. Використання REST API є стандартним підходом для забезпечення зв’язку між клієнтською та серверною частинами через HTTP запити. REST API дозволяє стандартизувати взаємодію, забезпечуючи легку інтеграцію та масштабованість.

Абстракція даних, реалізована через ORM, допомагає ізолювати бізнес-логіку від деталей зберігання даних.

Віртуалізація та контейнеризація, дозволяють створити однорідне середовище для розробки, тестування та розгортання, забезпечуючи консистенцію та ефективність всієї інфраструктури.

2.3 Моделювання бази даних на концептуальному рівні

При проєктуванні бази даних для вебзастосунку психологічної допомоги основною задачею є забезпечення ефективного зберігання та швидкого доступу до даних про користувачів, психологів, статті-запити, теги-категорії, підписки та коментарі. Важливо розробити структуру, яка б дозволяла легко масштабуватися та адаптуватися до змінних вимог користувачів та бізнесу.

Для нашого застосунку важливо обрати базу даних, яка забезпечує надійність, здатність до масштабування та зручність управління транзакціями. Хоча NoSQL бази даних пропонують гнучкість та швидкість обробки великих

обсягів неструктурованих даних, SQL бази даних краще підходять для завдань, де потрібна строга схема даних і забезпечення цілісності та надійності транзакцій. Основні переваги SQL баз даних включають міцну транзакційну підтримку, легкість у запитах, що забезпечує ефективне зв’язування даних між таблицями. Тому для нашого застосунку буде обрано SQL-орієнтовану систему управління базами даних.

На основі наших виявлених вимог та сутностей, що диктуватимуть структуру даних нашого застосунку, можна змоделювати відповідні таблиці сутностей з їх атрибутами та полями. Для зручності моделювання їх буде виділено у три наступні блоки.

2.3.1 Моделювання сутностей

Користувачі: основна таблиця для зберігання інформації про користувачів. Таблиця users матиме містити наступні атрибути, що представлені в таблиці 2.1.

Таблиця 2.1 – Атрибути таблиці users

Назва	**Опис**
1	**2**
id (serial)	Унікальний ідентифікатор користувача, який буде генеруватися автоматично, цей атрибут виступає як первинний ключ в таблиці.
email (varchar)	Електронна адреса користувача, яка є унікальною в системі та може бути використана тільки одним обліковим записом.
name (varchar)	Ім’я користувача, що використовується для ідентифікації та відображення у користувацькому інтерфейсі.

Продовження таблиці 2.1

1	2
password (varchar)	Пароль користувача, який використовується для автентифікації. Важливо забезпечення належного шифрування паролів перед зберіганням у базі даних.
createdAt (date)	Дата та час створення облікового запису користувача. Це поле автоматично встановлюється на поточну дату і час при створенні запису та відображається у системі як «created_at».
updatedAt (date)	Дата та час останнього оновлення облікового запису. Це поле автоматично оновлюється кожного разу, коли запис користувача модифікується, і також відображається як «updated_at».

Таблиця 2.2 – Атрибути таблиці user_info

Назва	Опис
bio (varchar)	Біографічна інформація користувача.
image (varchar)	Аватар користувача, в таблиці зберігається шлях або URL до зображення профілю користувача.
location (varchar)	Географічне розташування користувача, яке може бути використане для локалізації контенту чи спільноти. Це поле також є необов’язковим.
websiteUrl (varchar)	Вебсайт користувача, поле для зберігання посилання на сайт або соціальну мережу для зв’язку з користувачем, має назву («website_url»).

Підписки: відносини між користувачами та психологами, які вони обирають для слідкування. Забезпечує зв’язок між користувачами та об’єктами підписки (що в нашому випадку є сутність Користувача), а отже атрибути таблиці представлені в таблиці 2.2.

Таблиця 2.2 – Атрибути таблиці follow

Назва	Опис
followerId (varchar)	Посилання на id таблиці користувача, що ідентифікує запис, який створив підписку на іншого користувача.
followedId (varchar)	Посилання на id таблиці користувача, що ідентифікує запис, на якого була створена підписка іншим користувачем.

Таким чином, визначено перший блок менеджменту користувача, як показано на рисунку 2.1.

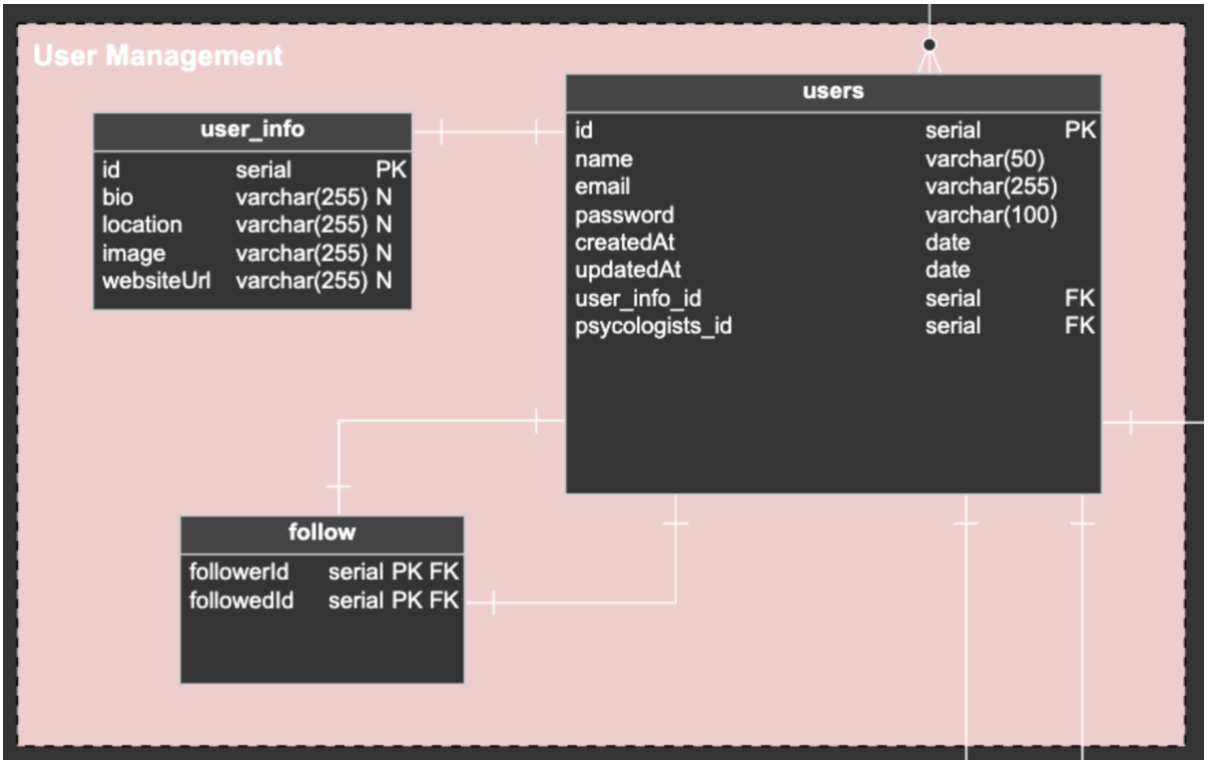

Рисунок 2.1 – Концептуальна діаграма блоку менеджменту користувача

Психолог: розширення таблиці користувачів з додатковими полями, такими як рейтинг. Забезпечує зв’язок «один-до-одного» з таблицею користувачів. Таблиця психолога матиме наступні атрибути, представлені в таблиці 2.3.

Таблиця 2.3 – Атрибути таблиці psychologists

Назва	**Опис**
id (serial)	Унікальний ідентифікатор психолога, який буде генеруватися автоматично. Цей атрибут виступає як первинний ключ в таблиці, оскільки в нашому випадку доречно зробити цю таблицю сильною сутністю, відповідно зовнішній ключ буде в таблиці користувачів на таблицю психологів.
rating (decimal)	Рейтинг психолога, цифрове значення яке має бути в діапазоні від 0 до 10.0

Теги: категорії або ключові слова, які використовуються для класифікації статей-запитів, а також спеціалізацій психологів, таким чином у застосунку одразу диктується потрібна бізнес логіка. Зв’язок «багато-до-багатьох» із психологами, оскільки психолог може мати багато тегів, а тег може мати багато психологів, через це нам потрібно зробити проміжну таблицю психологів до тегів. Таблиця тегів матиме наступні атрибути, представлені в таблиці 2.4.

Таблиця 2.4 – Атрибути таблиці tags

Назва	**Опис**
id (serial)	Унікальний ідентифікатор тегу, який буде генеруватися автоматично. Цей атрибут виступає як первинний ключ в таблиці.
name (varchar50)	Назва тегу, має бути унікальна.

Отже, визначено наступний блок психолога, в який також занесено теги, а відповідно й проміжну таблицю між ними, який показано на рисунку 2.2.

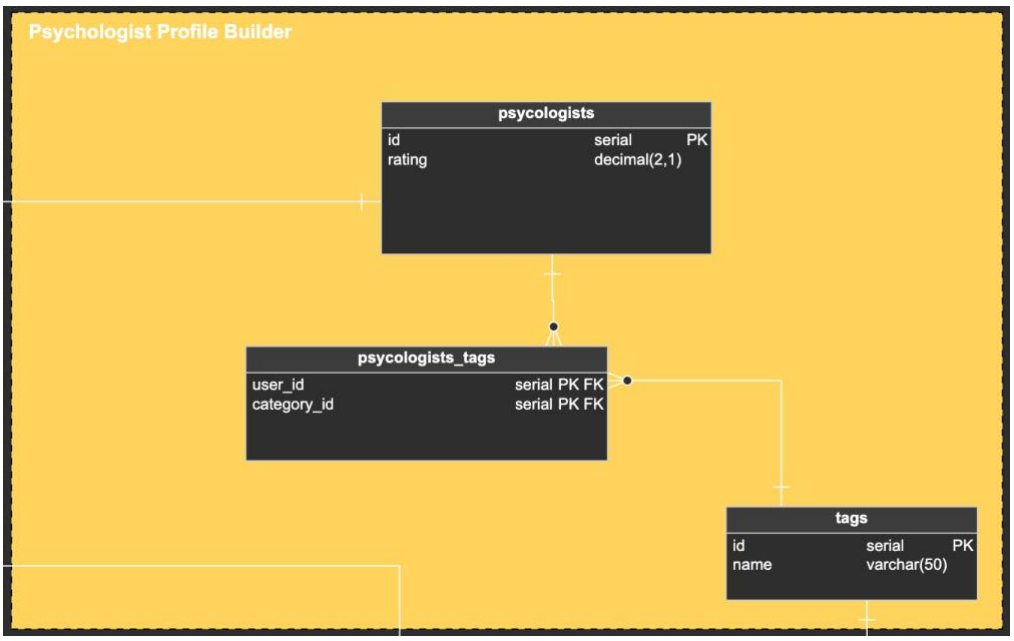

Рисунок 2.2 – Концептуальна діаграма блоку психологів та тегів

Статті-запити: таблиця, що містить інформацію про статті або запити на допомогу. Атрибути представлені в таблиці 2.5.

Таблиця 2.5 – Атрибути таблиці articles

Назва	**Опис**
1	**2**
id (varchar)	Унікальний ідентифікатор статті, що генерується автоматично за допомогою функції uuid(). Цей атрибут виступає як первинний ключ в таблиці.
slug (varchar)	Унікальний текстовий ідентифікатор статті, який використовується у URL адресах для поліпшення SEO і зручності доступу до статті.
title (varchar)	Заголовок статті, який використовується для представлення теми або основної ідеї контенту.
body (varchar)	Основний текст статті. Це поле може містити великі обсяги тексту і часто включає форматування.

Продовження таблиці 2.5

1	2
image (varchar)	Шлях або URL до зображення, яке асоціюється зі статтею. Це може бути обкладинка або ключове зображення, що використовується для ілюстрації теми, для зберігання даних можливо винесення в окрему таблицю.
createdAt (date)	Дата та час створення статті. Це поле автоматично встановлюється на поточну дату і час при створенні статті та відображається у системі як «created_at».
updatedAt (date)	Дата та час останнього оновлення статті. Це поле автоматично оновлюється кожного разу, коли стаття модифікується, і також відображається як «updated_at».
authorId (varchar)	Ідентифікатор автора статті, який вказує на користувача, який її створив. Це поле забезпечує зв’язок з таблицею користувачів.

Коментарі: включає в себе коментарі до статей, одна стаття може мати багато коментарів та окремий коментар може бути прив’язаний тільки до однієї статті. Так само з таблицею користувачів, зв’язок «один-до-багатьох». Атрибути сутності коментарів представлені в таблиці 2.6.

Таблиця 2.6 – Атрибути таблиці comments

Назва	Опис
1	**2**
id (varchar)	Унікальний ідентифікатор коментаря, який генерується автоматично. Цей атрибут служить як первинний ключ в таблиці.

Продовження таблиці 2.6

1	2
content (varchar)	Вміст коментаря, який містить текст, написаний користувачем. Це основне поле, де зберігається текст коментаря.
authorId (varchar)	Ідентифікатор автора коментаря, який зв’язує коментар з конкретним користувачем в системі. Це поле вказує на обліковий запис користувача, який створив коментар, і посилається на «author_id».
articleId (varchar)	Ідентифікатор статті, до якої додається коментар. Це поле встановлює зв’язок між коментарем та статтею, до якої він належить, і посилається на «article_id».
createdAt (date)	Дата та час створення коментаря. Це поле автоматично встановлюється на поточну дату і час у момент створення коментаря та відображається в системі як «created_at».
updatedAt (date)	Дата та час останнього оновлення коментаря, що автоматично оновлюється при редагуванні коментаря, відображається поле в системі як «updated_at»

Ще однією сутністю створено таблицю favorites для бізнес логіки (назва атрибутів не є принциповою), яка буде посилатись на id статті та юзера, якому вона сподобалась, таким чином забезпечується зручне зберігання даних про вподобання.

Також, слід зауважити, що у статті може бути багато тегів, а у тегів багато статей, а отже знов виникає зв’язок «багато-до-багатьох», тобто потрібно створити проміжну таблицю article_to_tag.

Отже, визначено наступний блок статей-запитів, що показано на рисунку 2.3.

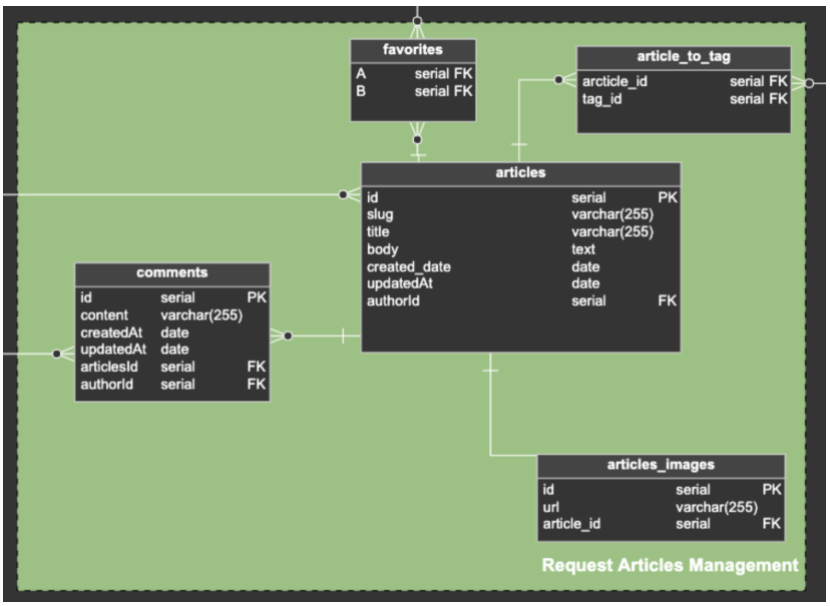

Рисунок 2.3 – Концептуальна діаграма блоку статей-запитів

2.3.2 Нормалізація та денормалізація

В контексті бази даних, нормалізація використовується для зменшення дублювання даних і забезпечення цілісності. Це досягається шляхом розбиття даних на кілька пов’язаних таблиць. У нашому випадку, нормалізація важлива для оптимального зберігання інформації про користувачів, їхні підписки, та взаємодію з психологами, яку вже було зроблено.

