Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

(повна назва)

Кафедра Програмної інженерії

(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для підтримки психічного здоров'я

(тема)

Виконав:

студент 4 курсу, групи ПЗПІ-20-7

Гамалій К. В.

(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення

(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія

(повна назва освітньої програми)

Керівник ст. викл. каф. ПІ Зибіна К. В.

(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ З.В. Дудар

(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 – Інженерія програмного забезпечення

(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Гамалію Костянтину Володимировичу

(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для підтримки психічного здоров'я

затверджена наказом по університету від 20.05.2024р. № 471 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18.06.2024 р.

3. Вихідні дані до роботи електронні ресурси за обраною тематикою, мінімальні вимоги до функціональності програми, загальні вимоги до архітектури системи, оптимізаційні методи та моделі, СУБД MySQL, середовище розробки PHP Storm, технології Laravel, Vue 3, Broadcast, Pusher

4. Перелік питань, що потрібно опрацювати в роботі аналіз предметної області, постановка задачі, перелік вимог до програмної системи, опис прийнятих проектних рішень, опис програмної реалізації, тестування розробленого програмного забезпечення

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз проблемної галузі, вибір найбільш придатних аналогів	16.05.2024 – 19.05.2024	Виконано
2	Створення Специфікації ПЗ, затвердження Специфікації ПЗ керівником кваліфікаційної роботи	19.05.2024 – 21.05.2024	Виконано
3	Аналіз та моделювання предметної області. Проектування БД. Проектування архітектури програмної системи.	25.05.2024	Виконано
4	Створення коду	26.05.2024 – 01.06.2024	Виконано
5	Тестування та дослідна експлуатація ПЗ	02.06.2024	Виконано
6	Оформлення пояснювальної записки	30.05.2024 – 06.06.2024	Виконано
7	Перевірка пояснювальної записки керівником, підготовка до перевірки на антиплагіат	07.06.2024	Виконано
8	Оцінка роботи рецензентом, отримання відзиву, проходження нормоконтролю	08.06.2024	Виконано
9	Здача роботи у електронний архів, допуск роботи до захисту завідувачем кафедри	12.06.2024	Виконано
10	Захист кваліфікаційної роботи	18.06.2024	

Дата видачі завдання 16 травня 2024 р.

Студент _____

(підпис)

Керівник роботи _____ асист. каф. ПІ Зибіна К. В.

(підпис) (посада, прізвище, ініціали

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра: 87 стор., 36 рис., 6 джерел.

ЗДОРОВ’Я, LARAVEL, DOCKER, VUEJS, BROADCAST, PUSHER, MySQL, CSRF, ЛІКАРІ, ПСИХІАТРИ, СЕНСОРИ, ПОШТА, PDF, HTTPS, MULTISELECT, KEYBOARD.

Об'єктом розробки - програмна система, що забезпечує підтримку психічного здоров'я користувачів шляхом надання відстеження стану та організації консультувань.

Проєкт розробляється з використанням сучасних технологій веб-розробки, що забезпечують високий рівень функціональності, безпеки та масштабованості. Основою системи є бекенд, написаний на Laravel, що забезпечує швидку обробку запитів, управління базами даних та реалізацію бізнес-логіки. Для зберігання даних використовується MySQL або інша реляційна СКБД, інтегрована через Laravel.

Docker використовується для контейнеризації додатку, що дозволяє легко розгортати і масштабувати систему в різних середовищах, забезпечуючи стабільність та ізольованість процесів. Це також спрощує налаштування та управління залежностями додатку.

Фронтенд частина реалізована за допомогою Vue 3, що дозволяє створювати інтерактивний та динамічний користувацький інтерфейс. Vue 3, завдяки своїй реактивності та компонентній архітектурі, значно спрощує розробку складних веб-додатків та підвищує ефективність взаємодії користувача з системою.

Для реалізації функцій мессенджера, таких як реального часу обмін повідомленнями, використовується Laravel Broadcast в поєднанні з Pusher. Ця

технологія дозволяє надсилати дані в реальному часі між сервером і клієнтами, забезпечуючи миттєве оновлення повідомлень без необхідності оновлення сторінки.

В результаті розробки була створена система, яка дозволяє надавати такі основні функції, як спілкування з лікарями та психіатрами через інтегрований чат в режимі реального часу, що використовує Laravel Broadcast та Pusher для миттєвого обміну повідомленнями; слідкування за станом здоров'я, яке забезпечується через інтуїтивний інтерфейс, реалізований на Vue 3, де користувачі можуть реєструвати та аналізувати свої психічні та емоційні стани; та адміністративну панель, що дозволяє керувати користувачами, лікарями, запитами та іншими аспектами системи, зручно організовану і доступну через браузер завдяки Laravel і Docker.

Програмна система складається з сервера та веб-клієнту.

HEALTH, LARAVEL, DOCKER, VUEJS, BROADCAST, PUSHER, MySQL, CSRF, DOCTORS, PSYCHIATRICS, SENSORS, MAIL, PDF, HTTPS, MULTISELECT, KEYBOARD.

The object of development for my diploma project is a software system designed to support users' mental health by providing tools for self-help, health monitoring, and organizing consultations.

The project is developed using modern web development technologies that ensure a high level of functionality, security, and scalability. The core of the system is a backend built on Laravel, which facilitates fast request processing, database management, and business logic implementation. For data storage, MySQL or another relational database management system is used, integrated through Laravel.

Docker is employed for containerizing the application, allowing easy deployment and scaling across different environments, ensuring stability and process isolation. This also simplifies the setup and management of the application's dependencies.

The frontend part is implemented using Vue 3, which enables the creation of an interactive and dynamic user interface. Vue 3, with its reactivity and component-based

architecture, significantly simplifies the development of complex web applications and enhances user interaction with the system.

To implement the messaging functionality, such as real-time message exchange, Laravel Broadcast combined with Pusher is used. This technology allows real-time data transmission between the server and clients, providing instant message updates without the need for page refreshes.

As a result of the development, a system was created that offers key features such as communication with doctors and psychiatrists through an integrated real-time chat, utilizing Laravel Broadcast and Pusher for instant messaging; health status monitoring facilitated by an intuitive interface built on Vue 3, where users can log and analyze their mental and emotional states; and an administrative panel that allows for managing users, doctors, queries, and other aspects of the system, conveniently organized and accessible through a web browser thanks to Laravel and Docker.

The software system consists of a server and a web client.

Я, Гамалій Костянтин Володимирович, студент гр. ПЗПІ-20-7, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система для підтримки психічного здоров’я», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

В сучасному світі, насиченому стрімкими темпами розвитку технологій та несамовитими темпами життя, питання психічного здоров’я стає все більш актуальним і нагальним. Нерідко люди знаходять себе в стресових ситуаціях, стикаються з різноманітними психологічними викликами, які впливають на їхню загальну якість життя. Враховуючи це, розробка та впровадження програмних систем для підтримки психічного здоров’я стає невід’ємною складовою стратегії збереження та покращення здоров’я населення.

Сучасні технології відкривають безліч можливостей для створення інноваційних рішень у галузі психічного здоров'я. Інтеграція програмних засобів у цей контекст може не тільки полегшити доступ до необхідних ресурсів, але й створити унікальні інструменти для ефективної підтримки та розвитку психологічного благополуччя. Ця дипломна робота спрямована на розробку програмної системи, яка не лише враховуватиме специфіку психологічних потреб користувачів, але й пропонуватиме інтегрований підхід до забезпечення їхнього психічного здоров'я.

Розглядаючи питання психічного здоров'я з точки зору інформаційних технологій, ми віддаймо перевагу не лише розумінню психологічних аспектів, але й використанню передових методів аналізу даних, штучного інтелекту та персоналізованих підходів. Зосереджуючись на інноваційних рішеннях, ми маємо намір створити програмну систему, що допомагатиме користувачам вправно впоратися з стресом, адаптуватися до життєвих викликів та підтримувати психічне здоров'я на оптимальному рівні.

Результатом цієї роботи буде високоефективний інструмент для підтримки психічного здоров'я, який може стати цінним доповненням до існуючих

психотерапевтичних та консультативних практик, сприяючи сталому покращенню якості життя суспільства.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

1.1.1 Розвиток предметної галузі

Аналіз предметної галузі, пов'язаної із розвитком програмних систем для підтримки психічного здоров'я, відкриває перед нами величезні можливості та водночас виклики, пов'язані із сучасним станом ментального здоров'я та ростом технологій. В добу стрімкого темпу життя, зростаючого психологічного навантаження та широкого використання цифрових платформ, питання психічного здоров'я стає необхідністю, а відповідь на цю потребу - програмні системи, спрямовані на підтримку та покращення психічного благополуччя.

Інтеграція програмних рішень у цій області дозволяє нам переглядати психічне здоров'я з інноваційної перспективи. Сучасні технології дозволяють збирати та обробляти дані про емоційний стан, стресові реакції, а також вести моніторинг інших аспектів психічного здоров'я. Це відкриває шлях до створення персоналізованих підходів до кожного користувача, враховуючи його унікальні потреби та особливості.

Зростання популярності месенджерів та соціальних мереж створює можливість інтегрувати програмні рішення для підтримки психічного здоров'я безпосередньо в ці платформи. Можливість отримання допомоги, користуючись звичайними комунікаційними засобами, може значно полегшити доступність та ефективність таких послуг для широкого кола користувачів.

Враховуючи сучасні тренди, також важливо зосереджуватися на розробці інструментів для стрімкого виявлення ризиків та попередження психічних проблем. Використання методів аналізу даних, штучного інтелекту та машинного навчання може допомогти розробити систему, яка буде в змозі передбачати погіршення стану психічного здоров'я та реагувати на це заздалегідь.

Завданням нашої програмної системи буде не лише реагувати на симптоми, але й сприяти психологічній резилієнтності, навчаючи користувачів стратегіям саморегуляції та психічного самозахисту. Розробка такої системи віддзеркалить не лише різноманітність психічних потреб сучасного суспільства, але й наші зусилля в створенні ефективних інструментів для їхнього забезпечення та підтримки.

1.1.2 Аналіз існуючих рішень

Схожих проектів із використанням штучного інтелекту, телеграм боту або онлайн допомоги кваліфікованого лікаря не було знайдено, але існує декілька схожих за тематикою сайти.

На сайті «American psychiatric association» (APA) [1] (див. рис. 2.1) представлена інформація про програмні системи для підтримки психічного здоров'я (PHS). На сайті є розділ, присвячений цифровим технологіям та психічному здоров'ю, де представлені дослідження та ресурси для психіатрів, які використовують цифрові технології в своїй практиці.

У розділі "Цифрові технології та психічне здоров'я" наведені огляди різних досліджень, які вивчали ефективність PHS. Дослідження показали, що PHS можуть бути ефективними у лікуванні таких психічних розладів, як депресія, тривога та посттравматичний стресовий розлад.

У розділі також представлені ресурси для користувачів і розробників PHS. Для користувачів представлені огляди різних програм, а також рекомендації щодо вибору PHS. Для розробників представлені ресурси, які допоможуть розробити ефективні PHS.

На основі інформації, представленої на сайті APA, можна зробити висновок, що PHS є перспективним напрямком у галузі психічного здоров'я. PHS мають ряд

переваг перед традиційними методами лікування, такими як доступність, невисока вартість, можливість використання в будь-який час і в будь-якому місці. Однак, PHS все ще недостатньо досліджені, і їх ефективність залежить від багатьох факторів, таких як тип психічного розладу, індивідуальні особливості користувача та якість програми.

Рисунок 1.1– Сторінка відображення сайту «Американської психіатричної асоціаці»

Плюси PHS

- доступність;
- невисока вартість;
- можливість використання в будь-який час і в будь-якому місці;
- можуть бути адаптовані до індивідуальних потреб користувача.

Мінуси PHS

- не можуть замінити традиційні методи лікування;
- можуть бути не ефективними для всіх користувачів;
- недостатньо досліджені.

На сайті «Mental Health America» (MHA) [2] (див. рис. 2.2) представлена інформація про програмні системи для підтримки психічного здоров'я (PHS). На сайті є розділ, присвячений цифровим технологіям та психічному здоров'ю, де представлені огляди різних програм, а також ресурси для користувачів і розробників.

У розділі "Цифрові технології та психічне здоров'я" наведені огляди різних досліджень, які вивчали ефективність PHS. Дослідження показали, що PHS можуть бути ефективними у лікуванні таких психічних розладів, як депресія, тривога та посттравматичний стресовий розлад.

У розділі також представлені ресурси для користувачів і розробників PHS. Для користувачів представлені огляди різних програм, а також рекомендації щодо вибору PHS. Для розробників представлені ресурси, які допоможуть розробити ефективні PHS.

На основі інформації, представленої на сайті MHA, можна зробити висновок, що PHS є перспективним напрямком у галузі психічного здоров'я. PHS мають ряд переваг перед традиційними методами лікування, такими як доступність, невисока вартість, можливість використання в будь-який час і в будь-якому місці. Однак, PHS все ще недостатньо досліджені, і їх ефективність залежить від багатьох факторів, таких як тип психічного розладу, індивідуальні особливості користувача та якість програми.

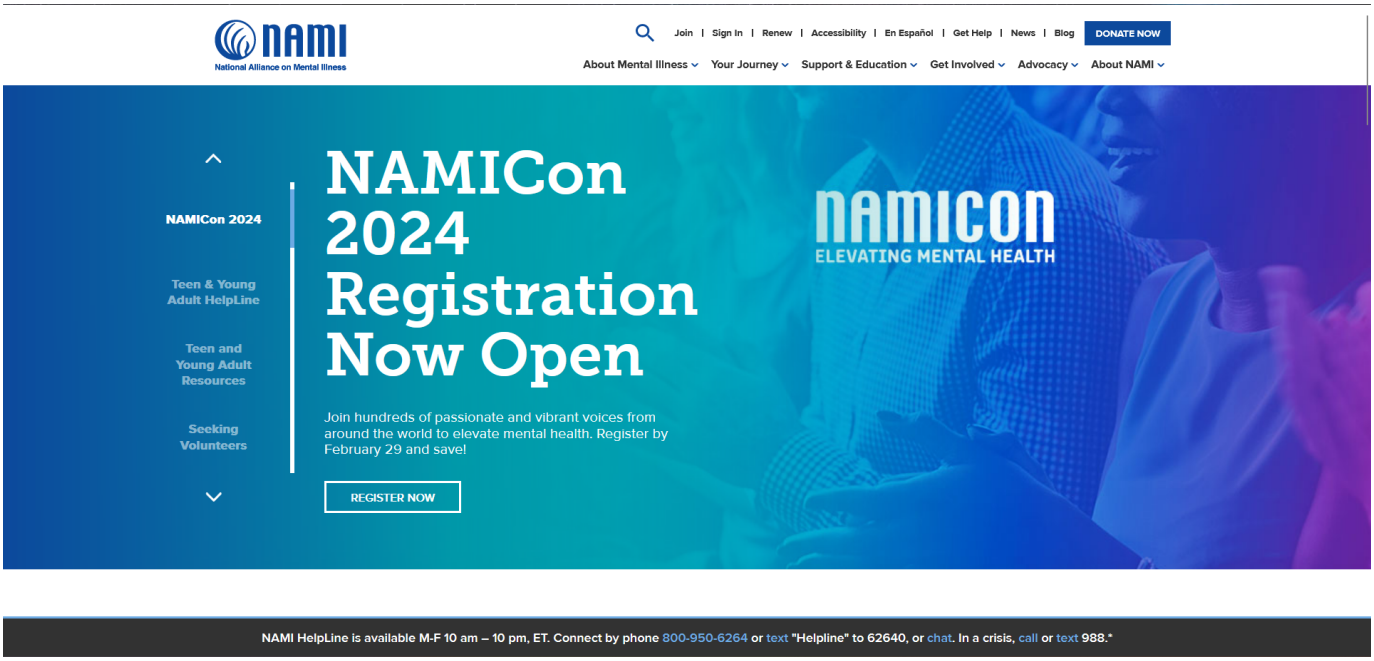


Рисунок 1.2 – Сторінка відображення сайту «Mental Health America»

Плюси PHS:

– доступність;
– невисока вартість;
– можливість використання в будь-який час і в будь-якому місці;
– можуть бути адаптовані до індивідуальних потреб користувача.

Мінуси PHS:

– не можуть замінити традиційні методи лікування;
– можуть бути не ефективними для всіх користувачів;
– недостатньо досліджені.

На сайті МНА представлені огляди різних PHS, які можна розділити на кілька категорій:

– інформаційно-освітні PHS. Ці програми надають користувачам інформацію про психічне здоров’я, симптоми психічних розладів та способи їх лікування;

– освітні PHS. Ці програми допомагають користувачам розвивати навички, необхідні для подолання психічних розладів, такі як навички управління стресом, навички самодопомоги та навички соціальної взаємодії;

– терапевтичні PHS. Ці програми використовують різні терапевтичні методи, такі як когнітивно-поведінкова терапія, щоб допомогти користувачам впоратися з психічними розладами;

– підтримуючі PHS. Ці програми надають користувачам підтримку та співтовариство, які можуть бути корисними для людей з психічними розладами.

1.1.3 Аналіз способів подальшої монетизації

Аналіз способів подальшої монетизації програмної системи для підтримки психічного здоров'я враховує потреби користувачів та забезпечує устойчивість проекту. Пропоную такий підхід до аналізу:

Опираючись на розгляд існуючих аналогів та враховуючи специфіку розроблюваної системи, основні методи монетизації включають в себе моделі "Підписка" та "Розміщення реклами". Основний функціонал системи буде доступний для всіх користувачів, забезпечуючи широкий охоплення аудиторії, проте частину значущих функцій слід зробити доступними лише під платну підписку.

У подальшому розвитку застосунку може варто розглядати впровадження декількох рівнів підписки, які надають різний функціонал. Наприклад, визначити різні рівні для ведення звичайного сімейного бюджету та спеціального - для бізнес-користувачів.

Додатково, можна розглядати можливість розміщення рекламних повідомлень через бота у приватних повідомленнях користувачів. Це можна зробити доступним

лише для платних підписників, забезпечуючи таким чином високий рівень контенту та уникату в рекламних повідомленнях.

Важливо враховувати, що рекламні повідомлення повинні бути тематично відповідними застосунку, наголошуючи на контролі витрат, фінансах та бізнесі, щоб не відвернути аудиторію. Такий гнучкий підхід до монетизації сприятиме забезпеченню стабільності та високої користі від розробленої програмної системи.

1.2 Виявлення проблем та актуалізація рішень

На основі аналізу предметної галузі та вивчення існуючих систем-аналогів, було проведено дослідження з метою визначення цільової аудиторії та ідентифікації потенційних проблем, які потребують вирішення для кінцевих користувачів.

Для більш точного визначення потреб користувачів та підвищення ефективності програмної системи, було проведено опитування серед групи потенційних користувачів. Цей підхід дозволив актуалізувати бачення майбутнього застосунка, враховуючи реальні потреби та очікування користувачів.

На основі отриманих даних були визначені ключові проблеми, що можуть виникнути в контексті підтримки психічного здоров'я. До цих проблем можуть відноситися низька інформованість користувачів, відсутність персоналізованих підходів та обмежений доступ до необхідних ресурсів.

З метою розв'язання цих проблем, планується розробка програмної системи, яка не лише враховуватиме специфіку психологічних потреб користувачів, але й пропонуватиме інтегрований підхід до забезпечення їхнього психічного здоров'я. Персоналізовані рекомендації, доступ до корисного контенту та інтерактивні інструменти мають на меті створити ефективну та зручну систему для підтримки психічного здоров'я у широкому колі користувачів.

1.2.1 Цільова аудиторія

Цільова аудиторія для розроблюваної програмної системи для підтримки психічного здоров'я орієнтована на широкий спектр користувачів з різним рівнем технічної освіти та потребами. Програмний застосунок буде спрямований на вирішення актуальних завдань щодо підтримки психічного здоров'я та забезпечення комфортного користування.

1. Користувачі, що ведуть контроль за своїм психічним здоров'ям:
   - опис: Особи, які розуміють важливість підтримки психічного здоров'я та шукають зручний інструмент для відстеження та покращення свого емоційного стану;
   - орієнтація: Додаток створений для тих, хто бажає активно вести контроль за своїм психічним здоров'ям та використовувати програму для його підтримки;
2. Користувачі месенджера (першочергова аудиторія):
   - опис: Люди, які активно користуються месенджерами та шукають засоби для підтримки свого психічного здоров'я в онлайн-середовищі;
   - орієнтація: Застосунок інтегрований в месенджер, спрощуючи використання та надаючи зручний інтерфейс для ведення контролю;
3. Користувачі поза месенджером (другочергова аудиторія):
   - опис: Люди, які шукають програмний продукт для підтримки психічного здоров'я за межами месенджера;
   - орієнтація*:* Простий та зрозумілий інтерфейс застосунка дозволить користувачам, які не користуються месенджерами, зручно використовувати його функціонал.

