

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МОНІТОРИНГУ

(назва теми)

ПСИХІЧНОГО ЗДОРОВ'Я

MENTAL HEALTH MONITORING SOFTWARE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-220сп

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ВАВРИК І.В.

(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
« » 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

БАВРИКА Іллі Васильовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення моніторингу психічного здоров'я. Mental Health Monitoring Software

керівник проєкту (роботи) к.т.н., доцент СКРУПСЬКИЙ Степан Юрійович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «04» квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 1 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Реалізація програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	БЕЛОВА А.В., асистент		

7. Дата видачі завдання « 4 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

_____ Ілля ВАВРИК
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Степан СКРУПСЬКИЙ
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 95 с., 25 рис., 2 табл., 3 дод., 10 джерел.

ПСИХОЛОГІЧНЕ ЗДОРОВ'Я, ПСИХОТЕРАПІЯ, НАСТРІЙ, ЕМОЦІЇ, КОНСУЛЬТУВАННЯ ПАЦІЄНТА, ПРОГРАМНА СИСТЕМА.

Об'єктом роботи є процес моніторингу психічного здоров'я людини. Предметом роботи є програмне забезпечення моніторингу психічного здоров'я. Мета роботи – розробити програмне забезпечення моніторингу психічного здоров'я для надання допомоги людині у виявленні та розв'язанні психологічних проблем.

Матеріали, методи та технічні засоби: мови Python, HTML, CSS, JavaScript, система керування базами даних MySQL.

Результати. До результатів роботи слід віднести і саму реалізовану програмну систему, і результати аналізу предметної області, де було визначено функції подібних програм і визначено відсутність взаємодії всередині спільноти, наявність платних засобів для аналізу, та результати проєктування програми.

Висновки. Розроблена програма направлена на підтримку фіксації настрою, станів, діяльності пацієнта, аналіз зміни настрою, власного стану індивідуально самим пацієнтом, моніторингу настроїв у спільноті та на базову підтримку людей, які потребують допомоги, лікарями. Програма орієнтована на те, щоб люди, які зіштовхуються з психологічними проблемами, могли їх вчасно виявити, на ранніх стадіях, та зрозуміли, що вони потребують професійної підтримки.

Галузь використання – індивідуальне використання, психічне здоров'я, психотерапія.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 95 pages, 25 figures, 2 tables, 3 appendixes, 10 sources.

MENTAL HEALTH, PSYCHOTHERAPY, MOOD, EMOTIONS, PATIENT COUNSELING, SOFTWARE SYSTEM.

The object of the work is the process of monitoring a person's mental health. The subject of the work is mental health monitoring software. The goal of the work is to develop mental health monitoring software to help people identify and solve psychological problems.

Materials, methods and technical means: Python, HTML, CSS, JavaScript languages, MySQL database management system.

The results. The results of the work include the implemented software system itself, and the results of the analysis of the subject area, where the functions of similar programs were determined and the lack of interaction within the community, the availability of paid tools for analysis, and the results of program design were determined.

Conclusions. The developed program is aimed at supporting the fixation of the patient's mood, conditions, activity, analysis of mood changes, own condition individually by the patient himself, monitoring of moods in the community and basic support of people in need of help by doctors. The program is aimed at ensuring that people who face psychological problems can detect them in time, in the early stages, and understand that they need professional support.

Field of use – individual use, mental health, psychotherapy.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	10
1.1 Аналіз програмних засобів моніторингу психічного здоров'я	10
1.2 Визначення концепції розроблюваної програми	13
1.3 Висновки за розділом 1 та постановка завдань роботи	14
2 Розробка архітектури програми.....	15
2.1 Вибір технологій розробки	15
2.2 Моделювання вимог до програми	15
2.3 Проєктування бази даних	20
2.4 Висновки за розділом 2	31
3 Реалізація програми	32
3.1 Порядок функціонування програми.....	32
3.2 Програмна реалізація вебсистеми	34
3.3 Висновки за розділом 3	43
4 Експлуатація та тестування програми	44
4.1 Призначення програми	44
4.2 Умови виконання програми	44
4.3 Виконання і тестування програми.....	44
4.4 Висновки за розділом 4	49
Висновки	50
Перелік джерел посилення	51
Додаток А Технічне завдання	52
Додаток Б Текст програми	57
Додаток В Слайди презентації.....	89

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ПЗ – програмне забезпечення.

ВСТУП

Одразу потрібно чітко окреслити, що за визначеннями Всесвітньої організації здоров'я психічне здоров'я є достатньо широким поняттям, яке включає в себе як відсутність психічних розладів, так і комплекс інших факторів, які визначають стан людини, коли вона здатна розвивати і реалізувати свої здібності, застосовувати ефективні стратегії подолання повсякденних стресів, реалізовувати свій власний зокрема професійний потенціал, сприяти розвитку громади [1]. Тож психічне здоров'я є широким поняттям, на яке впливає велика кількість факторів, які якщо не враховувати, можна цілісно погіршити життя людини у всіх його складових. З іншого боку через незнання, відсутність інформації про проблеми, невміння використовувати психологічні практики величезна кількість людей в Україні може мати наслідки, які в подальшому буде дуже складно подолати.

Поточна ситуація з психічним здоров'ям людей в Україні відповідає ситуації в інших регіонах, які постраждали від тривалих конфліктів і війни: кожна п'ята (приблизно 22 %) людина, яка пережила війну чи інший конфлікт впродовж останніх 10 років, має тривогу, депресію, посттравматичний стресовий розлад, біполярний розлад або шизофренію, тому за оцінками Всесвітньої організації охорони здоров'я приблизно 9,6 мільйонів людей в Україні можуть мати проблеми з психічним здоров'ям, що є величезною цифрою [2]-[3].

Зрозуміло, що на даний момент ця ситуація вже фактично сформувалася, але відслідкувати її є проблемним, поки війна триває. Тому в більшості випадків люди залишаються з власними проблемами один на один, а тому потребують допомоги в цьому процесі. Сучасні комп'ютерні технології сприяють тому, щоб застосовуватися і для цього процесу.

У даній роботі запропоновано розробити програмне забезпечення (ПЗ) моніторингу психічного здоров'я для надання допомоги людині у виявленні та розв'язанні психологічних проблем. Це є настільки важливим зокрема

через те, що моніторинг психічного здоров'я дозволяє знаходити тригери, які приводять до проблем у людини, за рахунок фіксування станів здобувати контроль над власним психічним здоров'ям, продовжувати власні дії, створюючи стратегію, що неможливо без відслідковування власних станів, надавати інформацію лікарю [4].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз програмних засобів моніторингу психічного здоров'я

Початковий аналіз дозволив встановити, що зараз існує ряд додатків, які можуть використовуватися для моніторингу психічного здоров'я. Вони мають певні спільні інструменти, але і свої особливості також. Акцент на цьому і зроблений під час аналізу кожного додатку окремо з виділених у розділі 1.

Moodfit [5]-[6] (рис. 1.1) – додаток, який використовується для відслідковування власних станів за настроєм, збереження, аналізу даних.

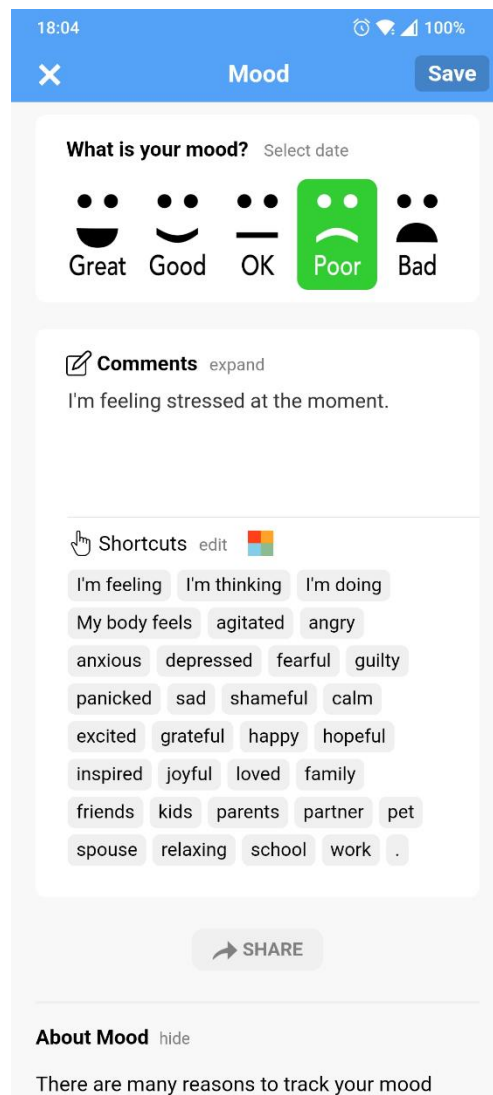


Рисунок 1.1 – Додаток Moodfit

Moodfit має платні функції, зокрема пов'язані з аналізом станів. Наприклад, для перегляду зміни настрою і пов'язаних з цим змін у видах діяльності (рис. 1.2) потрібна сплата. Тобто фактично в умовах, коли в Україні багато людей мають проблеми з психічним здоров'ям і могли би на початкових етапах допомогти собі у виявленні причин та пов'язаних проблем, це є обмеженням і не сприяє масовій підтримці.

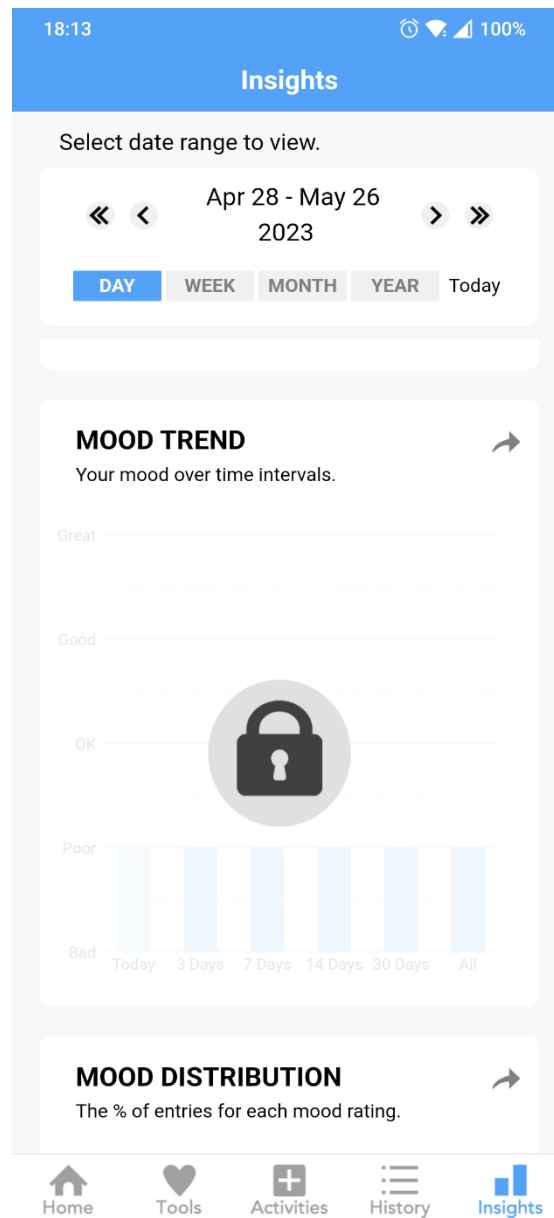


Рисунок 1.2 – Платні можливості в додатку Moodfit

Програма дозволяє фіксувати власний настрій за 5-рівневою шкалою, вносити діяльність, яку виконував користувач, отримати інструкції щодо

певних видів діяльності, вносити власні записи про результати діяльності, встановлювати власні цілі на день, переглядати історію виконаних запитів користувачем, аналізувати отримані дані. Але з усіх засобів аналізу тільки виведення за допомогою графіку зміни настрою є безкоштовним. Також програма дозволяє пройти тести для визначення власного стану.

Daylio [7]-[8] (рис. 1.3) є прикладом ще одного подібного додатку. Базовий ланцюжок роботи дозволяє користувачу визначати свій настрій за 5-бальною шкалою і тут же визначити діяльність, яку виконував користувач.