Попри те, денормалізація може бути застосована для оптимізації читання даних, шляхом введення деякого дублювання інформації для швидкого доступу.

Наприклад, таблиця з зображеннями до статей хоча й має право на логічне окреме існування, але більш доречним буде внести до таблиці самих статей, бо це лише одне поле в таблиці.

Далі, таблиця користувачів та інформації, хоча їх об’єднання буде створювати перевантаження сутності користувача інформацією, але сама третя нормальна форма нам говорить, що дотримання третьої нормальної форми, хоча теоретично й бажано, не завжди є практичним. Якщо, наприклад,

є таблиця «Клієнти» та було б бажано усунути всі можливі внутрішньо-польові залежності, потрібно було б створити окремі таблиці для міст, поштових індексів, представників з продажу, класів клієнтів та будь-яких інших факторів, які можуть дублюватися в кількох записах. Теоретично, нормалізацію варто переслідувати. Однак багато малих таблиць можуть погіршити продуктивність або перевищити ліміти відкритих файлів і пам’яті. Звідси, більш доцільно застосовувати третю нормальну форму тільки до даних, які часто змінюються. Якщо деякі залежні поля залишаються, необхідно проєктувати застосунок так, щоб користувач мав підтвердити всі пов’язані поля, коли будь-яке з них змінюється [25].

Отже, після проведення денормалізації моделі БД, вона матиме наступний вигляд (рис. 2.4).

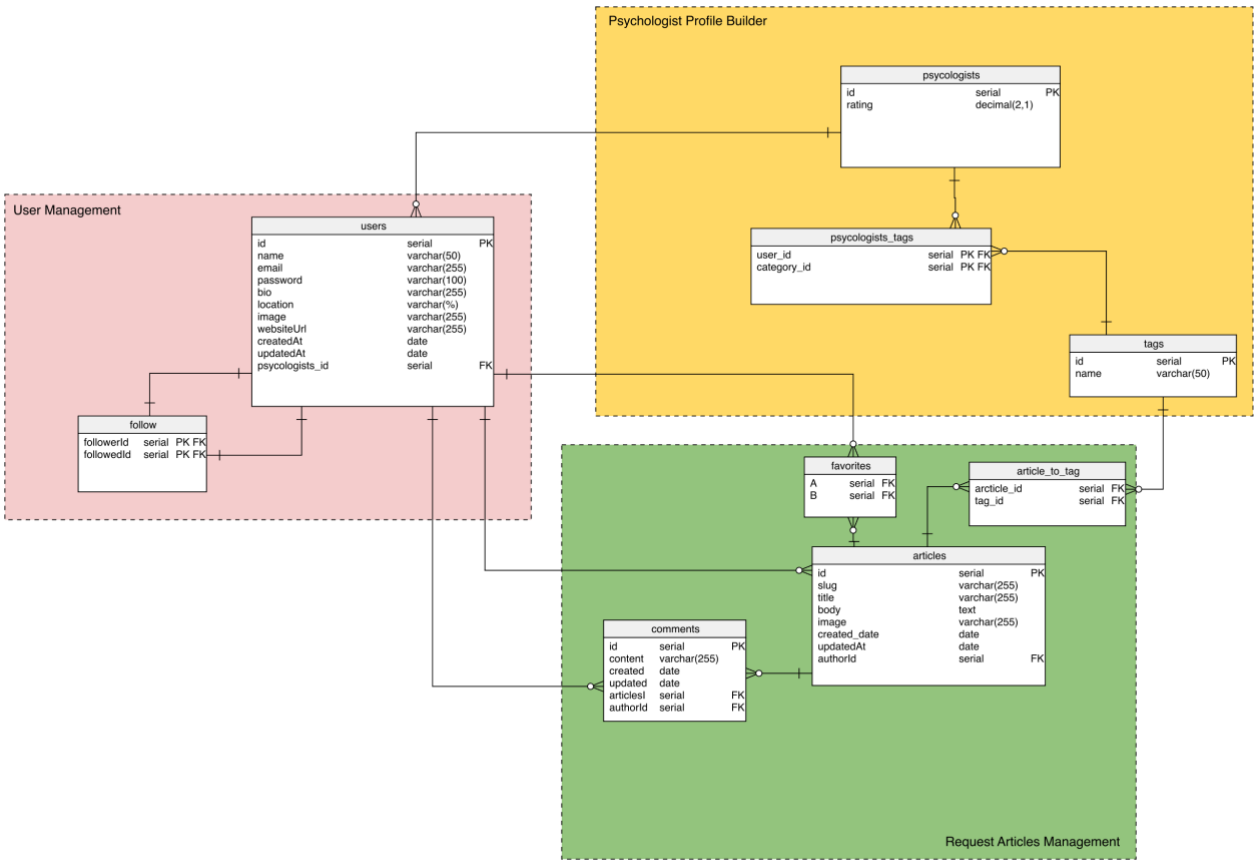

Рисунок 2.4 – Повна концептуальна діаграма БД

2.4 Визначення стеку технологій

Для розробки вебзастосунку було обрано TypeScript, оскільки він забезпечує строгу типізацію та підвищену надійність коду, що є критично важливим для зниження кількості помилок у розробці складних систем. TypeScript дозволяє використовувати одну мову програмування для написання як клієнтської, так і серверної частин, що спрощує розробку та підтримку проєкту, забезпечуючи більшу консистенцію коду і легкість в обміні кодовими ресурсами між різними частинами застосунку.

На бекенді було вирішено використовувати Node.js з фреймворком NestJS. Node.js відомий своєю високою продуктивністю у обробці асинхронних операцій та є ідеальним для реалізації мережевих застосунків. NestJS надає структурований підхід до розробки програмного забезпечення з інтеграцією TypeScript, що дозволяє більш чітко організовувати код та впроваджувати складні архітектурні рішення, забезпечуючи модульність та легке тестування компонентів системи.

Для фронт-енду було вибрано React.js, як високоефективний інструмент для розробки інтерактивних односторінкових застосунків (SPA) [26]. React.js сприяє створенню швидкодіючих вебсторінок, які можуть динамічно оновлювати свій вміст без перезавантаження, підвищуючи тим самим користувацький досвід. Це дозволяє ефективно управляти станами вебдодатків, спрощує передачу даних між компонентами та інтеграцію з іншими API.

За систему управління базами даних було обрано PostgreSQL через її потужні можливості у зберіганні та управлінні реляційними даними. PostgreSQL підтримує складні запити, транзакційність та надійність, що є важливим для застосунків, де цілісність даних має критичне значення. Реляційна модель дозволяє ефективно моделювати зв’язки між даними користувачів, статтями та коментарями, що забезпечує зручність обробки та аналізу великої кількості інформації.

Для спрощення взаємодії з базою даних було вибрано використання PrismaORM, що забезпечує потужний набір інструментів для взаємодії з базою даних у спосіб, що відповідає принципам сучасної програмної інженерії. PrismaORM дозволяє генерувати типізовані запити, що підвищує безпеку та надійність операцій з даними, а також сприяє швидкій розробці за рахунок використання готових абстракцій.

Кожен елемент технологічного стеку було обрано з метою забезпечення найбільшої ефективності, продуктивності та зручності розробки, що є ключовими факторами для успішного реалізації проєкту з вебзастосунку психологічної допомоги.

Для інтегрованого середовища розробки (IDE) було обрано WebStorm від JetBrains, який є одним з найпотужніших та найефективніших інструментів для розробки JavaScript-додатків, включно з підтримкою TypeScript та Node.js. WebStorm надає розширені можливості для редагування коду, включаючи автоматичне завершення, аналіз коду, рефакторинг та інтеграцію з різноманітними системами контролю версій. Це IDE також включає підтримку сучасних фреймворків як React.js, що дозволяє розробникам ефективно працювати з фронтендом. Використання WebStorm сприяє підвищенню продуктивності розробки та полегшує процес пошуку та виправлення помилок, що робить його ідеальним вибором для нашого проєкту.

Docker було вибрано як ключовий інструмент для контейнеризації та оркестрації середовища розробки і виробництва. Docker дозволяє розробникам пакувати додатки разом із їхніми залежностями у стандартні контейнери, які можуть бути запущені на будь-якому комп’ютері. Це забезпечує консистентність середовища на всіх етапах розробки, тестування та впровадження, значно знижуючи «розриви» між локальними розробниками та виробничим середовищем. Крім того, Docker спрощує управління версіями, розгортання та масштабування застосунків, забезпечуючи легке оновлення та відкат змін. Це робить Docker незамінним інструментом для забезпечення ефективності та надійності інфраструктури проєкту.

Для тестування ендпойнтів бек-енду було обрано використання Postman, що є потужним інструментом для розробки та тестування API. Postman дозволяє розробникам швидко створювати, обробляти та відправляти HTTP запити до різних ендпойнтів, а також переглядати відповіді сервера. Цей інструмент підтримує різні методи HTTP, такі як GET, POST, PUT, DELETE, що є необхідним для тестування різноманітних функцій API.

За допомогою Postman можна легко організувати тестові запити в колекції, які можуть бути поділені між членами команди, що сприяє співпраці та ефективності у процесі розробки. Інструмент також підтримує автоматизацію тестів і інтеграцію з CI/CD пайплайнами через власні скрипти та API. Це дозволяє автоматизувати тестування API після кожного оновлення коду, забезпечуючи стабільність і надійність вебзастосунку.

Використання Postman значно підвище ефективність тестування і забезпечує високу якість бек-енд розробки, дозволяючи швидко виявляти та виправляти помилки в API, що важливо для загальної успішності проєкту.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Налаштування середовища

3.1.1 Ініціалізація проєкту

Для розробки вебзастосунку з використанням сучасних технологій та методів, основа проєкту розділена на дві основні частини: клієнтську (фронт-енд) і серверну (бек-енд). Кожна частина буде розміщена у власній папці в рамках одного проєкту, що сприяє чіткому розділенню розробки та легкості управління кодом.

Початок роботи над проєктом передбачає встановлення необхідних залежностей через npm (Node Package Manager), який використовується для управління пакетами як на клієнті, так і на сервері. Для кожної частини проєкту будуть створені відповідні package.json файли, які зберігають інформацію про проєкт і список залежностей, необхідних для його виконання.

Клієнтська частина розташована у папці client, де використовується Vite як інструмент збірки для React-додатків. Vite надає швидку збірку та гарячу заміну модулів (HMR), що значно покращує досвід розробки фронт-енду. Структура папки включає підкаталоги для компонентів, хуків, стилів та служб API, що забезпечує логічне та ефективне розташування ресурсів проєкту.

Оскільки розробляється клієнтська частина з використанням React та TypeScript, скористаємось наступною командою в консолі у відповідній кореневій директорії нашого проєкту:

Лістинг 3.1. Консольна команда для ініціалізації клієнтської частини:

```
npm create vite@latest client -- --template react-ts
```

Серверна частина розташована у папці server, використовуючи Node.js з фреймворком NestJS. NestJS забезпечує структурований підхід до розробки серверних застосунків з використанням модулів, сервісів і контролерів, що

полегшує підтримку та масштабування проєкту. Фреймворк заснований на TypeScript, що дозволяє використовувати всі переваги строгої типізації для підвищення якості коду.

Для ініціалізації серверної частини спочатку нам знадобиться глобально встановити cli для фреймоврку, зробити це можна наступною командою у консолі:

Лістинг 3.2. Консольна команда для встановлення cli фреймворку NestJS:

npm i -g @nestjs/cli

Після цього буде ініціалізовано серверну частину у проєкті за допомогою встановленого cli:

Лістинг 3.3. Консольна команда для ініціалізації серверної частини:

nest new server

Такий підхід до структуризації та ініціалізації проєкту забезпече чітке розділення обов’язків між клієнтською та серверною частинами, а також одразу встановить усі необхідні залежності та створить початкові файли конфігурації.

3.1.2 Налаштування віртуалізації через Docker

Для розгортання та тестування вебзастосунку в ізольованому середовищі використовується Docker, що дозволяє створити легко переносимі та консистентні контейнери для кожної частини проєкту, не встановлювати більшість програмного забезпечення локально на машину, а також надає

можливість іншим розробникам запускати застосунок без потреби налаштування окремого проєкту.

Для роботи с Docker у обох директоріях проєкту було створено по відповідному Dockerfile.

Файл Dockerfile у директорії client налаштовує середовище для клієнтської частини, використовуючи базовий образ node:alpine через його легкість та ефективність. У Dockerfile йде налаштування наступних команд:

– встановлення робочої директорії в контейнері (/app);

– копіювання файлів package.json та package-lock.json до робочої директорії та виконання npm install для встановлення залежностей;

– копіювання всіх файлів проєкту до робочої директорії;

– відкриття порту 8080 для зв’язку з вебсервером;

– виконання команди npm run dev для запуску клієнтської частини.

Отже, Dockerfile для клієнту буде виглядати, як показано у лістингу 3.4.

Лістинг 3.4. Dockerfile для клієнту:

```
FROM node:alpine
LABEL authors="vitaliisam"
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8080
CMD [ "npm", "run", "dev"]
```

Схожий до клієнтського, Dockerfile у директорії server також використовує node:alpine. Процес будівництва образу включає аналогічні кроки, з виключенням відкриття порту 3000 для доступу до API сервера, а також список команд для запуску серверу буде іншим (npm run start:dev).

Далі, важливим кроком буде додати файл .dockerignore в обох директоріях (client і server). Він допоможе уникнути включення в образ непотрібних файлів, тобто ігноруватиме такі файли та директорії як node_modules, логі, конфігураційні файли та інші несуттєві файли, що забезпечує чистоту та мінімальний розмір кінцевого Docker образу.

Лістинг 3.5. Файл змінних .dockerignore:

```
Dockerfile
.dockerignore
node_modules
npm-debug.log
dist
.gitignore
.git
.env
config
build
docker-compose.yaml
README.md
```

Після конфігурування докер-файлів для клієнту та серверу, було створено у корневому директорії всього проєкту файл «docker-compose.yaml», в якому розписано конфігурацію для композиції контейнерів. Він буде включати в себе чотири сервіси, а саме базу даних (PostgreSQL), інструмент управління базою даних (pgAdmin), бек-енд (Node.js з NestJS) та фронт-енд (React.js).

Перший сервіс бази даних (db) PostgreSQL використовує образ postgres:14.1-alpine з Docker Hub та має наступні змінні:

– env_file: задає файл середовища, який містить змінні середовища для конфігурації;

– restart: з параметром «always» гарантує, що контейнер перезапускається автоматично при виході;

– environment: задає додаткові змінні середовища, такі як ім’я користувача та пароль для бази даних;

– container_name: надає ім’я контейнеру для легшого доступу;

– ports: відображення портів між хостом і контейнером;

– volumes: збереження даних бази даних на постійному диску замість тимчасового сховища контейнера.

Наступний сервіс – pgAdmin, вебінтерфейс управління базами даних PostgreSQL. З нових параметрів буде змінна «depeds_on», що вказує на те, що pgAdmin повинен запускатися після запуску бази даних. А також пару змінних середи конфігурації облікових записів за замовчуванням для доступу до pgAdmin.

Після цього необхідно налаштувати сервіси backend і frontend з сервером і клієнтом відповідно, в якому вказаний їх контекст з докер-файлами, що створені раніше, їх порти, файли середовища та інше. Їх буде налаштовано так, що сервер буде залежати від БД, а клієнт від серверу. В клієнті буде додано змінні «stdin_open» та «tty» для забезпечення інтерактивного режиму роботи контейнера для розробки.

Наприкінці визначаються томи, які задають конфігурацію для зберігання даних, що гарантує їх збереження навіть після перезапуску або видалення контейнерів. Файл композиції контейнерів буде виглядати, як показано у лістингу 3.6.