Цільова аудиторія має різноманітний спектр потреб та передбачає можливість швидкого та ефективного ведення контролю за психічним здоров'ям, щоб кожен

користувач міг легко забезпечити собі психологічний комфорт у повсякденному житті.

1.2.2 Аналіз бізнес-цілей

Аналіз бізнес-цілей для розроблюваної програмної системи для підтримки психічного здоров'я включає наступні ключові аспекти:

БЦ-1: Посилення ментального благополуччя користувачів. Реалізація функцій, спрямованих на активну підтримку та поліпшення психічного здоров'я користувачів, надаючи їм доступ до ресурсів та інструментів для саморегулювання емоційного стану.

БЦ-2: Автоматизація ведення денника емоцій та подій. Забезпечення можливості автоматичного введення та аналізу інформації про емоційний стан користувача, сприяючи автоматизації процесу ведення денника подій.

БЦ-3: Підтримка користувачів у стресових ситуаціях. Розробка інтерфейсу та функціоналу, спрямованих на виявлення та реагування на стресові ситуації, надаючи користувачам ефективні інструменти для подолання труднощів.

БЦ-4: Аналіз та надання рекомендацій для покращення психічного стану. Реалізація системи аналізу даних для надання персоналізованих рекомендацій та стратегій для поліпшення психічного здоров'я користувачів.

БЦ-5: Підвищення усвідомленості користувачів щодо їхнього емоційного стану. Розробка засобів для візуалізації та надання інформації про емоційний стан користувачів, щоб підвищити їхню самосвідомість та розуміння власних емоцій.

БЦ-6: Сприяння психологічній підтримці в онлайн-форматі. Забезпечення можливості отримання психологічної підтримки в онлайн-режимі через програму, спрощуючи доступ до кваліфікованої допомоги.

БЦ-7: Створення унікального середовища для підтримки спільноти користувачів. Розробка функціоналу для створення та участі в онлайн-спільноті, де користувачі можуть ділитися досвідом, отримувати поради та взаємну підтримку.

БЦ-8: Розвиток системи персоналізованих рекомендацій. Постійне вдосконалення системи аналізу даних для забезпечення точних та персоналізованих рекомендацій користувачам у сфері психічного здоров'я.

БЦ-9: Забезпечення конфіденційності та безпеки інформації. Вдосконалення системи безпеки для забезпечення конфіденційності особистих даних користувачів та створення довіри до системи.

1.2.3 Аналіз потреб користувачів

Аналіз потреб користувачів для програмної системи підтримки психічного здоров'я включає такі ключові вимоги:

ПК-1: Можливість ведення щоденника емоцій. Користувачі мають бажання легко та швидко реєструвати свої емоції та події, які впливають на їх психічне здоров'я.

ПК-2: Інтерактивність та можливість взаємодії з фахівцями. Користувачі хочуть мати можливість залишати коментарі, ставити запитання під публікаціями та спілкуватися з лікарями/психологами через приватні повідомлення.

ПК-3: Запис на консультацію. Користувачі мають потребу в інструменті для запису на онлайн-консультацію з фахівцем.

ПК-4: Зручність та простота використання. Користувачі хочуть мати інтуїтивно зрозумілий інтерфейс, оптимізований для різних пристроїв, з ефективним пошуком та зручною навігацією.

ПК-5: Персоналізація. Користувачі хочуть отримувати персоналізовані рекомендації та мати можливість збереження улюблених статей, відстеження історії консультацій та управління налаштуваннями профілю.

ПК-6: Конфіденційність та безпека. Користувачі мають потребу в захисті особистих даних, конфіденційності консультацій та надійних механізмах аутентифікації та авторизації.

ПК-7: Підтримка та допомога. Користувачі хочуть мати доступ до розділу з часто задаваними питаннями та можливість звернення до служби підтримки, а також навчальні матеріали щодо використання функціоналу сайту.

1.2.4 Аналіз припущень та залежностей

Аналіз припущень та залежностей для програмної системи підтримки психічного здоров'я включає наступні елементи:

П.1: Необхідність комплексної архітектури. Передбачається, що розробка системи вимагатиме наявності архітектурних рішень на різних рівнях інфраструктури, зокрема, робота з хмарними послугами та консультативними службами.

З.1: Залежність від хмарних надавачів та консультативних послуг. Використання хмарних послуг та консультативних служб буде ключовим для функціонування системи, і необхідно припускати належну доступність та підтримку від цих сервісів.

П.2: Можливість використання мессенджера. Передбачається, що користувачі будуть в змозі зручно взаємодіяти з системою через мессенджер для підтримки психічного здоров'я.

З.2: Розробка компонентів для розширення функціоналу на інші платформи. Розробка програмних компонентів повинна бути зорієнтована на можливість подальшого розширення функціоналу системи на інші платформи та середовища.

1.3 Постановка задачі

Мета даної кваліфікаційної роботи – розробка програмної системи, спрямованої на підтримку психічного здоров'я користувачів. Звертаючись до електронних ресурсів з обраною тематикою та враховуючи мінімальні вимоги до функціональності програми, розробляється комплексна система, що надає користувачам ефективні інструменти для моніторингу та управління своїм емоційним станом.

Обрана архітектура системи базується на багаторівневій структурі, що включає серверний рівень з централізованим API, клієнтську частину у вигляді Telegram бота та веб-сторінки для адміністрації. Використовуються ефективні інструменти розробки, зокрема середовище PHPStorm для серверної частини, Valentina Studio для роботи з СУБД MySQL та Vue 3 для клієнтської частини.

Важливою складовою є оптимізація системи. Застосовуються моделі та методи оптимізації для забезпечення швидкодії та ефективності використання ресурсів. Система використовує СУБД MySQL для забезпечення надійного зберігання та ефективного доступу до даних.

Для розробки використовуються такі технічні засоби та мови програмування: PHPStorm – середовище розробки для серверної частини, Valentina Studio – інструмент для роботи з базою даних MySQL, Vue 3 – фреймворк для реалізації клієнтської частини, Php – мова програмування для серверної частини, Telegraf – інструмент для оптимізації та моніторингу роботи Telegram бота.

Заплановані функції системи включають реєстрацію користувачів, можливість змінювати мову інтерфейсу, моніторинг та аналіз психічного стану, аналіз витрат в різних валютах, генерацію звітів та графічних даних, а також можливість ефективної комунікації через захищений канал у Telegram.

Після проведеного аналізу та врахування особливостей розробки програмної системи для підтримки психічного здоров’я, можна визначити наступні ключові функції та можливості системи:

Ф1. Реєстрація та управління користувачами.

Ф2. Мультиязиковість.

Ф3. Багатофункціональний моніторинг та пагінація даних.

Ф4. Оцінка емоційного стану.

Ф5. Психологічні поради та вправи.

Ф6. Повідомлення та нагадування.

Ф7. Система підтримки у кризових ситуаціях.

Ф8. Графіки та статистика.

Ф9. Інтеграція з мессенджерами.

Ф10. Захист приватності та конфіденційності.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Розроблювана програмна система повинна мати дві складові частини:

- серверний програмний застосунок, який надає своє API клієнтам;
- веб-додаток у вигляді бею сторінки.

Система повинна надавати можливість реєстрації користувачів з урахуванням їх особистої інформації та історії для подальшого використання системних функцій. Також вона має підтримувати локалізований двома мовами, зручний у використанні та інтуїтивно зрозумілий інтерфейс та надавати доступні функції її адміністрування. Користувачам слід мати можливість створювати та оновлювати свої профілі, а також вести щоденник психічного стану, включаючи відображення емоційного стану, побажань, та інших важливих аспектів.

Сервер системи повинен функціонувати як окремий, самостійний програмний продукт, що втілює архітектурний стиль REST [3] та має API-інтерфейси для зовнішніх користувачів (веб-клієнти). Доступ до ресурсів сервера має забезпечуватися через HTTPS у взаємодії з клієнтами. Важливо забезпечити захист системи, зокрема:

– реалізувати механізм аутентифікації для перевірки справжності учасників системи через автентифікаційні токени;
– наявність механізму авторизації для визначення прав доступу учасників та їх можливостей у системі;
– забезпечити захист персональних даних користувачів системи.

Для підтвердження прав користувачів слід використовувати концепцію CSRF токену [4], який містить закодовану інформацію про роль користувача та його ідентифікатор або email

Серверна частина має забезпечувати керування трьома рівнями доступу до системи: для звичайних користувачів системи, для лікарів або психологів та для

адміністраторів. Потрібно реалізувати сервіси, які забезпечують функціональні можливості обох рівнів доступу. Сервер повинен також мати можливість адміністрування системи, включаючи блокування та розблокування облікових записів, а також можливість створювати резервні копії всіх даних шляхом використання API для доступу до бази даних. Крім того, слід забезпечити локалізацію службових повідомлень, інтерфейсу та тексту у генерованих файлах статистичного або аналітичного змісту на українській та англійській мовах.

У серверній частині програмної системи для підтримки психічного здоров'я необхідно організувати логіку коду за допомогою розподілу на різні логічні шари:

– презентаційний шар контролерів: Він відповідає за обробку запитів від клієнтів, їх валідацію та логування на цьому рівні;

– шар сервісів, де зосереджена основна логіка API для виклику бізнес-процесів системи;

– шар даних, де знаходяться програмні репозиторії, які забезпечують доступ до даних у базі та управління самою базою даних.

Додатково до цього, на сервері мають бути налаштовані конфігураційні компоненти, що відповідають за різні аспекти системи:

– налаштування безпеки: забезпечення захисту системи від зовнішніх загроз та зловмисних атак;

– документація API: створення документації для API, яка допоможе розробникам розуміти функціональність системи;

– web взаємодія: Налаштування середовища для взаємодії з веб-клієнтами, забезпечення оптимальної швидкості та продуктивності;

– поштовий клієнт: налаштування електронної пошти для відправлення повідомлень користувачам системи;

– шаблони електронних сповіщень: створення шаблонів для різних видів сповіщень користувачам;

– логування службової інформації: запис і зберігання інформації про дії та події, що відбуваються в системі для належного аналізу та відладки;

– обробка виключень: налаштування механізмів для обробки та відловлювання винятків і помилок;

– валідація вхідних даних: перевірка правильності введених даних користувачами перед їхнім збереженням у системі

– робота з серверною файловою системою: налаштування доступу до файлів на сервері та їх управління

– доступ до бази даних та створення резервних копій: забезпечення доступу до бази даних для збереження та витягування інформації, а також автоматичне створення резервних копій для забезпечення безпеки даних

У системі підтримки психічного здоров’я, клієнтська частина (веб-клієнт) буде надсилати запити на сервер, який в свою чергу повинен перевіряти їх правильність, валідність авторизації, а також коректність переданих даних. Після цього сервер виконує обробку цих даних, виокремлює їх та передає на обробку до відповідних методів сервісного шару. Також він відповідає за обробку виключних ситуацій та генерацію відповідей у вигляді HTTP відповідей відповідно до загальноприйнятих стандартів.

Серверна частина системи повинна реалізувати сервісну логіку для різних режимів доступу, включаючи створення, редагування та видалення записів про витрати користувача та аналізу поточних витрат. Додатково, значною функціональністю є можливість автоматичної генерації звітів у форматі .pdf, які містять табличні дані про зміни у стані здоров'я користувача протягом місяця. Надіслання даних на електронну пошту для подальшого використання користувачем є також важливою можливістю.

Для передачі даних між рівнями системи використовується серіалізація та десеріалізація у форматі JSON. Важливою частиною є максимальне узагальнення

даних для можливості подальшого використання їх різними ботами у мессенджерах та мобільними застосунками.

Основний клієнтський інтерфейс програмної системи розгорнутий у Docker та окрему URL адресу та працює незалежно від інших частин програмної системи.

Клієнтський веб-додаток має надавати доступ до адаптивного користувацького інтерфейсу, взаємодіяти з сервером через REST API за допомогою протоколу HTTP. Додаток також повинен підтримувати можливість зміни мовного інтерфейсу на українську та англійську.

Основні функції клієнтського веб-додатку включають управління різноманітними аспектами взаємодії користувачів з платформою, забезпечення доступу до послуг та обміну інформацією. Однією з ключових функцій є можливість користувачів переглядати та взаємодіяти з послугами, що надаються лікарями та психіатрами. Користувачі можуть легко знайти необхідні послуги, скориставшись інтерфейсом для пошуку та сортування, а також зберігати обрані послуги у своєму профілі для швидкого доступу у майбутньому. Ця функціональність дозволяє користувачам ефективно керувати своїм вибором медичних послуг, зберігаючи час і зусилля на повторний пошук.

Крім цього, додаток підтримує функцію реального часу для обміну повідомленнями між користувачами та лікарями або психіатрами. Це забезпечує швидку та зручну комунікацію, яка є критичною для надання своєчасної підтримки та консультацій. Інтерфейс додатку дозволяє користувачам легко ініціювати чати з медичними працівниками, що значно підвищує якість обслуговування та задоволеність користувачів.

Для лікарів та психіатрів додаток надає інструменти для управління своїми послугами та комунікацією з пацієнтами. Вони можуть створювати та редагувати свої послуги, додавати описи та теги для кращої видимості у пошуку. Також вони мають доступ до чатів з пацієнтами, що дозволяє їм ефективно відповідати на запити та надавати консультації. Такий підхід забезпечує не тільки високу якість

обслуговування, але й полегшує роботу медичних працівників, оптимізуючи їхні процеси взаємодії з пацієнтами.

Адміністраторський профіль у системі буде представлений у веб-застосунку, розробленому з використанням Vue. Адміністратор матиме повний доступ до функціоналу та можливостей, реалізованих серверною частиною системи. Серед цих можливостей буде доступ до створення та завантаження файлів резервного копіювання даних системи.

На адміністраторській панелі будуть відображені профілі користувачів, які він зможе блокувати, розблоковувати, видаляти або редагувати за необхідності. Кожен профіль буде мати яскраву візуальну позначку статусу блокування для швидкого огляду.

Адміністратор також матиме доступ до форми налаштування записів, формування інформаційних блоків, дії із ролями користувачів, а саме схвалення ролі лікаря або психолога. Це дозволить йому змінювати параметри системи за потреби, редагувати або видаляти щось цензурне та керувати доступами.

Окрім цього, підкреслюється суттєвість сумісності системи з різними пристроями та операційними системами. Забезпечення коректної роботи на різних платформах та пристроях є важливим аспектом для забезпечення максимальної доступності користувачів. Такий підхід дозволяє системі стати більш доступною та зручною для використання, незалежно від обраного пристрою або операційної системи користувача. Незалежно від того, чи вони використовують комп’ютер, планшет або смартфон, або працюють під управлінням Windows, macOS, Linux або будь-якої мобільної операційної системи, вони можуть легко отримувати доступ до функціоналу системи та використовувати її для підтримки свого психічного здоров’я.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

Для спрощення проектування програмної системи були створені діаграми UML. Серед них варто виділити Use Case для системи загалом, Deployment для системи загалом та ER.

Діаграма варіантів використання (Use Case) для всієї системи описує способи взаємодії з нею. Вона надає зрозуміле зображення можливих сценаріїв використання та їх взаємозв'язків.

Рисунок 3.1 – Діаграма варіантів використання програмної системи

На зазначеній діаграмі відображено трьох акторів програмної системи: користувача, лікаря/психолога та адміністратора. Кожен з них має свої власні

варіанти використання системи в цілому, відповідні їхнім рівням доступу та функціональним потребам.

Користувач має змогу додавати або скасовувати зустрічі із лікарями або психіатрами, має змогу під’єднати датчики для швидкого інформування об наявних проблемах психічного здоров’я та інформаційного відображення статистики у вигляді графіків. Реєстрація користувача та можливість редагування профілю, що надає збереження датчику, налаштування мови, його данних у вигляді електронної скриньки, ФІО та фотографії. Окрім цього, користувач може отримувати інформаційні повідомлення про стан його здоров’я або заплонованих зустрічей.

Врач обо психолог у свою чергу мають змогу створювати інформаційні блоки та бути інформованими системою якщо якийсь із користувачів захотів записатися або щось дізнатися. Також для них ця система виступає як початковою практикою у здобуванні навиків своєї професії шляхом отримання коментарів проведення консультацій із користувачами, що також покращить іх рейтинг для працевлаштування.

Адміністратор існує для того, щоб перевіряти глобальну роботу програмної системи, робити резервну копію баз даних та відновлюватися з нею, має доступ до кожного запису даних та кожного профілю, слідкує за реєстраціями лікарів, психологів та скарг.

Проектування програмної системи почалося з UML-моделювання окремих рівнів системи після створення загальної діаграми варіантів використання. Перший етап включав проектування веб-частини системи, яка відокремлена від серверного програмного забезпечення. У цьому контексті було побудовано діаграму станів, яка ілюструє зміну станів об'єктів у часі. На цій діаграмі представлені стани та переходи користувача під час отримання аналітичних звітів з системи. Користувачі переходять між станами, використовуючи вбудовані клавіатури, щоб поступово уточнювати свої наміри, аналогічно переходам між екранами у звичайних додатках.

Користувач може обрати отримання щомісячних даних графіків зі станом

здоров’я на відправку до електронної пошти перейшовши до самих графіків та натиснувши на відповідну кнопку відправки.

Також слід зазначити, що користувач може у будь який час вийти із будь якого стану шляхом натискання на інші вкладки або кнопки повернутися назад у браузері.

Програмна система, що розробляється, має дві складові частини:

– сервер;

– веб-частина;

На рисунку 4.2 зображено діаграму розгортання програмної системи, що ілюструє загальну архітектуру системи та шари кожного з рівнів і їх взаємодію.

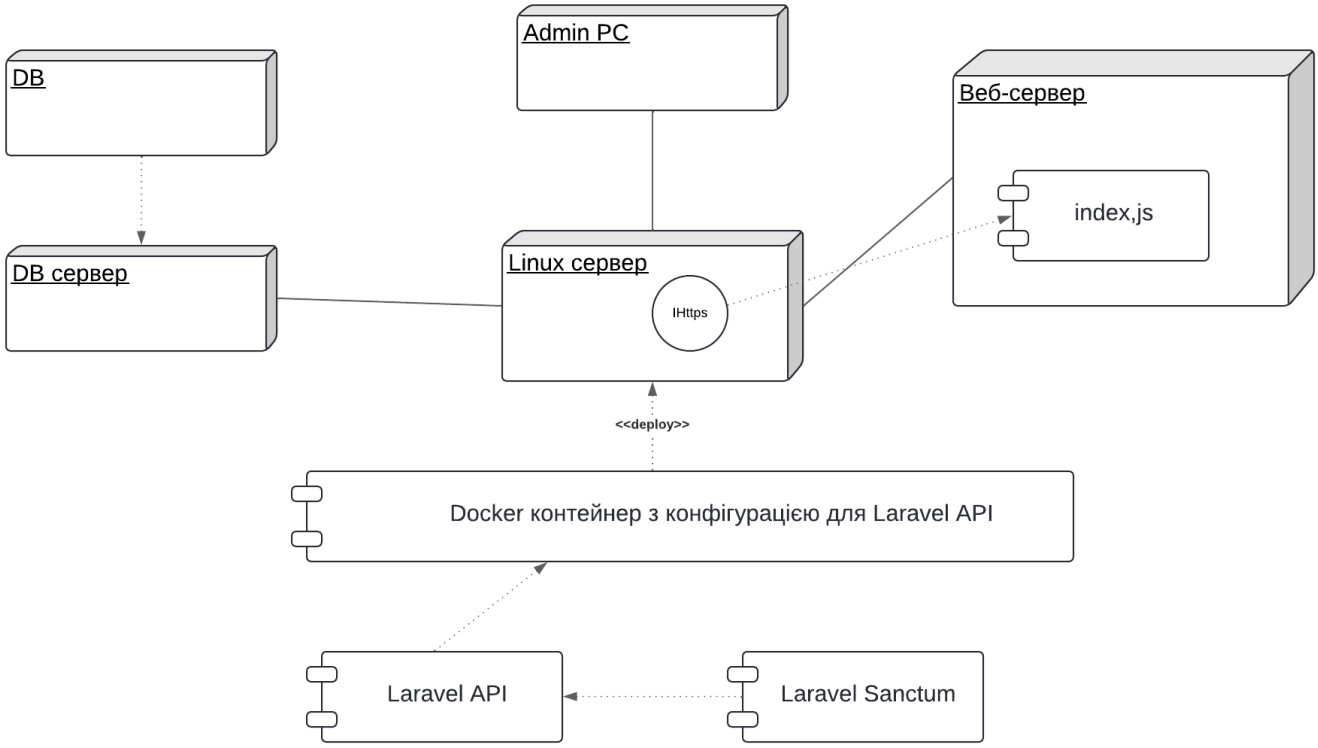


Рисунок 3.2 – Діаграма розгортання та загальна архітектура системи

Серверна частина програмної системи реалізована з використанням фреймворку Laravel, який забезпечує швидку і ефективну розробку веб-додатків. Для зберігання даних використовується база даних MySQL[5], яка інтегрована в

контейнер Docker. Це забезпечує легку розгортку та керування середовищем бази даних, а також забезпечує консистентність середовища між різними розробниками та на різних етапах розробки проекту.