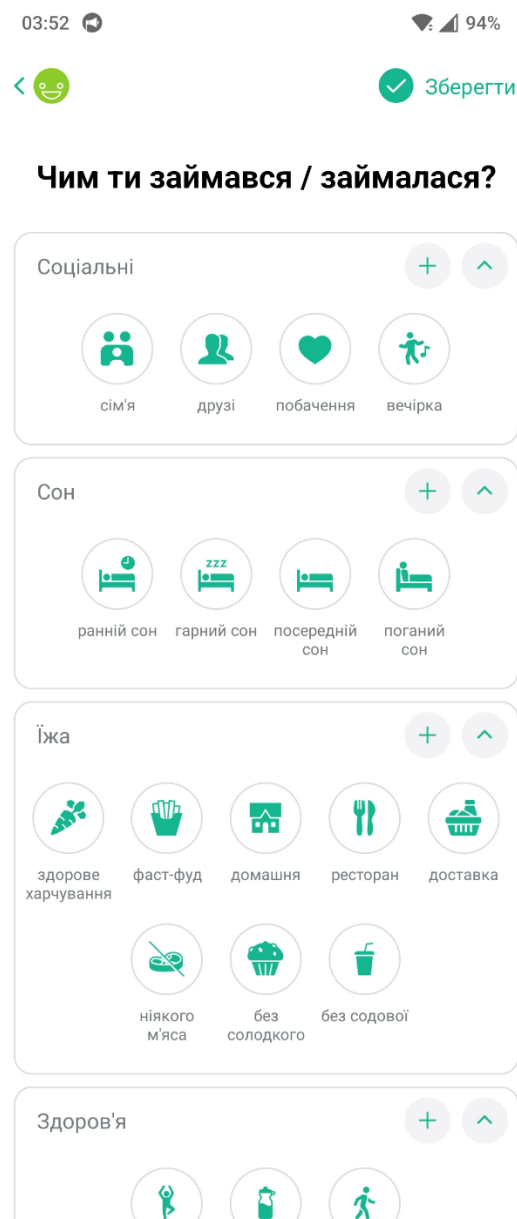


Рисунок 1.3 – Додаток Daylio

Програма дозволяє переглядати зміни настрою протягом останнього місяця, зв'язок за місяць обраного виду діяльності з настроєм, але на основі стандартного кольору за кожен день. Тобто не можна побачити, як кількісні показники тривалості діяльності пов'язані зі зміною настрою. А для визначення кількісних змін потрібно отримати платну версію.

1.2 Визначення концепції розроблюваної програми

До результатів аналізу було внесено додатки Moodfit і Daylio, порівнюючи їх з ПЗ, що розробляється (табл. 1.1).

Таблиця 1.1 – Порівняння програмних засобів моніторингу психічного здоров'я

Критерій	Moodfit	Daylio	IVMood
Фіксування настроїв	+	+	+
Внесення виконаної діяльності	+	+	+
Інструкції з психологічних практик	+	—	+
Наявність платних функцій	+	+	—
Аналіз залежності зміни настроїв від діяльності	Платний	Платний	Безкоштовний
Відслідковування змін настроїв у спільноті	—	—	+
Консультавання лікарями	—	—	+

Загалом Moodfit надає більше загальних можливостей, адже додаток містить зокрема можливість перегляду інструкцій з певних практик,

фіксування результатів виконання діяльності за цими інструкціями, якщо користувач має таке бажання. Daylio таких можливостей не має. Однак обидві програми мають в своєму складі платні функції, якими в безкоштовній версії скористатися не можна. До таких функцій зокрема належить аналіз залежності зміни настроїв від діяльності. Тобто фактично знаходити причини власного стану, посилаючись на збережені дані, у безкоштовному варіанті щонайменше достатньо складно і точно незручно, що призведе в такому випадку до того, що програма не буде використовуватися з цією метою.

ПЗ, що розробляється, окрім подолання проблеми з недоступністю функцій аналізу повинно надати можливість взаємодіяти в межах спільноти, відкриваючи власний настрій та його зміни іншим учасникам спільноти, консультуватися з лікарями, надаючи їм внесені дані. Це має допомогти людям зокрема в Україні виявляти власні проблеми з психічним здоров'ям, причини, які до цього призвели, відслідковувати подальший стан.

1.3 Висновки за розділом 1 та постановка завдань роботи

У першому розділі набір засобів моніторингу психічного здоров'я був проаналізований. До остаточно порівняння було внесено Moodfit і Daylio. У ПЗ, що розробляється, запропоновано реалізувати основні функції з цих додатків, а також інших, які не приводились у розділі 1. Окрім того вирішено надати можливість роботи в спільноті, консультування з лікарями зі знаходженням контактів через програму та забезпечення засобів аналізу власних станів безкоштовно.

Завдання, які потрібно виконати в роботі:

- аналіз програмних засобів моніторингу психічного здоров'я і визначення концепції програми;
- проєктування ПЗ моніторингу психічного здоров'я;
- реалізація ПЗ моніторингу психічного здоров'я.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір технологій розробки

Технології, за допомогою яких мала виконуватися розробка, базувались на мові програмування, в якості якої проаналізовано мови Python [9] і C# (табл. 2.1). Кожна з них має і недоліки, і переваги, зокрема Python – це не найшвидша мова програмування, але вона пришвидшує процес розробки, тому готову програму можна отримати раніше, до того ж мати більш гнучкі можливості і додаткові бібліотеки. Тому Python – це базова мова розробки програми в роботі. Додаткові мови включають HTML, CSS, JavaScript.

Таблиця 2.1 – Порівняння мов C# і Python для розробки

Критерій	C#	Python
Використовувана парадигма програмування	Об'єктно-орієнтована	Мультипарадигмальна
Варіант типізації	Статична	Динамічна
Порівняний вплив на тривалість розробки	Базовий	Зменшує
Вплив на швидкість роботи програми	Зменшує	Базовий
Бібліотеки розширень	Базова	Переважає за кількістю

2.2 Моделювання вимог до програми

Пацієнт може використовувати наступні сценарії для роботи з програмою:

- контроль поточного настрою;
- демонстрування динаміки зміни настрою протягом періоду часу;

– демонстрування основної діяльності, яка призводить до певного настрою;

- демонстрування зміни настрою зі змінами в обраній діяльності;
- авторизація;
- пошук лікаря;
- звернення до лікаря за консультацією;
- оцінювання пацієнтом лікаря;
- демонстрування консультацій;
- долучення пацієнта до спільноти;
- створення спільноти пацієнтів;
- демонстрування зміни настроїв пацієнтів спільноти за період часу;
- відтворення кроків процедури;
- фіксування виконання діяльності;
- проведення опитування пацієнта щодо його стану;
- фіксування прийому ліків;
- аналізування власного стану;
- додавання діяльності до типового розкладу;
- відображення типового розкладу;
- відображення повідомлення про подолання випробування;
- демонстрування всіх випробувань пацієнта;
- демонстрування доступних пацієнту випробувань;
- демонстрування історії діяльності пацієнта;
- демонстрування історії приймання ліків пацієнтом;
- демонстрування історії результатів опитування пацієнта;
- демонстрування історії аналізу станів пацієнта;
- демонстрування доступних опитувань;
- демонстрування доступних діяльностей;
- демонстрування переліку спільнот пацієнта;
- створення випробування;
- демонстрування профілю лікаря (рис. 2.1).

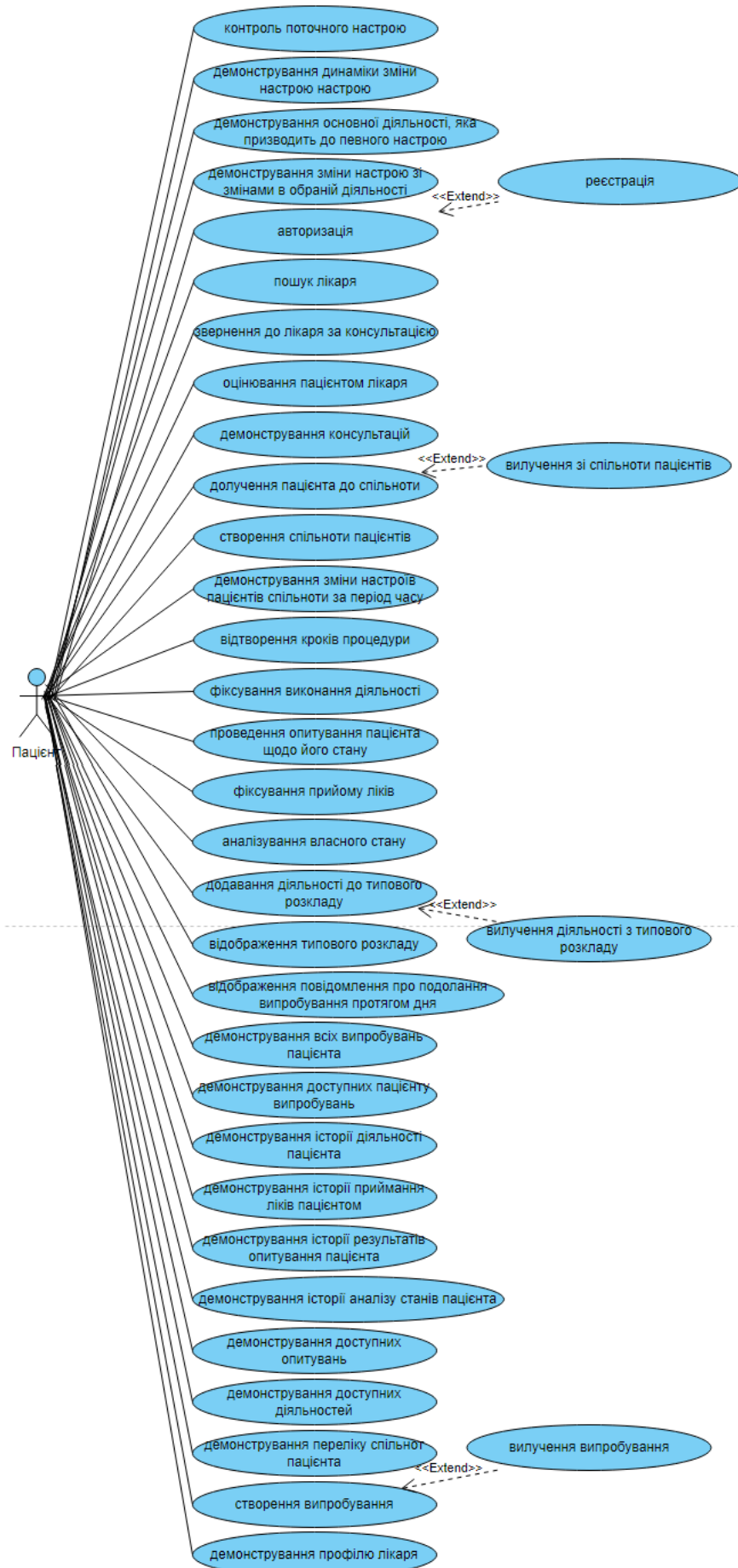


Рисунок 2.1 – Діаграма варіантів використання системи пацієнтом

Альтернативними сценаріями є наступні:

- реєстрація, що реалізується в тому випадку, якщо пацієнт не має облікового запису;
- вилучення діяльності з типового розкладу, якщо більше немає потреби у постійному виконанні раніше доданої діяльності;
- вилучення випробування, якщо пацієнт більше не потребує інформування про наявне випробування;
- вилучення зі спільноти пацієнтів, якщо пацієнт більше не бажає бути учасником спільноти.

Лікар може використовувати такі сценарії для роботи (рис. 2.2):

- демонстрування динаміки зміни настрою пацієнта, що звернувся за консультацією, протягом періоду часу;
- демонстрування основної діяльності, яка призводить певного настрою у пацієнта, який звернувся за консультацією;
- демонстрування зміни настрою пацієнта, який звернувся за консультацією, зі змінами в обраній діяльності;
- авторизація;
- консультування пацієнта;
- демонстрування звернень за консультацією;
- демонстрування історії пацієнта.

Менеджер програми може використовувати такі сценарії (рис. 2.3):

- створення облікових записів лікарів;
- авторизація;
- створення випробування;
- демонстрування доступних випробувань;
- демонстрування доступних опитувань;
- демонстрування доступних діяльностей;
- створення опитування для перевірки психологічного стану;
- створення процедури за певною діяльністю.