Лістинг 3.6. Файл композиції docker.compose.yaml:

```
version: '3.8'
services:
  db:
    image: postgres:14.1-alpine
    env_file:
```

```
    - server/.env
  restart: always
  environment:
    - POSTGRES_USER=root
    - POSTGRES_PASSWORD=root
  container_name: postgres
  ports:
    - '5432:5432'
  volumes:
    - db:/var/lib/postgresql/data

pgadmin:
  image: dpage/pgadmin4
  restart: always
  container_name: nest-pgadmin4
  environment:
    - PGADMIN_DEFAULT_EMAIL=admin@admin.com
    - PGADMIN_DEFAULT_PASSWORD=pgadmin4
  ports:
    - '5050:80'
  depends_on:
    - db

backend:
  container_name: backend
  build:
    context: ./server
    dockerfile: Dockerfile
  env_file:
    - ./server/.env
```

```
    image: server
    ports:
      - "3000:3000"
    depends_on:
      - db
    volumes:
      - ./server:/app

  frontend:
    container_name: client
    build:
      context: ./client
      dockerfile: Dockerfile
    env_file:
      - ./client/.env
    image: client
    ports:
      - "8080:8080"
    stdin_open: true
    tty: true
    depends_on:
      - backend

volumes:
  db:
    driver: local
```

3.2 Реалізація фізичного рівня БД

Для реалізації фізичного рівня БД скористаємось Prisma ORM, це інструмент для роботи з базами даних у застосунках, що підтримують Node.js та TypeScript. Він пропонує зручний спосіб взаємодії з реляційними базами даних через використання високорівневої абстракції, що спрощує розробку та підтримку додатків.

Для цього в директорії серверу буде створено нову папку під назвою «prisma», в якій будуть зберігатись міграції для бази даних і сам файл «schema.prisma». Основою Prisma є мова опису моделей, яка визначається у цьому файлі. Ця мова є декларативною та сильно типізованою, що дозволяє чітко визначати структуру даних, яка буде використовуватися в базі даних. Мова включає в себе:

– типи полів, які можуть бути примітивними типами даних, такими як String, Int, Boolean, DateTime тощо, а також складнішими типами, які описують зв’язки між моделями через @relation;

– атрибути поля: кожне поле може мати атрибути, такі як @id для первинного ключа, @default для значення за замовчуванням, @unique для унікальності, та інші, ці атрибути дозволяють детально налаштовувати поведінку кожного поля;

– міграції: Prisma використовує мову опису моделей для генерації міграцій, які автоматично застосовуються до бази даних для оновлення її структури відповідно до останніх змін у моделях.

Для підключення до БД буде створений у директорії сервера файл «.env», в якому описуються змінні середовища, це забезпечить гнучкість та приватність, оскільки можна визначити цей файл не включеним до загального репозиторію проєкту. У файлі середовищ буде визначено змінну «DATABASE_URL», що буде рядком URL підключення до БД, яка створена за допомогою DOCKER.

Далі, у файлі «schema.prisma» описується підключення, що буде URL з файлу середовищ та провайдера для генерації, корекції, а також міграцій, виглядати це буде наступним чином:

Лістинг 3.7. Описання підключення та провайдера у файлі «schema.prisma»:

```
generator client {
  provider = "prisma-client-js"
}
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}
```

Далі, буде створено моделі-сутності, які описані в підрозділі 2.3. Моделі визначаються ключовим словом «model *Назва сутності*» у вигляді об’єкту та всередині нього описується назву атрибуту, його тип та через декоратори параметри атрибуту (такі як унікальність, значення за замовчуванням первинні ключі, тощо), а також зв’язки з іншими сутностями.

Описання моделі сутності статті «Article» розглянуто у лістингу 3.8.

Лістинг 3.8. Описання сутності «Article» у файлі «schema.prisma»:

```
model Article {
  id        String    @id @default(uuid())
  slug      String    @unique
  title     String
  body      String    @default("")
  image     String    @default("")
  tagList   Tag[]
  createdAt DateTime  @default(now()) @map("created_at")
```

```
  updatedAt DateTime  @updatedAt @map("updated_at")
  author    User      @relation("articles", fields: [authorId], references: [id])
  authorId  String    @map("author_id")
  favorited User[]    @relation("favorites")
  comments  Comment[]
  @@map("articles")
}
```

Спочатку йдуть назви атрибутів (id, назва, автор, тощо), вказується їх відповідний тип та параметри. Для атрибуту «id» це буде @id, а також значення за замовчуванням, що генерується автоматично за допомогою функції uuid(). Використання UUID гарантує унікальність ідентифікаторів на глобальному рівні, що є важливим для систем з розподіленими базами даних.

За допомогою параметру @map можна вказувати точну назву атрибуту у БД, а також зв’язків та констрейнтів (обмежень).

Для зв’язку з іншою сутністю використовується параметр @relation, в якому вказуєть, до якої сутності йду зв’язок-відношення та за якими полями. Якщо зв’язок «один-до-багатьох», вказується сутність як масив сутностей (як, наприклад, з користувачами та тегами).

В кінці надається назва сутності у БД, таким чином, можна використовувати моделі у застосунку з рівнем абстракції, а також сама БД буде мати контекст, до чого саме йде звернення через таку мову опису.

Аналогічним чином буде створено та описано наступні сутності схеми (лістинг 3.9).

Лістинг 3.9. Моделі-сутності у файлі «schema.prisma»:

```
model User {
  id          String      @id @default(uuid())
  email       String      @unique
  name        String
```

```
    password      String
    bio           String       @default("")
    image         String       @default("")
    location      String       @default("")
    websiteUrl    String       @default("") @map("website_url")
    createdAt     DateTime     @default(now()) @map("created_at")
    updatedAt     DateTime     @updatedAt @map("updated_at")
    articles      Article[]    @relation("articles")
    favorites     Article[]    @relation("favorites")
    following     Follow[]     @relation("follower")
    followers     Follow[]     @relation("following")
    comments      Comment[]
    psychologist  Psychologist? @relation(fields: [psychologistId],
references: [id])
    psychologistId String?      @unique
    @@map("users")
  }
  model Psychologist {
    id     String  @id @default(uuid())
    tagList Tag[]
    rating  Decimal @default(0) @db.Decimal(2, 1)
    User    User?
    @@map("psychologists")
  }
  model Tag {
    id          String       @id @default(uuid())
    name        String       @unique
    articles    Article[]
    psychologists Psychologist[]
    @@map("tags")
```

```
}
model Follow {
  follower    User?    @relation("follower", fields: [followerId], references: [id])
  followerId  String   @map("follower_id")
  following   User?    @relation("following", fields: [followingId], references: [id])
  followingId String   @map("following_id")
  createdAt   DateTime @default(now()) @map("created_at")
  @@id([followerId, followingId])
}
model Comment {
  id        String   @id @default(uuid())
  content   String
  author    User     @relation(fields: [authorId], references: [id])
  authorId  String   @map("author_id")
  article   Article  @relation(fields: [articleId], references: [id])
  articleId String   @map("article_id")
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
}
```

Завершальним кроком підключення буде впровадження модуля та сервісу Prisma у головний модуль нашого серверу «app.module.ts» (лістинг 3.10).

Лістинг 3.10. Впровадження сервісу та модуля у файлі «app.module.ts»:

```
@Module({
  imports: [
    PrismaModule,
```

```
    ConfigModule.forRoot(),
  ],
  controllers: [AppController],
  providers: [AppService, PrismaService],
})
export class AppModule {}
```

Далі, для генерації таблиць у нашій фізичній БД необхідно виконати міграції на основі опису сутностей, для цього скористаємось консольною командою (обов'язково робити у директорії серверу, оскільки саме там визначена Prisma).

Лістинг 3.11. Консольна команда міграції Prisma:

```
npx prisma migrate dev
```

Після того, як міграції успішно виконані, можна побачити створені таблиці у нашій фізичній БД (рис. 3.1).

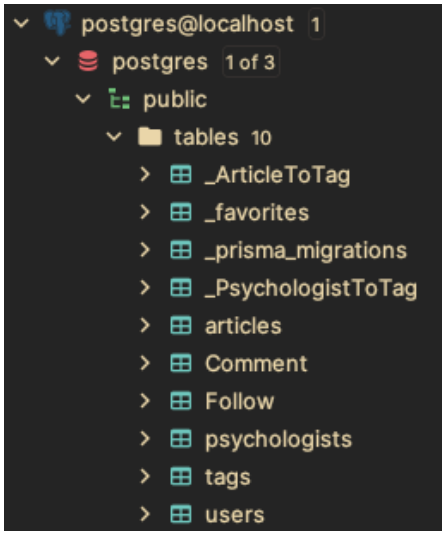

Рисунок 3.1 – Таблиці у фізичній БД

Отже, після створення таблиць визначена наступна схематика сутностей у застосунку (рис. 3.2).

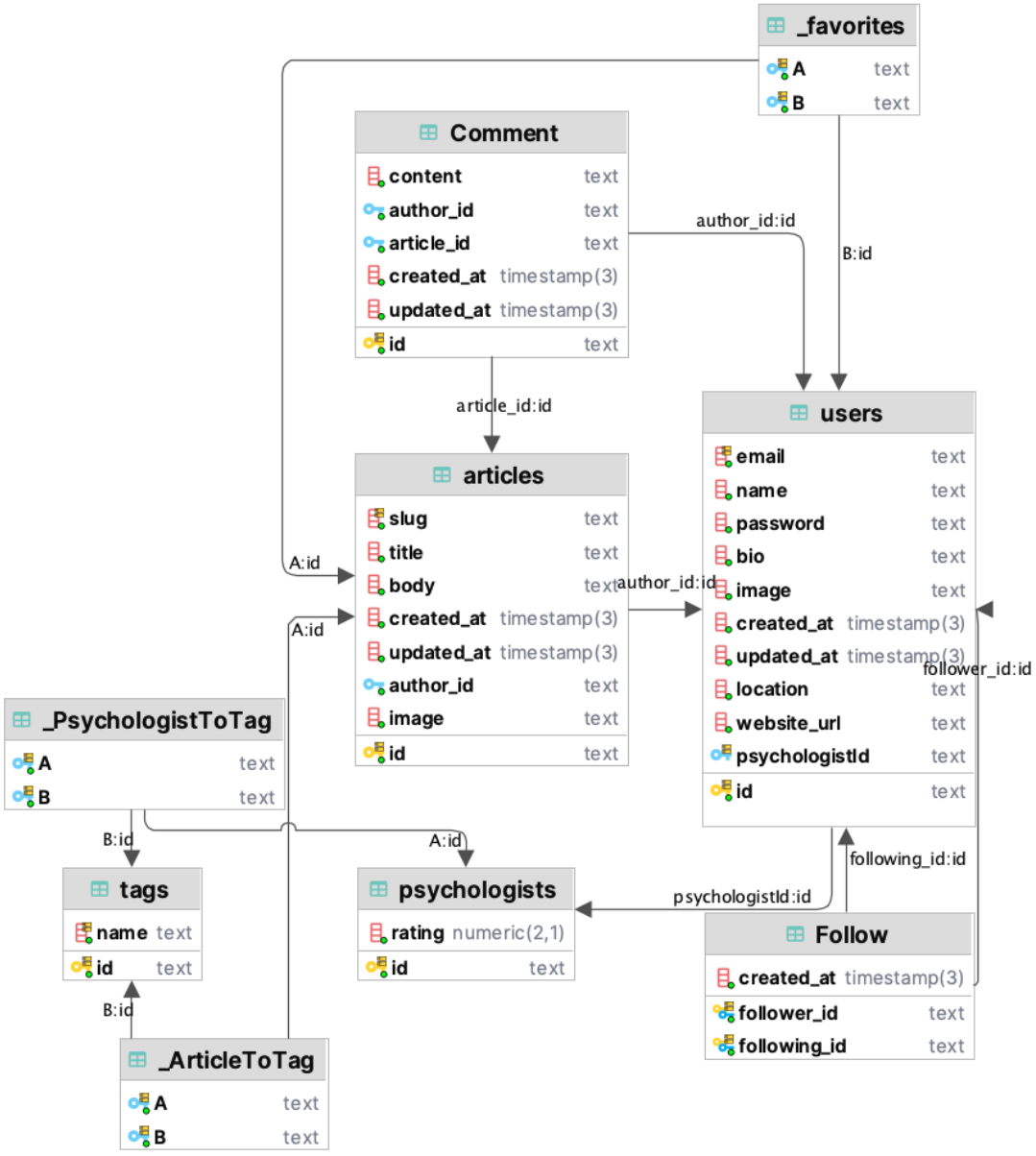

Рисунок 3.2 – Діаграма сутностей у застосунку

3.3 Реалізація серверної частини застосунку

Для побудови серверу використовується NestJS – фреймворк для побудови ефективних і масштабованих серверних додатків. Він використовує сучасні концепції JavaScript і TypeScript, забезпечуючи інтеграцію з іншими бібліотеками та інструментами. Фреймворк має модульну структуру, що дозволяє легко організовувати код у логічно відокремлені частини, тим самим

спрощуючи розробку та тестування. Тобто кожна сутність має свої окремі модулі, що є основою концепції архітектури застосунку.

В структуру, що диктує NestJS, для кожної сутності необхідним є використання модулів, контролерів, провайдерів (сервісів), та мідлвеєрів. Це забезпечує чітку структуру та організацію коду, де кожен компонент виконує визначену роль:

– модулі визначають межі для пов’язаних функціональностей. Кожен модуль інкапсулює провайдерів, контролери, інші модулі та експортує необхідні сервіси, що дозволяє розділити застосунок на декомпоновані блоки;

– контролери відповідають за обробку вхідних запитів і повернення відповідей до клієнта. Вони визначають маршрути та пов’язані з ними обробники, що забезпечують взаємодію з клієнтом, тобто впроваджують створення API. Робиться це через відповідні декоратори для визначення маршрутів (наприклад, @Get, @Post, @Put, @Delete) та впровадження залежностей через конструктор для інтеграції сервісів;

– провайдери (сервіси) забезпечують бізнес-логіку та можуть бути ін’єктовані в контролери або інші сервіси. Це дозволяє підтримувати низьку зв’язність та високу згуртованість коду;

– мідлвеєри використовуються для виконання коду до або після обробки маршрутів. Вони можуть використовуватися для логування, автентифікації, гардів або інших загальних задач обробки запитів.

3.3.1 Модуль аутентифікації

Модуль аутентифікації є ключовою складовою вебзастосунку для психологічної допомоги, оскільки він відповідає за реєстрацію та верифікацію користувачів з забезпеченням безпечного доступу до особистих і чутливих даних. Модуль включає декілька важливих компонентів, розташованих у відповідних директоріях для організації коду.

Автентифікацію буде реалізовано за допомогою JWT-токенів. JSON Web Token (JWT) використовується для безпечної передачі інформації між клієнтом та сервером як компактний, самодостатній токен. Токени JWT містять закодовані дані користувача та цифровий підпис, що гарантує їхню цілісність та автентичність.

Для цього створено папку auth у директорії серверу, а також наступні внутрішні директорії та файли.

Першою буде папка конфігурації Config, в якій у файлі jwt.config.ts описуєтья секретний ключ для серверу, вказаний у файлі середовища серверу .env (необхідно додати нову змінну) та конфігурацію, яка включає параметри токенів, такі як термін дії.

Наступною буде папка dto (Data Transfer Objects), в якій є структури даних, що використовуються для передачі даних між різними частинами аплікації, зокрема між клієнтами та сервером у мережевих запитах. Таким чином можна чітко визначити, які дані передаються в конкретних операціях, включаючи поля, що мають бути заповнені, та валідаційні правила, які можуть застосовуватися. Це позбавляє від зайвих кроків перевірки інформації на відповідність.

У цій директорії створюються три файли:

– login.dto.ts: описує структуру даних для входу користувачів;

– refresh-token.dto.ts: описує дані для оновлення токенів;

– register.dto.ts: визначає дані, які користувач повинен надати при реєстрації.

Виглядати це буде наступним чином, на прикладі лістингу 3.12.