Клієнтська частина системи реалізована з використанням фреймворку Vue 3, який надає потужні засоби для створення інтерактивних та відзивчивих користувацьких інтерфейсів. Використання Vue 3 дозволяє створювати складні клієнтські додатки зі зручною структурою та ефективним управлінням станом додатку.

Зв'язок між компонентами моєї системи здійснюється за допомогою протоколу HTTPS для забезпечення безпеки та конфіденційності даних. Для взаємодії між сервером та базою даних використовується інтерфейс IMySQLConnector. Зазвичай сервер бази даних є окремою машиною від серверного додатку програмної системи. Однак спочатку їх можна з'єднати та тримати базу даних локально. Це сприятиме збільшенню швидкості читання та редагування даних, а також забезпечить їхню безпеку, оскільки ніщо не буде передаватися через мережу Інтернет. Проте такий підхід має й свої недоліки, зокрема, якщо відбудеться поломка на основному сервері, постраждають і дані. Крім того, можуть виникнути проблеми при масштабуванні системи, оскільки базі даних може не вистачати вільного місця на дисках сервера. Отже, найоптимальнішою стратегією буде спочатку тримати локальну базу даних, а потім перейти до використання хостингових сервісів для таких серверів. Це дозволить забезпечити надійність та доступність даних, а також покращить масштабованість системи.

Користувачам надається можливість пройти аутентифікацію та авторизацію для отримання доступу до даних компанії та списку її маршрутів. Звичайні користувачі проймають стандартну аутентифікацію, вводячи свій логін та пароль. Після цього зв'язок підтримується за допомогою Authentication Token.

Лікарі та психологи мають свою власну окрему аутентифікацію, яка відправляється на розгляд адміністрації для подальшого входу на сайт. Після

підтвердження створення аккаунту адміністрацією вони також вводять свій логін та пароль, а зв'язок підтримується Authentication Token.

Компонент бізнес логіки реалізовано у наступних аспектах програмної системи:

Коли користувач обирає генерацію звіту з оцінкою категорії за певний місяць, сервер сканує дані користувача за цей період отримуючи дані через протокол. Застосовуючи відповідні коефіцієнти, сервер визначає рівень психічного здоров'я в різних категоріях. Після цього клієнтська частина генерує зображення з рендером графіка. Зображення надсилається користувачеві для подальшого аналізу свого психічного стану.

Веб-клієнт використовується для взаємодії з користувачем та надання доступу до функціоналу системи. Він має кілька рівнів, починаючи з презентаційного, який відповідає за відображення інформації та забезпечення зручного інтерфейсу для користувача. Для цього використовується фреймворк Vue, який дозволяє створювати динамічні та інтерактивні компоненти. Для забезпечення гнучкості та ефективності веб-клієнта використовуються допоміжні компоненти, які дотримуються принципів SOLID. Ці компоненти включають в себе утилітарні класи для відображення даних, форматування інформації, а також забезпечення локалізації застосунку. Вони спрощують розробку, підтримку та розширення функціоналу веб-клієнта, забезпечуючи високий рівень якості та стабільності програмної системи.

Щоб забезпечити ефективну комунікацію між компонентами програмної системи, більшість запитів використовують заголовок "Content-Type": "application/json". Це означає, що дані передаються у форматі JSON у тілі запитів методу POST. Цей підхід спрощує обробку та обмін даними між компонентами системи, забезпечуючи стандартизацію та зручність у взаємодії.

Механізм підтримки різних мов на веб-сайті вбудований за допомогою допоміжної бібліотеки i18n. Для кожної мови створюється окремий файл, який реєструється у класі i18n, підключеному у файлі index.js. Все, що потрібно зробити,

- це надати переклад для кожного поля в об'єкті, який спілкується з кодом Vue. Цей підхід суттєво спрощує процес локалізації системи на різні мови та дозволяє швидко виправляти помилки прямо під час роботи.

Після аналізу концептуальної моделі предметної області були створені наступні сутності та зв’язки між ними:

– сутність «User(звичайний користувач) з наступними атрибутами:» ID користувача (унікальний ідентифікатор), ID ролі (зовнішній ключ), ФІО, день народження, електронна пошта, зображення профілю, пароль, токен аутентифікації, ID сенсора пульсу (зовнішній ключ), ID шагометру (зовнішній ключ);

– сутність «Doctor(лікар або психолог)» з наступними атрибутами: ID лікаря або психолога (унікальний ідентифікатор), ID ролі (зовнішній ключ), ФІО, день народження, електронна пошта, пароль, зображення профілю, токен аутентифікації

– сутність «Document(документи сутності Doctor)» з наступними атрибутами: ID документа (унікальний ідентифікатор), ID лікаря або психолога(зовнішній ключ), ID типу документа (зовнішній ключ), файл, опис файлу;

– сутність «DocumentType(типи документів сутності Document)» з наступними атрибутами: ID типу документа (унікальний ідентифікатор), назва типу;

– сутність «Role(ролі користувачів, лікарів або психологів)» з наступними атрибутами: ID ролі (унікальний ідентифікатор), назва ролі;

– сутність «Service(послуги створені лікарями або психологами)» з наступними атрибутами: ID послуги (унікальний ідентифікатор), ID лікаря або психолога (зовнішній ключ), назва послуги, її опис, дата публікації, визначення репартування послуги;

– сутність «Tag(тег послуги)» з наступними атрибутами: ID тегу (унікальний ідентифікатор), назва тегу;

– сутність «ServiceTag(вспоміжна таблиця зв’язків таблиці послуг та тегів)» з наступними атрибутами: ID послуги (зовнішній ключ), ID тегу (зовнішній ключ);

– сутність «Chat(чат для спілкування в послузі)» з наступними атрибутами: ID чату (унікальний ідентифікатор), ID послуги (зовнішній ключ), ID лікаря або психолога (зовнішній ключ), ID користувача (зовнішній ключ), текст повідомлення, дата написання повідомлення, назва ролі відправника;

– сутність «SavedService(таблиця збережених послуг користувача)» з наступними атрибутами: ID збереженої послуги (унікальний ідентифікатор), ID користувача (зовнішній ключ), ID послуги (зовнішній ключ);

– сутність «HeartRateSenso(дані сенсору пульсу)» з наступними атрибутами: ID запису з сенсору (унікальний ідентифікатор), ID користувача (зовнішній ключ), значення сенсору;

– сутність «SensorPedometer(дані сенсору шагометру)» з наступними атрибутами: ID запису з сенсору (унікальний ідентифікатор), ID користувача (зовнішній ключ), значення сенсору;

На основі усього зазначеного вище була побудована ER-діаграма (схема бази даних) для програмної системи. Дана схема зображена на рисунку 4.3.

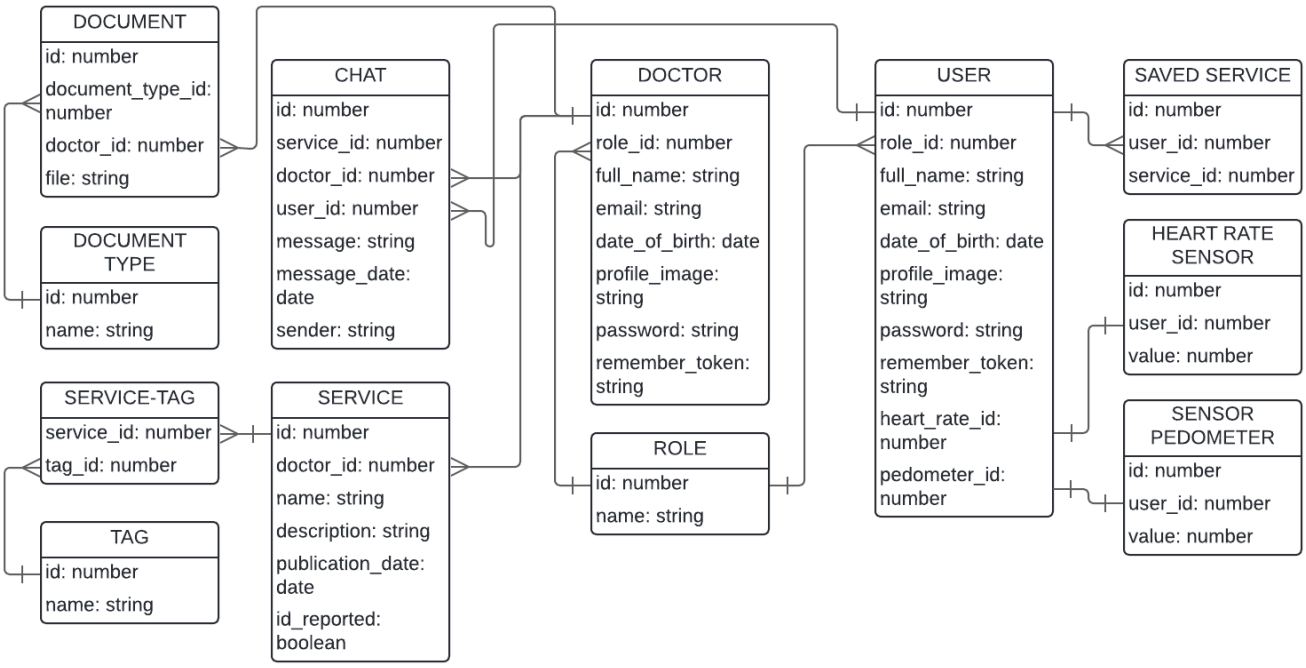


Рисунок 3.3 – ER-діаграма для програмної системи

Отже, була розроблена схема бази даних, що містить 12 реляційних таблиць, які між собою мають різні види відношень та знаходяться у третій нормальній формі.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Загальні відомості

Програмна система була повністю розроблена на комп'ютері з операційною системою Windows з використанням підсистеми WSL з операційною системою Linux дистрибутиву Ubuntu.

Була використана СУБД MySQL, оскільки вона забезпечує високу продуктивність, надійність та масштабованість для зберігання великих обсягів даних. MySQL підтримує транзакції та реплікацію, що є критично важливим для стабільної роботи системи, особливо у випадках великої кількості одночасних запитів і користувачів. Крім того, MySQL легко інтегрується з Laravel, спрощуючи налаштування і управління базою даних. Її популярність і широке використання в промисловості забезпечують доступність великої кількості ресурсів і підтримки, що важливо для забезпечення безперебійної роботи і майбутнього розширення системи.

Програмна система була розроблена з урахуванням вимог до низького споживання ресурсів, що дозволяє її ефективне функціонування на апаратному забезпеченні з помірними характеристиками. Зокрема, при тестуванні на п’ятиядерному 64-розрядному процесорі система продемонструвала стабільну роботу без помітних проблем або уповільнень. Завдяки оптимізації коду та архітектури, система не потребує затратних графічних обчислень, що знижує потребу у потужних відеокартах. Це робить її сумісною навіть з пристроями, які не мають спеціальних графічних прискорювачів. Що стосується використання оперативної пам’яті, система використовує її в мінімальних обсягах, що забезпечує плавну роботу навіть на комп'ютерах з обмеженими ресурсами. Під час тестування із кількома десятками одночасних користувачів було виявлено, що споживання пам’яті залишалося на низькому рівні, і навіть комп'ютер з 2 ГБ оперативної пам’яті зміг забезпечити належну продуктивність. Крім того, для комфортного використання всіма функціями

системи достатньо будь-якого сучасного дисплея, що здатен відобразити веб-сторінку, що робить систему доступною для широкого кола користувачів.

З програмного забезпечення на комп’ютері повинні присутні інтерпретатор мови програмування PHP, js, для Laravel та VueJs, встановленні модулі Broadcast, Docker-compose, npm, програмний застосунок Docker Desktop.

4.2 Виклик та завантаження

Так як програмна система складається з двох частин, а саме веб-клієнту та серверної частини, розглянемо налаштування та запуск кожної з частин окремо.

Почнемо з серверної частини. Оскільки сервер побудований за допомогою фреймворку Laravel, вирішення потенційних проблем стає легшим завдяки відповідним електронним ресурсам. Однак, для запуску сервера потрібно підготувати середовище Docker контейнерів. Цей процес досить простий: достатньо запустити Docker Desktop та виконати стандартну команду «docker-compose up» у терміналі. У разі виникнення проблем, система автоматично надасть необхідну інформацію для їх вирішення. Після успішного завершення роботи програми, можна використати команду «docker-compose down», щоб припинити роботу Docker. Додатково, для створення онлайн-чату використовується технологія Broadcast, для підключення якої слід розкоментувати BroadcastServiceProvider у відповідному файлі проекту: config -> app.php. Перед початком роботи з базою даних на сервері рекомендується створити відповідні міграції моделей. Для цього у контейнері Docker можна використати команду «php artisan migrate». Крім того, база даних повинна містити записи для таблиць DocumentType та Role, що можна створити за допомогою seeders у папці database -> seeders та викликати їх командою «php artisan db –class=НазваКласу».

Для ефективного запуску клієнтської частини програми необхідно спочатку завантажити всі необхідні бібліотеки за допомогою команди «npm install». Це

дозволить встановити всі необхідні залежності і гарантувати коректну роботу програми. Після завершення цього процесу можна переходити до запуску програми за допомогою команди «npm run serve». Цей крок дозволить запустити локальний сервер і відкрити програму у веб-браузері без будь-яких помилок, пов'язаних з відсутністю необхідних бібліотек чи залежностей.

4.3 Опис фізичної моделі бази даних

Для зберігання та маніпулювання даними в моєму дипломному проекті використовується система керування базами даних MySQL. Вибір цієї СУБД обумовлений її потужністю, надійністю та широким функціоналом, що ідеально відповідає вимогам проекту. Docker використовується для контейнеризації додатку, що забезпечує легке розгортання та масштабування системи в різних середовищах, забезпечуючи стабільність та ізольованість процесів. Laravel, як фреймворк для розробки бекенду, використовується для швидкої обробки запитів, управління базами даних та реалізації бізнес-логіки. Створення та модифікація таблиць реалізована за допомогою механізму міграцій Laravel, який дозволяє ефективно вносити зміни до структури бази даних. Цей механізм автоматизує процес створення та оновлення таблиць, що дозволяє розробникам зосередитися на інших аспектах розробки. Також використано Docker для зручного розгортання та масштабування бази даних у різних середовищах. Приклад скрипту, що був створений вручну та успішно застосований у базі:

```
public function up(): void
    {
        Schema::table('chats', function (Blueprint $table) {
            $table->string('sender')->nullable()->after('message');
        });
    }
```

З генерації скрипту видно, що було додано стовпець "sender" (роль відправника)

до таблиці "chats". Це дозволяє створювати власні файли міграції для будь-яких операцій, але рекомендується використовувати вбудовані механізми Laravel для автоматизації цього процесу.

Одним ілюстративним прикладом скрипту, згенерованого для створення основної таблиці в системі, як, наприклад, таблиці "Service", є наступний фрагмент коду:

```
public function up(): void
    {
        Schema::create('services', function (Blueprint $table) {
            $table->id();
            $table->foreignId('doctor_id')->constrained('doctors');
            $table->string('name');
            $table->text('description');
            $table->date('publication_date');
            $table->boolean('is_reported')->default(false);
            $table->timestamps();
        });
    }
```

У методі "up", за допомогою об'єкту "migrations", вказуються основні компоненти таблиці, такі як її назва, поля та обмеження. Для таблиці "Service" визначено такі поля, як ідентифікатор, індекс зовнішнього ключа "doctor_id", який зв'язаний з таблицею "doctors", назва послуги, її опис, дата публікації, прапорець "is_reported" зі значенням за замовчуванням "false" та мітки часу оновлення. Для кожного з полів використовуються спеціальні класи моделі де для кожного типу даних – свій об’єкт.

У параметрі поля "id" визначено основний ключ, який вимагає унікальності та подальших обмежень. Аналогічно встановлюється і зовнішній ключ для таблиці "services".

Ми також можемо розглянути приклад коду для моделі таблиці "Збережені послуги", яку користувачі можуть використовувати:

```
public function up(): void
    {
        Schema::create('saved_services', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('service_id');
```

```
            $table->foreignId('user_id')->constrained('users')-
>onDelete('cascade');
            $table->timestamps();
        });
    }
```

Цей скрипт створює таблицю "Збережені послуги", де кожен запис має унікальний ідентифікатор ("id"). Поле "service_id" зберігає ідентифікатор послуги, яку зберігає користувач. Поле "user_id" вказує на користувача, який зберігає цю послугу, та встановлює зовнішній ключ, який зв'язується з таблицею "users". Опція "onDelete('cascade')" вказує на те, що при видаленні користувача будуть автоматично видалятися всі його збережені послуги. Крім того, в таблиці є поля "created_at" та "updated_at", які відповідають за дату та час створення та оновлення запису відповідно.

4.4 Опис серверної частина програмної системи

Розробка серверної частини програмної системи на платформі Windows часто стикається з викликами сумісності, особливо коли потрібно використовувати інструменти, які традиційно більш сумісні зі стеком розробки Linux. Для вирішення цих проблем використовується Windows Subsystem for Linux (WSL) з Ubuntu. WSL дозволяє запускати середовище Linux, таке як Ubuntu, безпосередньо на Windows, що надає можливість використовувати улюблені інструменти та ресурси Linux, не залишаючи середовище Windows.

Ubuntu, як один з найпопулярніших дистрибутивів Linux, був вибраний для використання в Windows Subsystem for Linux (WSL) з-за своєї широкої підтримки, стабільності та відкритого джерела. Це дозволяє розробникам отримати доступ до розширеного арсеналу інструментів та бібліотек, що полегшує роботу та вирішення можливих проблем на платформі Windows. Крім того, Ubuntu відомий своєю

безпекою, що має важливе значення в сучасних програмних системах, де одна помилка може призвести до серйозних наслідків, включаючи втрату даних і довіри користувачів. Безкоштовність Ubuntu також забезпечує широку популярність серед користувачів, що сприяє збільшенню кількості ресурсів та підтримки. Крім того, Ubuntu має велику кількість корисних програм для створення резервних копій, що є важливим аспектом будь-якої програмної системи. Щорічні оновлення Ubuntu забезпечують передбачуваність та можливість планування вдосконалень, що сприяє ефективному управлінню ресурсами та забезпечує актуальність системи.

Підключення необхідних бібліотек та залежностей до проекту здійснюється за допомогою npm, який є потужним менеджером пакетів. Він дозволяє легко встановлювати та оновлювати різноманітні пакети. Використання npm спрощує управління залежностями та забезпечує консистентність серед розробників.

Щодо передачі даних між серверною та клієнтською частинами програмної системи, Axios є одним з найефективніших інструментів. Він дозволяє виконувати різноманітні HTTP-запити, такі як GET, POST, PUT, DELETE, забезпечуючи надійну та ефективну комунікацію між сторонами. Використання Axios дозволяє розробникам легко обмінюватися даними та взаємодіяти з сервером, забезпечуючи стабільну та продуктивну роботу програмної системи.

Налаштування можливості використання маршруту визначається використанням CSRF токену у форматі Bearer, що допомагає забезпечити безпеку та автентифікацію в системі шляхом встановлення шляхів у спільну групу Middleware auth:sanctum. Токени CSRF призначені для запобігання атакам CSRF, що унеможливлює зловмисникам створювати повністю дійсні HTTP-запити, які можуть бути використані для атаки на користувача-жертву. Оскільки зловмисник не може заздалегідь визначити або передбачити значення маркера CSRF користувача, він не може створити запит з усіма потрібними параметрами для виконання дії в програмі. Laravel автоматично генерує цей токен для кожного сеансу користувача. Коли користувач взаємодіє з додатком, Laravel перевіряє, чи співпадають значення токена

CSRF, надісланого користувачем з форми, зі значенням, збереженим у сеансі. Якщо токени співпадають, запит вважається дійсним, і він буде оброблений. Це дозволяє захистити додаток від небажаних атак, забезпечуючи безпеку передачі даних між клієнтом і сервером. У звичайній практиці не виникає ситуації, коли функція отримує недійсний токен, але все ж краще передбачити і врахувати неймовірні випадки. Тобто, якщо користувача не вдається знайти, відповідь може містити HTTP-код помилки 402 - «Немає авторизації». В іншому випадку об’єкт користувача серіалізується і повертається до веб-додатку.