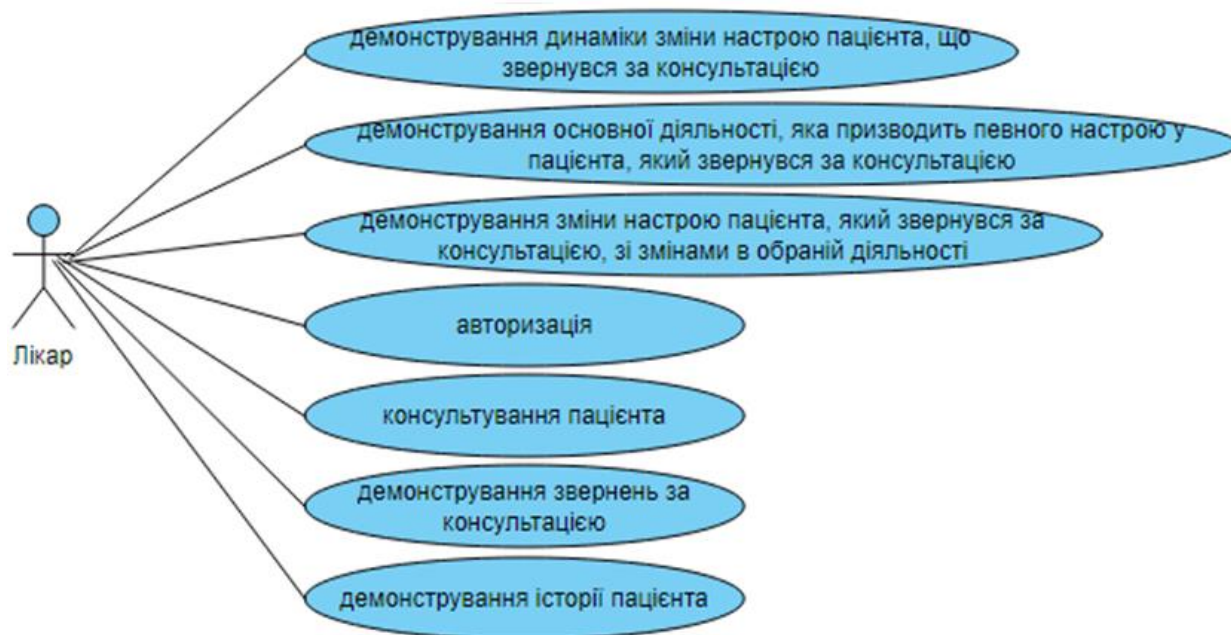


Рисунок 2.2 – Діаграма варіантів використання системи лікарем



Рисунок 2.3 – Діаграма варіантів використання системи менеджером

Альтернативними сценаріями є наступні:

- вилучення випробування, якщо випробування більше не вважається доцільним;
- внесення змін до опитування, якщо в процедурі опитування відбулись зміни або були внесені неточності;

– внесення змін до процедури за певною діяльністю, якщо в процедурі відбулись зміни або були внесені неточності.

2.3 Проєктування бази даних

База даних (БД) створена як реляційна за допомогою системи MySQL.

Облікові записи користувачів зберігаються в БД у таблиці User з наступними полями.

Поле id є ідентифікатором облікового запису.

Поле username є логіном облікового запису, який використовується для авторизації та ідентифікації користувачів у програмі.

Поле password є логіном цього облікового запису.

Поле first_name є іменем користувача.

Поле last_name є прізвищем користувача.

Поле birth_date є датою народження користувача.

Поле photo є фотографією користувача.

Поле is_staff є позначкою про те, що користувач є менеджером або ні.

Облікові записи асоціюються з конкретними користувачами, які розрізняються виконуваними ролями. В залежності від ролі вони характеризуються і різними характеристиками.

Лікар є користувачем програми, але окрім облікового запису про нього зберігається в БД цілий ряд характеристик. Для цього використовується таблиця Doctor з наступними полями.

Поле id є ідентифікатором лікаря, який працює з програмою.

Поле user є обліковим записом цього лікаря, тому представляється зовнішнім ключем з таблиці User.

Поле description є інформацією про лікаря.

Поле specialization є спеціалізацією лікаря, тому представляється зовнішнім ключем з таблиці Specialization.

Поле review є оцінкою, яку всередньому пацієнти виставили цьому лікарю.

Самі спеціалізації лікарів зберігаються в БД у таблиці Specialization з наступними полями.

Поле id є ідентифікатором спеціалізації, що використовується в програмі для визначення особливостей вмінь лікаря.

Поле name є назвою спеціалізації лікаря.

Пацієнт також є користувачем програми, але окрім облікового запису про нього зберігається і додаткова характеристика. Для цього використовується таблиця Patient з наступними полями.

Поле id є ідентифікатором пацієнта, який працює з програмою.

Поле user є обліковим записом цього пацієнта, тому представляється зовнішнім ключем з таблиці User.

Поле shared визначає, чи готовий пацієнт об'єднуватися з іншими користувачами у суспільства, ділячись з ними даними про свій настрій. Якщо ні, то знайти в пошуку такого користувача неможливо буде.

Дані самих суспільств, у які об'єднуються користувачі в програмі, зберігаються в БД у таблиці Community з наступними полями.

Поле id є ідентифікатором суспільства.

Поле name є назвою суспільства користувачів.

Суспільства складаються з окремих пацієнтів, відповідно потрібно зберігати про це дані. Вони зберігаються в БД у таблиці CommunityPatient з наступними полями.

Поле id є ідентифікатором входження пацієнта до суспільства.

Поле community є суспільством, до якого входить пацієнт, тому представляється зовнішнім ключем з таблиці Community.

Поле patient є пацієнтом, який входить в це суспільство, тому представляється зовнішнім ключем з таблиці Patient.

Поле approved визначає, чи погодився цей пацієнт брати участь в цьому суспільстві. Якщо користувача було додано до суспільства, а не він

зробив це сам, то він має встановити цю позначку, щоб інші учасники суспільства змогли отримати дані про його настрій.

Програма має дозволяти фіксувати не тільки настрій, але і результати діяльності. Діяльність може стосуватися певних ментальних процедур, фізичних тощо. Кожна фізична діяльність, тобто заняття спортом також характеризується конкретним відповідним видом спорту. У інших випадках це можуть бути конкретні процедури. Види, процедури об'єднано в єдине поняття процедури. Тобто виходить, що діяльність фіксується на основі зв'язки між діяльністю, що є узагальнюючим поняттям, і конкретної процедури, яка в подальшому, якщо необхідно, може визначатися нижчим рівнем (фактично кроками).

Вищий рівень, сама діяльність, зберігається в БД у таблиці Activity з наступними полями.

Поле id є ідентифікатором діяльності.

Поле name є назвою діяльності.

Поле photo є фотографією діяльності.

Процедури, які стосуються діяльності, зберігаються в БД у таблиці Activityvariant з наступними полями.

Поле id є ідентифікатором процедури діяльності.

Поле activity є діяльністю, до якої відноситься ця процедура, тому представляється зовнішнім ключем з таблиці Activity.

Поле variant є назвою процедури, яка використовується в програмі.

Поле title є назвою процедури.

Може бути так, що діяльність передбачає тільки одну процедуру, в такому випадку вже в програмі при виведенні це має враховуватися і презентуватися користувачу як єдиний рівень. Тобто якщо похід до психотерапевта не має ніяких окремих процедур, то фактично створюється запис у таблиці Activity, а під нього вводиться єдиний запис у таблиці Activityvariant.

Окремі кроки процедур діяльності зберігаються в БД у таблиці Activityvariantstep з наступними полями.

Поле id є ідентифікатором кроку процедури.

Поле activityvariant є процедурою діяльності, до якої відноситься цей крок, тому представляється зовнішнім ключем з таблиці Activityvariant.

Поле ask є запитанням або описом дій, які потрібно виконати на цьому кроці.

Поле photo є фотографією, яка демонструє цей крок.

Поле title є назвою кроку процедури.

Поле answered визначає, чи цей крок є запитанням, чи тільки описом. Тобто поле визначає, чи потрібно давати відповідь на це запитання, чи достатньо просто прочитати цей крок і виконати його. Якщо відповідь не потрібна, то фіксується завершення роботи над процедурою повністю, а дії за кроками не фіксуються. Якщо це запитання, то фіксується за запитанням за кожним кроком відповідь для пацієнта.

Як було зазначено вище, для кроків процедур, які передбачають запитання, потрібно зберігати відповіді. Вони зберігаються в БД у таблиці PatientActivitystep з наступними полями.

Поле id є ідентифікатором відповіді пацієнта стосовно кроку процедури.

Поле activityvariantstep є кроком процедури, стосовно якої внесено відповідь, тому представляється зовнішнім ключем з таблиці Activityvariantstep.

Поле answer є як раз самою наданою пацієнтом відповіддю.

Поле answered_date є датою і часом, коли було надано цю відповідь.

Виконання процедури, кроки яких не передбачають запитань, зберігаються за цілими процедурами в БД у таблиці PatientActivity з наступними полями.

Поле id є ідентифікатором виконання процедури деякої діяльності пацієнтом.

Поле `patient` є пацієнтом, який виконував цю процедуру, тому представляється зовнішнім ключем з таблиці `Patient`.

Поле `activityvariant` є процедурою діяльності, тому представляється зовнішнім ключем з таблиці `Activityvariant`.

Поле `comment` є коментарем пацієнта щодо виконання цієї процедури цього разу.

Поле `duration` визначає тривалість виконання процедури.

Поле `done_date` є датою і часом, коли розпочалось виконання процедури або коли виконувалась процедура, якщо вона нетривала.

Настрій пацієнта є одним з найбільш важливих моніторингових засобів, оскільки відносно нього оцінюється стан, результати виконання будь-якої діяльності. Тому контроль настрою пацієнта зберігається в БД у таблиці `PatientMood` з наступними полями.

Поле `id` є ідентифікатором контролю настрою пацієнта.

Поле `patient` є пацієнтом, настрою якого було проконтрольовано цього разу, тому представляється зовнішнім ключем з таблиці `Patient`.

Поле `mood` є конкретним значенням настрою (жахливий, поганий, помірний, добрий, прекрасний).

Поле `shared` додатково визначає, чи варто цей запис показувати іншим учасникам спільнот, до яких входить пацієнт: за замовчуванням значення показується, але якщо встановлено в цьому полі відповідну позначку, то значення приховується від всіх окрім самого пацієнта.

Поле `date` є часом, коли були виконано контроль настрою.

Опитувальники для перевірки психологічного стану зберігаються в БД у таблиці `Mentaltest` з наступними полями.

Поле `id` є ідентифікатором опитування для перевірки психологічного стану пацієнта.

Поле `name` є назвою відповідного опитування (тесту).

Поле `description` є описом цієї перевірки.

Поле `typical` є визначенням того, чи в цьому тесті типові відповіді. Тобто є такі тести щодо стану людини, які передбачають наявність одних і тих самих відповідей на всі запитання тесту. Типовий тест визначає, що всі відповіді є типовими (однаковими для всіх запитань опитування).

Оскільки опитування можуть бути різними, то і результати можуть відповідати різним маркерам. Співвідношення між числовими результатами опитування і маркерами зберігаються в БД у таблиці `MentaltestResult` з наступними полями.

Поле `id` є ідентифікатором маркеру результату опитування.

Поле `mentaltest` є опитувальником для перевірки психологічного стану, для якого встановлено цей маркер, тому представляється зовнішнім ключем з таблиці `Mentaltest`.

Поле `name` є назвою цього маркера (здоровий, нездоровий, наприклад).

Поле `result` є числовим значенням, якому відповідає цей маркер.

Запитання до опитувальника зберігаються в БД у таблиці `MentaltestQuestion` з наступними полями.

Поле `id` є ідентифікатором запитання з опитувальника для перевірки психологічного стану.

Поле `mentaltest` є опитувальником для перевірки психологічного стану, для якого встановлено цей маркер, тому представляється зовнішнім ключем з таблиці `Mentaltest`.

Поле `question` є текстом запитання.

Варіанти відповідей на запитання до опитувальника зберігаються в БД у таблиці `MentaltestQuestionAnswer` з наступними полями.

Поле `id` є ідентифікатором варіанту відповіді на запитання до опитувальника.

Поле `mentaltestquestion` є запитання до опитувальника, до якого відноситься варіант відповіді, тому представляється зовнішнім ключем з таблиці `MentaltestQuestion`.

Поле `answer` є текстом варіанту відповіді.

Поле `result` є числовим значенням (балом), яке можна отримати у випадку вибору цього варіанту відповіді на запитання.

Результати опитування конкретного пацієнта на основі певного опитувальника зберігаються в БД у таблиці `PatientMentaltest` з наступними полями.

Поле `id` є ідентифікатором результату опитування пацієнта щодо його психологічного стану.

Поле `patient` є пацієнтом, тому представляється зовнішнім ключем з таблиці `Patient`.

Поле `mentaltest` є опитувальником, за яким пацієнт проходив перевірку психологічного стану, тому представляється зовнішнім ключем з таблиці `Mentaltest`.

Поле `result` є кількістю балів, яку пацієнт отримав за цією перевіркою.

Поле `date` є часом, коли пацієнт проходив перевірку.

Самі відповіді на запитання опитувальника пацієнтом зберігаються в БД у таблиці `PatientMentaltestQuestionAnswer` з наступними полями.

Поле `id` є ідентифікатором відповіді пацієнта на запитання опитувальника.

Поле `patientmentaltest` є результатом опитування пацієнта, до якого відноситься ця конкретна відповідь, тому представляється зовнішнім ключем з таблиці `PatientMentaltest`.

Поле `answer` є отриманою від пацієнта відповіддю, тому представляється зовнішнім ключем з таблиці `MentaltestQuestionAnswer`.

Консультації пацієнтів лікарями зберігаються в БД у таблиці `Consult` з наступними полями.

Поле `id` є ідентифікатором консультації пацієнта лікарем.

Поле `doctor` є лікарем, який консультував пацієнта, тому представляється зовнішнім ключем з таблиці `Doctor`.