Лістинг 3.12. DTO для реєстрації, файл «register.dro.ts»:

```
export class RegisterDto {
  @IsEmail()
  email: string
```

```
  @IsString()
  @IsNotEmpty()
  name: string

  @MinLength(6)
  @IsString()
  password: string

  @IsNotEmpty()
  isPsychologist: boolean
}
```

Схожим чином працюють й інтерфейси, але вони визначають структуру, яку мають наслідувати класи або об’єкти, що їх імплементують. Вони встановлюють чіткий контракт для методів і властивостей, забезпечуючи консистентність і низький рівень зв’язності між компонентами системи, що полегшує тестування та масштабування. Використання інтерфейсів також сприяє використанню поліморфізму, дозволяючи об’єктам різних класів взаємодіяти через загальний інтерфейс. Оскільки використовується typescript, це надає безпечність використання даних та необхідний контекст при розробці. В папці interfaces буде створено два файли:

– auth.ts: інтерфейс, який описує структуру респонсу автентифікації на основі токену;

– token.ts: інтерфейси для структуризації інформації токенів (id, email, iat, exp).

Наступною буде директорія guards, в якій створено власний JwtGuard у вигляді декоратора, це забезпечує безпеку та авторизацію користувачів на певних маршрутах або методах контролера. Цей декоратор використовується для перевірки наявності та валідності JWT (JSON Web Token), що передається у заголовках запиту. Якщо токен відсутній або недійсний, доступ до обраних

частин застосунку буде заблокований, тим самим запобігаючи несанкціонованому доступу та забезпечуючи, що тільки автентифіковані користувачі мають доступ до критичних ресурсів. Тобто, можна його використати на запиті при редагуванні статті та якщо користувач не авторизований та не є автором статті – доступу на редакцію не буде надано. Такий підхід захисту реалізується у файлі «jwt.guard.ts», успадковоючись від класа AuthGuard, бібліотеки passport.

Також, для перевірки на валідність буде створено файл зі стратегію валідації токенів у дикекторії strategies: jwt.strategy.ts – стратегія для Passport.js, яка визначає методи валідації JWT.

У корені директорії «auth» буде три файли:

– auth.service.ts: сервіс, що містить бізнес-логіку для реєстрації, входу, та видачі токенів, містить наступні методи:

1) async register(dto: RegisterDto): Promise<AuthResponse>: асинхронний метод для реєстрації користувача, приймає DTO для реєстрації, та повертає AuthResponse (тобто користувача з авторизацією);

2) async login(dto: LoginDto): Promise<AuthResponse>: асинхронний метод авторизації користувача, приймає відповідне DTO, також повертає авторизованого користувача;

3) async getNewTokens(dto: RefreshTokenDto): Promise<AuthResponse>: асинхронний метод видачі токену авторизації користувачеві, з відповідними DTO та респонсом;

4) private async generateTokens(user: User): Promise<Tokens>: приватний асинхронний метод генерації самих токенів, що приймає користувача, та повертає токен;

5) private hashPassword(str: string): string: приватний метод для хешування паролів при створенні користувачів;

– auth.controller.ts: контролер, який приймає HTTP запити для реєстрації та аутентифікації. В контролері можна працювати з запитами, використовуючи методи, що визначені у сервісі модуля:

1) POST метод register: за адресою /auth/register (якщо розробляється локально, то повна адреса буде виглядати наступним чином: http://localhost:3000/auth/login), приймає дані реєстрації користувача (RegisterDto) та викликає метод register() сервісу AuthService для створення нового користувацького облікового запису. Повертає дані авторизації, зокрема токени доступу та оновлення, якщо реєстрація успішна;

2) POST метод login: за адресою /auth/login, приймає облікові дані користувача (LoginDto) для входу в систему та викликає метод login() сервісу AuthService. Повертає дані авторизації, включаючи токени доступу та оновлення, якщо аутентифікація успішна;

3) POST метод refresh-token: за адресою /auth/refresh-token, приймає токен оновлення (RefreshTokenDto) та викликає метод getNewTokens() сервісу AuthService для оновлення токенів доступу. Повертає оновлені дані авторизації, включаючи нові токени доступу та оновлення;

– auth.module.ts: основний модуль, який інтегрує всі компоненти модуля аутентифікації. Тобто головний файл всього модулю автентифікації, який потім імпортується в модуль всієї серверної частини, виглядає він як показано на лістингу 3.13.

Лістинг 3.13. Модуль «auth.module.ts»:

```
@Module({
  imports: [
    PrismaModule,
    JwtModule.registerAsync({
      imports: [ConfigModule],
      inject: [ConfigService],
      useFactory: getJwtConfig,
    }),
    ConfigModule,
  ],
```

```
    controllers: [AuthController],
    providers: [AuthService, JwtStrategy, JwtGuard],
})
export class AuthModule {}
```

Таким самим чином побудовані й інші модулі з їх відповідною бізнес-логікою та API, а саме модуль статті (article), користувача (user), тегу (tag), підписок (follow), коментарів (comment), а також сервіс обробки та збереження зображень(немає власного API), що будуть зберігатися на сторонньому сервісі (cloudinary).

API для модуля користувача (user):

– GET /user: використовує JwtGuard, витягує параметр id з токену, повертає поточного залогіненого користувача по id;

– GET /user/psychologists: приймає параметри запиту (GetPsychologistsQueryParamsDto), повертає усіх психологів;

– GET /user/:id: приймає id, повертає користувача по id;

– PUT /user: використовує JwtGuard, витягує параметр id з токену, приймає UpdateUserDto, оновлює користувача;

– PUT /user/update-avatar: використовує JwtGuard, використовує FileInterceptor для обробки файлу, приймає зображення як «avatar», витягує параметр id з токену, оновлює аватар користувача;

– PUT /user/change-password: використовує JwtGuard, витягує параметр id з токену, приймає ChangePasswordDto, змінює пароль користувача;

– DELETE /user: використовує JwtGuard, витягує параметр id з токену, видаляє поточного користувача;

– POST /user/:fuid/follow: використовує JwtGuard, витягує параметр id з токену, приймає fuid, додає користувача до списку на кого підписані;

– DELETE /user/:fuid/follow: використовує JwtGuard, витягує параметр id з токену, приймає fuid, видаляє користувача зі списку на кого підписані.

API для модуля статей (articles):

– GET /articles: приймає параметри запиту (GetArticlesQueryParamsDto), повертає всі статті;

– GET /articles/:id: приймає id, повертає одну статтю по id;

– GET /articles/author/:authorId: приймає authorId та параметри запиту (GetArticlesQueryParamsDto), повертає статті автора за authorId;

– GET /articles/user/reading-list: використовує JwtGuard, витягує параметр id з токену, повертає список читання користувача;

– POST /articles: використовує JwtGuard, UseInterceptors для обробки зображення, витягує параметр id з токену, приймає CreateArticleDto та зображення, створює статтю;

– PUT /articles/:id: використовує JwtGuard, UseInterceptors для обробки зображення, витягує параметр uid з токену, приймає id, UpdateArticleDto та зображення, оновлює статтю;

– DELETE /articles/:id: використовує JwtGuard, витягує параметр uid з токену, приймає id, видаляє статтю;

– POST /articles/:id/favorite: використовує JwtGuard, витягує параметр id з токену, приймає id, додає статтю до обраних;

– DELETE /articles/:id/favorite: використовує JwtGuard, витягує параметр id з токену, приймає id, видаляє статтю з обраних.

API для модуля коментарів (comment):

– GET /comments/:articleId: приймає articleId, повертає всі коментарі до статті за articleId;

– POST /comments/:articleId: використовує JwtGuard, витягує параметр authorId з токену, приймає articleId та CreateCommentDto, створює коментар до статті.

API для модуля підписки (follow):

– GET /:id/following: використовує JwtGuard, приймає id та параметри запиту (GetFollowsDto), повертає список користувачів, на яких підписаний користувач з заданим id;

– GET /:id/followers: використовує JwtGuard, приймає id та параметри запиту (GetFollowsDto), повертає список користувачів, які підписані на користувача з заданим id.

API для модуля тегів (tag):

– GET /tags: приймає параметри запиту (GetTagsDto), повертає список усіх тегів;

– GET /tags/:tagName: приймає tagName та параметри запиту (GetTagsDto), повертає статті, асоційовані з конкретним тегом;

– POST /tags: використовує JwtGuard, приймає масив даних для створення тегів (CreateTagsDto[]), створює нові теги та повертає кількість створених тегів.

Після завершення файлова структура серверу матиме наступний вигляд (рис. 3.3).

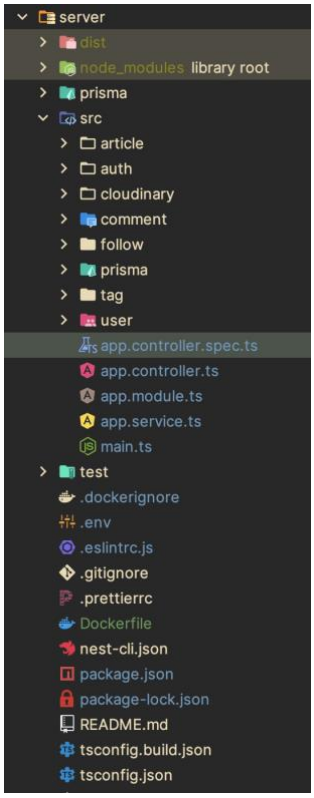

Рисунок 3.3 – Файлова структура серверу

Як всі компоненти взаємодіють між собою без бібліотек та утиліт можна візуалізувати наступним чином, як показано на рисунку 3.4.

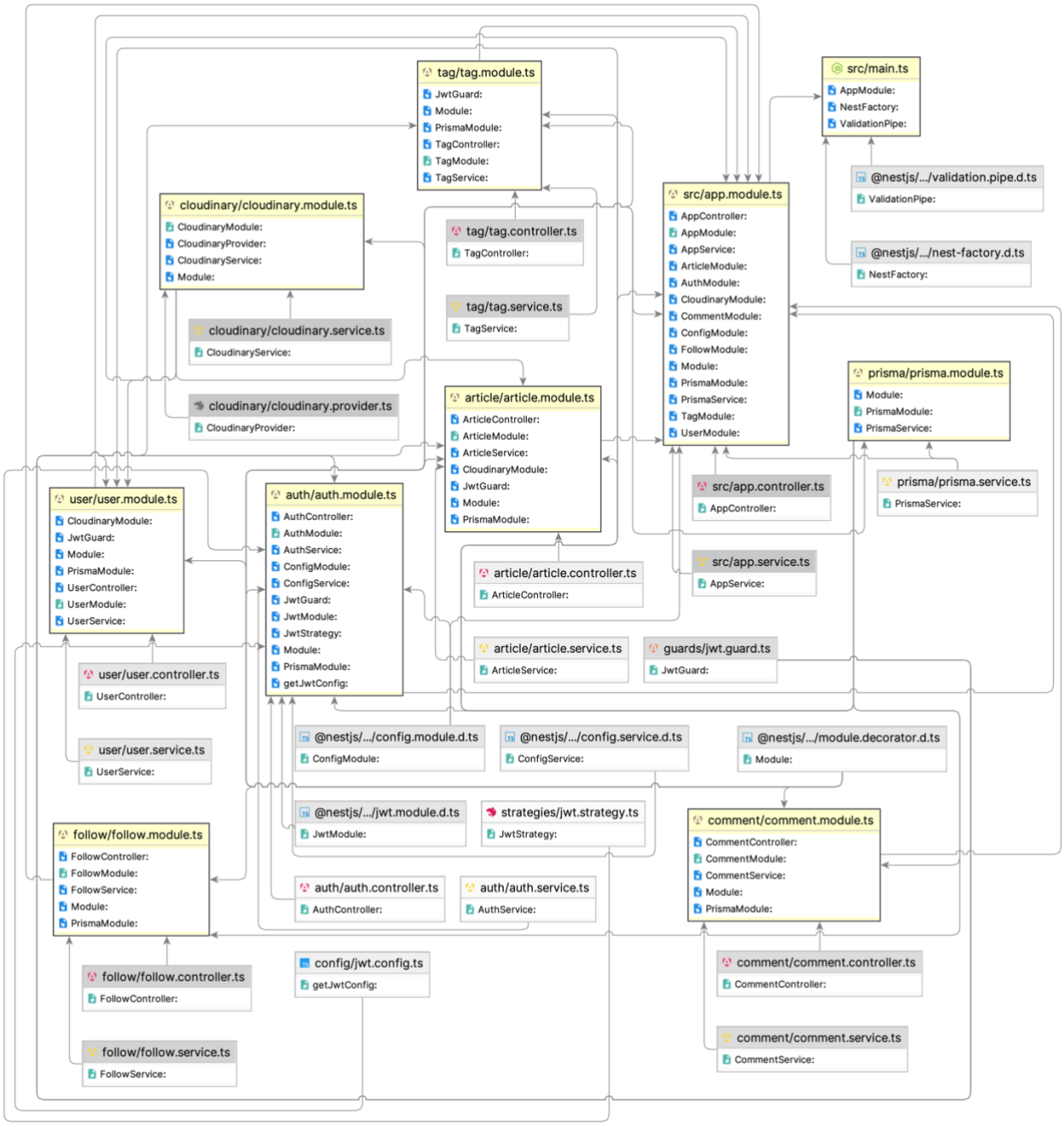

Рисунок 3.4 – UML-діаграма компонент серверу

3.3.2 Тестування серверної частини

3.3.2.1 Тестування енд-поінтів з Postman

Тестування ендпоінтів через Postman – це процес виконання HTTP-запитів до API для перевірки його функціональності та відповідності специфікації. Тестування ендпоінтів модуля auth включає:

– реєстрація нового користувача: створення POST-запиту до ендпоінта api/auth/register з необхідними даними користувача (електронна адреса, пароль, ім’я, чи є психологом, та за бажанням інші поля) у тілі запиту у вигляді json-формату (рис. 3.5). На це отримаємо респонс (відповідь серверу) зі статусом серверу, з перевіркою відповіді на успішну реєстрацію та отримання токенів аутентифікації (рис. 3.6);