Маршрутизація визначена в файлі api.php відповідної папки routes. Вона є простою і складається з запитів Route бібліотеки Illuminate. Також більша частина шляхів поміщена в спільну группу захисту auth:sanctum. Щоб додати новий запит до системи, потрібно лише створити новий шлях у цьому файлі та визначити його обробку. Нижче наведено приклад таких шляхів:

```
Route::group(['middleware' => 'auth:sanctum'], function () {
    Route::get('/auth/getUsers', [UserController::class, 'getUsers']);
    Route::get('/auth/getUser/{id}',              [UserController::class,
'getUser']);
    Route::get('/auth/getDoctor/{id}',            [UserController::class,
'getDoctor']);
    Route::post('/auth/user/update/{id}',         [UserController::class,
'updateUser']);
});
```

Повний код маршрутизації можна побачити у додатку Б.

Важливо відзначити, що існує глобальна маршрутизація у файлі проекту, яка використовує маршрутизацію додатка. Таким чином, повний URL-шлях для отримання користувача з програмної системи буде наступним: https://<ip>:<port>/<application_name>/auth/getUser.

Створення моделей для міграцій у Laravel є ключовим елементом у розробці програмних систем, оскільки вони забезпечують зручний спосіб роботи з базою даних на рівні об'єктів. Моделі відображають структуру та логіку взаємодії з даними, що були визначені у міграціях, дозволяючи розробникам маніпулювати даними, не занурюючись у складні SQL-запити. Це спрощує код і робить його більш зрозумілим

та підтримуваним. Наприклад, після створення міграції для таблиці services, яка визначає поля і їх типи, модель Service дозволить легко взаємодіяти з цими даними, забезпечуючи методи для створення, читання, оновлення та видалення записів у таблиці:

```
class Service extends Model
{
    use HasFactory;

    protected $fillable = [
        'doctor_id',
        'name',
        'description',
        'publication_date',
        'is_reported'
    ];

    public function doctor()
    {
        return $this->belongsTo(Doctor::class);
    }

    public function chat()
    {
        return $this->hasMany(Chat::class);
    }

    public function tags()
    {
        return $this->belongsToMany(Tag::class);
    }
}
```

Крім того, моделі у Laravel можуть включати додаткову логіку бізнес-процесів, як-от визначення зв'язків між таблицями, наприклад, відношення "один до багатьох" або "багато до багатьох", що робить роботу з базою даних більш інтуїтивною та ефективною. Таким чином, створення моделей є необхідним для ефективної організації коду та забезпечення гнучкості у роботі з даними, що в кінцевому підсумку сприяє стабільності та масштабованості всієї системи.

Подальше оброблення логіки відбувається у контроллері та зазначеним в шляху методі контроллеру. Таким чином програма розуміє який тип запиту був здійснений, який контроллер слід використовувати і в якому методі оброблювати логіку. До прикладу отримання звичайних користувачів відбувається наступним чином:

```
    public function getUser($id)
        {
            try {
                $user = User::findOrFail($id);

                return response()->json([
                    'status' => true,
                    'message' => 'User logged in successfully',
                    'user'       =>       $user->load('role',       'savedServices',
'sensorPedometers', 'heartRateSensors'),
                ], 200);
            } catch (\Throwable $th) {
                return response()->json([
                    'status' => false,
                    'message' => $th->getMessage()
                ], 500);
            }
        }
```

Тут відбувається отримання даних юзера знайденого по вказаному id в маршруту та подальше відповідь на клієнтську частину даних або помилки.

Для обробки запиту на редагування послуги, що включає складніший підхід, ніж просте оновлення даних, необхідно також врахувати взаємодію з тегами, прикріпленими до цієї послуги. Якщо тег вже існує, він просто прикріплюється до послуги; якщо ні – створюється новий тег у базі даних. Нижче наведено приклад такої функції:

```
    public function update(Request $request, $id)
        {
            try {
                $service = Service::findOrFail($id);
                $tags = $request->input('tags');

                $service->name = $request->input('name');
                $service->description = $request->input('description');
                $service->is_reported = $request->input('is_reported');
                $service->save();

                $newTags = [];
                foreach ($tags as $tag) {
                    if (is_null($tag['id'])) {
                        $newTag = Tag::create(['name' => $tag['name']]);
                        $newTags[] = $newTag->id;
                    } else {
                        $newTags[] = $tag['id'];
                    }
                }
                $service->tags()->sync($newTags);

                return response()->json([
```

```
                'status' => true,
                'data' => $service->load('tags'),
                'message' => 'Service updated successfully',
            ], 200);
        } catch (\Throwable $th) {
            return response()->json([
                'status' => false,
                'message' => $th->getMessage()
            ], 500);
        }
    }
```

Можна побачити, що ця функція починається з пошуку послуги за її ідентифікатором у базі даних. Далі вона отримує масив тегів з вхідного запиту, оновлює основні поля послуги та зберігає зміни. Потім, функція перевіряє, чи кожен тег має ідентифікатор: якщо ні, створюється новий тег, і його ідентифікатор додається до списку. Інакше, до списку додається існуючий ідентифікатор тегу. Після цього виконується синхронізація тегів з послугою. Нарешті, якщо все пройшло успішно, повертається JSON-відповідь з оновленими даними послуги і повідомленням про успішне оновлення; у разі помилки – повертається відповідь з кодом помилки та повідомленням.

Процес валідації даних, які надходять до сервера від користувача, щоб забезпечити їх відповідність певним правилам та критеріям перед подальшою обробкою є важливою частиною обробки запиту. Цей процес важливий для підтримки цілісності та безпеки системи, зменшення ризику помилок і захисту від зловмисних дій. У Laravel валідація даних є вбудованою функцією, яка забезпечує простий та ефективний механізм для перевірки вхідних запитів. Створюється відповідний Request, що зберігає в собі правила віладації даних:

```
class StoreRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     */
    public function authorize(): bool
    {
        return true;
    }

    /**
```

```
         * Get the validation rules that apply to the request.
         *
         *                          @return                          array<string,
\Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>
         */
        public function rules(): array
        {
            return [
                'full_name' => 'required|max:255',
                'email' => 'required|email|unique:users,email',
                'date_of_birth' => 'date',
                'password' => 'required'
            ];
        }

        public function messages()
        {
            return [
                'full_name.required' => 'This field must be filled in',
                'full_name.max'  =>  'This  field  must  be  less  than  255
characters',
                'email.required' => 'This field must be filled in',
                'email.email'  =>  'Your  mail  should  follow  the  format
"example@gmail.com"',
                'email.unique' => 'This mail is already taken',
                'date_of_birth' => 'This field must be a date',
                'password.required' => 'This field must be filled in',
            ];
        }
    }
```

У контексті розробки програмних систем, використання репозиторіїв та інтерфейсів до них є важливим практичним підходом, що значно покращує архітектуру та підтримуваність коду. Репозиторій виконує роль абстракції для доступу до даних, відокремлюючи бізнес-логіку програми від специфіки зберігання даних. Це дозволяє розробникам змінювати підходи до зберігання (наприклад, змінювати базу даних або спосіб отримання даних) без впливу на інші частини коду.

Одним з таких прикладів є логіка обробки сповіщень під час спілкування в чаті:

```
class ChatRepository implements ChatRepositoryInterface
{
    public function list($doctor_id, $user_id, $service_id)
    {
        return Chat::query()
            ->where('service_id', $service_id)
            ->where(function ($query) use ($doctor_id, $user_id) {
                $query->where('doctor_id', $doctor_id)
                    ->where('user_id', $user_id);
            })
            ->orderBy('created_at', 'asc')
```

```
            ->get();
    }

    public function send($data) {
        $message = new Chat;
        $message->fill($data)->save();
        return $message;
    }
}
```

З коду помітно унаслідковування інтерфейсу ChatrepositoryInterface для зазначення розробнику про реалізацю необхідних методів:

```
Interface ChatRepositoryInterface{

    public function list($doctor_id, $user_id, $service_id);
    public function send($data);
}
```

Документи, які користувачі завантажують у систему, зберігаються у спеціально створеній підпапці documents в каталозі storage/app/public. Цей підхід забезпечує легкий доступ до документів, таких як договори, звіти, або будь-які інші файли, які можуть бути завантажені користувачами. Завдяки цьому, всі файли організовані і легко доступні за визначеними URL, наприклад, https://<домен>/storage/documents/<назва_файлу>. Це значно спрощує процес роботи з документами, роблячи їх доступними без складних процедур аутентифікації або додаткових налаштувань. Крім того, такий підхід полегшує управління та підтримку документів, дозволяючи легко знаходити і працювати з потрібними файлами.

Використання каталогу storage/app/public для зберігання документів та зображень забезпечує кілька важливих переваг. По-перше, це дозволяє зберігати всі файли в одному місці, що полегшує управління і підтримку їх доступності. По-друге, URL до файлів стають передбачуваними та легко використовуваними в додатку, що спрощує відображення контенту на веб-сторінках. По-третє, зберігання файлів у публічному каталозі забезпечує швидкий доступ до них з клієнтської сторони, що покращує продуктивність і зручність використання додатка.

Логіку зберігання можна побачити у наступному коді створення лікаря або психіатра:

```
public function createDoctor(Request $request)
```

```
        {
            try {
                if ($request->hasFile('profile_image')) {
                    $path         =         $request->file('profile_image')-
>store('public/images');
                    $path = str_replace('public/', '', $path);
                } else {
                    $path = null;
                }

                $user = Doctor::create([
                    'full_name' => $request->full_name,
                    'email' => $request->email,
                    'date_of_birth' => $request->date_of_birth,
                    'profile_image' => $path,
                    'password' => Hash::make($request->password),
                    'role_id' => intval($request->registrationType)
                ]);

                $documentTypes            =            json_decode($request-
>input('document_types'), true);

                foreach ($documentTypes as $index => $type) {
                    $uploadedFile   =   $request->file("documents_{$index}")-
>store('public/documents');
                    $typeId = intval($type);

                    if ($uploadedFile) {
                        $document = new Document([
                            'document_type_id' => $typeId,
                            'doctor_id' => $user->id,
                            'file' => $uploadedFile
                        ]);

                        $user->documents()->save($document);
                    }
                }

                return response()->json([
                    'status' => true,
                    'message' => 'User created successfully',
                    'user' => $user->load('role', 'documents'),
                    'token'     =>     $user->createToken("API     TOKEN")-
>plainTextToken
                ], 200);
            } catch (\Throwable $th) {
                return response()->json([
                    'status' => false,
                    'message' => $th->getMessage()
                ], 500);
            }
        }
```

Документи в documents і зображення в images завжди легко знайти і

використовувати, що робить систему більш гнучкою та зручною для користувачів. Такий підхід до організації файлів сприяє загальній ефективності роботи додатка і забезпечує його стабільну та продуктивну роботу.

4.5 Опис веб-клієнту програмної системи

Веб-клієнт нашої програмної системи був побудований з використанням сучасного фреймворку Vue 3, що забезпечує гнучкість, продуктивність і легкість у розробці компонентів. Vue 3 є потужним інструментом для створення односторінкових додатків (SPA), що дозволяє реалізувати інтуїтивно зрозумілий та інтерактивний інтерфейс користувача. З допомогою Vue 3 ми створили динамічну та масштабовану систему, яка відповідає вимогам сучасного веб-додатку.

Одним із ключових аспектів нашого клієнта є використання Pinia – сучасного та ефективного рішення для управління станом у додатках на Vue 3. Pinia дозволяє централізовано управляти станом додатку, що спрощує роботу з даними та забезпечує узгодженість у всіх компонентах. Це особливо важливо у великих додатках, де синхронізація стану між різними частинами додатку може стати складним завданням. Pinia надає зрозумілу та прозору структуру для роботи з глобальним станом, забезпечуючи легкий доступ до нього з будь-якого місця в додатку. Завдяки цьому ми можемо ефективно працювати з даними користувачів, їхніми налаштуваннями та іншою інформацією, яка впливає на роботу всього додатку.

Для забезпечення реального часу і миттєвого оновлення даних у нашій програмній системі ми інтегрували Pusher – сервіс, що дозволяє легко реалізовувати функціональність push-повідомлень та оновлень у реальному часі. Використання Pusher у нашому додатку дозволяє миттєво передавати повідомлення між сервером і клієнтом, забезпечуючи швидку та ефективну комунікацію. Це особливо важливо для

додатків, які потребують високої інтерактивності та швидкої реакції на дії користувачів, таких як чати, панелі адміністратора або будь-які інші компоненти, де оновлення даних повинно відбуватися в реальному часі.

Для забезпечення ефективного спілкування між клієнтською частиною нашої програмної системи, побудованої на Vue 3, та сервером, ми використовуємо Axios – потужний і гнучкий HTTP-клієнт. Axios значно спрощує процес виконання запитів до сервера, дозволяючи легко реалізовувати операції типу GET, POST, PUT та DELETE. Він підтримує асинхронну обробку запитів, що дозволяє нашому додатку реагувати на дії користувачів у реальному часі без необхідності перезавантаження сторінки. Використання Axios забезпечує автоматичне управління заголовками запитів, включаючи передачу токенів для аутентифікації, таких як CSRF токени в заголовку Bearer, що є критично важливим для безпеки і захисту від атак. Крім того, Axios надає зручний механізм обробки помилок і повторних спроб, що підвищує стійкість нашої системи до тимчасових збоїв у мережі або серверних помилок. Завдяки простому і зрозумілому API, Axios інтегрується у Vue 3 без зайвих ускладнень, що дозволяє розробникам зосередитися на створенні функціональних можливостей додатку, а не на низькорівневих аспектах роботи з мережею.

Таким чином на клієнтській частині було створено налаштування зв’язку із сервером у папці services:

```
import axios from "axios";

let access_token = localStorage.getItem('token')

const instance = axios.create({
    baseURL: 'http://localhost:85/api',
});

instance.interceptors.request.use((config) => {
    config.headers.Authorization = `Bearer ${access_token}`;
    return config;
});

export default instance;
```

Отримання повідомлень від сервера включає в себе не лише використання

технології Pusher для отримання миттєвих оновлень, але й пряме взаємодію з сервером через фреймворк Laravel. Крім Pusher, наш веб-клієнт використовує бекенд, побудований на Laravel, для отримання та обробки даних. За допомогою Axios, ми надсилаємо запити до сервера, а Laravel обробляє ці запити та відправляє назад відповіді. Це може бути отримання нових даних, зміни в існуючих записах чи будь-яка інша інформація, яка потребує оновлення в клієнтській частині. Такий двобічний обмін даними забезпечує зручну та ефективну роботу програмної системи, дозволяючи користувачам отримувати актуальну інформацію у реальному часі та взаємодіяти з додатком без затримок. Отримані від сервера дані ефективно зберігаються в Pinia - становій управляючій бібліотеці, яка використовується в нашому веб-клієнті для керування станом додатка. Pinia дозволяє легко та ефективно зберігати, оновлювати та отримувати дані у клієнтській частині програми. Завдяки цьому механізму, дані, отримані від сервера через Axios або Pusher, автоматично синхронізуються зі станом додатка у Pinia. Це забезпечує консистентність та актуальність інформації, яка відображається користувачеві у веб-інтерфейсі. Використання Pinia дозволяє ефективно керувати станом додатка та забезпечує його плавну та зручну роботу. Ось як виглядає структура запитів та зберігання даних в Pinia:

```
export const useChatStore = defineStore('chat', {
    state: () => ({
        messages: null,
        chats: []
    }),
    actions: {
        async getChatMessages(doctor_id, user_id, service_id) {
            await
AxiosInstanse.get(`/chat/${doctor_id}/${user_id}/${service_id}`).then((respo
nse) => {
                this.messages =  response.data.data;
            })
        },

        async sendMessage(data, doctor_id, user_id) {
            AxiosInstanse.post(`/chat/send/${doctor_id}/${user_id}`,
data)
        },
```

```
            async getDoctorChats(service_id, doctor_id) {
                await
AxiosInstanse.get(`/chat/${service_id}/${doctor_id}`).then((response) => {
                    this.chats =  response.data.data;
                })
            },

            async deleteDoctorChat(data, t) {
                await AxiosInstanse.post(`/chat/destroy`, data).then(() =>
{
                    Swal.fire({
                        icon: 'success',
                        title: t('chat.swal.error_title'),
                        text: t('chat.swal.error_text'),
                        timer: 2500,
                        showConfirmButton: false,
                    });
                }).catch(() => {
                    Swal.fire({
                        icon: 'error',
                        title: t('chat.swal.error_title'),
                        text: t('chat.swal.error_text'),
                        timer: 2500,
                        showConfirmButton: false,
                    });
                });
            }
        },
    })
```

У функціях дій (actions) визначені методи для взаємодії з сервером. Наприклад, метод "getChatMessages" відправляє запит для отримання повідомлень чату за вказаними параметрами, і отримані дані зберігає у стані "messages". Аналогічно, методи "sendMessage" та "getDoctorChats" відповідають за надсилання повідомлень і отримання історії чатів відповідно.

Також варто відзначити, що перед кожним запитом до сервера автоматично додається токен для аутентифікації користувача. Це забезпечує безпеку та автентичність кожного запиту із клієнта до сервера. Використання токену в запитах дозволяє перевірити права доступу та упевнитися у коректності виконання дій у системі.

У проекті використовуються різні технології для створення веб-додатка. Однією з ключових технологій є Vue 3, який забезпечує масштабовану та динамічну

реактивність веб-інтерфейсу. Для ефективного керування станом додатка і взаємодії між компонентами використовується бібліотека Pinia. Вона дозволяє створювати глобальний стан за допомогою store, який зберігає дані та забезпечує їхню доступність у всьому додатку. Крім того, для збереження стану застосунку при оновленні сторінки використовується плагін piniaPluginPersistedstate.

Для міжнародного перекладу інтерфейсу використовується бібліотека vue-i18n, яка дозволяє легко локалізувати додаток для різних мовних середовищ. Різні мовні файли, такі як uk.json та en.json, зберігають переклади текстів у різних мовах.

Для забезпечення взаємодії з сервером та отримання реального часу оновлень використовується пакет laravel-echo та бібліотека Pusher. Це дозволяє налаштовувати прослуховувачів подій на сервері та реагувати на них у реальному часі.

Крім того, додаток використовує бібліотеку Bootstrap для стилізації та розмітки веб-інтерфейсу, яка надає готові компоненти та класи для швидкого створення стильних інтерфейсів.

Нижче наведений фрагмент коду, де виконуються основні налаштування додатка, включаючи створення Pinia, встановлення Pusher, підключення маршрутизатора, створення міжнародних перекладів та монтування додатка у DOM:

```
window.Pusher = require('pusher-js');

window.Echo = new Echo({
    broadcaster: 'pusher',
    key: '74bef44c38b3ccd601bb',
    cluster: 'eu',
    forceTLS: true
});

const pinia = createPinia();
const app = createApp(App);
pinia.use(piniaPluginPersistedstate);

pinia.use(({ store }) => {
```

```
    store.router = markRaw(router);
});
app.component('header-app', HeaderApp);

app.component('sidebar-app', SidebarApp);
app.use(pinia);
app.use(router);
app.mount('#app');

const i18n = createI18n({
    locale: "uk",
    messages: {uk, en}
});

app.use(i18n);
```

Опановуючи інтерфейс та спілкування з користувачем, можна зазначити, що основне меню представлене у вигляді верхньої панелі Header, де наведено ключові функції та можливості сайту. Клацання на будь-яку з посилань автоматично перенаправляє користувача на відповідну сторінку, де відбувається відповідна логіка. Зміна мови впливає на вміст сторінки, тому важливо, щоб програма правильно інтерпретувала команди на обох мовах, які користувач може вибрати – українській та англійській. Більшість сайтів починаються з процедури авторизації або реєстрації, вигляд яких представлено на рисунках 4.1 – 4.2.