Поле `patient` є пацієнтом, якого консультував лікар, тому представляється зовнішнім ключем з таблиці `Patient`.

Поле `meeting` є посиланням на зустріч, яку вирішив провести лікар для консультування пацієнта.

Поле `create_date` є часом створення запису про консультацію (коли пацієнт звернувся до лікаря).

Поле `consult_date` є часом проведення консультації (коли лікар вніс результуючу інформацію в цей запис).

Поле `shared` визначає, чи можна надавати лікарю всю інформацію про пацієнта з програми.

Поле `question` є запитанням або проблемою, з якою пацієнт звернувся до лікаря на консультацію.

Поле `prescription` є результуючим записом лікаря з порадою про подальші дії, про лікування за результатами консультації.

Поле `closed` є позначкою про завершення консультації.

Поле `review` є оцінкою, виставленою пацієнтом цьому лікарю після консультації.

Ліки, які може приймати пацієнт і які відповідно впливають на його стан, також мають зберігатися в БД. Перелік самих препаратів зберігається в БД у таблиці `Medicine` з наступними полями.

Поле `id` є ідентифікатором препарату.

Поле `name` є назвою препарату.

Приймання ліків пацієнтом зберігаються в БД у таблиці `PatientMedicine` з наступними полями.

Поле `id` є ідентифікатором приймання ліків пацієнтом.

Поле `patient` є пацієнтом, який приймав ліки, тому представляється зовнішнім ключем з таблиці `Patient`.

Поле `medicine` є препаратом, який приймав пацієнт, тому представляється зовнішнім ключем з таблиці `Medicine`.

Поле `date` є часом приймання ліків пацієнтом.

Поле `comment` є коментарем до приймання ліків. Зокрема в цьому полі може вказуватися кількість ліків.

Пацієнт в програмі може не тільки фіксувати результати, але і створювати для себе певні плани щодо виконання тієї чи іншої діяльності. Це передбачає планомірне виконання відповідних процедур деякої діяльності. Такі дані зберігаються в БД у таблиці `Dailygoal` з наступними полями.

Поле `id` є ідентифікатором запланованої процедури діяльності.

Поле `patient` є пацієнтом, який запланував цю діяльність для себе, тому представляється зовнішнім ключем з таблиці `Patient`.

Поле `activityvariant` є процедурою діяльності, ціль виконувати яку ставить перед собою пацієнт, тому представляється зовнішнім ключем з таблиці `Activityvariant`.

Поле `next_date` є наступною датою, коли ця процедура має бути виконана. Тобто це фактично позначка про те, що сьогодні цю процедуру пацієнт вже виконав.

Поле `end_date` є датою, коли виконання процедури має бути завершено. Якщо не вказано, передбачається, що виконання має продовжуватися без обмеження.

Поле `state` є станом цього плану, тобто виконувати цю процедуру далі чи це вже неактуально.

Пацієнт може самостійно намагатися проаналізувати, пояснити свій стан. Це особливо важливо для негативних станів під час виявлення депресії. Результати аналізу пацієнтом власного стану зберігаються в БД у таблиці `Analyse` з наступними полями.

Поле `id` є ідентифікатором аналізу пацієнтом власного стану.

Поле `patient` є пацієнтом, який виконав аналіз, тому представляється зовнішнім ключем з таблиці `Patient`.

Поле `title` є назвою, тобто фактично проблемою, яка сталася у пацієнта.

Поле situation є детальним описом ситуації, якою її спостерігав пацієнт.

Поле thoughts є описом негативних думок, до яких у пацієнта призвела ця ситуація.

Поле reason є описом причини, яка на думку пацієнта, призвела до такого його стану.

Поле date є часом, коли цей аналіз відбувся.

Виконуючи аналіз власного стану, пацієнт може визначати емоції, які він у відповідній ситуації відчуває. Емоції відносно ситуації зберігаються в БД у таблиці AnalyseEmotions з наступними полями.

Поле id є ідентифікатором емоції пацієнта під час ситуації, яка виникла і аналізується ним.

Поле analyse є аналізом стану пацієнта, якого стосується ця емоція, тому представляється зовнішнім ключем з таблиці Analyse.

Поле emotion, є емоцією, яку в цьому стані відчуває пацієнт, тому представляється зовнішнім ключем з таблиці Emotion.

Емоції, з яких може вибирати пацієнт, визначаючи свій стан, зберігаються в БД у таблиці Emotion з наступними полями.

Поле id є ідентифікатором емоції.

Поле name є назвою емоції.

Пацієнт може не тільки планувати процедури, які він бажає виконати, але і фактично відслідковувати певний набір процедури, визначаючи його як з одного боку певне досягнення, а відповідно з іншого боку як випробування, яке він намагається подолати. Ці випробування можуть бути або типовими, визначеними на рівні всієї програми, або визначеними самим пацієнтом. Випробування зберігаються в БД у таблиці TypicalChallenge з наступними полями.

Поле id є ідентифікатором випробування.

Поле name є назвою випробування.

Поле `typical` є позначкою про те, що це типове випробування, тобто створене менеджером для всіх пацієнтів.

Якщо це не типове (на рівні всієї програми) випробування, то це фактично дані про встановлену назву випробування пацієнтом. А ось дані про те, який пацієнт це випробування створив і як він його долає, або про те, як він долає типове випробування, зберігаються в БД у таблиці `Challenge` з наступними полями.

Поле `id` є ідентифікатором випробування пацієнта.

Поле `patient` є пацієнтом, який долає випробування, тому представляється зовнішнім ключем з таблиці `Patient`.

Поле `typicalchallenge` є випробування, яке долає пацієнт, тому представляється зовнішнім ключем з таблиці `TypicalChallenge`.

Поле `create_date` є датою створення цього випробування (визначення для пацієнта типового випробування або створення його пацієнтом).

Поле `duration` є кількість днів поспіль, протягом якої на даний момент виконує всі процедури з випробування пацієнт.

Поле `max_duration` є максимальною за весь час кількістю днів, протягом якої на даний момент виконував процедури певної діяльності цей пацієнт.

Самі процедури з випробування зберігаються в БД у таблиці `ChallengeActivityvariant` з наступними полями.

Поле `id` є ідентифікатором процедури певної діяльності з випробування.

Поле `typicalchallenge` є випробуванням, до якого ця процедура належить, тому представляється зовнішнім ключем з таблиці `TypicalChallenge`.

Поле `activityvariant` є процедурою певної діяльності, яку потрібно протягом дня хоча би раз виконати пацієнту, тому представляється зовнішнім ключем з таблиці `Activityvariant` (рис. 2.4).

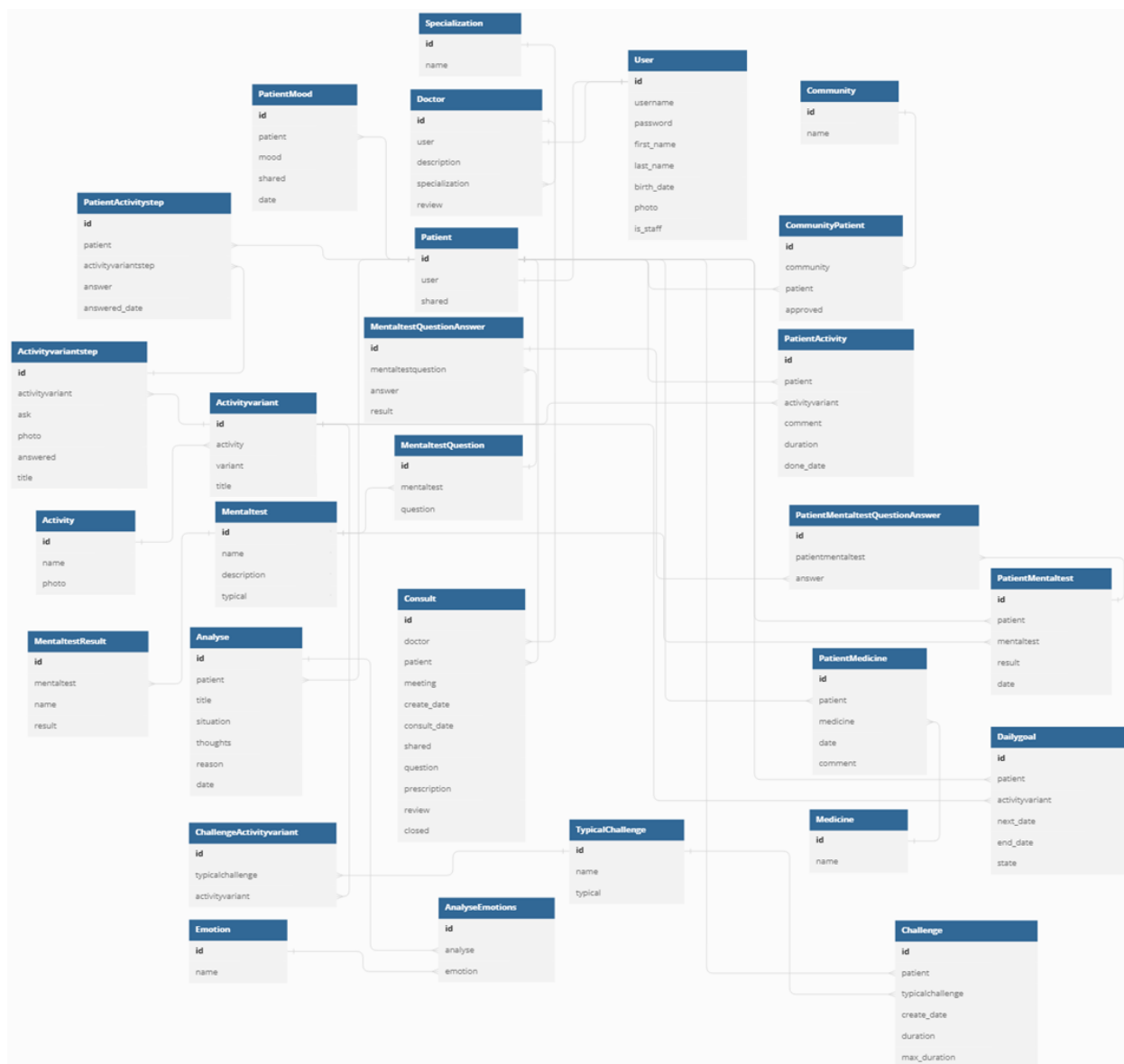


Рисунок 2.4 – Схема БД

2.4 Висновки за розділом 2

У другому розділі мова Python обрана для реалізації, виконано моделювання роботи пацієнтів, лікарів та менеджерів з ПЗ, визначено універсальну структуру БД, що дозволяє в подальшому визначати всі потрібні практики як окремі види і процедури діяльності, розширюючи вміст програми.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Порядок функціонування програми

Функціонування програми побудовано на послідовному виконанні тих сценаріїв роботи, які були на діаграмах варіантів використання системи представлені в розділі 2. Порядок роботи залежить від ролі користувача.

Для лікаря цей порядок визначається тільки переліком пацієнтів, які звернулися до нього з запитами на консультацію. З цього переліку лікар може відкривати конкретний запит і вносити в нього потрібну інформацію, що стосується зустрічі, якщо це необхідно, відповіді на проблему, яку виявив у себе пацієнт.

Для менеджера програми цей порядок роботи визначається основним меню, з якого він може після авторизації обрати потрібний варіант роботи, що включає лікарів, звідки можна додавати нових лікарів для роботи з програмою, випробування, опитування та діяльності.

Для пацієнта після виконання авторизації відображається одразу його типовий розклад, тобто цілі, які він встановив собі на день (рис. 3.1). Після цього пацієнт може використовувати меню для того, щоб обрати необхідну далі дію, що включає: демонстрування спільнот пацієнта, фіксування власного настрою, демонстрування динаміки зміни настрою, демонстрування доступних діяльності, демонстрування доступних опитувань, демонстрування історії приймання ліків, демонстрування історії операцій, аналізування власного стану, демонстрування випробувань, пошук лікарів, демонстрування консультацій, а також завершення роботи та вихід з облікового запису. Усі ці функції пацієнт може використовувати тільки після авторизації. Без авторизації робота фактично неможлива, адже все, що виконується в програмі, персоналізовано. Обираючи кожний пункт меню, пацієнту стає доступним виконання основних дій за ним та додаткові функції (наприклад, створення нових записів за посиланням з виведенням переліку існуючих), а також можливість обрати інший пункт за потреби.