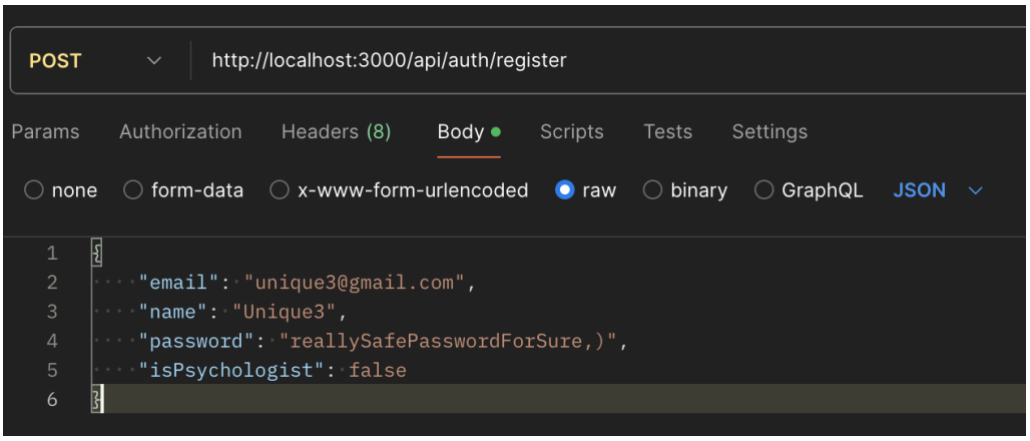

Рисунок 3.5 – POST-запит на реєстрацію користувача

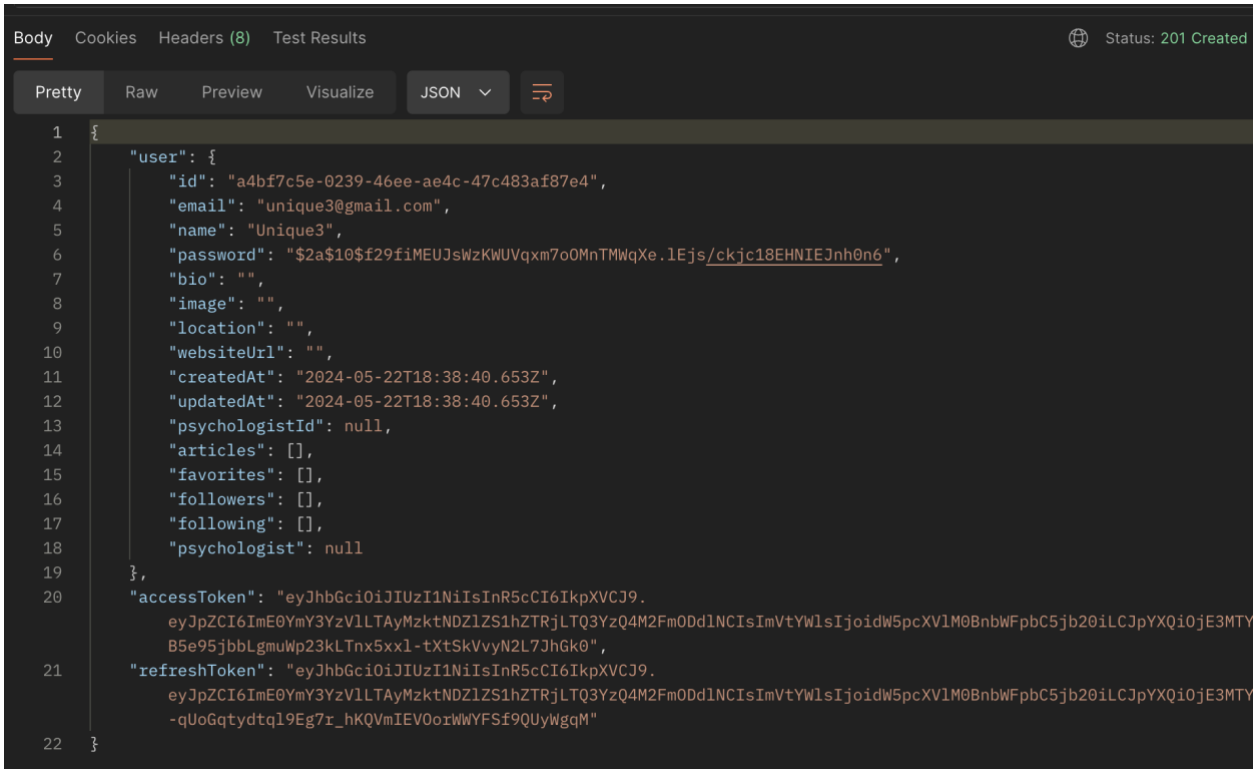

Рисунок 3.6 – Відповідь сервера про успішне створення користувача (статус «201»)

– вхід користувача: виконання POST-запиту до ендпоінта api/auth/login (рис. 3.7-3.8) з відповідними обліковими даними користувача (електронна адреса, пароль). Перевірка відповіді на успішний вхід та отримання токенів, якщо дані авторизації користувача вірні, відповідь сервера буде містити відповідного користувача з токенами та статусом запиту «201» про успішне створення токенів та вхід у системи. Якщо ж дані авторизації не відповідають жодному користувачеві, то відповідь буде мати в собі повідомлення про помилку та статусом «400» про невдачу (рис. 3.9);

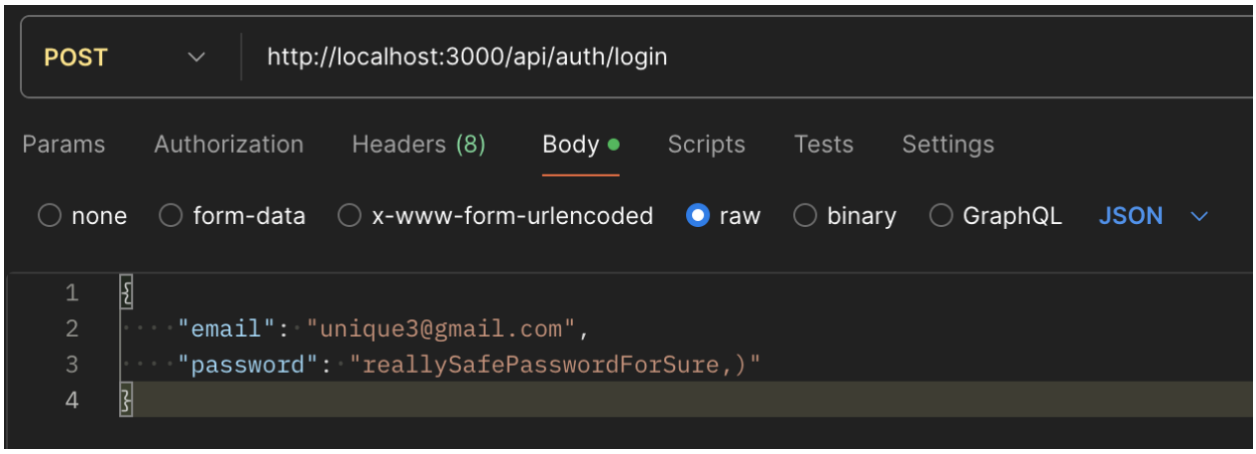

Рисунок 3.7 – POST-запит на авторизацію користувача з правильними даними авторизації

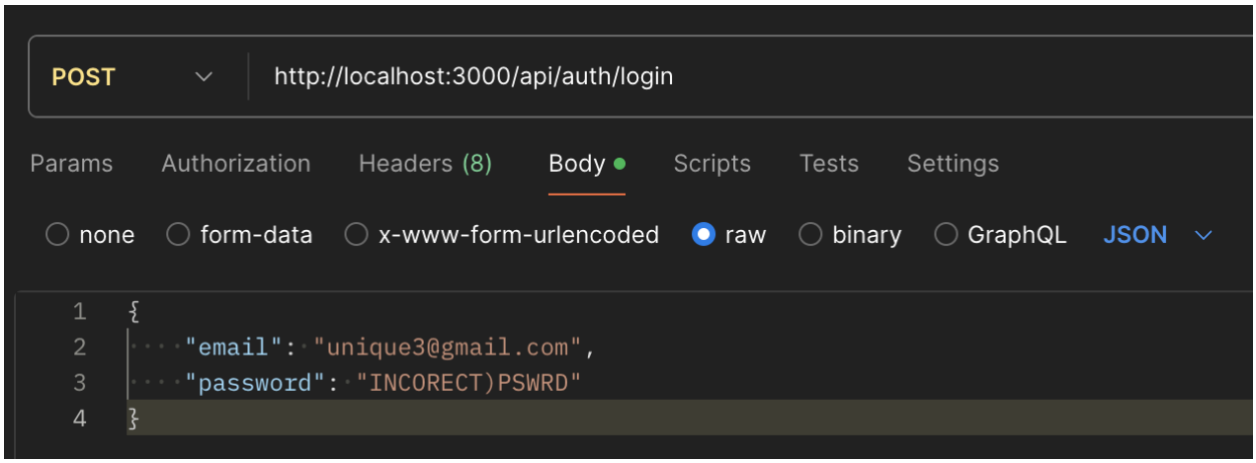

Рисунок 3.8 – POST-запит на авторизацію користувача з неправильними даними авторизації

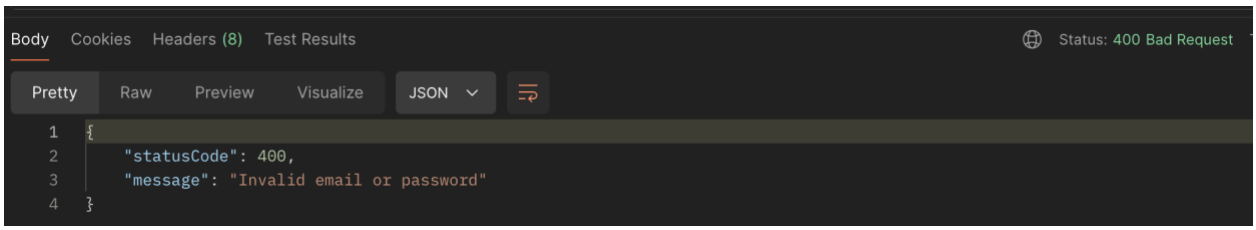

Рисунок 3.9 – Відповідь серверу про невдачу авторизації (статус «400»)

– оновлення токенів: виконання POST-запиту до ендпоінта api/auth/refresh-token з оновлюючими даними токенів (рис. 3.10). Перевірка відповіді на успішне оновлення та отримання нових токенів, при правильному токені статут запиту буде «201».

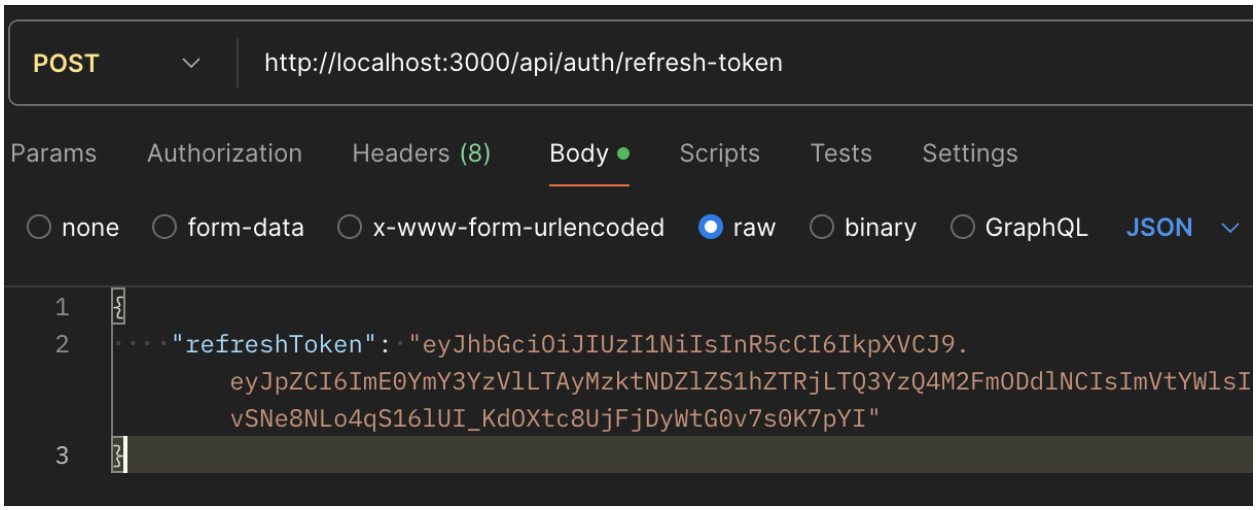

Рисунок 3.10 – Запит на оновлення токену

Тестування ендпоінтів може включати перевірку валідації вхідних даних, обробку помилок і відповідей у разі невдачі, перевірку забезпечення безпеки, такої як захист від SQL-ін’єкцій та перехоплення токенів.

3.3.2.2 Написання юніт-тестів

Для перевірки окремих компонентів програми, таких як функції, методи класів або окремі модулі, в ізоляції від інших частин системи будуть реалізовано юніт-тести. Вони дозволяють перевірити правильність роботи

окремих одиниць коду без необхідності запускати всю систему або її частину. Це допомагає виявити помилки та покращити стабільність програми. Юніт-тести зазвичай мокають (фальсифікують) залежності, такі як сервіси чи зовнішні інтеграції, щоб перевірити поведінку окремого компонента в ізоляції.

Приклад юніт-тестів буде розглянуто у лістингу 3.14.

Лістинг 3.14. Юніт-тести контролера автентифікації «auth.controller.spec.ts»:

```
describe('AuthController', () => {
  let controller: AuthController
  let authService: AuthService

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      controllers: [AuthController],
      providers: [
        {
          provide: AuthService,
          useValue: {
            register: jest.fn(),
            login: jest.fn(),
            getNewTokens: jest.fn(),
          },
        },
      ],
    }).compile()

    controller = module.get<AuthController>(AuthController)
    authService = module.get<AuthService>(AuthService)
  })
```

```
        it('should be defined', () => {
            expect(controller).toBeDefined()
        })

        describe('register', () => {
            it('should return a new authentication response including user and
tokens', async () => {
                const dto: RegisterDto = {
                    email: 'test@example.com',
                    password: 'password123',
                    name: 'Test User',
                    isPsychologist: false,
                }
                const user: User = {
                    id: '1',
                    email: dto.email,
                    name: dto.name,
                    password: dto.password,
                    psychologistId: null,
                    bio: '',
                    createdAt: new Date(),
                    updatedAt: new Date(),
                    websiteUrl: '',
                    image: '',
                    location: '',
                }
                const result: AuthResponse = { accessToken: 'someToken',
refreshToken: 'someRefreshToken', user: user }
```

```
            jest.spyOn(authService, 'register').mockImplementation(() =>
Promise.resolve(result))

            expect(await controller.register(dto)).toBe(result)
            expect(authService.register).toHaveBeenCalledWith(dto)
        })
    })

    describe('login', () => {
        it('should return an authentication response upon successful login',
async () => {
            const dto: LoginDto = {
                email: 'test@example.com',
                password: 'password123',
            }
            const user: User = {
                id: '1',
                email: dto.email,
                name: 'Test User',
                password: dto.password,
                psychologistId: null,
                bio: '',
                createdAt: new Date(),
                updatedAt: new Date(),
                websiteUrl: '',
                image: '',
                location: '',
            }
            const result: AuthResponse = { accessToken: 'accessToken',
refreshToken: 'refreshToken', user: user }
```

```
        jest.spyOn(authService, 'login').mockResolvedValue(result)

        expect(await controller.login(dto)).toBe(result)
        expect(authService.login).toHaveBeenCalledWith(dto)
      })
    })

    describe('getNewTokens', () => {
      it('should return refreshed tokens along with the user', async () => {
        const dto: RefreshTokenDto = { refreshToken: 'oldRefreshToken' }
        const user: User = {
          id: '1',
          email: 'user@example.com',
          name: 'Existing User',
          password: 'existingpassword',
          psychologistId: null,
          bio: '',
          createdAt: new Date(),
          updatedAt: new Date(),
          websiteUrl: '',
          image: '',
          location: '',
        }
        const result: AuthResponse = { accessToken: 'newAccessToken',
refreshToken: 'newRefreshToken', user: user }

        jest.spyOn(authService, 'getNewTokens').mockResolvedValue(result)

        expect(await controller.getNewTokens(dto)).toBe(result)
```

```
        expect(authService.getNewTokens).toHaveBeenCalledWith(dto)
      })
    })
  })
```

Юніт-тести для контролера AuthController перевіряють правильність його роботи шляхом створення фейкових об’єктів сервісів, які він використовує. Для цього «мокається» (фальсифікується) поведінка сервісів, визначаючи, які значення вони повинні повертати при виклику з контролера. Потім викликаються методи контролера, і перевіряється, чи вони поводяться відповідно до очікувань, співставляючи отримані результати з очікуваними значеннями, описуючи в кожному тесті, що ми повинні отримати. Це допомагає забезпечити, що контролер правильно взаємодіє зі своїми залежностями та обробляє їхні результати, забезпечуючи надійну роботу програми. Результати відпрацювання даних юніт-тестів зображено на рисунку 3.5.