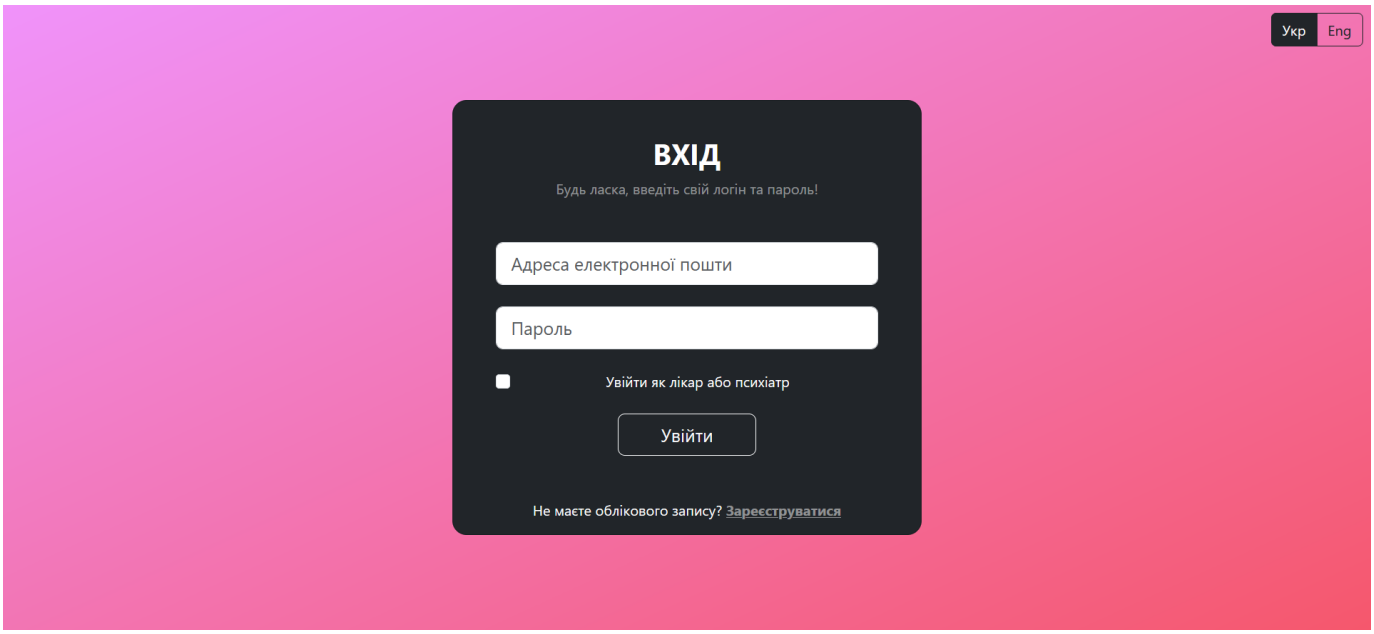


Рисунок 4.1 – Вигляд сторінки авторизації

Рисунок 4.2 – Вигляд сторінки реєстрації

Створення функцій валідації на Vue дозволяє забезпечити надійну та консистентну перевірку введених даних користувачем перед їх відправленням на сервер. Цей процес включає в себе розробку методів або компонентів, які перевіряють введені дані на відповідність заданим критеріям, таким як обов'язковість поля, формат електронної пошти, мінімальна та максимальна довжина тощо. Використання функцій

валідації допомагає забезпечити коректну взаємодію з користувачем, запобігаючи надсиланню невірних або неповних даних, що може призвести до помилок у роботі програми. Такий підхід до валідації дозволяє підтримувати високу якість програмного продукту та покращує його використання для кінцевих користувачів. Таким чином було створено обробку даних для цих форм у відповідних файлах валідації в папці validations. Ось приклад одного з таких файлів валідації авторизації:

```
export function validateEmail(email) {
    if (!email) {
        return 'email_empty';
    } else if (!/\S+@\S+\.\S+/.test(email)) {
        return 'email_format';
    } else {
        return null;
    }
}

export function validatePassword(password) {
    if (!password) {
        return 'password_empty';
    } else if (password.length < 7) {
        return 'password_length';
    } else {
        return null;
    }
}
```

Під час створення окремих компонентів та сторінок, кожен з них зазвичай зберігається у відповідній папці, що допомагає підтримувати порядок у структурі проекту. Наприклад, компоненти можуть бути розташовані у папці "components", а окремі сторінки - у папці "views" або "pages".

Створення окремих компонентів та сторінок є важливою практикою у розробці веб-додатків з багатьма перевагами. Розбиття інтерфейсу на компоненти дозволяє підтримувати структурованість та організацію коду, сприяючи більшій читабельності та підтримці коду у майбутньому. Кожен компонент відповідає за конкретну функціональність або візуальний елемент, що спрощує виконання змін та розвиток додатку.

Створення окремих сторінок також сприяє чіткості та структурованості проекту. Кожна сторінка відповідає за певну частину функціональності додатку та може бути

незалежною одиницею розробки, що дозволяє команді працювати паралельно над різними частинами проекту.

Розділення на компоненти та сторінки також забезпечує можливість повторного використання коду. Компоненти, які виконують певні функції, можна використовувати на різних сторінках або навіть у різних проектах, що зменшує зусилля при розробці та підтримці програмного забезпечення. На рисунку 4.3 можна побачити відповідну архітектуру системи.

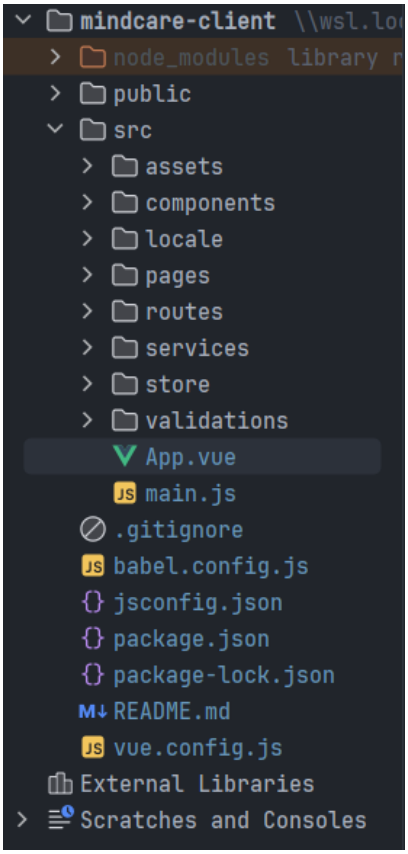


Рисунок 4.3 – Вигляд архітектури проекту

У програмній системі існують різні види потраплення на сторінку, залежно від ролі користувача. Звичайний користувач має можливість переглядати всі доступні послуги на проекті, долучати їх до свого списку улюблених або подавати скарги на них. У верхній панелі навігації (хедері) для звичайного користувача доступні маршрути до даних сенсорів, списку улюблених послуг, налаштувань профілю та

виходу з системи (рисунок 4.4).

Лікар або психіатр, увійшовши в систему, бачить лише свої послуги, які він надає. Для цієї категорії користувачів також доступні маршрути на налаштування профілю та виходу з системи (рисунок 4.5).

Такий підхід до відображення функціональності в залежності від ролі користувача допомагає забезпечити персоналізований та зручний досвід взаємодії з програмною системою для кожного типу користувача.

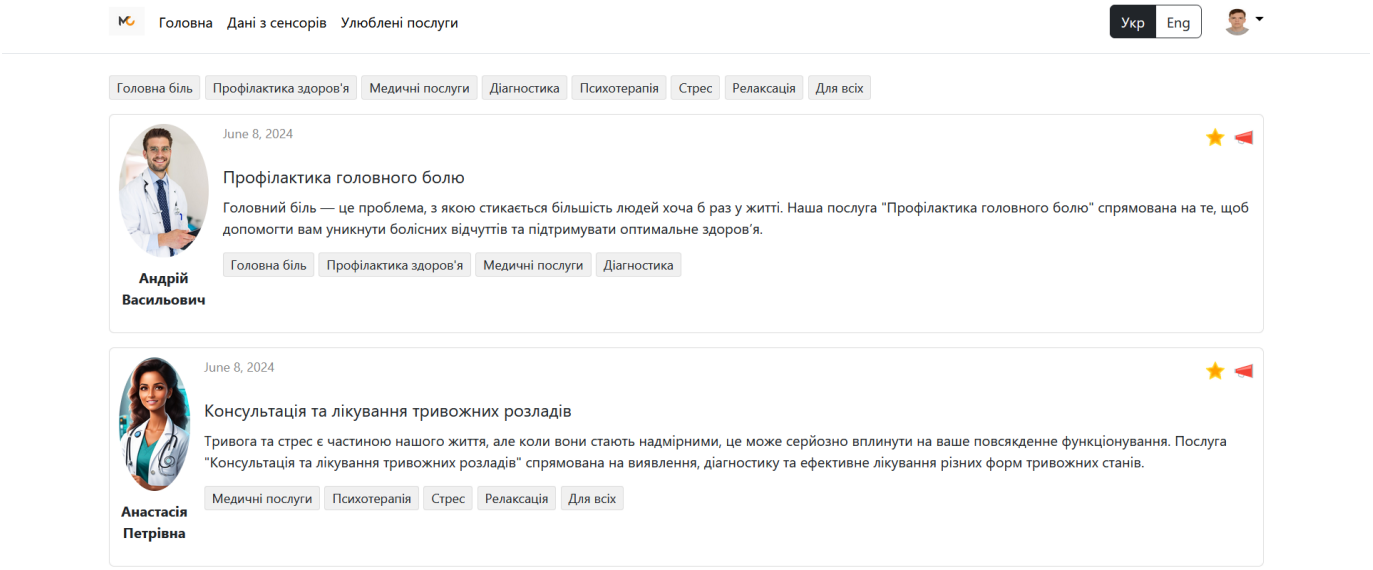


Рисунок 4.4 – Вигляд головної сторінки звичайного користувача

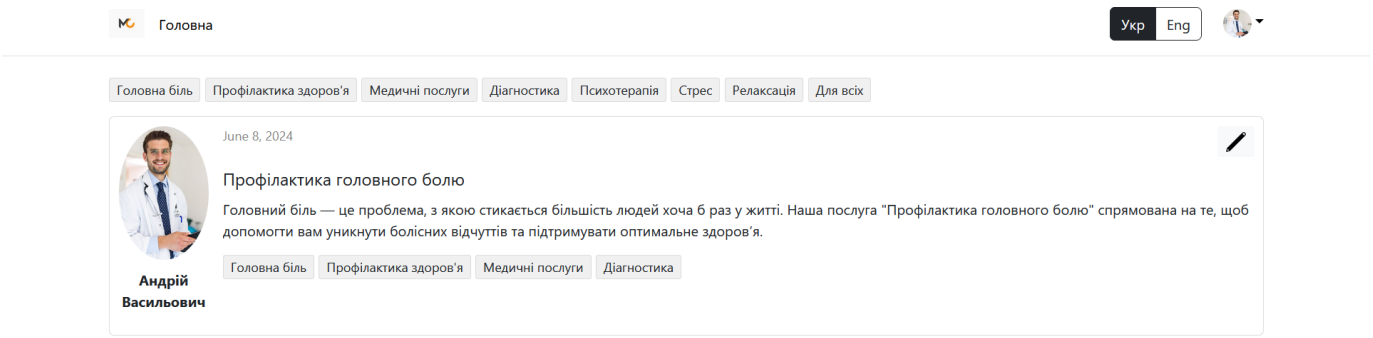


Рисунок 4.5 – Вигляд головної сторінки лікаря

Меню налаштувань у програмній системі надає користувачам можливість

змінювати дані свого профілю. В цьому меню доступні різноманітні опції, такі як зміна особистих даних, включаючи ім'я, прізвище, адресу електронної пошти та інші (рисунок 4.6). Окрім цього, в меню налаштувань можуть бути доступні інші параметри, такі як мовні налаштування, зміна теми інтерфейсу або вибір інших персоналізованих налаштувань, які дозволяють користувачам налаштувати досвід використання системи під свої індивідуальні потреби та вподобання.

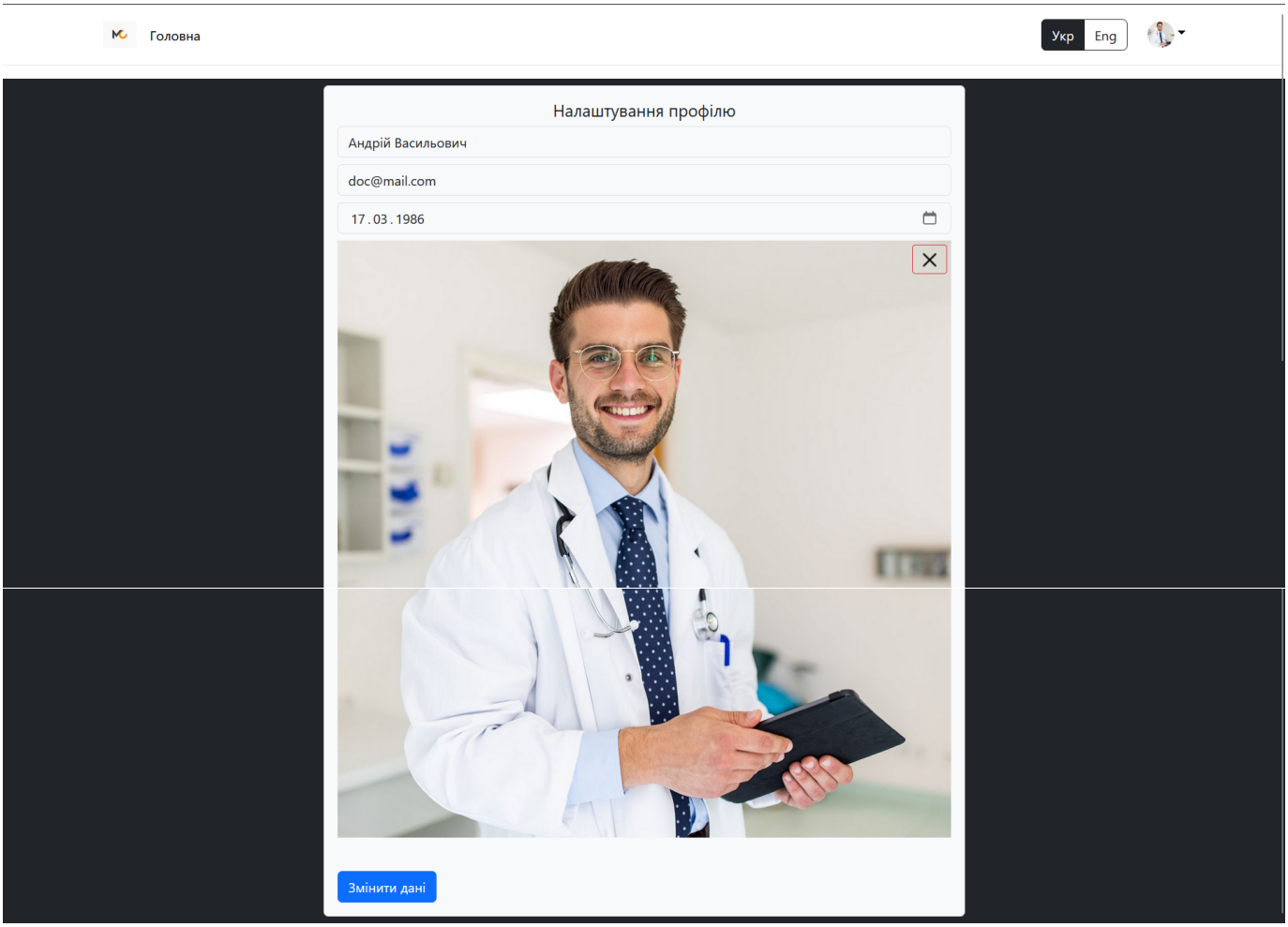


Рисунок 4.6 – Вигляд сторінки оновлення даних користувача

У панелі лікарів/психіатрів доступна функціональність створення нової послуги для надання пацієнтам. Під час створення нової послуги, лікар чи психіатр має можливість ввести назву послуги, що чітко відображає її характер та мету (рисунок 4.7). Опис послуги дозволяє докладно розповісти пацієнтам про її суть, особливості та

переваги. Крім того, під час створення послуги, лікар чи психіатр може вказати теги, які краще всього описують тематику та специфіку цієї послуги. Теги допомагають пацієнтам швидше знайти потрібну послугу серед доступних у системі та полегшують навігацію користувачів в розділі послуг. Завдяки Vue-Multiselect, користувачі можуть легко вибирати необхідні теги з доступного списку або вводити нові теги, які вони вважають потрібними для конкретної послуги. Цей компонент надає зручний інтерфейс для додавання тегів, дозволяючи користувачам швидко та ефективно встановлювати ключові параметри для кожної послуги. Крім того, Vue-Multiselect забезпечує можливість налаштування різноманітних параметрів відображення та поведінки компонента, що робить його ідеальним інструментом для створення динамічних та зручних інтерфейсів у вашому веб-додатку. Після введення усіх необхідних даних та параметрів, нова послуга буде доступна для перегляду та запису пацієнтами, що дозволить лікарям/психіатрам ефективно організувати робочий процес та надати максимально якісні медичні послуги своїм клієнтам.

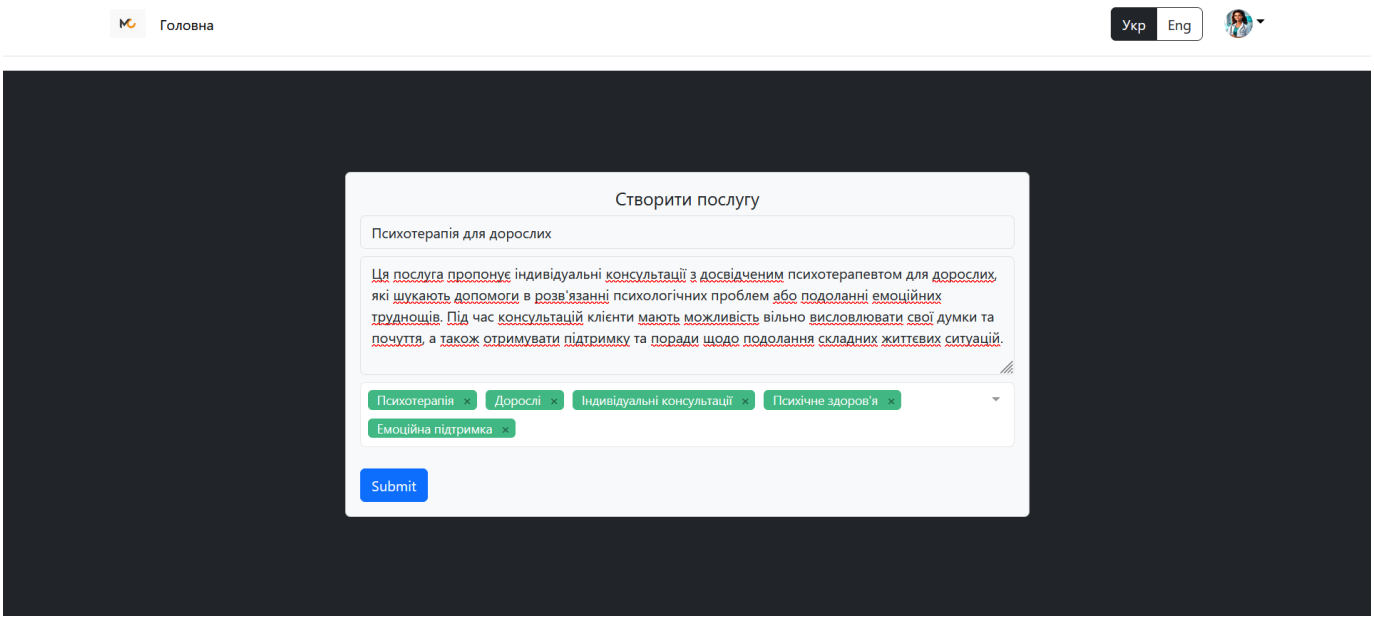


Рисунок 4.7 – Сторінка створення нової послуги психіатра

Після успішного створення відповідна послуга з’явиться на головній сторінці

психіатра. Вигляд створеної послуги наведено на рисунку 4.8.

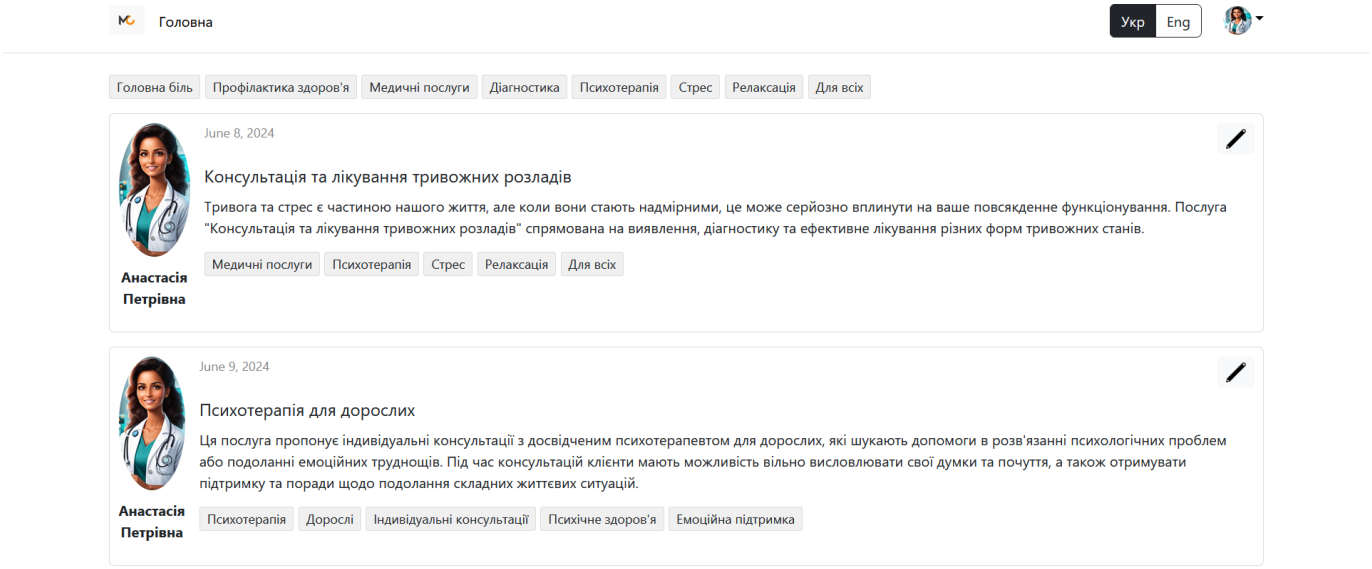


Рисунок 4.8 – Вигляд головної сторінки після створення нової послуги

Крім створення та перегляду послуг, програмна система також надає можливість користувачам швидко знаходити необхідні послуги за допомогою фільтрації за тегами. Це дозволяє знайти послуги, які відповідають конкретним потребам чи інтересам користувача, що забезпечує більш зручний та ефективний пошук. Після вибору певного тегу зі списку, система автоматично фільтрує всі доступні послуги за цим критерієм, відображаючи лише ті, які мають відповідний тег. Цей функціонал дозволяє користувачам швидко зорієнтуватися серед великої кількості послуг та знайти саме ті, які їм потрібні у данний момент.