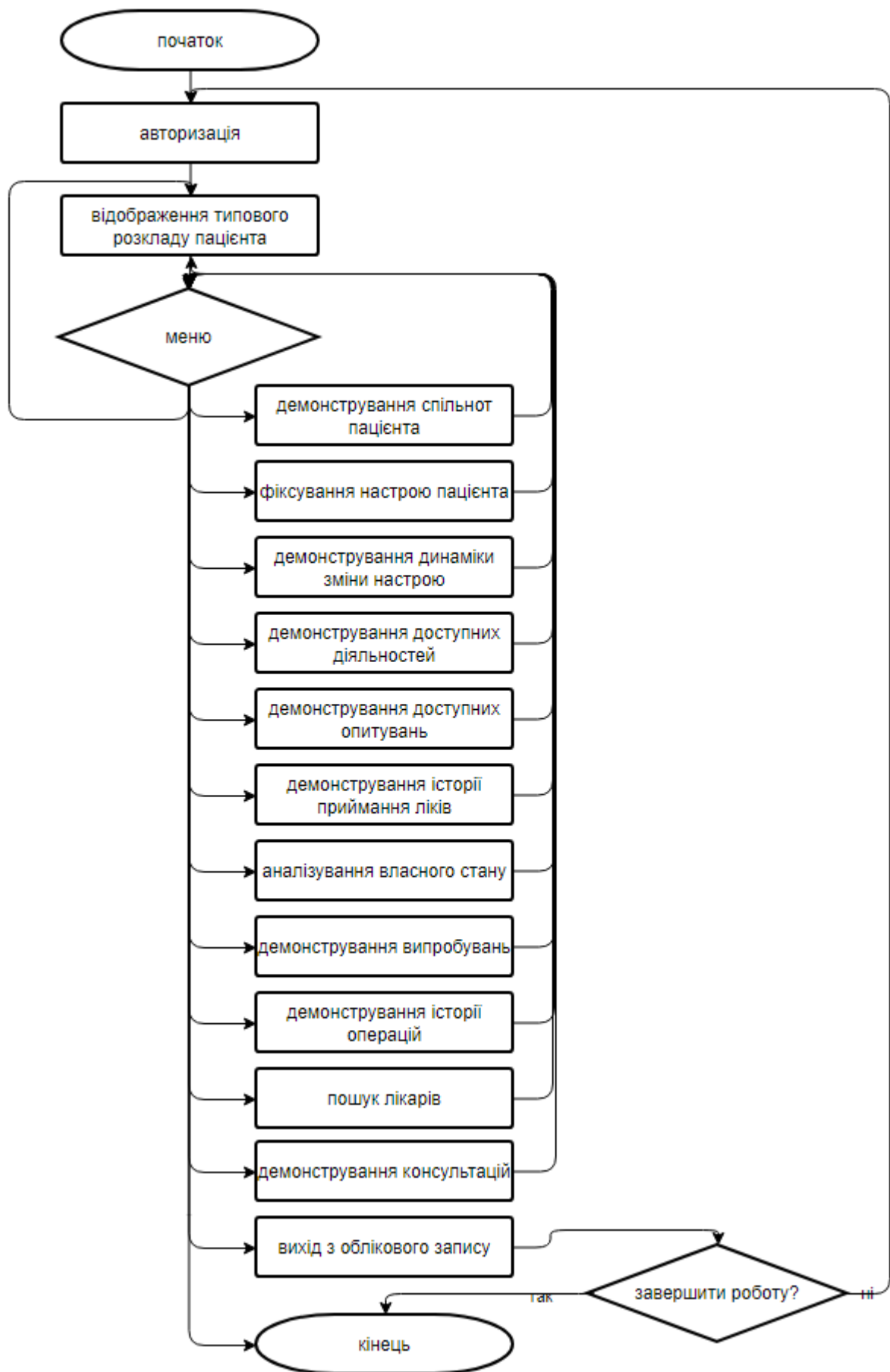


Рисунок 3.1 – Схема функціонування програми для пацієнта

3.2 Програмна реалізація вебсистеми

Демонстрування динаміки зміни настрою пацієнта протягом періоду часу реалізовано за допомогою функції `patient_moods` (`request`, `start` = None, `end` = None), яка дозволяє універсально встановити період часу за допомогою параметрів початку інтервалу `start` та завершення інтервалу `end`. Ці параметри визначаються числами, розділеними між собою дефісами, зазначаючи першим число, потім номер місяця, а останнім – повний рік. Підтримується можливість і відсутності значень цих параметрів, тоді повертаються всі зафіксовані настрої пацієнта. На стороні клієнта за допомогою `javascript`-вбудованих рішень визначається інтервал, що відповідає місяцю, року або будь-якому іншому потрібному інтервалу, а також формується сам графік. Пацієнт, для якого витягаються зафіксовані настрої, визначається обліковим записом, за яким він авторизувався.

Демонстрування історії приймання ліків пацієнтом реалізовано за допомогою функції `patient_medicines` (`request`, `medicine` = None, `start` = None, `end` = None), де сам ідентифікатор препарату визначається параметром `medicine`, який може бути не визначений, тоді відображається історія приймання всіх ліків, а інтервал часу, за який потрібно вивести історію, формується як датами значеннями параметрів `start` і `end`.

Демонстрування історії діяльності пацієнта реалізовано за допомогою функції `patient_activities` (`request`, `activity` = None, `activity_variant` = None, `start` = None, `end` = None), в якій можна визначити безпосередньо один з двох основних параметрів: `activity` – ідентифікатор діяльності, `activity_variant` – ідентифікатор процедури з відповідної діяльності. У такому випадку в історії виводяться тільки записи про діяльність пацієнта за вказаним видом або процедурою. У першому випадку відповідно повертаються записи про всі процедури, що стосуються цього виду діяльності. Параметри `start` і `end` дозволяють обмежити проміжок часу, за який ця історія виводиться.

Демонстрування історії результатів опитування пацієнта реалізовано за допомогою функції `patient_tests` (`request`, `mentaltest = None`, `start = None`, `end = None`), в якій можна вказати конкретний тест (опитування) номером у параметрі `mentaltest`, результати якого потрібно продемонструвати. Якщо параметр не встановлено, всі опитування виводяться разом. Параметри `start` і `end` дозволяють обмежити проміжок часу, за який ця історія виводиться.

Демонстрування історії аналізу станів пацієнта реалізовано за допомогою функції `patient_analyses` (`request`, `start = None`, `end = None`), що повертає всі спроби пацієнтом, який авторизувався, аналізувати свої стани за період часу від `start` до `end`.

Демонстрування історії пацієнта лікарю реалізовано за допомогою функції `patient_history` (`request`, `patient`, `start = None`, `end = None`), в якій визначається пацієнт з логіном `patient`, а лікарський запис визначається за записом авторизованого користувача. За цим лікарем перевіряється, що цей пацієнт оформив запит на консультацію і цей запит не закритий. Таким чином, якщо пацієнт в минулому звертався до лікаря, то в майбутньому історія більше не буде доступна лікарю. Окрім того, якщо пацієнт до лікаря взагалі не звертався, то дані також не будуть надані. Дані складаються з виконаних пацієнтом процедур, опитувань, ліків, які приймав пацієнт та настроїв.

Демонстрування доступних діяльностей пацієнту, що важливо при виборі діяльності для планування, реалізовано за допомогою функції `activities` (`request`).

Демонстрування доступних опитувань пацієнту реалізовано за допомогою функції `mentaltests` (`request`).

Демонстрування переліку спільнот пацієнта реалізовано за допомогою функції `patient_communities` (`request`). У результаті формуються два переліки: `communities_approved` – спільноти, участь в яких підтвердив пацієнт, та `communities_new` – спільноти, які чекають підтвердження пацієнтом.

Демонстрування доступних пацієнту випробувань реалізовано за допомогою функції `typical_challenges (request)`, яка визначає спочатку всі випробування, які є типовими, а потім виділяє з них тільки ті, які ще для себе не встановив на виконання авторизований пацієнт.

Демонстрування всіх випробувань пацієнта реалізовано за допомогою функції `challenges (request)`. У результаті визначаються тільки ті випробування, які сам додав собі авторизований пацієнт.

Демонстрування звернень за консультацією до лікаря реалізовано за допомогою функції `doctor_consults (request)`. Усі консультації впорядковуються за датою створення. Спочатку виводяться нові. Визначаються консультації, запит на які був адресований лікарю, який авторизувався на даний момент.

Демонстрування консультацій пацієнта реалізовано за допомогою функції `patient_consults (request)`, в якій формується перелік консультацій та запитів на консультацію до лікарів, які були створені пацієнтом, що авторизувався в системі.

Демонстрування профілю лікаря реалізовано за допомогою функції `doctor_profile (request, doctor)`. Лікар визначається своїм логіном, заданим у параметрі `doctor`. Вся інформація, включаючи спеціалізацію лікаря, повертається і заповнює відповідну сторінку.

Пошук лікаря реалізовано за допомогою функції `find_doctor (request, specialization = None, name = None, review = None, years = None, minyear = None)`. Усі параметри пошуку визначені у параметрах функції: `specialization` – ідентифікатор спеціалізації лікаря, `name` – ім'я лікаря, `review` – мінімальна середня оцінка лікаря, `years` – максимальна кількість років та `minyear` – мінімальна кількість років лікаря. Значення всіх цих параметрів може бути і пропущено, тоді критерієм пошуку такий параметр не встановлюється. Ім'я лікаря універсально перевіряється. Якщо пробіл в імені відсутній, то пошук відбувається за прізвищем. Якщо пробіл присутній, то відділяється перша і друга частина, а потім перевіряються і відбираються всі лікарі, у яких перша

частина відповідає частині імені, а друга – частині прізвища та навпаки. У підсумку всі відсортовані за середнім балом записи про лікарів повертаються для формування сторінки з результатами пошуку.

Демонстрування зміни настроїв пацієнтів спільноти за період часу реалізовано за допомогою функції `community_moods` (`request`, `community`, `start = None`, `end = None`), що має параметри `community`, який визначає ідентифікатор спільноти, який має бути обов'язково визначено, та інтервал часу, за який має бути відображено історію, починаючи з моменту часу `start` і до моменту часу `end`. Повернення даних за цією спільнотою відбувається тільки в тому випадку, якщо пацієнт, який авторизувався, входить до складу цієї спільноти, при чому він до того ж підтвердив свої входження. Якщо це так, то відбираються всі пацієнти, які входять до цієї спільноти, а за ними відбираються всі зафіксовані настрої зі вказаного інтервалу часу, які дозволив відкривати іншим пацієнт. Впорядковані за датою, від найстарішої до найновішої, записи про настрої повертаються в результаті.

Демонстрування основної діяльності, яка призводить до позитивного або негативного настрою реалізовано за допомогою функції `activityvariants_moods` (`request`, `start = None`, `end = None`, `mood = None`). Ця функція має параметр `mood`, що визначає конкретний настрій (жахливий, поганий, помірний, добрий, прекрасний), та параметри `start` і `end`, що визначають інтервал часу, протягом якого потрібно виконати цій аналіз. Відповідно протягом заданого інтервалу часу вибираються всі записи про настрої та всі записи про процедури стосовно певної діяльності. Далі відбувається співставлення. Якщо конкретний настрій встановлено, то дані вибираються тільки за ним. Якщо не встановлено, то дані відбираються окремо за двома групами: позитивні настрої (значення 4 і 5) та негативні (значення 1 і 2). За відібраними настроями далі аналізується кожна процедура. За кожною процедурою окремо аналізується кожний випадок, коли було зафіксовано цю процедуру. За ним встановлюється у відповідність конкретний настрій. Це найближчий настрій, який після цієї діяльності було

зафіксовано в програмі. Якщо це встановлений в параметрах настроїв або якщо це один з настроїв спочатку з позитивної групи, а потім з негативної, то кількість випадків для цієї процедури збільшується на 1. Так відбувається аналіз для всіх випадків. У підсумку для кожної процедури щодо певної діяльності визначається кількість разів, коли відповідний настроїв або група настроїв були встановлені після виконання цієї діяльності. На клієнтській стороні за допомогою JavaScript ці дані впорядковуються і виводяться на сторінці, вказуючи які процедури були зафіксовані найчастіше з результатом у вигляді відповідного настрою.

Ті ж самі дії, але не для процедур з діяльностей, а самих загальних діяльностей реалізовані у функції `activity_moods (request, start = None, end = None, mood = None)`, яка має ті самі параметри, що і попередня описана.

Відображення типового розкладу пацієнта реалізовано за допомогою функції `patient_goals (request)`, в якій всі цілі користувача на день щодо виконання різних процедур діяльностей повертаються для авторизованого пацієнта.

Контроль поточного настрою пацієнта реалізовано за допомогою функції `control_mood (request)`, яка працює за допомогою форми `PatientMoodForm`, вносячи настроїв авторизованого пацієнта в БД.

Звернення до лікаря за консультацією реалізовано за допомогою функції `ask_consult (request)`, яка працює на основі форми `ConsultForm`, створюючи відповідний об'єкт в результаті. Коли лікар працює з цим об'єктом, він фактично його редагує.

Створення спільноти пацієнтів реалізовано за допомогою функції `create_community (request)`, яка на основі форми `CommunityForm` дозволяє тільки створити спільноту, а також внести до неї першого підтвердженого учасника – авторизованого пацієнта.