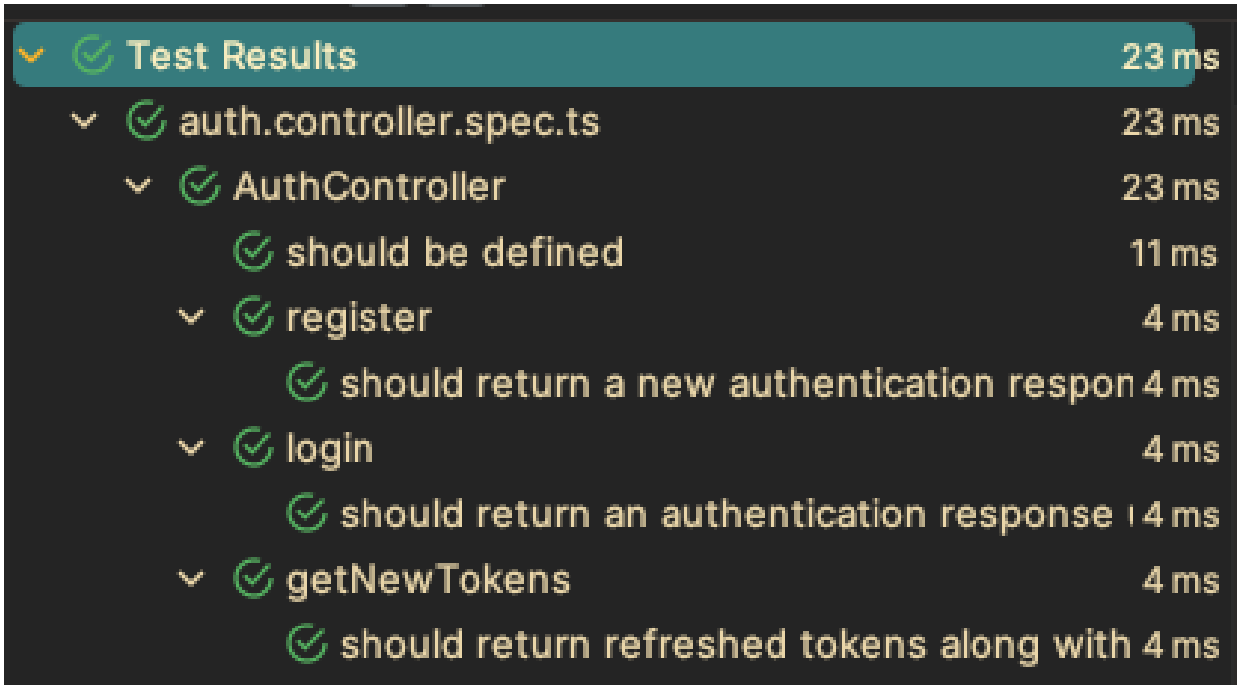

Рисунок 3.11 – Результати відпрацювання юніт-тестів модуля автентифікації, усі тести пройдено успішно

3.3.3 Реалізація клієнтської частини застосунку

Клієнтська частина вебзастосунку для психологічної допомоги розроблена на базі бібліотеки React, яка є вибором багатьох розробників завдяки своїй гнучкості та ефективності в створенні інтерактивних користувацьких інтерфейсів. React використовує декларативний підхід до побудови компонентів, що дозволяє розробникам швидко реагувати на зміни даних, автоматично оновлюючи UI, коли стан програми змінюється.

3.3.4 Організація коду та компонентна структура

В архітектурі клієнтської частини особливу увагу приділено модульності та повторному використанню коду. Застосунок розділений на декілька ключових модулів (наприклад, `auth`, `articles`, `tags`), кожен з яких містить компоненти, сервіси для роботи з API та власні стилі.

Структура модуля включає:

– компоненти відповідають за візуалізацію сторінок і UI елементів;

– API файли забезпечують зв’язок з сервером через асинхронні запити;

– стилі відповідають за оформлення компонентів;

– хуки дозволяють забезпечує реактивні можливості при роботі з функціональними компонентами, як управління станом компонентів, виконання побічних ефектів, використання контексту, мемоізації та інших реактивних функцій.

Компоненти розроблені таким чином, що кожен має чітко визначену функцію, це спрощує тестування та підтримку коду. Застосування CSS модулів або Styled Components забезпечує ізоляцію стилів, уникаючи конфліктів і проблем масштабування великих застосунків. А також зберігання медіа файлів, таких як іконок та інших у окремій директорії «public» сприяє безпеці застосунку.

3.3.5 Маршрутизація та взаємодія з сервером

React Router використовується для управління навігацією між різними сторінками застосунку. Маршрутизація організована таким чином, що вона підтримує як публічний доступ, так і приватний доступ до ресурсів, який вимагає аутентифікації. Також, для реалізації реактивної поведінки компонентів будуть використовуватись хуки React'у, такі як «useState», «useEffect», і «useContext», використовуються. Вони дозволяють компонентам реагувати на зміни даних в реальному часі, що забезпечує високу відповідність інтерфейсу вимогам користувачів.

Організацію маршрутизації к усім API серверу описано в лістингу 3.15.

Лістинг 3.15. Маршрутизація клієнтської частини:

```
export const Routes = () => {
   const location = useLocation()
   useEffect(() => {
      window.scrollTo({ top: 0 })
   }, [location])

   return (
      <AnimatePresence initial={false} mode="wait">
         <RouterRoutes>
            {/* Private routes */}
            <Route element={<PrivateRoutes />}>
               <Route path="/add-article" element={<AddArticle />} />
               <Route path="/edit-article/:id" element={<EditArticle />} />
               <Route path="/user/:id" element={<Profile />} />
               <Route path="/user/:id/following" element={<Follow />} />
               <Route path="/user/:id/followers" element={<Follow />} />
               <Route path="/user/edit-profile" element={<EditProfile />}>
```

```
                <Route index path="*" element={<ProfileSettings />} />
                <Route path="account" element={<AccountSettings />} />
              </Route>
              <Route path="/reading-list" element={<ReadingList />} />
            </Route>

            {/* Public routes */}
            <Route path="/" element={<Home />} />
            <Route path="/articles/:id/:slug" element={<Article />} />
            <Route path="/login" element={<Login />} />
            <Route path="/register" element={<Register />} />
            <Route path="/tags" element={<Tags />} />
            <Route path="/tag/:tagName" element={<Tag />} />
            <Route path="/search" element={<Search />} />
            <Route path="*" element={<NotFound />} />
          </RouterRoutes>
        </AnimatePresence>
      )
    }
```

Ця конфігурація дозволяє контролювати доступ до певних сторінок, а також забезпечує плавний перехід між різними розділами застосунку.

Взаємодія з сервером відбувається через API ендпоінти, організовані у відповідних модулях для кожної сутності. Ці ендпоінти використовують Redux Toolkit для керування станом застосунку, забезпечуючи централізоване управління даними, які надходять від сервера, вказуючи відповідні параметри для запитів та отримання цих даних, а також побудову самих запитів до серверу з їх методами та URL. Вони описані в директорії features, з відповідними піддиректоріями. Прилад API для авторизації файлу features/auth/authApi.ts (лістинг 3.16).

Лістинг 3.16. Опис API до авторизації серверу:

```
export const authApi = api.injectEndpoints({
  endpoints: (build) => ({
    register: build.mutation<AuthResponse, AuthData>({
      query: (body) => ({
        url: 'auth/register',
        method: 'POST',
        body,
      }),
    }),

    login: build.mutation<AuthResponse, Omit<AuthData, 'name'>>({
      query: (body) => ({
        url: 'auth/login',
        method: 'POST',
        body,
      }),
    }),
  }),
})

export const { useRegisterMutation,  useLoginMutation } = authApi
```

Отже, реалізація клієнтської частини на базі React із використанням передових практик маршрутизації, управління станом та інтерактивності забезпечує створення масштабованого, ефективного та інтуїтивно зрозумілого користувацького інтерфейсу.

Взаємодію базових компоненти клієнтської частини показано на наступній UML-діаграмі (рис. 3.5).

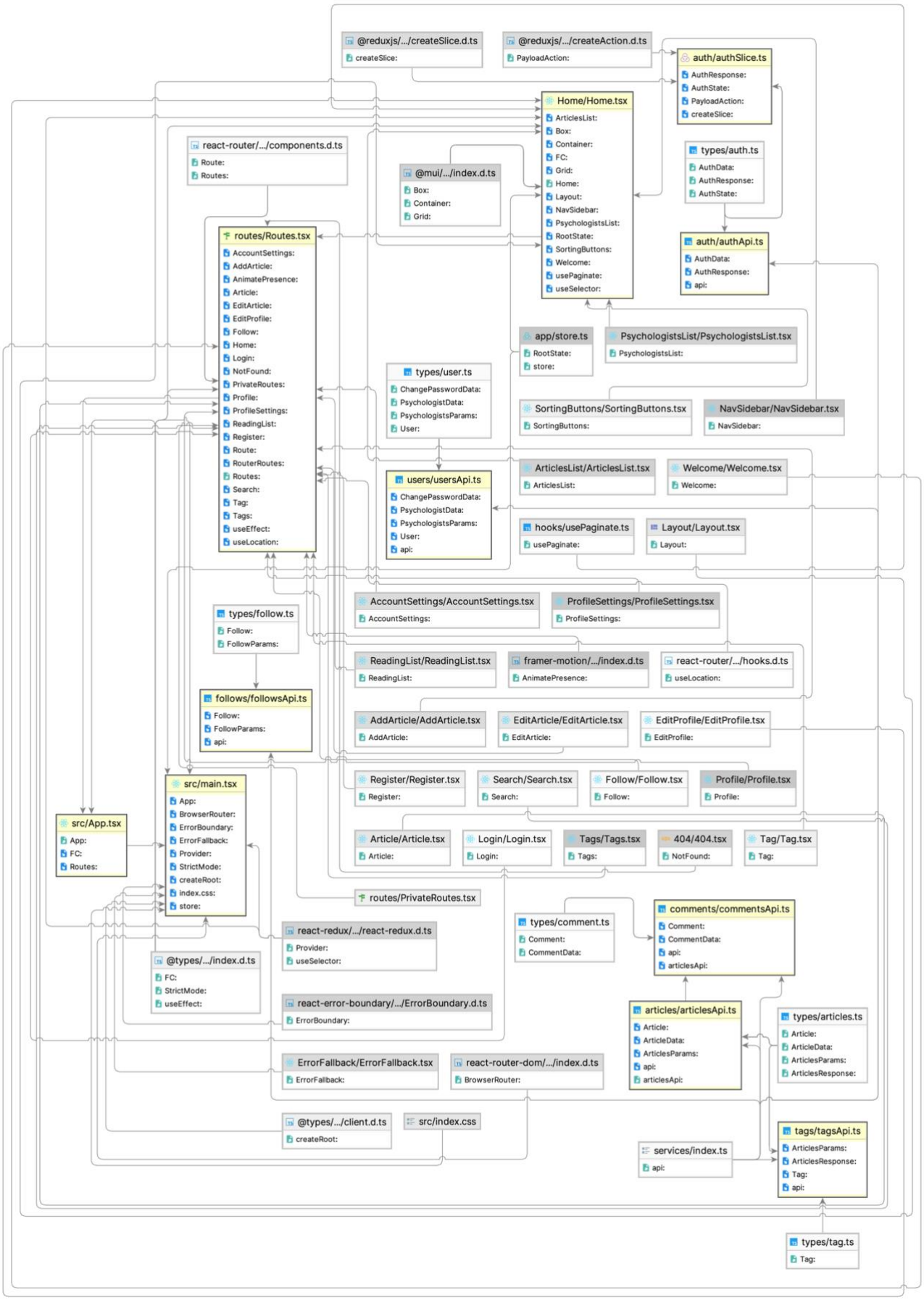

Рисунок 3.12 – UML-діаграма базових компонент клієнту (основні модулі, сторінки та керування API)

3.4 Перспективи розвитку

Розроблений застосунок вже має достатній функціонал для забезпечення повноцінного функціонування, попри те, ще залишається багато можливостей для його удосконалення та розширення. На сьогоднішній день платформа ефективно забезпечує базові потреби користувачів і спеціалістів, але подальший розвиток та інтеграція нових технологій та сервісів можуть значно підвищити якість та доступність послуг. Інновації, зокрема в областях штучного інтелекту та машинного навчання [22], відкривають нові перспективи для покращення діагностичних інструментів та персоналізації лікування, що може революціонізувати підходи до психологічної підтримки та розширити можливості професійного росту для психологів. Незважаючи на значні досягнення, перед розробниками все ще стоять завдання, пов’язані з оптимізацією користувацького інтерфейсу, забезпеченням безпеки даних та інтеграцією з іншими платформами та інструментами, що вимагає неперервних зусиль та інвестицій у розвиток проєкту.

Оскільки даний застосунок був реалізований самостійно та за обмежений проміжок часу, він несе в собі базовий функціонал, в реальних умовах на досягнення корпоративного рівня витрачаються роки та зазвичай потрібна повноцінна команда. Найближчим часом застосунок можна наповнити більш практичним функціоналом для роботи з психологами, впровадити повноцінний чат, зробити валідацію сертифікації психологів, а також зручні інтеграції з іншими сервісами. Також, за наявності часу та ресурсів, застосунок можливо вивести на більш професійний рівень, впровадивши покращені аналітичні засоби для психологів. Це можуть бути інструменти для аналізу ефективності консультацій та відстеження прогресу пацієнтів, що можуть допомогти спеціалістам краще планувати та адаптувати свої методики.

З більшим розвитком застосунку зросте й кількість користувачів та даних, а відповідно, необхідно буде забезпечити масштабування

інфраструктури, щоб підтримувати високу продуктивність та доступність застосунку:

– перехід на мікросервісну архітектуру: розподілення компонентів застосунку по різних мікросервісах для підвищення ефективності обробки запитів та поліпшення управління ресурсами;

– використання хмарних рішень: інтеграція з хмарними платформами для забезпечення гнучкості ресурсів та оптимізації витрат на підтримку інфраструктури;

– покращення систем безпеки: оновлення та впровадження новітніх стандартів безпеки для захисту персональних даних користувачів, зокрема використання сучасних методів шифрування та автентифікації.

Розвиток застосунку також включає інновації та дослідження нових можливостей застосування технологій в перспективному майбутньому. Завдяки вже впровадженій архітектурі й достатній базі застосунку можливі наступні інтеграції:

– інтеграція штучного інтелекту: використання AI для аналізу тексту консультацій[27], автоматичного виявлення настрою користувачів, та надання рекомендацій психологам щодо методів лікування;

– дослідження взаємодії користувача: аналіз поведінки користувачів у застосунку для покращення інтерфейсу та взаємодій на основі отриманих даних.

Ці стратегії та інновації дозволять не тільки підтримувати актуальність та конкурентоспроможність вебзастосунку, але й значно покращити якість надання психологічних послуг, роблячи їх більш доступними для широкого кола користувачів.

Розробка нових функціональних можливостей і технологій відкриває перспективи для покращення досвіду використання, розробки та ефективності взаємодії з платформою. Запровадження інноваційних підходів забезпече більш ефективне та індивідуалізоване обслуговування клієнтів у сфері психологічної підтримки.

3.5 Інструкція користувача

3.5.1 Налаштування програми

Для запуску програми буде потрібен інстанс Docker [28] Пілся цього завантажте код програми з GitHub репозиторію. Відкрийте термінал і виконайте наступну команду, щоб клонувати репозиторій (лістинг 3.17).

Лістинг 3.17. Консольна команда копіювання проєкту та перехід у відповідну директорію:

```
git clone https://github.com/SamVitalii/nest-psychological-assistance-app
cd nest-psychological-assistance-app
```

Також, альтернативний варіантом є завантаження файлів проєкту напряму (рис. 3.17)

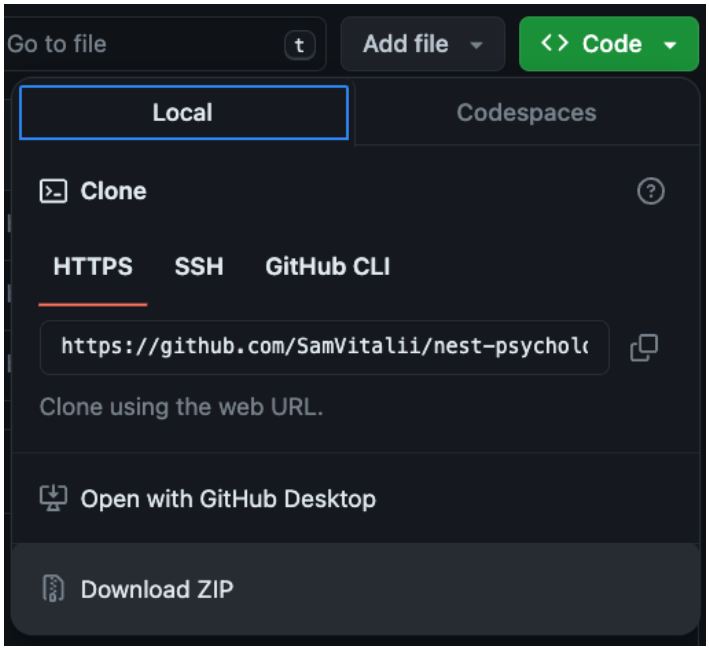

Рисунок 3.13 – Вигляд вікна завантаження файлів проєкту

Перейшовши в директорію проєкту, треба додати файли змінних середовищ.