Програмна система надає можливість коригувати дані про послугу шляхом натискання на кнопку редагування, розташовану в правому верхньому куті карточки послуги. Ця кнопка дозволяє зручно та швидко вносити зміни до інформації про послугу, включаючи назву, опис та теги. Після натискання на кнопку редагування, користувач отримує доступ до форми, де може внести необхідні зміни та підтвердити їх.

Також веб-додаток надає можливість звичайним користувачам додавати певні послуги до списку улюблених для подальшого швидкого доступу шляхом натискання на кнопку у вигляді зірки у правому верхньому куту послуги. Ця функція дозволяє зручно зберігати та організовувати послуги, які сподобалися користувачеві, щоб вони могли швидко знайти їх у майбутньому (рисунок 4.9). Після додавання послуги до списку улюблених, вона стає доступною у спеціальному розділі «Улюблені послуги» (рисунок 4.10), де користувач може швидко знайти та переглянути її. Крім того, користувачі можуть видаляти послуги зі списку улюблених у будь-який момент натиснувши на кнопку у правому верхньому куту послуги у вигляді кошику для сміття, щоб підтримувати актуальність та організованість своїх обранням. Ця можливість допомагає забезпечити комфортне та індивідуалізоване користування веб-додатком, дозволяючи користувачам швидко знаходити та працювати з обраними послугами.

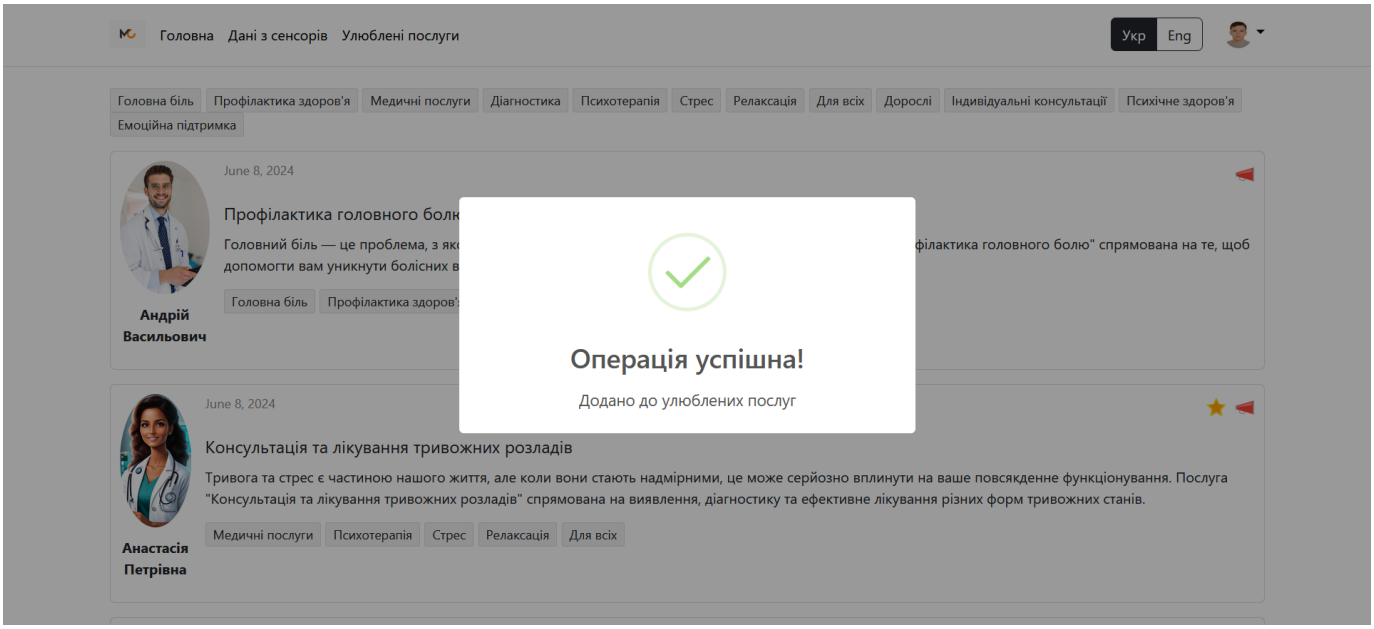


Рисунок 4.9 – Процес додання послуги до улюблених

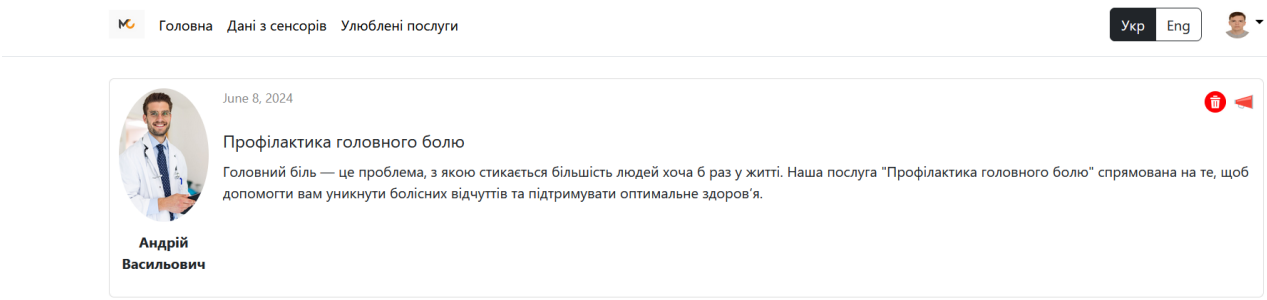


Рисунок 4.10 – Вигляд сторінки улюблених послуг

Дані, що надходять із сенсорів пульсу та шагомеру, відіграють важливу роль у функціонуванні програмної системи. Ці дані відображають стан здоров'я користувачів та їхню фізичну активність в реальному часі. Для заповнення бази даних та створення реалістичної обстановки були використані seeders, які автоматично генерували дані протягом двох років. Кожну годину, seeders створювали рандомні дані пульсу та шагомеру, відображаючи широкий спектр можливих значень цих параметрів. Цей процес дозволяє системі мати доступ до реалістичних даних про стан здоров'я та фізичну активність користувачів, що допомагає в управлінні їхнім благополуччям та фітнес-програмами. На рисунку 4.11 наведено вигляд статистики.

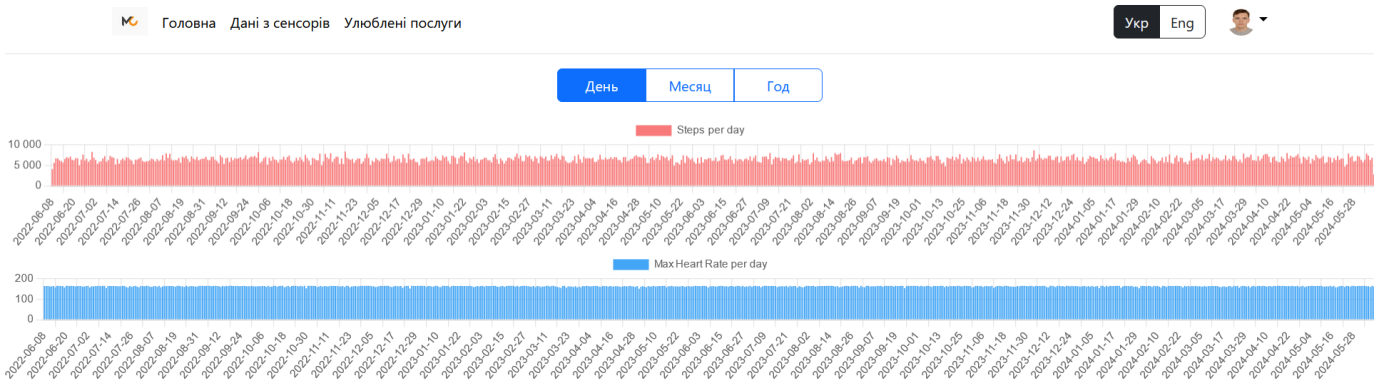


Рисунок 4.11 – Вигляд сторінки даних з сенсорів

У статистиці доступна можливість переглядати дані за різними періодами часу, включаючи день, місяць та рік. Це дозволяє користувачам отримати детальніший аналіз своєї фізичної активності та пульсу в певний період часу. Вони можуть

відстежувати зміни у своєму стані здоров'я та фізичній активності протягом різних часових проміжків, що допомагає їм приймати більш обґрунтовані рішення щодо свого благополуччя та фітнес-цілей.

Для онлайн спілкування в Laravel необхідно виконати команду php artisan queue:work у контейнері mindcare_app. Після цього користувач може натискати на відповідну іконку лікаря або психіатра на сторінці послуги та відкривати з ним мессенджер у реальному часі. У випадку лікаря або психіатра, при натисканні на їхню іконку в послузі, відкривається меню, в якому відображаються різні користувачі, які звернулися до цього фахівця. Користувач може обрати потрібного собі фахівця і розпочати спілкування з ним у мессенджері (рисунок 4.12). Це забезпечує зручну та швидку можливість звернутися до лікаря або психіатра для отримання консультації чи допомоги в режимі реального часу.

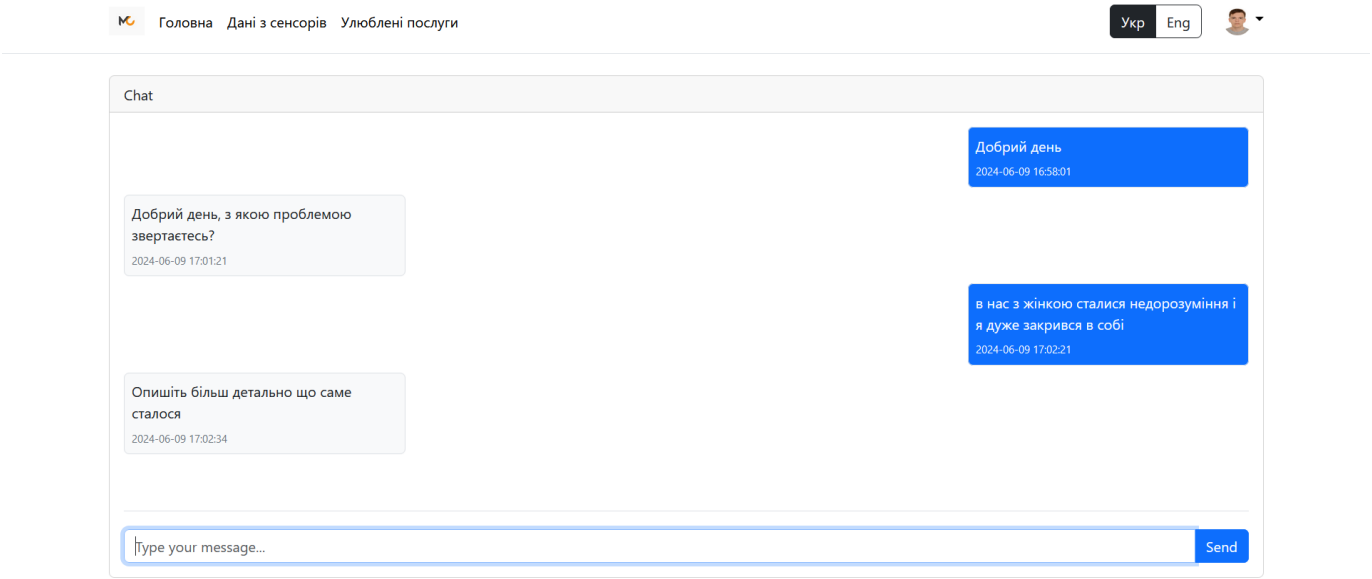


Рисунок 4.12 – Вигляд інтерфейсу месендеру

У випадку лікаря або психіатра, при натисканні на їхню іконку в послузі, відкривається меню, в якому відображаються різні користувачі, які звернулися до цього фахівця, що наведено на рисунку 4.13.

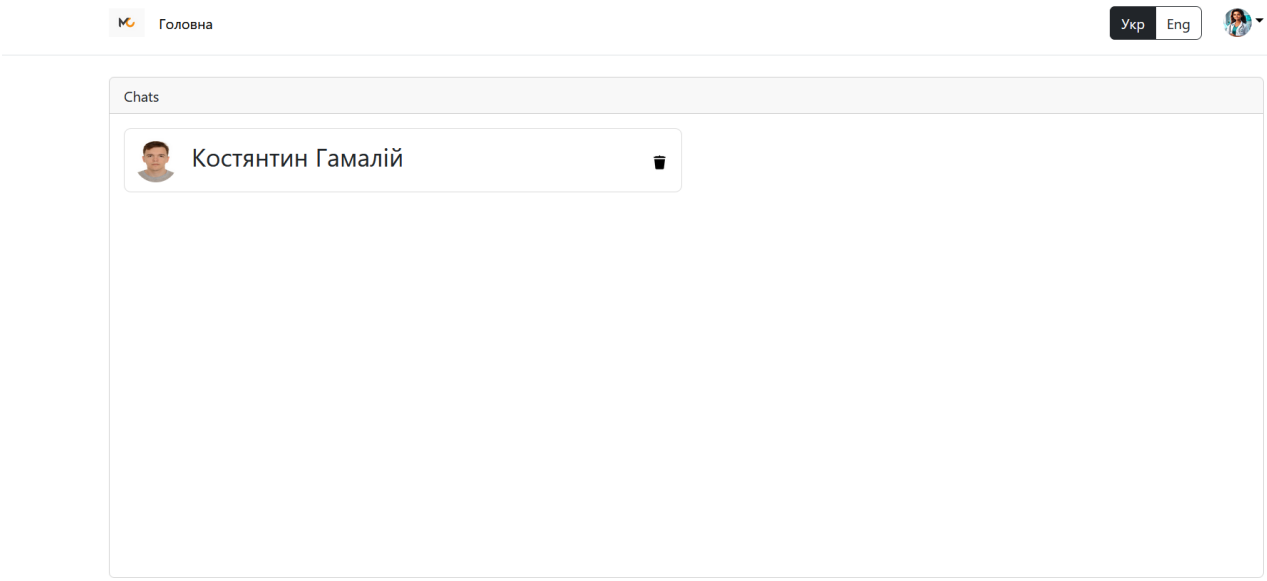


Рисунок 4.13 – Сторінка мессенджу користувачів на цій послузі

Користувач може обрати потрібного собі фахівця і розпочати спілкування з ним у мессенджері. Це забезпечує зручну та швидку можливість звернутися до лікаря або психіатра для отримання консультації чи допомоги в режимі реального часу. Інтерфейс чату у лікаря абог психіатра виглядає так само, як і у користувача на рисунку 4.12.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Обґрунтування вибору видів тестування

Усі ми схильні до помилок, і це нормально. Однак, коли мова йде про програмне забезпечення, навіть маленькі помилки можуть мати серйозні наслідки. Тому тестування стає ключовою складовою процесу розробки програм. Це надійний спосіб переконатися, що наш код працює належним чином і не зламається при внесенні змін.

Сьогодні кодова база програм може бути дуже об'ємною, містити десятки тисяч рядків коду. Переконатися, що нові функціональності не руйнують те, що вже є, - завдання непросте. Важливо, щоб всі частини програми працювали узгоджено, мали коректний функціонал і взаємодіяли між собою без проблем.

Використання тестів допомагає забезпечити стабільність програмного продукту. Вони допомагають переконатися, що все працює належним чином, навіть після внесення змін. Тестування є гарантією безпеки, що нові функції не порушать роботу того, що вже було створено, і дозволяє ефективно додавати новий функціонал без великих зусиль на перевірку старого.

Під час розробки програмної системи критично важливо, щоб всі її компоненти працювали належним чином і взаємодіяли між собою без проблем. Використання тестів допомагає переконатися, що програма працює як очікується і відповідає усім вимогам користувачів.

Тести на Laravel - це потужний інструмент для забезпечення якості програмного забезпечення. Однією з переваг Laravel є вбудована система тестування, яка дозволяє легко створювати, запускати та аналізувати тести для наших додатків. У Laravel використовується PHPUnit - один з найпопулярніших фреймворків для тестування в PHP. З його допомогою можна писати різноманітні тести: від простих функціональних до складних інтеграційних. Laravel також надає вбудовані можливості для симуляції HTTP-запитів, використання бази даних у тестах, підтримки фейків для генерації

тестових даних та інше. Це дозволяє легко та ефективно тестувати різні аспекти додатку, від перевірки логіки до тестування інтерфейсу користувача.

Завдяки вбудованим засобам тестування в Laravel можна швидко створювати та запускати тести, що дозволяє впевнено розвивати додатки, забезпечуючи їхню стабільність і надійність.

5.2 Опис тестування серверної частини системи

У Laravel тести пишуться у спеціальній папці з назвою "tests". Для створення нового тесту використовується команда "php artisan make NameTest", яка автоматично створює файл тесту з вказаною назвою. Потім тести можна запустити за допомогою команди "php artisan test test/Feature/NameTest.php", яка виконує всі тести, розміщені у файлі "NameTest.php" у папці "Feature".

Ось приклад структури тесту:

```
class DoctorTest extends TestCase
{
    use RefreshDatabase;
    use WithFaker;

    public function testCreateDoctorAndServices()
    {
        $doctor = Doctor::factory()->create();

        for ($i = 0; $i < 100; $i++) {
            Service::factory()->create([
                'doctor_id' => $doctor->id,
            ]);
        }

        $this->assertCount(100, $doctor->services);
    }
}
```

Цей тест кейс створений для перевірки функціоналу створення докторів та послуг у системі. Він включає в себе ряд кроків для перевірки правильності процесу створення доктора та його послуг.

Спочатку, використовуючи функціонал фабрики моделей, створюється доктор за допомогою методу "Doctor::factory()->create()". Цей метод автоматично створює об'єкт доктора з випадковими даними, що допомагає забезпечити реалістичність тесту.

Далі, за допомогою циклу, який повторюється сто разів, створюється сто послуг для цього доктора. Кожна послуга також створюється за допомогою фабрики моделей, і їй автоматично призначається ідентифікатор доктора за допомогою поля 'doctor_id', яке вказує на ідентифікатор поточного доктора.

Після створення ста послуг для доктора, тест перевіряє кількість послуг, що належать цьому доктору, за допомогою методу "assertCount". Цей метод перевіряє, чи кількість послуг, повернених зворотньо, дорівнює ста, що було створено в цьому тесті.

Цей тест допомагає переконатися, що процес створення докторів та їх послуг працює належним чином і що всі послуги, створені для конкретного доктора, дійсно зберігаються в базі даних і можуть бути отримані за допомогою відповідних методів.

Давайте тепер зробимо тестові кейси для перевірки відправлення повідомлень між користувачем та лікарем у нашій системі медичних консультацій.

Почнемо з перевірки відправлення повідомлення від користувача до лікаря:

```
class testSendMessageFromUserToDoctor extends TestCase
{
    use RefreshDatabase;

    public function testSendMessageFromUserToDoctor()
    {
        $user = User::factory()->create();
        $doctor = Doctor::factory()->create();

        $messageContent = "Це тестове повідомлення від користувача до
доктора.";
        Chat::create([
            'user_id' => $user->id,
            'doctor_id' => $doctor->id,
            'message' => $messageContent,
```

```
            'message_date' => now(),
            'sender' => 'user',
        ]);

        $this->assertDatabaseHas('chats', [
            'user_id' => $user->id,
            'doctor_id' => $doctor->id,
            'message' => $messageContent,
            'sender' => 'user',
        ]);
    }
}
```

У цьому тесті ми спочатку створюємо об'єкти користувача та лікаря, які будуть учасниками чату. Потім ми відправляємо тестове повідомлення від користувача до лікаря через модель Chat. Після цього ми перевіряємо, чи відображається це повідомлення в базі даних з відповідними параметрами, такими як ID користувача та лікаря, текст повідомлення та його відправник.