Фіксування виконання діяльності пацієнтом реалізовано за допомогою функції `fix_activity (request, activity, step = None)`. Процедура з діяльності визначається своїм ідентифікатором у параметрі `activity`. Параметром

визначається номер кроку в послідовності кроків за цією процедурою. Тобто це не ідентифікатор, а порядковий номер у впорядкованій за ідентифікаторами послідовності всіх кроків процедури. Підтримка функції реалізується формою `PatientActivityForm`, в якій вносяться дані про виконання авторизованим пацієнтом процедури з діяльності, яку було задано. Якщо крок не було вказано, то відбирається перший крок з послідовності, якщо вказано, то відповідно той крок, який відповідає вказаній позиції. За цим кроком перевіряється, чи він передбачає внесення відповіді пацієнтом, чи тільки прочитання інструкції. Якщо передбачає, то підтримується робота через форму `PatientActivitystepForm`. Також при внесенні цієї процедури діяльності за пацієнтом перевіряються всі випробування, які встановив для себе пацієнт. Якщо за якимось з випробувань усі процедури, які заплановані на поточний день, пацієнтом були виконані, то воно додається до переліку `challenges`, що забезпечує виконання функції відображення повідомлення про подолання випробування протягом дня. Цією ж функцією реалізується відтворення кроків процедури пацієнту для тих випадків, коли крок не передбачає відповіді пацієнта, а тільки демонстрацію інструкції.

Фіксування прийому ліків пацієнтом реалізовано за допомогою функції `fix_medicine (request)`, яка підтримується на основі форми `PatientMedicineForm`, вносячи для авторизованого пацієнта вказані ліки.

Аналізування власного стану пацієнтом реалізовано за допомогою функції `fix_analyse (request)`, яка підтримується на основі форми `AnalyseForm`, вносячи для авторизованого пацієнта результати аналізу пацієнтом власного стану.

Створення випробування менеджером реалізовано за допомогою функції `create_typical_challenge (request)`, яка підтримується на основі форми `TypicalChallengeForm`. За випробуванням вносяться процедури діяльності, які мають бути кожен день виконані для подолання даного випробування.

Створення випробування пацієнтом реалізовано за допомогою функції `create_challenge (request)`, яка підтримується на основі форми `ChallengeForm` і

створює відповідне випробування з фіксацією рекордів і поточного стану для авторизованого пацієнта.

Додавання діяльності до типового розкладу пацієнта реалізовано за допомогою функції `add_activity (request, activity)`, яка підтримується на основі форми `DailygoalForm`. Параметр `activity` вказує ідентифікатор процедури діяльності, яку потрібно внести до типового розкладу пацієнта.

Створення процедури за певною діяльністю реалізовано за допомогою функції `create_activityvariant (request)`, що підтримується на основі форми `ActivityvariantForm`, з якою взаємодіє менеджер. Якщо авторизований користувач не має менеджерського статусу, то форма не повертається для роботи.

Внесення змін до процедури за певною діяльністю реалізовано за допомогою функції `edit_activityvariant (request, id)`, в якій параметр `id` визначає ідентифікатор процедури, яку потрібно змінювати. Внесення змін відбувається за допомогою форми `ActivityvariantForm`. Відбувається перевірка менеджерського статусу авторизованого користувача.

Консультування лікарем пацієнта реалізовано за допомогою функції `edit_consult (request, id)`, в якій параметром `id` визначається ідентифікатор заявки на консультацію, яку створив пацієнт. Якщо така заявка існує і якщо вона стосується лікаря, який авторизувався, то на основі окремої форми `DoctorConsultForm`, що забезпечує доступ до полів лікаря в консультації, відбувається внесення необхідних даних.

Оцінювання пацієнтом лікаря реалізовано за допомогою функції `assess (request, consult, point)`, в якій параметром `consult` визначається ідентифікатор консультації, а параметром `point` – сама оцінка від 1 до 5. У функції відбувається перевірка, що авторизований пацієнт є тим, хто створив заявку на цю консультацію. Після збереження оцінювання переобчислюється середній бал лікаря, дані вносяться в БД.

Вилучення діяльності з типового розкладу пацієнта реалізовано за допомогою функції `delete_goal (request, goal)`, де параметр `goal` визначає

ідентифікатор цілі пацієнта з виконання діяльності в типовому розкладі. Якщо такий об'єкт для такого пацієнта існує, то він вилучається.

Вилучення випробування пацієнтом реалізовано за допомогою функції `delete_challenger (request, challenge)`, у якій для авторизованого пацієнта вибирається випробування з ідентифікатором `challenge`, а якщо воно існує, то вилучається з БД.

Вилучення зі спільноти пацієнтів реалізовано за допомогою функції `delete_community (request, community, patient = None)`, де параметр `community` визначає ідентифікатор спільноти, а `patient` – логін пацієнта, якого потрібно вилучити зі спільноти. Логін може бути пропущений, тоді відбувається вилучення зі спільноти авторизованого пацієнта. Якщо логін заданий, то спочатку перевіряється, що авторизований пацієнт входить до спільноти `community`, а якщо це так, то перевіряється, що пацієнт `patient` входить до неї. Якщо це також вірно, то запис про входження цього пацієнта в спільноту вилучається.

Долучення пацієнта до спільноти реалізовано за допомогою функції `add_to_community (request, community, patient)`. Долучення відбувається пацієнтом, який вже входить у спільноту. Він визначає пацієнта, якого потрібно додати до спільноти, задаючи його логін у параметрі `patient`. Додавання відбувається до спільноти з ідентифікатором `community`. Для авторизованого користувача перевіряється, що він є учасником цієї спільноти. Якщо це так, то створюється новий запис про участь пацієнта `patient` у спільноті.

Створення опитування для перевірки психологічного стану реалізовано за допомогою функцій:

- `create_test (request)`, що на основі форми `MentaltestForm` реалізує створення самого загального опитування;
- `create_test_question (request)`, що на основі форми `MentaltestQuestionForm` реалізує створення запитання у опитуванні;

- `create_test_answer (request)`, що на основі форми `MentaltestQuestionAnswerForm` реалізує створення варіанту відповіді в запитанні.

Усі ці функції перевіряють менеджерський статус у авторизованого користувача.

Внесення змін до опитування реалізовано за допомогою функцій:

- `edit_test (request, id)`, що на основі форми `MentaltestForm` реалізує редагування загального опитування з ідентифікатором `id`;

- `edit_testresult (request, id)`, що на основі форми `MentaltestResultForm` реалізує редагування результату опитування з ідентифікатором `id`;

- `edit_testquestion (request, id)`, що на основі форми `MentaltestQuestionForm` реалізує редагування запитання з ідентифікатором `id` у опитуванні;

- `edit_testanswer (request, id)`, що на основі форми `MentaltestQuestionAnswerForm` реалізує редагування варіанту відповіді з ідентифікатором `id` в запитанні.

Проведення опитування пацієнта щодо його стану реалізовано за допомогою функції `save_test_answer (request, mentaltest)`, яка створює запис про проходження опитування з ідентифікатором `mentaltest` авторизованим пацієнтом, якщо цей запис ще не створено, а далі на основі форми `PatientMentaltestQuestionAnswerForm` реалізує внесення відповіді на запитання в БД. Якщо в цей день вже вносився результат відповіді на це запитання, то результат замінюється. Тобто зберігаються тільки по одному результату на день.

Відтворення всього опитування на одній сторінці для менеджера реалізується функцією `full_test (request, id)`, де `id` визначає ідентифікатор опитування.

Демонстрування зміни настрою зі змінами в обраній діяльності реалізується функцією `patient_activity_moods (request, activity, start = None, end = None)`, параметр якої `activity` визначає ідентифікатор процедури з

діяльності, а start і end визначають період, за яким потрібно виконувати обчислення. За заданим інтервалом за вказаною процедурою відбувається для кожного дня обчислення тривалості виконання цієї процедури або кількості, якщо тривалість для неї не вказується. Це співставляється з настроєм пацієнта в цей день.

Кожна з цих функцій повертає сторінку, заповнену відповідними даними за логікою фреймворку Django, формуючи саму сторінку за допомогою розмітки, визначеної на основі шаблону [10].

3.3 Висновки за розділом 3

У результаті виконання третього розділу програмно реалізовано вебдодаток для моніторингу психічного здоров'я людини. Створено усі функції, які було заплановано, та описано їх у цьому розділі.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма надає можливість користувачам відслідковувати власне психічне здоров'я, контролювати і фіксувати настрій, стани, виконувати діяльність, отримувати процедури психологічних практик, аналізувати динаміку зміни станів, отримувати консультації від лікарів, а лікарям аналізувати накопичені дані за пацієнтами та надавати їм консультації.

4.2 Умови виконання програми

Умови виконання полягають в забезпеченні потрібної програмної та апаратної частини, що в сумі дозволяють створити сервер з доступом з клієнтських пристроїв до нього для роботи. Програмні вимоги в першу чергу включають засоби мови Python, систему MySQL. Апаратні вимоги включають: процесор, мінімальна конфігурація – Intel Core 2 або подібний, оперативна пам'ять з доступом не менше ніж до 4 Гб, жорсткий диск з доступним обсягом не менше ніж 100 Гб.

4.3 Виконання і тестування програми

Тестування програми виконувалось на різних пристроях, включаючи як персональні комп'ютери під керуванням Windows 10 і 11, браузерів Chrome, Firefox, так і мобільними пристроями під керуванням Android 12 і 13. Коректне відображення інтерфейсів було перевірено. Також було перевірено коректність виконання функцій, які були визначені вимогами в розділі 2.

Основний потік роботи пацієнта з програмою був представлений у розділі 3, тому тут зроблений акцент на окремих режимах роботи, що забезпечують основну логіку використання програми.

Одразу після початку роботи пацієнта (рис. 4.1) йому демонструється типовий розклад, який містить у вигляді зображень всі цілі пацієнта, тобто процедури з діяльності, які визначив для себе для виконання протягом дня пацієнт. Кожне зображення обмежено кольором: зелений колір вказує на те, що цю діяльність в цей день було вже виконано пацієнтом, а червоний колір вказує, що ця діяльність ще не виконувалась.

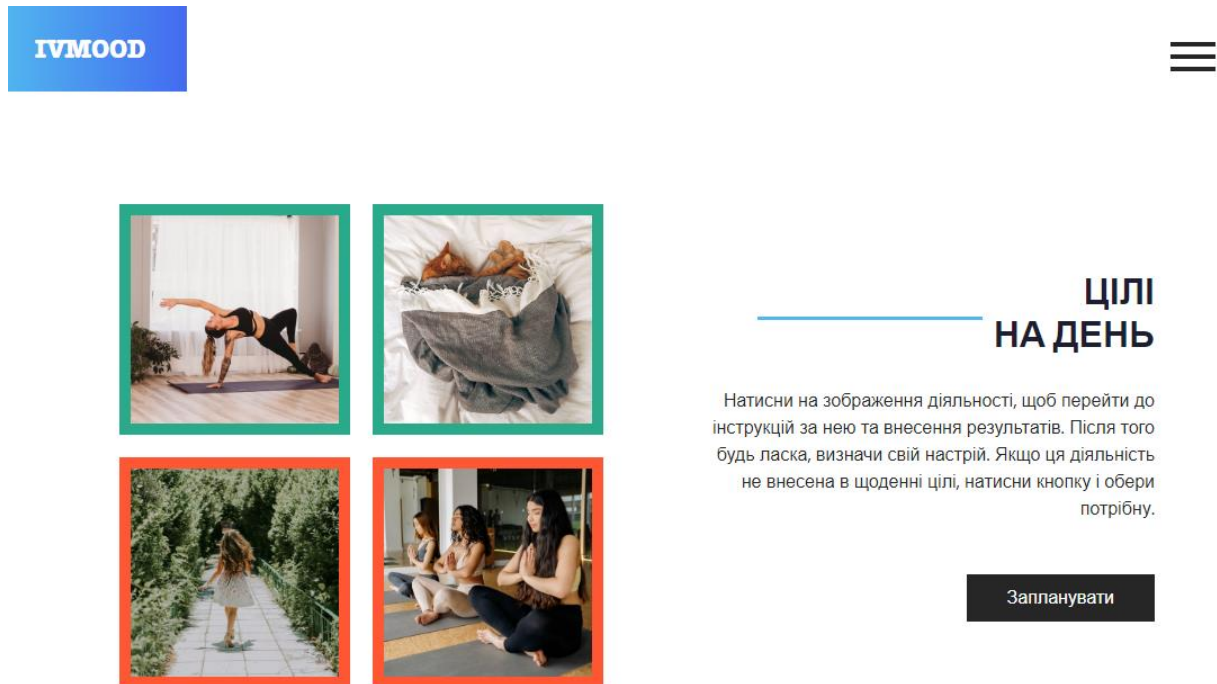


Рисунок 4.1 – Типовий розклад

Для того щоб зафіксувати діяльність, яка ще не виконувалась, потрібно натиснути на зображення процедури, тоді відкриється окреме вікно для перегляду інструкцій процедури. З цієї ж сторінки можна вилучити діяльність, якщо вона більше не потрібна в стандартному розкладі.