Для серверу (додайте в корінь директорії server «.env» файл) він виглядатиме як показано у лістингу 3.18.

Лістинг 3.18. Файл середовища серверу «.env»:

```
PORT=3000
JWT_SECRET=secret

POSTGRES_USER=root
POSTGRES_PASSWORD=root
#DB_HOST=localhost
DB_HOST=db
DB_PORT=5432
POSTGRES_DB=postgres

DATABASE_URL=postgresql://${POSTGRES_USER}:${POSTGRES_PASSWORD}@${DB_HOST}:${DB_PORT}/${POSTGRES_DB}
PG_ADMIN_EMAIL=admin@admin.com
PG_ADMIN_PASSWORD=pgadmin4

CLOUD_NAME=YOUR_CLOUD_NAME
CLOUDINARY_API_KEY=YOUR_API_KEY
CLOUDINARY_API_SECRET=YOUR_API_SECRET
```

Ви можете залишити усі змінні у файлі, або налаштувати свої власні, попри те, вам потрібно вставити свої власні змінні для хмарного середовища збереження медіа файлів. Отримати їх можна на офіційному сайті сервісу [29], зареєструвавшись та відкривши меню «API Keys». Зауважте, що при локальному запуску проєкту (не через Docker) змінна хосту БД повинна бути встановлена на локальну.

Для клієнту додайте в корінь директорії client «.env» файл (лістинг 3.18).

Лістинг 3.19. Файл середовища клієнту «.env»:

VITE_BASE_URL=http://localhost:3000/api
PORT=8080

За потреби його змінні також можна скорегувати під власні потреби. Після цього ви зможете запустити всі компоненти програми, використовуючи docker-compose. Для цього ведіть наступну команду для побудови та запуску контейнерів (лістинг 3.20).

Лістинг 3.20. Консольна команда запуску контейнерів:

docker-compose up

Ця команда автоматично налаштує та запустить усі необхідні сервіси, визначені у файлі docker-compose.yaml, включаючи базу даних, міграції, сервер та клієнт застосунку.

3.5.2 Використання програми

Крок 1. Реєстрація та вхід

Після запуску програми перейдіть на головну сторінку вебзастосунку.

Використовуйте кнопку «Create Account» (рис. 3.15) для створення нового облікового запису, вказавши необхідні дані (ім’я користувача, електронну пошту, пароль та також вашу роль) у формі автентифікації (рис. 3.16). Або кнопку «Login» для авторизації, якщо вже маєте аккаунт користувача у відповідній формі (рис. 3.17).

Рисунок 3.15 – Кнопки для автентифікації користувача

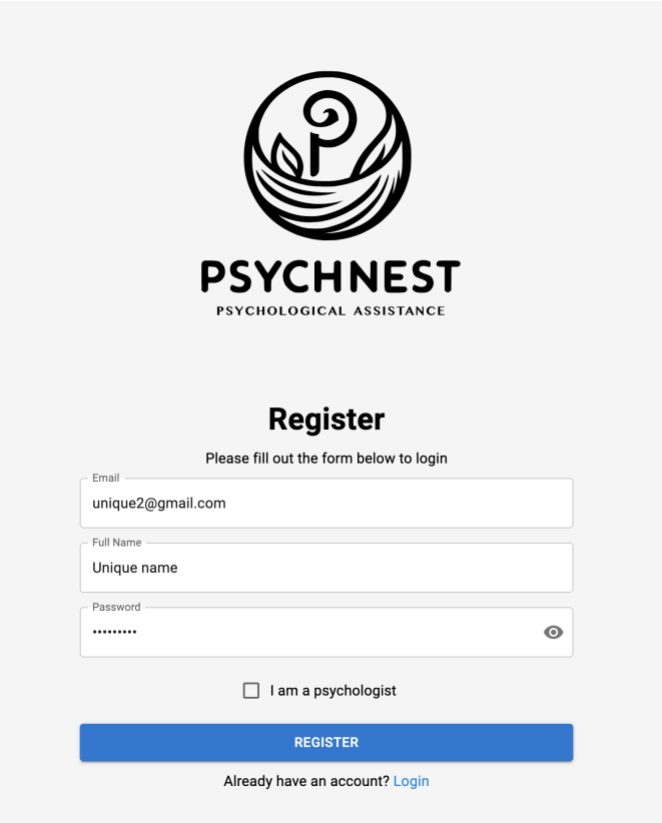

Рисунок 3.16 – Форма реєстрації користувача

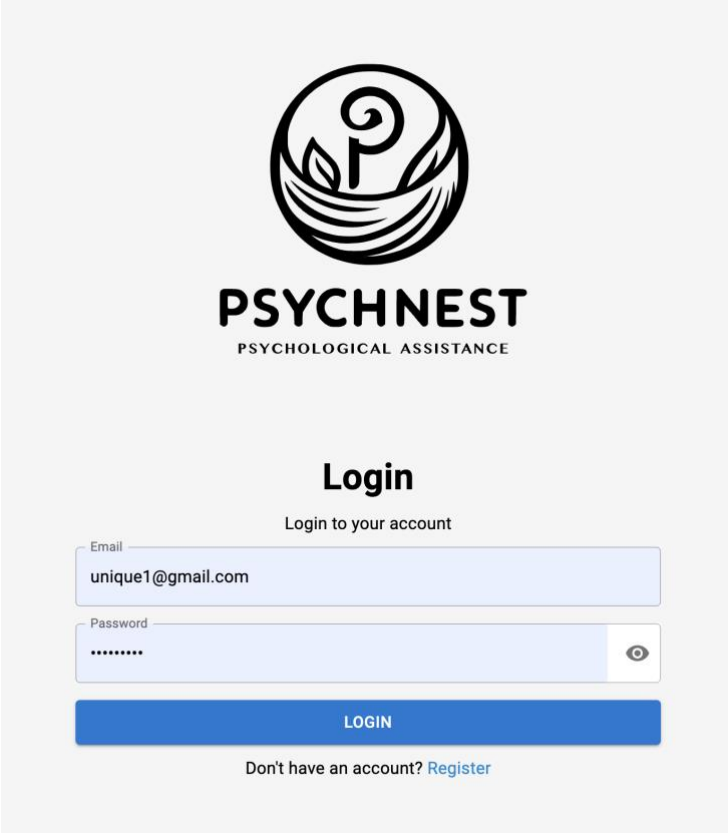

Рисунок 3.17 – Форма авторизації користувача

Після реєстрації або входу у систему, вас перекине на головну сторінку вебзастосунку. В залежності від типу вашого профілю в вас будуть відображатись статті, або список психологів. А також меню пошуку та фільтрації (рис. 3.18).

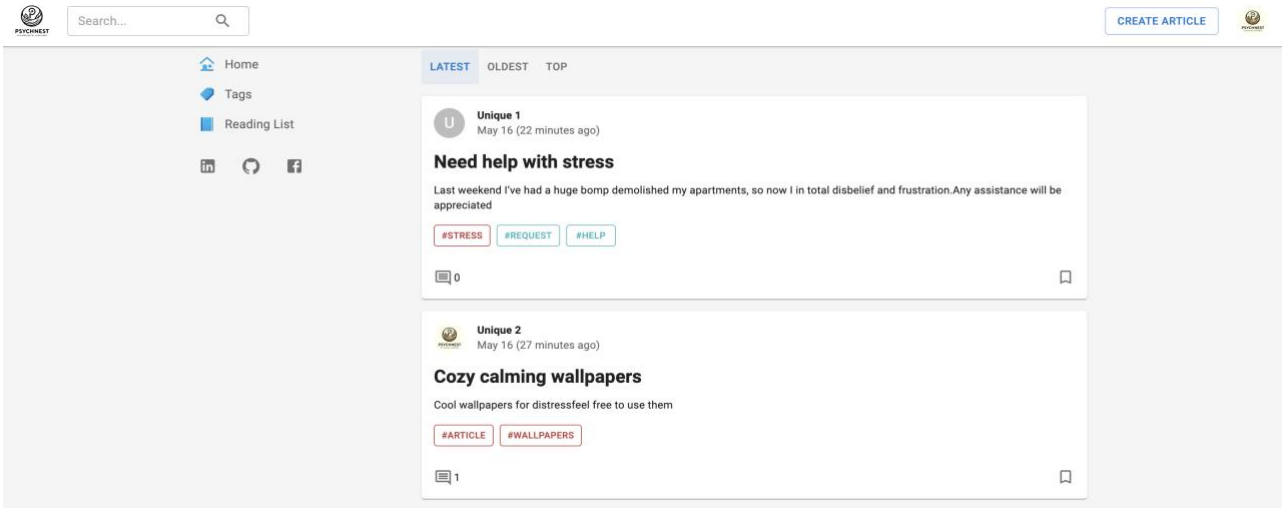

Рисунок 3.18 – Головна сторінка

Крок 2. Робота зі статтями

Використовуйте меню або пошукову стрічку для навігації по статтях або створення нових.

На сторінці біля профілю користувача (правий верхній кут) ви можете створити нову статтю (рис. 3.19), використовуючи кнопку «Create Arcticle».

Для редагування або видалення ваших статей перейдіть на сторінку самої статті, натиснувши на неї, та якщо ви є автором цієї статті, скористайтеся відповідними кнопками на сторінці перегляду статті «Edit Article», або для її видалення «Delete Article» (рис. 3.20).

Для того, щоб залишити коментар, незалежно від того, чи ви є автором статті, спуститься униз сторінки перегляду статті, та написавши свій коментар у відповідному полі «Add to the discussion», натисніть кнопку підтвердження відправки коментаря «Submit» (рис. 3.20).

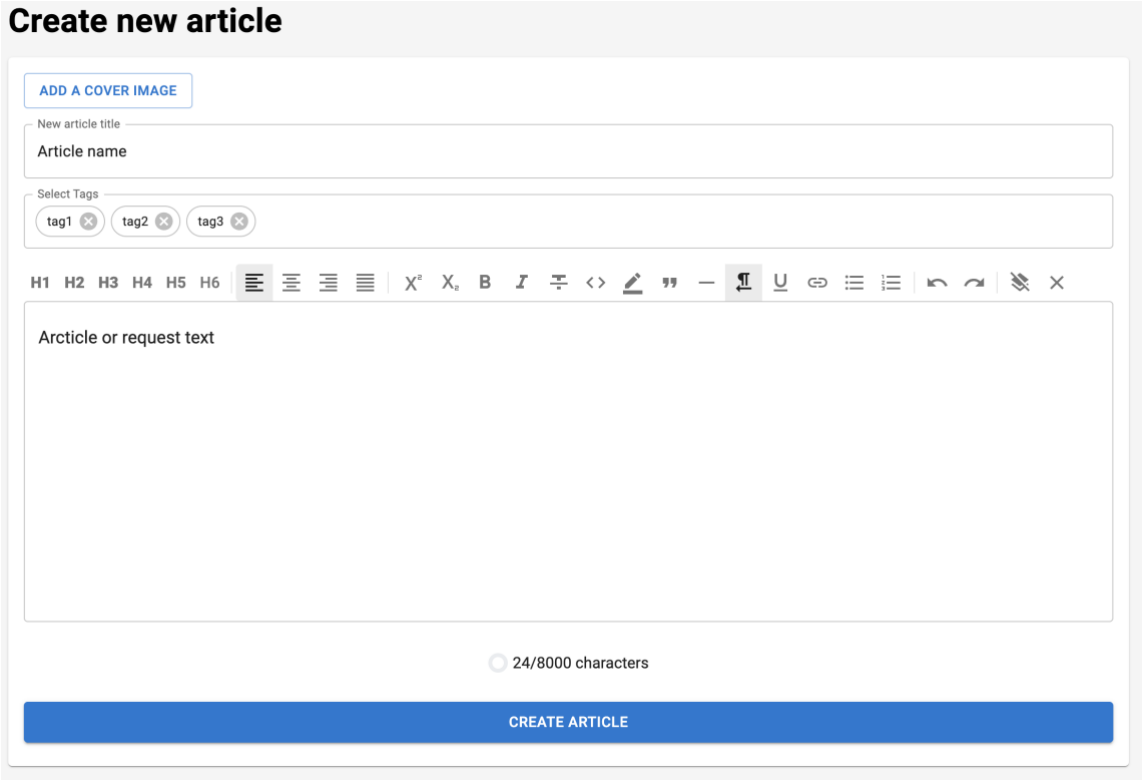

Рисунок 3.19 – Створення статті

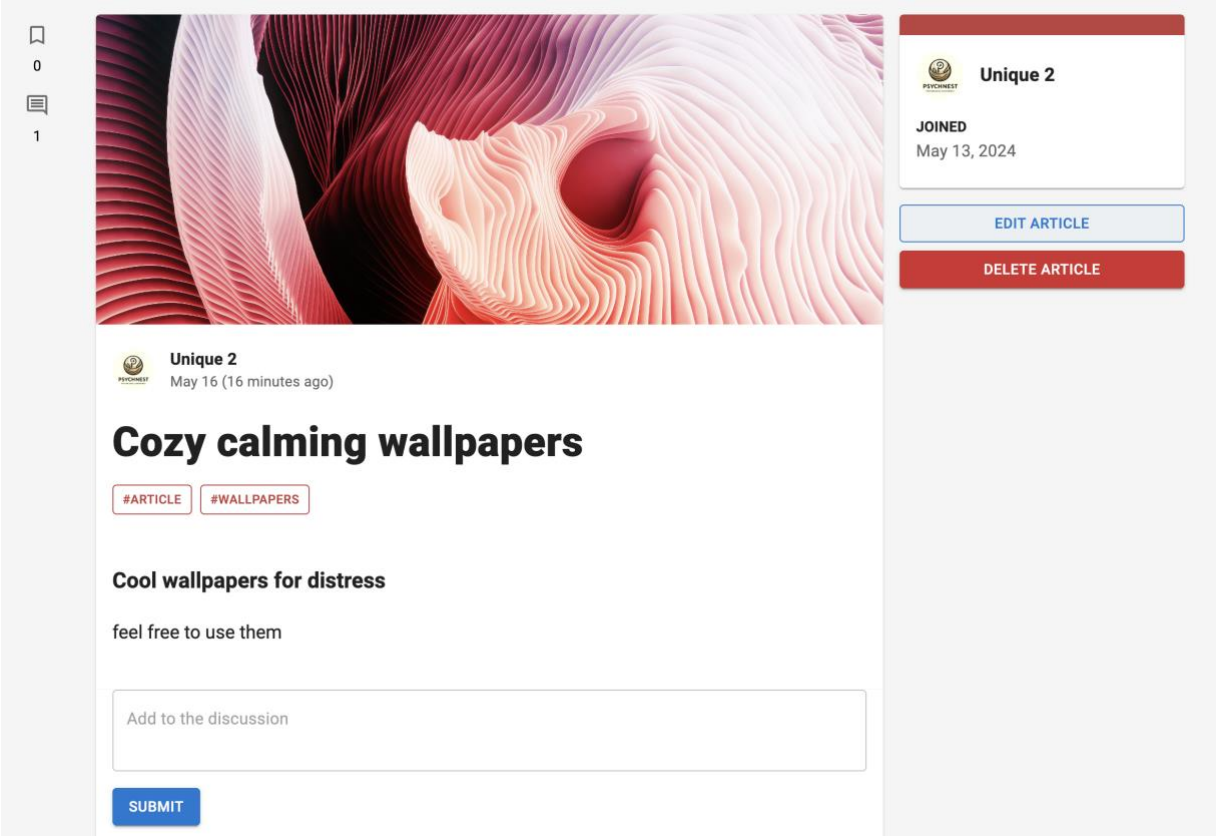

Рисунок 3.20 – Сторінка статті, редагування, коментування

Крок 3. Взаємодія з іншими користувачами

Ви можете переглядати профілі інших користувачів, підписуватися на них або відписуватися, відвідавши їхні профілі. Залишайте коментарі під статтями інших користувачів, щоб висловити вашу думку або поставити запитання.