Тепер перейдемо до перевірки відправлення повідомлення від лікаря до користувача:

```
class testSendMessageFromDoctorToUser extends TestCase
{
    use RefreshDatabase;

    public function testSendMessageFromDoctorToUser()
    {
        $user = User::factory()->create();
        $doctor = Doctor::factory()->create();

        $messageContent = "Це тестове повідомлення від лікаря до
користувача.";
        Chat::create([
            'user_id' => $user->id,
            'doctor_id' => $doctor->id,
            'message' => $messageContent,
            'message_date' => now(),
            'sender' => 'doctor',
        ]);

        $this->assertDatabaseHas('chats', [
            'user_id' => $user->id,
            'doctor_id' => $doctor->id,
            'message' => $messageContent,
            'sender' => 'doctor',
        ]);
    }
}
```

У цьому тесті ми виконуємо аналогічні кроки, але відправляємо тестове повідомлення від лікаря до користувача. Після цього ми також перевіряємо, чи відображається це повідомлення в базі даних з відповідними параметрами, такими як ID користувача та лікаря, текст повідомлення та його відправник.

Обидва ці тестові кейси допоможуть переконатися, що механізм відправлення повідомлень працює правильно, а інформація коректно зберігається в базі даних і може бути отримана при потребі для подальшої обробки або відображення у користувацькому інтерфейсі.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Наукове впровадження проекту

На основі тематики кваліфікаційної роботи була написані тези для XXXII Міжнародної науково-практичної конференції "Інформаційні технології: Наука, техніка, технологія, освіта, здоров’я", який проходив з 22 по 25 травня 2024 року в онлайн-режимі на базі ХНУРЕ.

Тези під назвою " ПРОГРАМНА СИСТЕМА ДЛЯ ПІДТРИМКИ ПСИХІЧНОГО ЗДОРОВ’Я" була опублікована у відповідному збірнику конференції [6].

У цій публікації представлені ключові аспекти програмної системи, розробленої для підтримки психічного здоров'я. Система враховує унікальні потреби кожного користувача і забезпечує комплексний підхід до моніторингу психологічного стану. Вона включає моніторинг фізіологічних показників за допомогою сучасних технологій, таких як датчики, що відстежують серцевий ритм, рівень стресу та якість сну, дозволяючи аналізувати динаміку змін у психічному стані. Також система надає користувачам можливість спілкуватися з кваліфікованими психологами та консультантами через вбудований чат або відеозв'язок, отримуючи індивідуальні поради та підтримку. Крім того, вона створює платформу для молодих фахівців або тих, хто шукає допомогу, дозволяючи їм знаходити досвідчених психологів для наставництва або консультацій, що сприяє обміну знаннями та досвідом.

ВИСНОВКИ

У ході розробки програмної системи для підтримки психічного здоров'я було зроблено важливі кроки для створення функціонального та ефективного інструменту. Веб-клієнт системи, розроблений на базі фреймворка Vue, забезпечує зручний та інтерактивний інтерфейс для користувачів, дозволяючи їм легко взаємодіяти з функціоналом системи. Зв'язок між клієнтською та серверною частинами системи забезпечується за допомогою протоколу HTTPS, що гарантує безпечну передачу даних.

Використання Docker дозволяє забезпечити незалежність від середовища розгортання та забезпечити легкість у розгортанні та масштабуванні системи. База даних MySQL використовується для зберігання та управління даними користувачів, забезпечуючи надійність та швидкий доступ до інформації.

Для забезпечення гнучкості та ефективності взаємодії з користувачами системи використовуються принципи SOLID, а також допоміжні компоненти, які спрощують розробку та підтримку програмного забезпечення. Бібліотека i18n дозволяє легко локалізувати систему на різні мови та забезпечує зручність у користуванні для користувачів з різних країн.

У цілому, розроблена програмна система відповідає вимогам сучасного ринку програмного забезпечення, забезпечуючи зручність, безпеку та ефективність у використанні для користувачів та забезпечуючи стабільність та надійність для розробників.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. American psychiatric association – URL: https://www.psychiatry.org/ (дата звернення: 17.01.2024).

2. Mental Health America – URL: https://www.nami.org/home (дата звернення: 17.01.2024).

3. Overview of REST API specification formats [Електронний ресурс] – URL: https://idratherbewriting.com/learnapidoc/pubapis_rest_specification_formats.html (дата звернення: 26.05.2022).

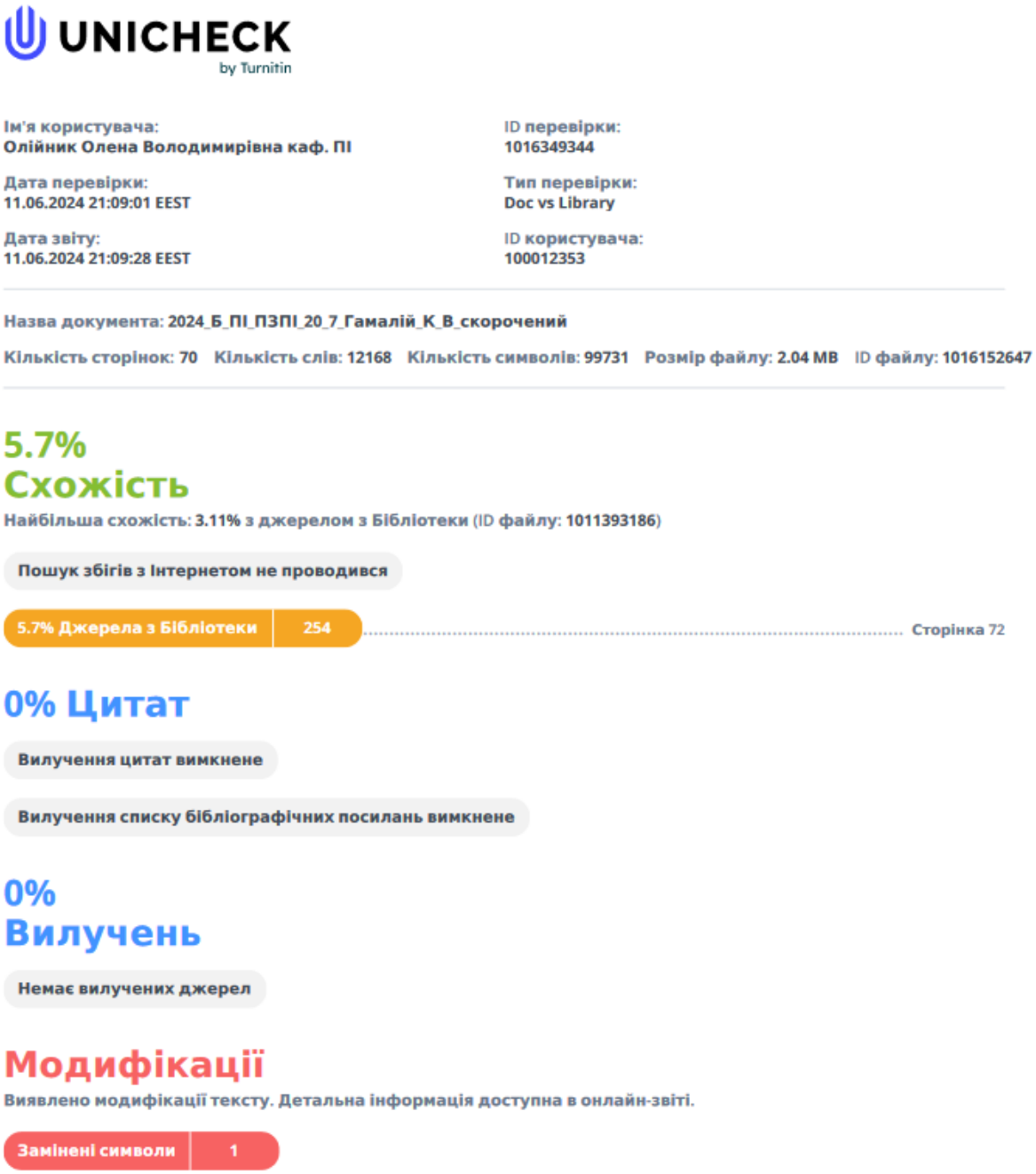
4. CSRF (Cross Site Request Forgery) [Електронний ресурс] – URL: https://book.hacktricks.xyz/pentesting-web/csrf-cross-site-request-forgery (дата звернення: 26.05.2022).

5. Бондаренко В. Г. MySQL. Вільямс, 2004. 1056 стр

6. Конференція «Інформаційні технології: наука, техніка, технологія, освіта, здоров'я (MicroCAD-2024)». URL: https://web.kpi.kharkov.ua/microcad/ (дата звернення: 16.05.2024)

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016349344

Дата перевірки:
11.06.2024 21:09:01 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
11.06.2024 21:09:28 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІ_20_7_Гамалій_К_В_скорочений

Кількість сторінок: 70 Кількість слів: 12168 Кількість символів: 99731 Розмір файлу: 2.04 MB ID файлу: 1016152647

5.7%
Схожість

Найбільша схожість: 3.11% з джерелом з Бібліотеки (ID файлу: 1011393186)

Пошук збігів з Інтернетом не проводився

5.7% Джерела з Бібліотеки 254 ..... Сторінка 72

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 1

Рисунок А.1 – Результат перевірки кваліфікаційної роботи бакалавра на плагіат

ДОДАТОК Б

Код файлу маршрутизації серверної частини програми

```
<?php

use App\Http\Controllers\Api\ChatController;
use App\Http\Controllers\Api\SavedServiceController;
use App\Http\Controllers\Api\ServiceController;
use App\Http\Controllers\Api\UserController;
use App\Http\Controllers\Api\RoleController;
use App\Http\Controllers\Api\TagController;
use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "api" middleware group. Make something great!
|
*/

Route::post('/auth/register', [UserController::class, 'createUser'])-
>name('register');
Route::post('/auth/registerDoctor', [UserController::class, 'createDoctor'])-
>name('register_doctor');
Route::post('/auth/login', [UserController::class, 'loginUser'])-
>name('login');
Route::post('/auth/loginDoctor', [UserController::class, 'loginDoctor'])-
>name('loginDoctor');
Route::post('/auth/logout', [UserController::class, 'logout'])-
>name('logout');
Route::post('/roles/store', [RoleController::class, 'store']);

Route::group(['middleware' => 'auth:sanctum'], function () {
    Route::get('/auth/getUsers', [UserController::class, 'getUsers']);
    Route::get('/auth/getUser/{id}', [UserController::class, 'getUser']);
    Route::get('/auth/getDoctor/{id}', [UserController::class, 'getDoctor']);
    Route::post('/auth/user/update/{id}', [UserController::class,
'updateUser']);
    Route::post('/auth/doctor/update/{id}', [UserController::class,
'updateDoctor']);
    Route::delete('/auth/user/destroy/{id}', [UserController::class,
'destroyUser']);
    Route::delete('/auth/doctor/destroy/{id}', [UserController::class,
'destroyDoctor']);

    Route::get('/tags', [TagController::class, 'index']);
    Route::get('/tags/{id}', [TagController::class, 'getTag']);
```

```
    Route::post('/tags/store', [TagController::class, 'store']);
    Route::post('/tags/update/{id}', [TagController::class, 'update']);
    Route::delete('/tags/destroy/{id}', [TagController::class, 'destroy']);

    Route::get('/services', [ServiceController::class, 'index']);
    Route::get('/services/{id}', [ServiceController::class,
'indexByDoctor']);
    Route::get('/services/get/{id}', [ServiceController::class,
'getService']);
    Route::get('/services/favorite/{id}', [ServiceController::class,
'getFavoriteServices']);
    Route::delete('/services/favorite/destroy/{serviceId}/{userId}',
[ServiceController::class, 'destroyFavoriteService']);
    Route::post('/services/store', [ServiceController::class, 'store']);
    Route::put('/services/report/{id}', [ServiceController::class,
'setReport']);
    Route::post('/services/update/{id}', [ServiceController::class,
'update']);
    Route::delete('/services/destroy/{id}', [ServiceController::class,
'destroy']);

    Route::post('/savedService/store', [SavedServiceController::class,
'store']);

    Route::get('/chat', [ChatController::class, 'show']);
    Route::get('/chat/{serviceId}/{doctorId}', [ChatController::class,
'getDoctorChats']);
    Route::post('/chat/store', [ChatController::class, 'store']);
    Route::post('/chat/destroy', [ChatController::class, 'destroy']);

    Route::get('/roles', [RoleController::class, 'index']);
    Route::get('/roles/{id}', [RoleController::class, 'getRole']);
    Route::put('/roles/update/{id}', [RoleController::class, 'update']);
    Route::delete('/roles/destroy/{id}', [RoleController::class, 'destroy']);

    Route::get('/chat/{doctor_id}/{user_id}/{service_id}',
[ChatController::class, 'index']);
    Route::post('/chat/send/{doctor_id}/{user_id}', [ChatController::class,
'send']);
});
```

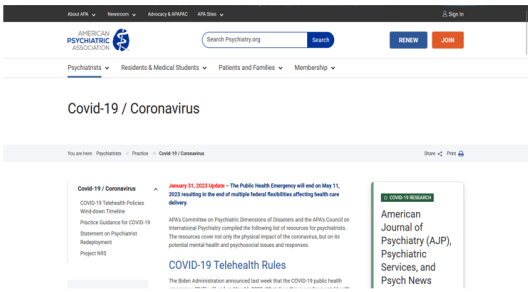
ДОДАТОК В

Слайди презентації

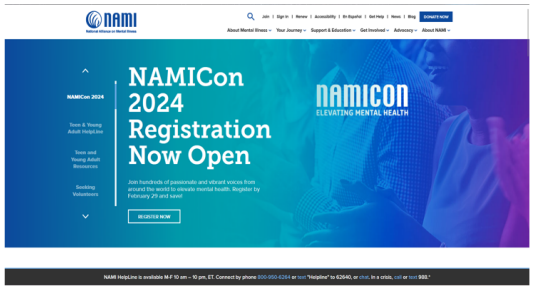
Рисунок В.1 – Слайд 1

Рисунок В.2 – Слайд 2

Аналоги



- Американська психіатрична асоціація



- Mental Health America

3

Рисунок В.3 – Слайд 3

Актуальність

Зі зростанням потреби у швидкому та зручному доступі до медичних послуг, особливо під час глобальних криз, таких як пандемія, онлайн-платформи стають важливим інструментом для забезпечення безперервного медичного обслуговування. Віддалений доступ до консультацій, можливість відстежувати стан здоров'я та отримувати своєчасні рекомендації від лікарів знижують навантаження на традиційні медичні заклади і сприяють кращій профілактиці захворювань. Інтеграція з сенсорами для моніторингу фізичних показників дозволяє забезпечити персоналізований підхід до кожного пацієнта