Після фіксації виконаної діяльності пацієнту потрібно відзначити свій поточний настрій (рис. 4.2). Для цього призначений окремий пункт меню та сторінка, де необхідно натиснути на потрібне зображення, що відповідає поточному настрою: верхній рядок це зліва направо жахливий і поганий, середній – помірний, нижній – добрий та прекрасний. Після натискання на

зображення робота зі сторінкою завершується. При наведенні відображається словесне представлення настрою.

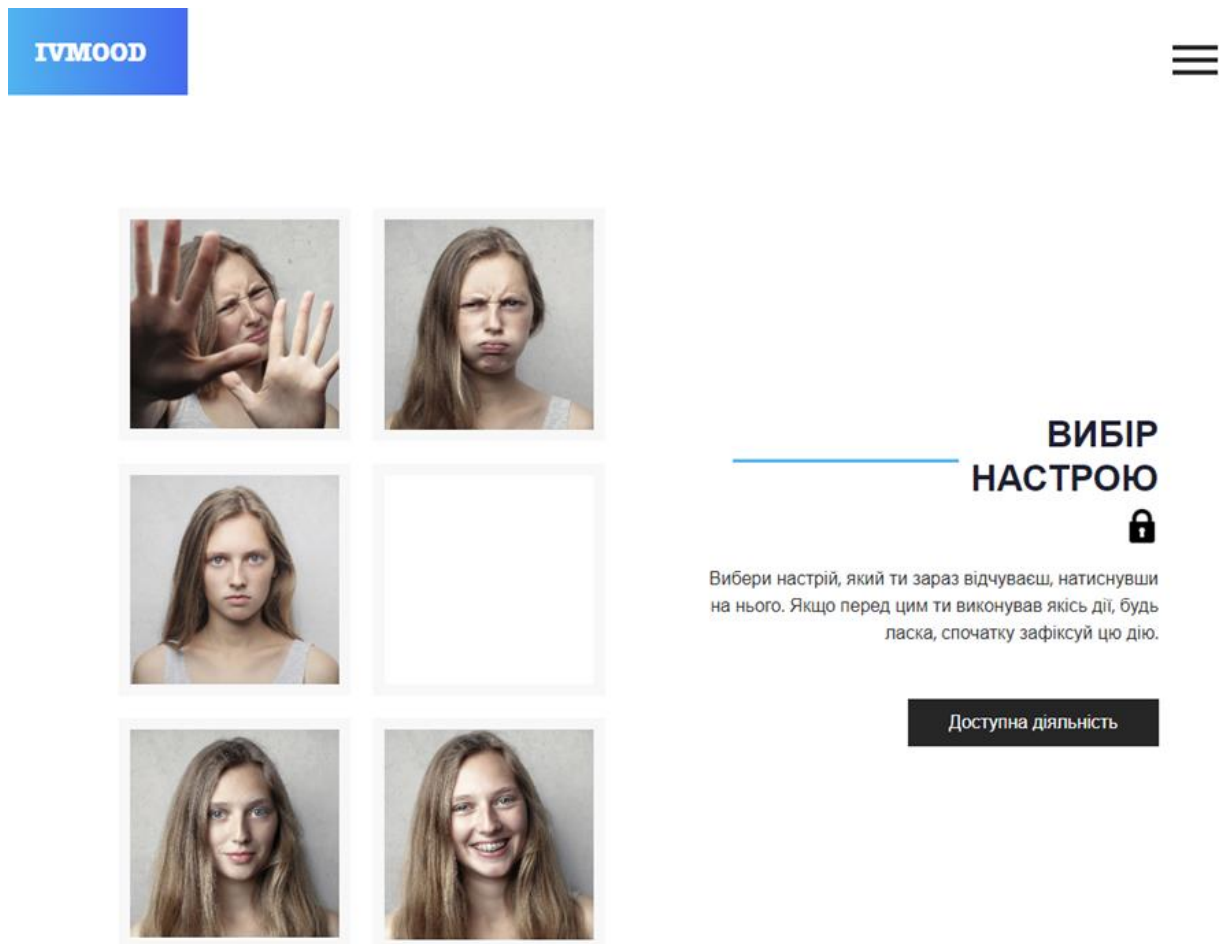


Рисунок 4.2 – Фіксування настрою

Зі сторінки демонстрації доступних діяльностей пацієнт може додати потрібну діяльність до свого розкладу. Можна також відкрити інструкцію за конкретною процедурою (рис. 4.3). Частина процедур передбачає внесення відповідей за кроками, частина – ні. Проте в будь-якому випадку, якщо пацієнт виконав цю діяльність, то йому пропонується внести коментар, відзначити, коли саме це відбулось і скільки тривало. Якщо тривалість не визначається, значить це одномоментна дія. На початковій сторінці процедури відображається одразу перший крок знизу під загальною інформацією про процедуру. Стрілки під фотографією дозволяють

переходити між кроками. Для першого кроку стрілка вліво дозволяє повернутися на сторінку вибору діяльностей.

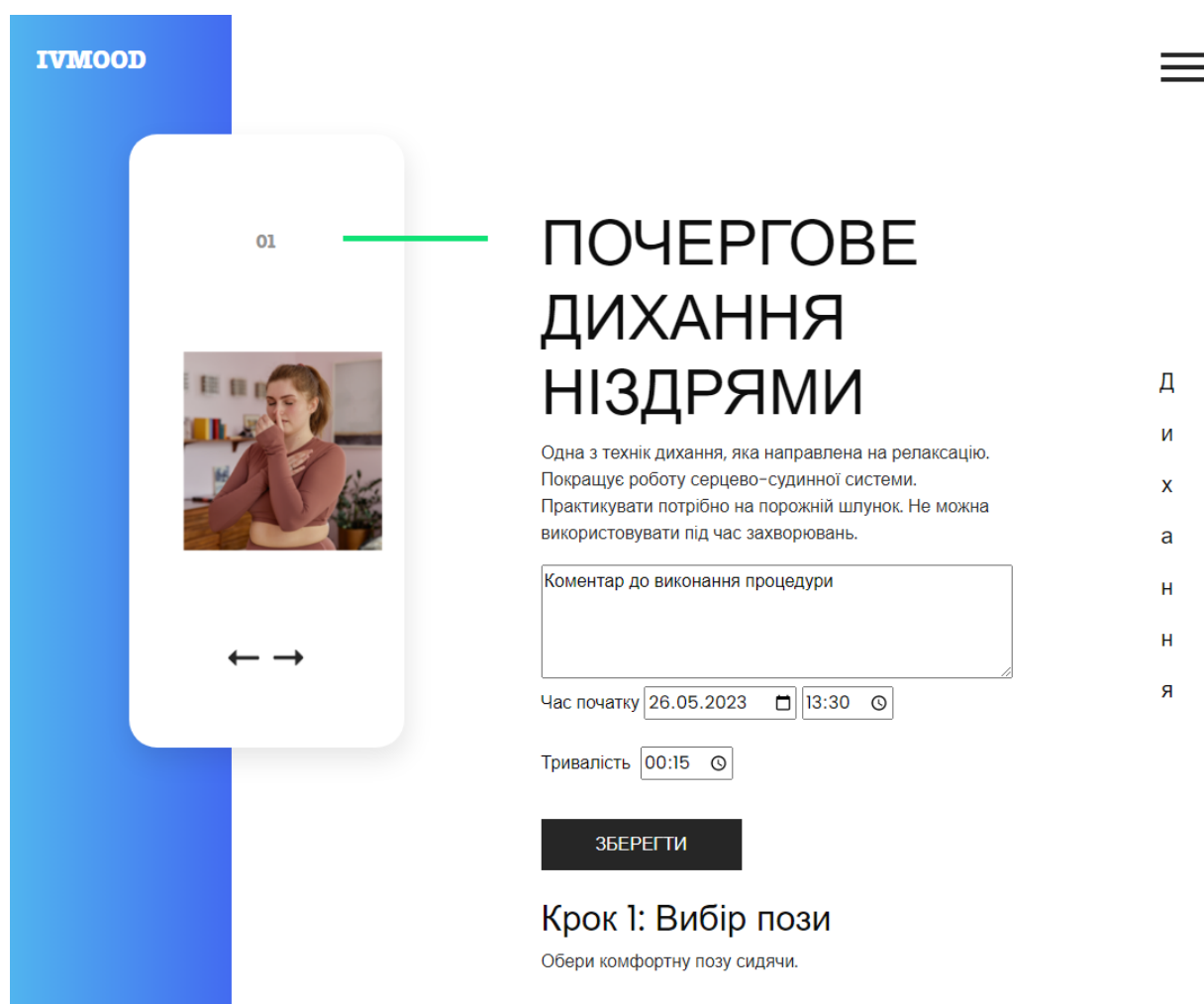


Рисунок 4.3 – Фіксування виконання діяльності

Як було визначено під час аналізу предметної області, функції аналізу настроїв є платними у існуючих програмах. У програмі IVMood ці функції доступні з меню за двома пунктами. Це або динаміка змін (рис. 4.4), або статистика з кількістю випадків, коли до деякого настрою призводила кожна процедура, відсортованими за кількістю випадків.

Аналіз змін настрою передбачає вибір діяльності з переліку або визначення тільки настрою. Тоді діяльність не відображається взагалі, тільки настроїв. Також потрібно задати інтервал часу. Стандартно відображається поточна дата і місяць від неї назад. На графіку виводяться за вказаний період

дати і точки, що відповідають тривалості виконання діяльності в цей день та значенню настрою. Кожен настрій відповідає одному з 5 значень. При наведенні відображається словесне представлення настрою.

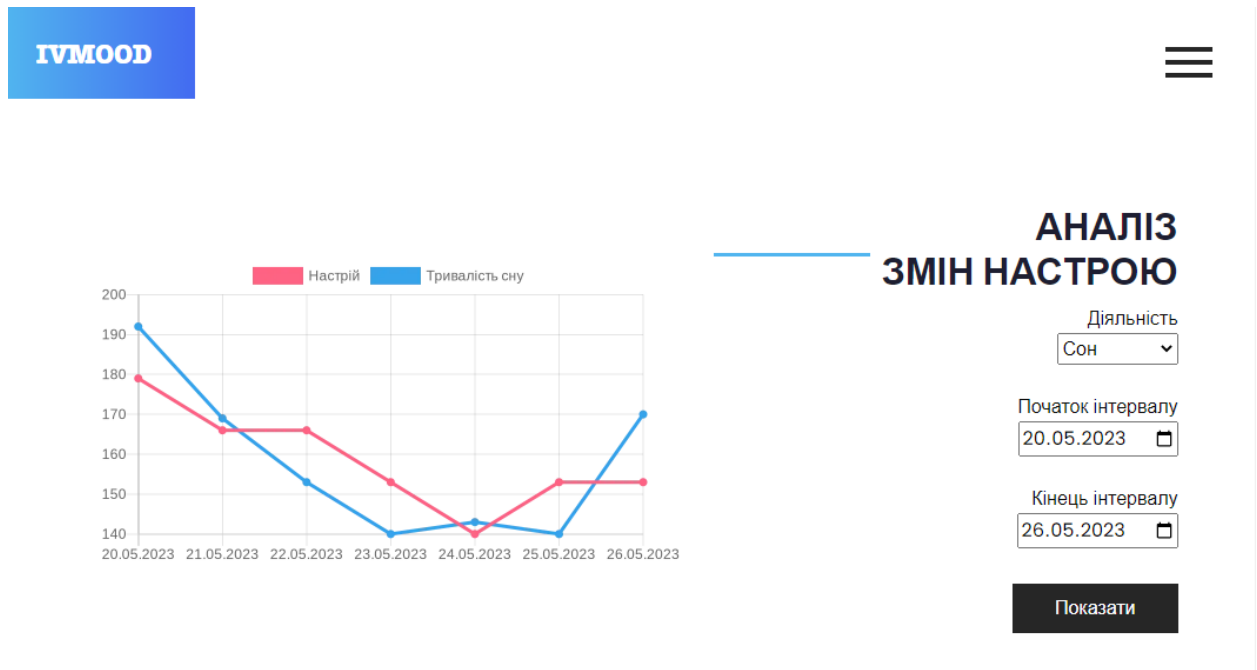


Рисунок 4.4 – Аналіз зміни настрою за діяльністю

Окрім пацієнта і лікаря, і менеджер мають свої режими роботи. Лікар має базовий інтерфейс, що складається з переліку оформлених до нього запитів (рис. 4.5). За кожним запитом відображається ім'я та фотографія пацієнта, дата створення запиту, опис проблеми від пацієнта. Лікар може відкрити потрібний запис, внести свою відповідь. За кожним запитом показується, чи він вже опрацьований лікарем, чи ні. З меню (правий верхній кут) лікар може вийти з облікового запису.