Крок 4. Налаштування особистого профілю

На сторінці вашого профілю (рис. 3.21) ви можете змінити особисту інформацію, фото профілю (рис. 3.22) або пароль через вкладки налаштувань (рис. 3.23).

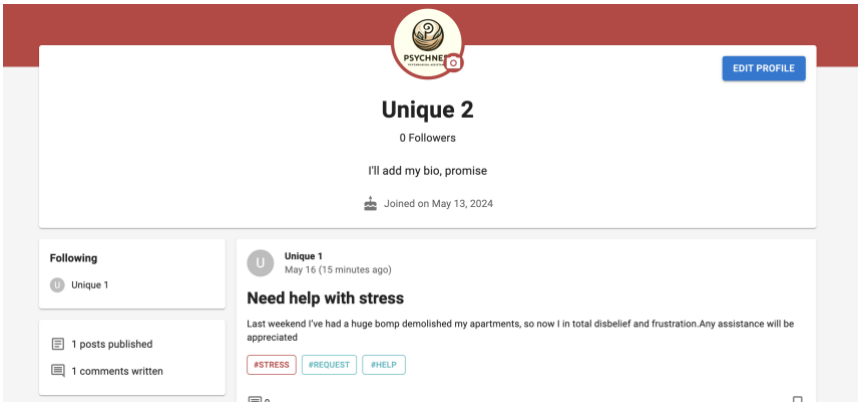

Рисунок 3.21 – Сторінка власного профілю

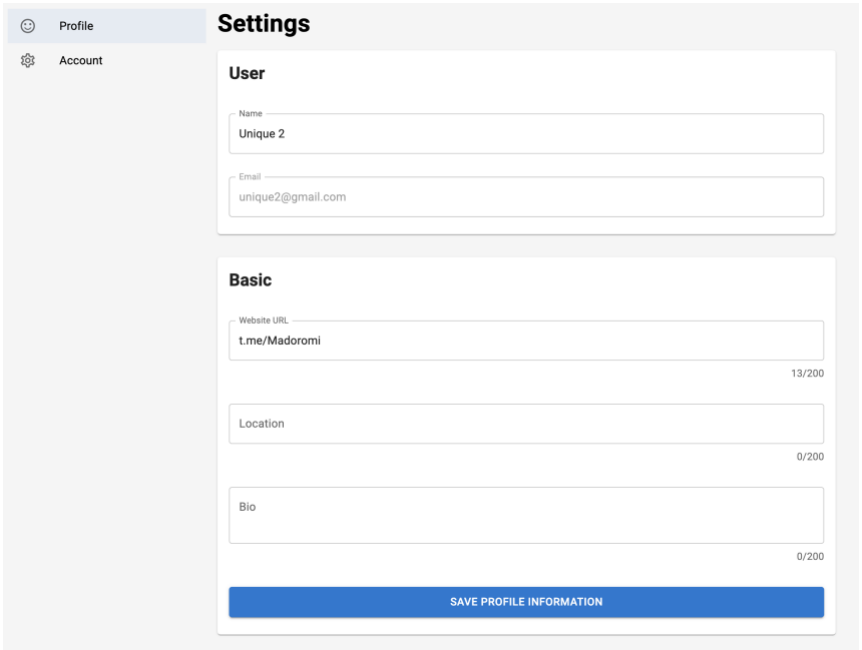

Рисунок 3.22 – Налаштування профілю

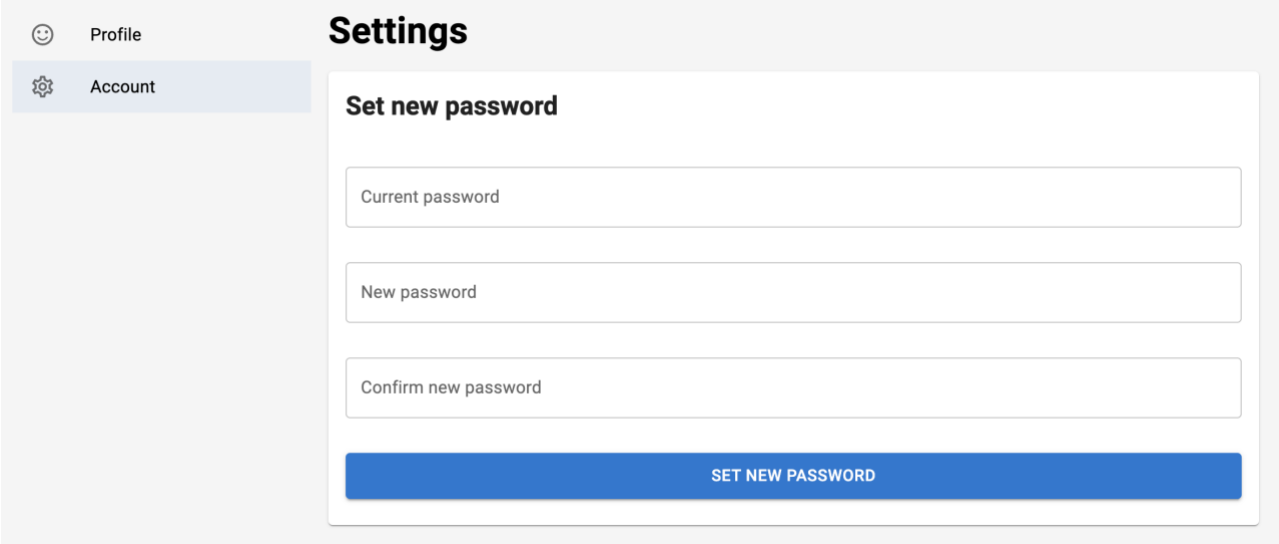

Рисунок 3.23 – Налаштування паролю

Окрім вже зазначеного функціоналу та пошуку з фільтрацією, користувач у головному меню (рис. 3.24) має доступ до збережених статей, а також тегів, де вже може шукати статі або запити за ними (рис. 3.25).

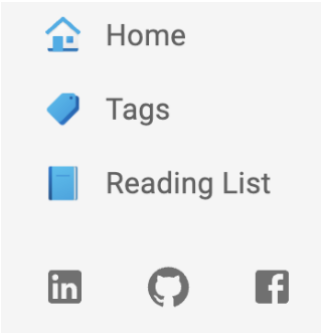

Рисунок 3.24 – Головне меню

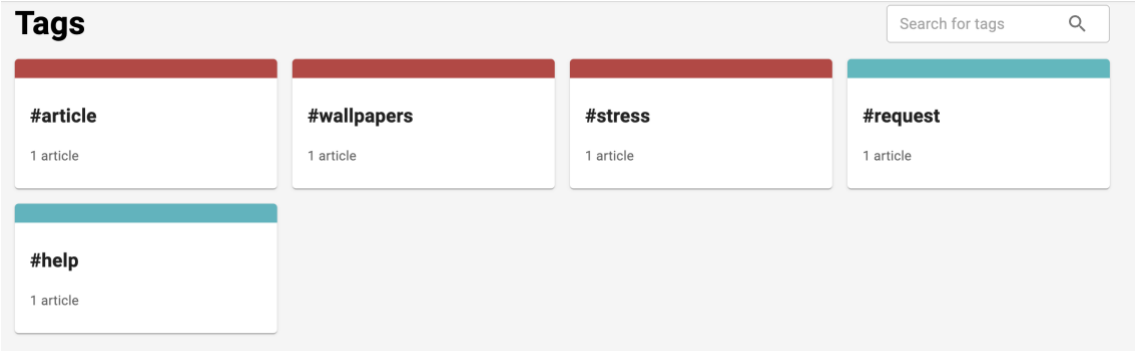

Рисунок 3.25 – Теги

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізований вебзастосунок для психологічної допомоги з використанням сучасних технологій. Застосунок було успішно спроєктовано, розроблено та тестовано, забезпечуючи стабільну та ефективну платформу для надання психологічних послуг користувачам.

Проєктування застосунку було ретельно сплановано з урахуванням потреб кінцевих користувачів, що дозволило створити інтуїтивно зрозумілий та доступний інтерфейс. Архітектура застосунку була розроблена таким чином, щоб забезпечити легке масштабування та інтеграцію нових функціональних можливостей, а також швидкість розробки.

Застосування сучасних технологій, таких як React.js, Node.js з NestJS, Prisma ORM, та Docker, забезпечило високий рівень продуктивності та безпеки застосунку. Розроблені модулі, а також методи взаємозв’язку та комунікації між клієнтською, серверною сторонами та рівнем даних відповідають сучасним вимогам до вебдодатків і надають ефективну взаємодію з користувачами. Усі відповідні ендпоінти API було протестовано, включаючи інструмент Postman. Це забезпечило ретельну перевірку всіх компонентів системи, а також підтвердило надійність та ефективність функціоналу застосунку.

Для можливості подальшого розвитку, а також впровадження можливості будь-кому скористатися застосунком, було надано детальні керівництва та документацію для користувачів та розробників, з відкриттям вихідного коду програми у репозиторії GitHub.

Проєкт має значний потенціал для подальшого розширення та інтеграції нових технологій, включаючи штучний інтелект для аналізу емоційного стану користувачів та інтеграції з медичними інформаційними системами.

Таким чином, мету роботи було досягнуто, результатом якої став розроблений якісний вебзастосунок, що демонструє важливість інтеграції

технологій у сферу охорони психічного здоров’я, відповідає актуальним потребам у сфері психологічної допомоги та надає потенціал для подальшого розвитку в цьому напрямку.

Результати роботи апробовано у вигляді тези доповіді під час XIX Міжнародної науково-практичної конференції «Modern trends are the driving force of scientific progress» [30].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Muthukrishna, M., Henrich, J., & Slingerland, E. (2021). Psychology as a historical science. *Annual Review of Psychology*, *72*, 717-749.

2. Bratman, G. N., Anderson, C. B., Berman, M. G., Cochran, B., De Vries, S., Flanders, J., ... & Daily, G. C. (2019). Nature and mental health: An ecosystem service perspective. *Science advances*, *5*(7), eaax0903.

3. Криштанович, С. В. (2022). Вступ, предмет, об'єкт, завдання та методи психології. У Педагогіка та психологія (Лекція 1). Львівський державний університет фізичної культури імені Івана Боберського.

4. Plakun, E. M. (2023). Psychodynamic therapy: an overview for trainees and their teachers: part 1–the basics. *Journal of Psychiatric Practice®*, *29*(2), 142-146.

5. Gautam, M., Tripathi, A., Deshmukh, D., & Gaur, M. (2020). Cognitive behavioral therapy for depression. *Indian journal of psychiatry*, *62*(Suppl 2), S223-S229.

6. Hernández-Peña, H., Lagomarsino-Montoya, M., Aguirre-Martínez, G., Mansilla-Sepúlveda, J., Estay-Sepúlveda, J. G., & Ganga-Contreras, F. (2022). On the Digital Age from Humanistic Psychology. *Environment and Social Psychology*, *7*(1).

7. Dallos, R., & Vetere, A. (2021). *Systemic therapy and attachment narratives: Applications in a range of clinical settings*. Routledge.

8. D'Alfonso, S. (2020). AI in mental health. *Current opinion in psychology*, *36*, 112-117.

9. Szinay, D., Jones, A., Chadborn, T., Brown, J., & Naughton, F. (2020). Influences on the uptake of and engagement with health and well-being smartphone apps: systematic review. *Journal of medical Internet research*, *22*(5), e17572.

10. Brecko, A., Pomšár, L., & Zolotová, I. Web Application for Personal Digital Health Records. *Acta Electrotechnica et Informatica*, *23*(3), 16-26.

11. Oberoi, O., Raj, S., Ongsakul, V., & Goyal, V. (2022). Benefits and Risks of Cloud Computing. In *Advancing Computational Intelligence Techniques for Security Systems Design* (pp. 105-124). CRC Press.

12. Тітов, С. В., & Тітова, О. В. (2013). Web-сайти органів місцевого самоврядування як складова впровадження е-урядування в Україні.

13. Ситніков, Д. Е., & Тітова, О. В. (2013). Аналіз вебсайтів органів місцевої влади як механізму забезпечення права доступу до публічної інформації. *Вісник Харківської державної академії культури*, (41), 134-142.

14. Stoll, J., Müller, J. A., & Trachsel, M. (2020). Ethical issues in online psychotherapy: A narrative review. *Frontiers in psychiatry*, *10*, 498439.

15. Snoswell, C. L., Chelberg, G., De Guzman, K. R., Haydon, H. H., Thomas, E. E., Caffery, L. J., & Smith, A. C. (2023). The clinical effectiveness of telehealth: a systematic review of meta-analyses from 2010 to 2019. *Journal of telemedicine and telecare*, *29*(9), 669-684.

16. Eysenbach, S., & Kazdin, A. E. (2017). Empirical trends in the treatment of anxiety and depression using cognitive-behavioral therapy delivered via the Internet. *Clinical Psychology: Science and Practice*, 24(2), pp. 177–191.

17. Hay, P., Bacalu, F., Bulik, C. M., Barrington, W., & Treasure, J. (2019). Internet-delivered interventions for eating disorders: A systematic review and meta-analysis. *Clinical Psychology Review*, 69, 101783, pp. 2-12.

18. Tvoroshenko, I., & Andrieieva, A. (2021). Development of web applications for remote learning of English.

19. Тітов, С. В., & Тітова, О. В. (2015). Оцінка юзабіліті освітніх сайтів: методи і технології. *Вісник Харківської державної академії культури. Серія: Соціальні комунікації*, (47), 127-134.

20. Cherednichenko, O., Yanholenko, O., & Iakovleva, O. (2013). *Web-Based monitoring and evaluation: research activity assessment case study* (Doctoral dissertation, EDIS Publishing Institution of the University of Zilina).

21. Тітов, С. В., & Тітова, О. В. (2014). Інформаційно-освітнє середовище навчального закладу: розвиток засобів і способів комунікаційної й інформаційної взаємодії. *Вісник Харківської державної академії культури*, (43), 144-150.

22. CoderOne. The Right way to write NestJS & Typescript clean-code - SOLID, 2023. *YouTube*. URL: https://www.youtube.com/watch?v=vE74gnv4VlY (дата звернення 02.05.2024).

23. Руденко Д.О., Колосок Е.В. (2021) Огляд можливостей Google Analytics для роботи з даними. 9 Міжнародна науково-практична конференція «Results of modern scientific research and development» (14-16 листопада, 2021) Barca Academy Publishing, Мадрид, Іспанія. 2021. С.162-165.

24. Руденко Д.О., Бондар В.О. (2020). Огляд можливостей використання стратегій об'єктно-орієнтованого маппінгу для зіставлення сутностей при розробці web додатків. Матеріали III Міжнародної науково-практичної конференції «Priority Directions of Science and Technology Development» Київ, Україна.с.377-380.

25. Database normalization description - Microsoft 365 Apps. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description (дата звернення 04.05.2024).

26. Танянський О., Руденко Д. Порівняльний аналіз популярних JavaScript-фреймворків та бібліотек для front-end розробки. Інформаційні системи та технології: матеріали міжнар. наук.-техн. конф., м. Харків, 10–15 вер. 2018 р. Харків, с. 347–349.

27. Руденко, Д., Лотвінова, В., & Безверха, Є. (2023). Навчання нейронної мережі в задачах обробки даних. *Collection of scientific papers «SCIENTIA»*, (September 22, 2023; Singapore, Singapore), 94-95.

28. Docker: Accelerated Container Application Development. *Docker*. URL: https://www.docker.com/ (дата звернення 13.04.2024).

29. Image and Video Upload, Storage, Optimization and CDN. *Cloudinary*. URL: https://console.cloudinary.com (дата звернення 20.04.2024).

30. Руденко Д.О., Самородов В. К. (2024). Вплив та використання сучасних методів розробки засобів для психологічної допомоги. Матеріали XIX Міжнародної науково-практичної конференції «Modern trends are the driving force of scientific progress» Лісабон, Португалія, с.101-103.