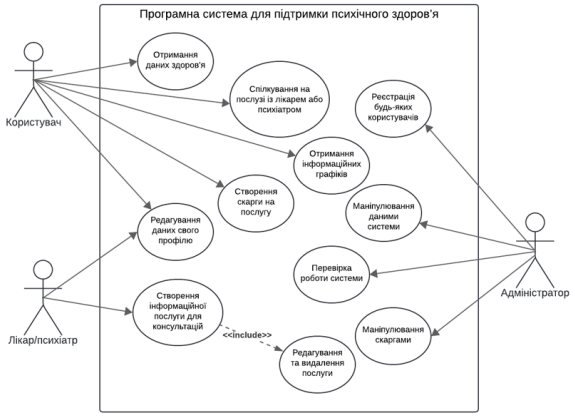
4

Рисунок В.4 – Слайд 4

Рисунок В.5 – Слайд 5

Рисунок В.6 – Слайд 6

Архітектура та проектування програмної системи

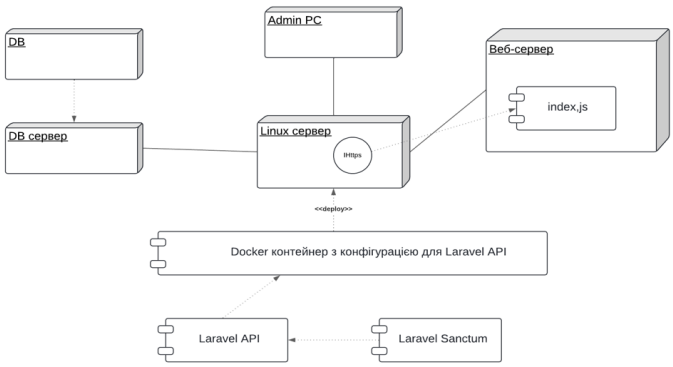


Діаграма варіантів використання програмної системи

7

Рисунок В.7 – Слайд 7

Архітектура та проектування програмної системи



Діаграма розгортання та загальна архітектура системи

8

Рисунок В.8 – Слайд 8

Рисунок В.9 – Слайд 9

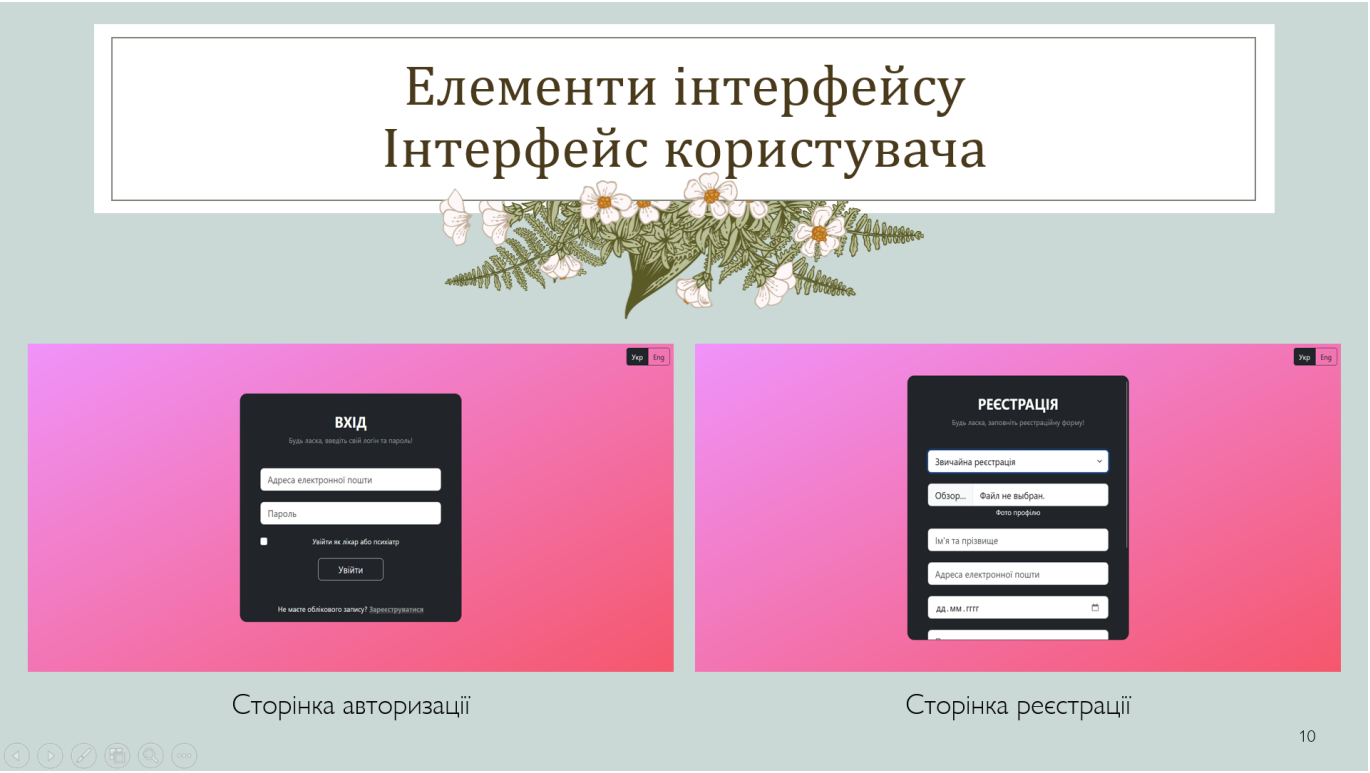


Рисунок В.10 – Слайд 10

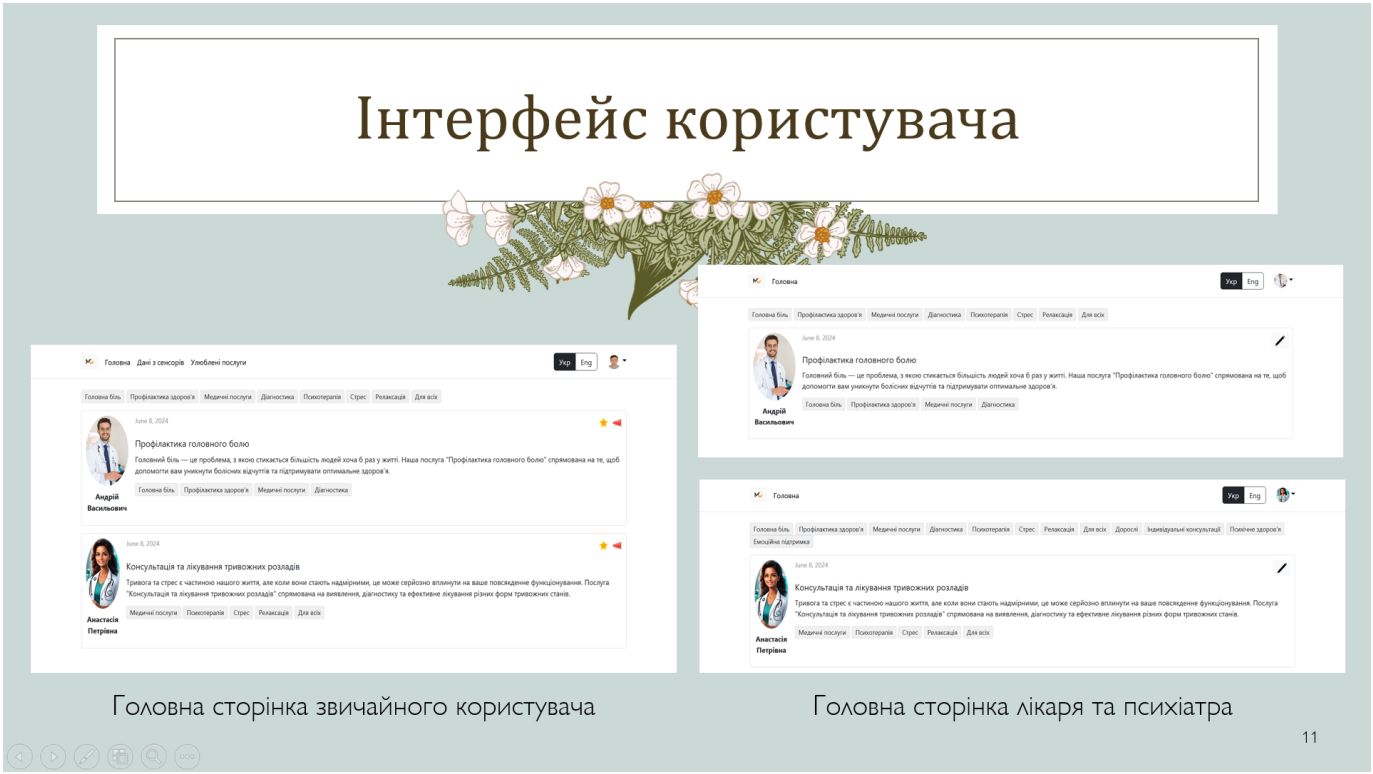


Рисунок В.11 – Слайд 11

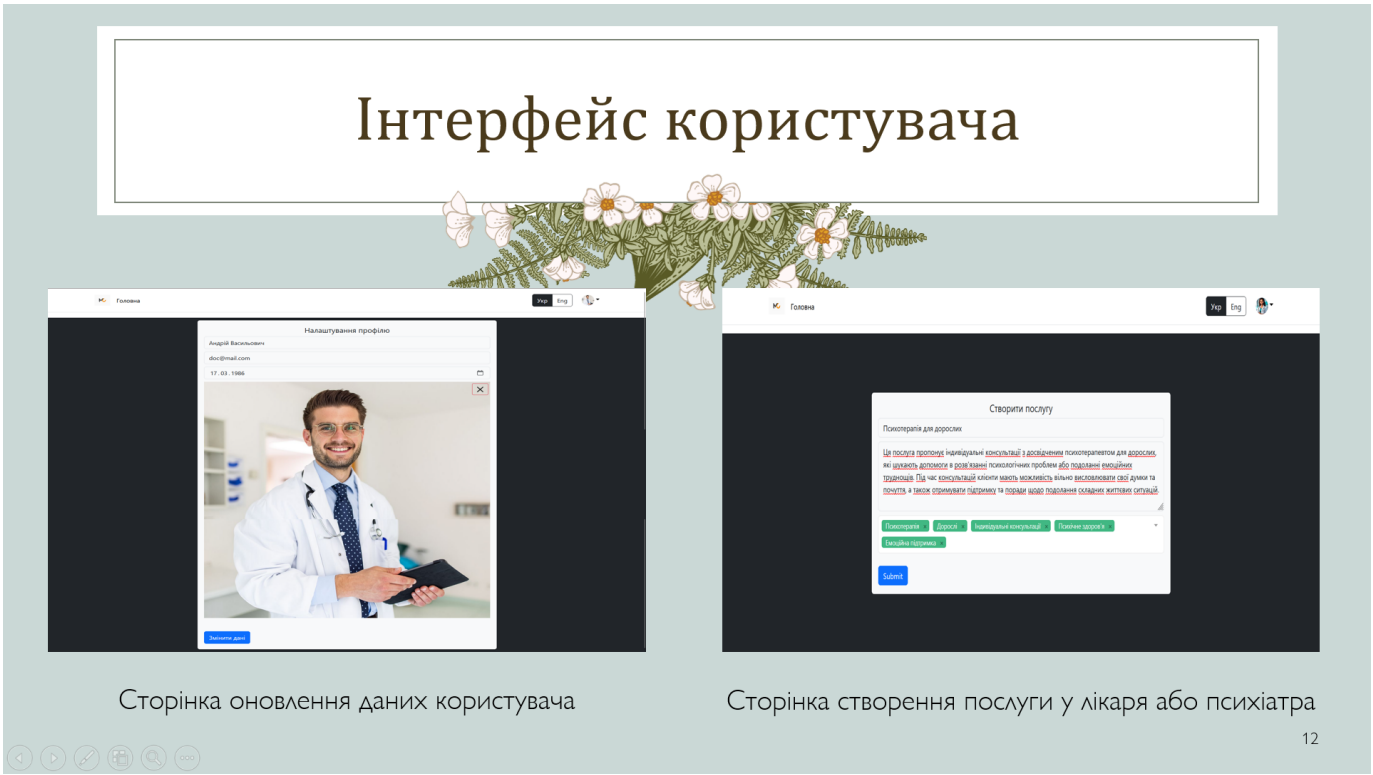


Рисунок В.12 – Слайд 12

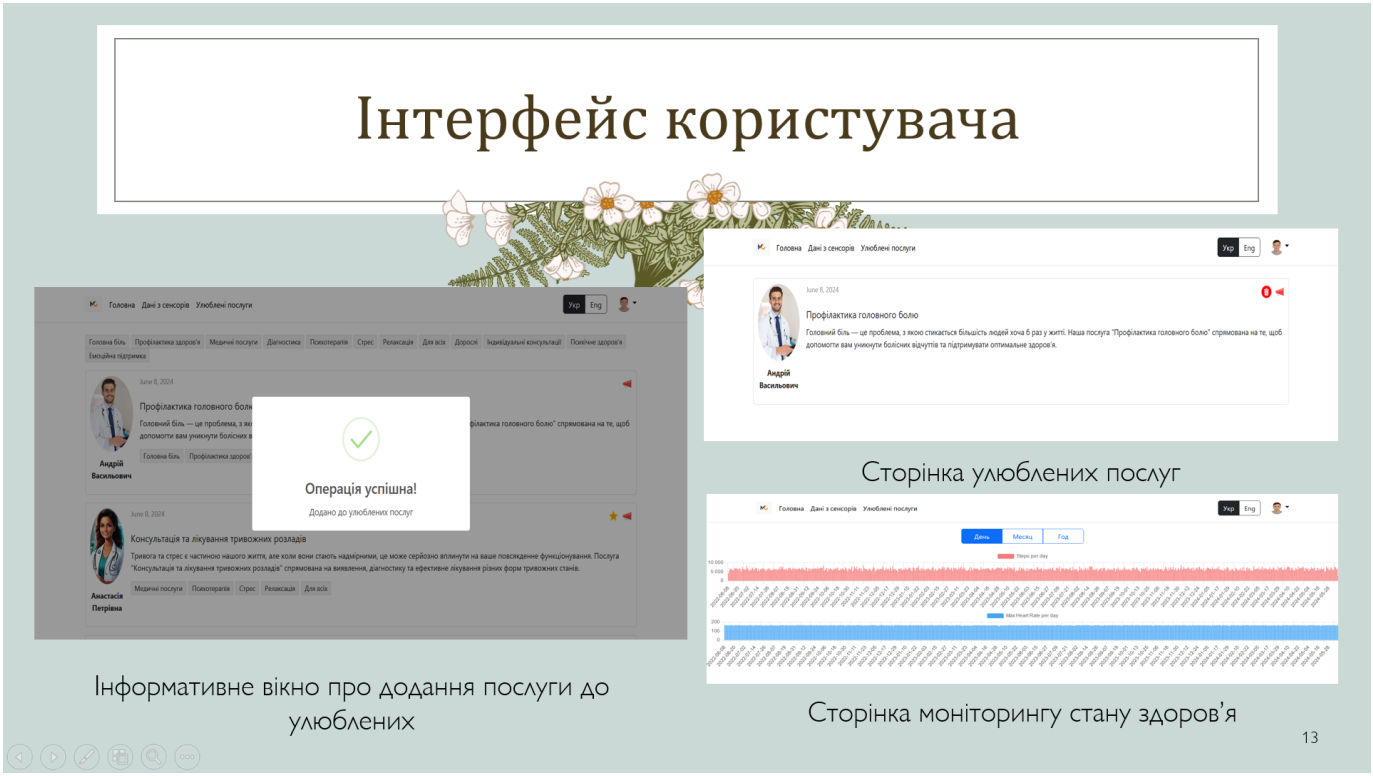


Рисунок В.13 – Слайд 13

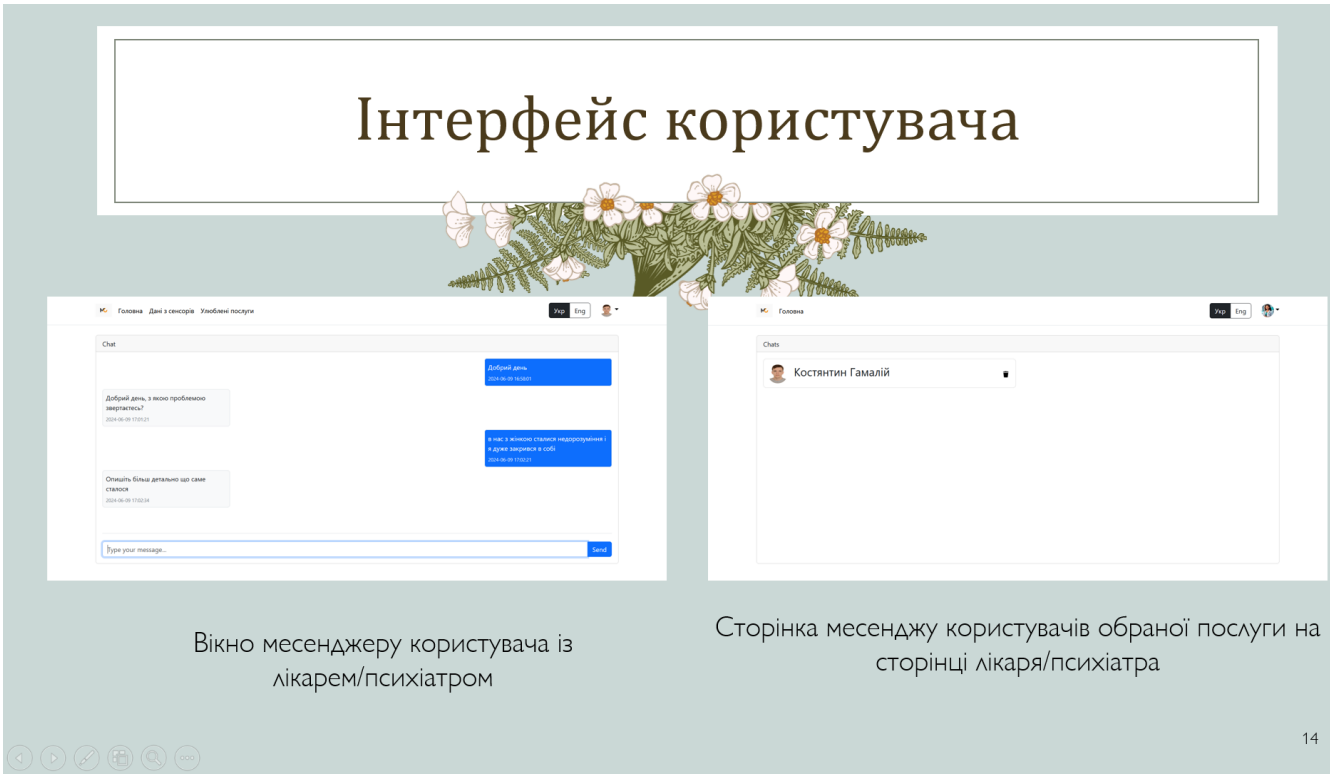


Рисунок В.14 – Слайд 14

Тестування

Однією з переваг Laravel є вбудована система тестування, яка дозволяє легко створювати, запускати та аналізувати тести для нашого додатку. У Laravel використовується PHPUnit - один з найпопулярніших фреймворків для тестування в PHP. З його допомогою можна писати різноманітні тести: від простих функціональних до складних інтеграційних. Laravel також надає вбудовані можливості для симуляції HTTP-запитів, використання бази даних у тестах, підтримки фейків для генерації тестових даних та інше. Це дозволяє легко та ефективно тестувати різні аспекти додатку, від перевірки логіки до тестування інтерфейсу користувача.

15

Рисунок В.15 – Слайд 15

Висновки

В результаті виконання роботи було спроєктовано та створено програмну систему, що:

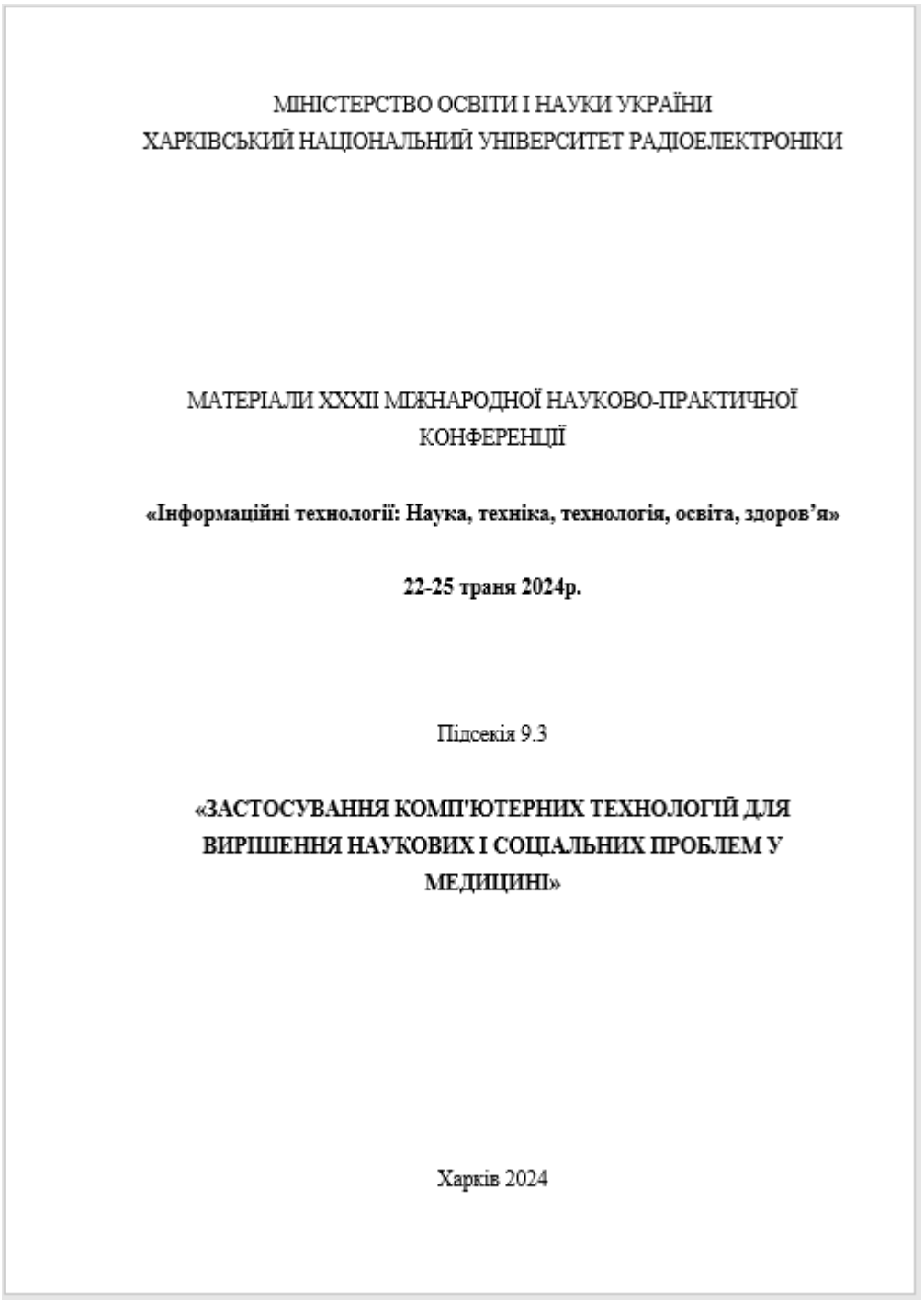
- Сприяє ефективній взаємодії між користувачами та медичними працівниками
- Забезпечує зручне управління медичними послугами
- Надає можливість оцінки та збереження медичних послуг
- Підтримує обмін повідомленнями між пацієнтами та лікарями
- Інтегрує функції контролю стану здоров'я
- Забезпечує масштабованість та стабільність роботи

16

Рисунок В.16 – Слайд 16

ДОДАТОК Г

Тези доповіді для науково-практичної інтернет-конференції



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МАТЕРІАЛИ XXXII МІЖНАРОДНОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ

«Інформаційні технології: Наука, техніка, технологія, освіта, здоров'я»

22-25 траня 2024р.

Підсекія 9.3

«ЗАСТОСУВАННЯ КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ ДЛЯ ВИРІШЕННЯ НАУКОВИХ І СОЦІАЛЬНИХ ПРОБЛЕМ У МЕДИЦИНІ»

Харків 2024

Рисунок Г.1 – обкладинка збірника

Інформаційні технології: наука, техніка, технологія, освіта, здоров'я. MicroCAD-2024

ПРОГРАМНА СИСТЕМА ДЛЯ ПІДТРИМКИ ПСИХІЧНОГО ЗДОРОВ'Я

Гамалій К. В., Зибіна К. В.
Національний університет
«Харківський національний університет радіоелектроніки», м. Харків

Психічне здоров'я визначається не лише відсутністю психічних розладів, а й загальним самопочуттям та здатністю справлятися зі стресом. Наприклад, під час війни або конфлікту люди зазнають значного психологічного тиску, що може призвести до появи психічних проблем.

Для розв'язання завдання підтримки психічного здоров'я людини, була розроблена програмна система, що враховує унікальні потреби кожного користувача. У цьому проекті була розроблена система для комплексного моніторингу психологічного стану користувачів, яка включає наступні етапи:

- Моніторинг за психологічним станом за допомогою підключених датчиків: система використовує сучасні технології для безперервного відстеження фізіологічних показників, таких як серцевий ритм, рівень стресу, якість сну та інші. Це дозволяє не лише виявляти можливі зміни у психологічному стані, але й аналізувати їх динаміку та тренди з часом;
- Отримання способу спілкування із досвідченими фахівцями та отримання вказівок щодо усунення проблем: користувачі мають можливість спілкуватися з кваліфікованими психологами та консультантами через вбудований чат або відеодзвінок. Це дає їм можливість отримувати індивідуальні поради та підтримку у вирішенні своїх проблем;
- Можливість початковим фахівцям знайти досвід: система також надає можливість молодим спеціалістам або тим, хто шукає допомогу, знайти досвідчених психологів і консультантів для наставництва або консультації. Це створює мережу взаємодопомоги та сприяє обміну знаннями і досвідом;

Психічне здоров'я не обмежується лише відсутністю психічних розладів, воно також включає загальне самопочуття та здатність ефективно впоратися зі стресом. У ситуаціях конфлікту чи емоційного тиску, люди можуть відчувати потребу в підтримці та ресурсах для покращення свого психічного стану.

Розроблена програмна система позитивно вплине на психічне становище користувачів, допоможе їм ефективніше справлятися зі стресом та покращить загальний рівень психічного здоров'я в суспільстві.

Рисунок Г.2 – матеріал тез