Авторизація для всіх користувачів спільна, але далі відображається той варіант роботи, якій ролі відповідає обліковий запис. Якщо в самому обліковому записі вказано значення `is_staff` як правдиве, то відображається сторінка для менеджера. Якщо ні, то відображається або сторінка для лікаря, або для пацієнта, в залежності від того, в якій таблиці БД є дані під цей обліковий запис.

СЕРГІЙ ІГНАТЕНКО

неопрацьовано

Останнім часом почав відчувати надто велику втому. Багато працюю, звик долати втому і стрес. Але зараз втрачаю бажання працювати далі і робити взагалі щось.

створено 26.05.2023 19:08

Консультувати



ВІКТОРІЯ ТИМКО

неопрацьовано

Маю деякі ідеї починати власну справу, але страх того, що не вдасться, вище за мене. Відкладаю все на потім, а потім так і не настає. Чи можна це здолати?

створено 26.05.2023 18:49

Консультувати



Рисунок 4.5 – Звернення до лікаря

4.4 Висновки за розділом 4

Програму повністю реалізовано, виконано перевірку її функцій, представлено основні режими роботи користувачів.

ВИСНОВКИ

У роботі необхідно було розробити ПЗ моніторингу психічного здоров'я для надання допомоги людині у виявленні та розв'язанні психологічних проблем. В основі це ПЗ фактично є засобом збереження станів психічного здоров'я, але це є корисним для знаходження причин того стану, в якому людина перебуває, для продовження лікування, адже для цього як раз і потрібен збір таких даних з наданням їх в подальшому лікарю. Враховуючи, що програми-аналоги мають тільки платні засоби аналізу та описану концепцію, вирішено було розробити програму з наданням консультацій лікаря через неї або зі встановленням контакту через неї, що зокрема дозволило лікарю надати ту інформацію, яку в програмі накопичив про своє психічне здоров'я пацієнт.

Програмну систему було розроблено мовами Python, HTML, CSS, JavaScript з використанням системи керування БД MySQL. До результатів роботи слід віднести і саму реалізовану програмну систему, і результати аналізу предметної області, де було визначено функції подібних програм і визначено відсутність взаємодії всередині спільноти, наявність платних засобів для аналізу, та результати проєктування програми. Користувачами програми є потенційні пацієнти, лікарі та менеджери програми. Необхідні їм функції були визначені як функціональні вимоги в розрізі кожного користувача, реалізовані та описані.

Розроблена програма направлена на підтримку фіксації настрою, станів, діяльності пацієнта, аналіз зміни настрою, власного стану індивідуально самим пацієнтом, моніторингу настроїв у спільноті та на базову підтримку людей, які потребують допомоги, лікарями. Програма орієнтована на те, щоб люди, які зіштовхуються з психологічними проблемами, могли їх вчасно виявити, на ранніх стадіях, та зрозуміли, що вони потребують професійної підтримки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке психічне здоров'я? – Товариство Червоного Хреста України [Електрон. ресурс]. – Режим доступу : <https://redcross.org.ua/mhpss/mental-health/>.
2. Scaling-up mental health and psychosocial services in war-affected regions [Electronic resource] : best practices from Ukraine. – Access mode : <https://www.who.int/news-room/feature-stories/detail/scaling-up-mental-health-and-psychosocial-services-in-war-affected-regions--best-practices-from-ukraine>.
3. Ukrainian refugees need mental-health care that their hosts lack [Electronic resource]. – Access mode : <https://www.economist.com/europe/2022/04/30/ukrainian-refugees-need-mental-health-care-that-their-hosts-lack>.
4. Pangilinan, J. The 11 Best Mood Tracker Apps for 2023 [Electronic resource] / J. Pangilinan. – Access mode : <https://www.happierhuman.com/best-mood-tracker-apps/>.
5. Moodfit. Fitness for Your Mental Health [Electronic resource]. – Access mode : <https://www.getmoodfit.com/>.
6. Moodfit [Electronic resource] : Mental Health Fitness. – Access mode : <https://play.google.com/store/apps/details?id=com.robleridge.Moodfit/>.
7. Daylio – Journal, Diary and Mood Tracker [Electronic resource]. – Access mode : <https://daylio.net/>.
8. Valera, S. Best Mood Tracker Apps [Electronic resource] / S. Valera. – Access mode : <https://www.verywellmind.com/best-mood-tracker-apps-5212922>.
9. Downey, A. Think Python [Text] : How to Think Like a Computer Scientist / A. Downey. – Sebastopol, California : O'Reilly Media, 2016. – 289 p.
10. Goid Free Website Template. Free CSS Templates. Free CSS [Electronic resource]. – Access mode : <https://www.free-css.com/free-css-templates/page291/goid>.

ДОДАТОК А
Технічне завдання

Вступ

Психічне здоров'я є дуже важливим для людини, впливаючи на всю її діяльність безпосередньо. Тому відслідковувати психічний стан людини, аналізувати зміни, виявити проблеми важливо вчасно. І ПЗ при цьому може бути достатньо корисним, надаючи можливість і виконувати моніторинг стану, і визначаючи необхідні для застосування практики, дозволяючи виконувати планування і контроль практик для покращення стану пацієнта. Зараз динамічне оточення, робота в стресових умовах додатково посилені війною в Україні, що призводить до того, що велика кількість людей відчуває проблеми з психічним здоров'ям. Не всі звертаються до психологів при цьому, не всі відслідковують свій стан, а тому використання ПЗ може допомогти виявити наявні проблеми вчасно.

A.1 Підстави для розробки

Завдання на дипломну кваліфікаційну роботу визначено на підставі наказу № 87 від 4 квітня 2023 р. за Національним університетом «Запорізька політехніка» за темою «Програмне забезпечення моніторингу психічного здоров'я».

A.2 Призначення розробки

Програма повинна надати можливість користувачам відслідковувати власне психічне здоров'я, контролювати і фіксувати настрій, стани, виконувану діяльність, отримувати процедури психологічних практик, аналізувати динаміку зміни станів, отримувати консультації від лікарів, а лікарям аналізувати накопичені дані за пацієнтами та надавати їм консультації.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Для підтримки описаного призначення програма повинна надавати користувачам наступний набір функцій для роботи:

- контроль поточного настрою пацієнта;
- демонстрування динаміки зміни настрою пацієнта протягом періоду часу (місяця, року);
- демонстрування основної діяльності, яка призводить до позитивного або негативного настрою;
- демонстрування зміни настрою зі змінами в обраній діяльності;
- авторизація і реєстрація пацієнта, авторизація лікаря;
- пошук лікаря;
- звернення до лікаря за консультацією;
- консультування лікарем пацієнта;
- вилучення зі спільноти пацієнтів;
- оцінювання пацієнтом лікаря;
- долучення пацієнта до спільноти;
- створення спільноти пацієнтів;
- демонстрування зміни настроїв пацієнтів спільноти за період часу;
- відтворення кроків процедури пацієнту;
- фіксування виконання діяльності пацієнтом;
- проведення опитування пацієнта щодо його стану;
- фіксування прийому ліків пацієнтом;
- аналізування власного стану пацієнтом;
- додавання діяльності до типового розкладу пацієнта;
- відображення типового розкладу пацієнта;
- вилучення діяльності з типового розкладу пацієнта;
- відображення повідомлення про подолання випробування протягом дня;

- демонстрування всіх випробувань пацієнта;
- вилучення випробування пацієнтом;
- створення випробування;
- демонстрування доступних пацієнту випробувань;
- демонстрування історії діяльності пацієнта;
- демонстрування історії приймання ліків пацієнтом;
- демонстрування історії результатів опитування пацієнта;
- демонстрування історії аналізу станів пацієнта;
- демонстрування звернень за консультацією до лікаря;
- демонстрування історії пацієнта лікарю;
- демонстрування доступних опитувань пацієнту;
- демонстрування доступних діяльностей пацієнту;
- демонстрування переліку спільнот пацієнта;
- створення опитування для перевірки психологічного стану;
- внесення змін до опитування;
- створення процедури за певною діяльністю;
- внесення змін до процедури за певною діяльністю;
- демонстрування профілю лікаря;
- демонстрування консультацій пацієнта.

A.3.2 Вимоги до складу та параметрів технічних засобів

Апаратні вимоги до серверної сторони:

- процесор у мінімальній конфігурації Intel Core 2 або еквівалент;
- оперативна пам'ять з доступом не менше ніж до 4 Гб;
- жорсткий диск з бажаним доступним обсягом не менше ніж 100 Гб,

а у випадку меншого обсягу контролювати обсяг даних у БД при внесенні фотографій речей, при цьому в будь-якому випадку потрібний обсяг може і значно збільшуватися, що залежить від кількості користувачів, кількості та якості фотографій, які вони завантажують.

А.4 Порядок контролю та приймання

Основою щодо реалізації контролю та приймання дипломної кваліфікаційної роботи є завдання на роботу і безпосередньо дане технічне завдання. За успішного виконання цих завдань у встановлені строки робота після проходження всіх підготовчих процедур вноситься на захист перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

```

from .models import PatientMood, Patient, PatientActivity, PatientMedicine,
Medicine, Activityvariant, PatientActivitystep, Analyse, Doctor, Consult, CommunityPatient,
TypicalChallenge, Challenge, Specialization, Dailygoal, Activityvariantstep, Community, Activity,
Mentaltest, MentaltestResult, MentaltestQuestion, MentaltestQuestionAnswer,
PatientMentaltest, PatientMentaltestQuestionAnswer

from django.shortcuts import redirect, render

from .forms import PatientMoodForm, ConsultForm, CommunityForm,
PatientActivityForm, PatientActivitystepForm, PatientMedicineForm, AnalyseForm,
TypicalChallengeForm, ChallengeForm, DailygoalForm, ActivityvariantForm,
DoctorConsultForm, DoctorForm, MentaltestForm, MentaltestQuestionForm,
MentaltestQuestionAnswerForm, MentaltestResultForm,
PatientMentaltestQuestionAnswerForm

from django.db.models import Q, Count
import datetime

def patient_moods(request, start = None, end = None):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if patient:
        start_no = False
        if start:
            start_parts = start.split("-")
            year = int(start_parts[2])
            month = int(start_parts[1] )
            day = int(start_parts[0] )
            start_date = datetime.datetime(year, month, date, 0, 0)
        else:
            start_no = True
        end_no = False
        if end:
            end_parts = end.split("-")
            year = int(start_end[2])
            month = int(start_end[1])
            day = int(start_end[0])
            end_date = datetime.datetime(year, month, date, 23, 59)
        else:
            end_no = True
        if end_no and not start_no:
            end_date = start_date + datetime.timedelta(months = 1)

```

```

elif not end_no and start_no:
    start_date = end_date - datetime.timedelta(months = 1)
elif end_no and start_no:
    start_date = datetime.datetime.now()
    end_date = start_date + datetime.timedelta(months = 1)
    moods = PatientMood.objects.filter(patient = patient, date__gte = start_date,
date__lte = end_date).order_by("date")
    return render(request, "ivmood/patientmoods.html", {"moods": moods })
else:
    return render(request, "ivmood/patientmoods.html", {"moods": [] })

def patient_activity_moods(request, activity, start = None, end = None):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if patient:
        start_no = False
        if start:
            start_parts = start.split("-")
            year = int(start_parts[2])
            month = int(start_parts[1] )
            day = int(start_parts[0] )
            start_date = datetime.datetime(year, month, date, 0, 0)
        else:
            start_no = True
        end_no = False
        if end:
            end_parts = end.split("-")
            year = int(start_end[2])
            month = int(start_end[1])
            day = int(start_end[0])
            end_date = datetime.datetime(year, month, date, 23, 59)
        else:
            end_no = True
        if end_no and not start_no:
            end_date = start_date + datetime.timedelta(months = 1)
        elif not end_no and start_no:
            start_date = end_date - datetime.timedelta(months = 1)
        elif end_no and start_no:
            start_date = datetime.datetime.now()

```

```

        end_date = start_date + datetime.timedelta(months = 1)
        moods = PatientMood.objects.filter(patient = patient, date__gte = start_date,
date__lte = end_date).order_by("date")
        activities = []
        while start_date <= end_date:
            curr_date_start = datetime.datetime(start_date.year, start_date.month,
start_date.day, 0, 0)
            curr_date_end = datetime.datetime(start_date.year, start_date.month,
start_date.day, 23, 59)
            patient_activities = PatientActivity.objects.filter(patient = patient,
done_date__gte = curr_date_start, done_date__lte = curr_date_end)
            duration = None
            times = patient_activities.values()[0]
            if times.duration:
                times = False
            else:
                times = True
            for patient_activity in patient_activities:
                if not duration and times:
                    duration = 1
                elif not duration and not times:
                    duration = patient_activity.duration
                elif duration and times:
                    duration = duration + 1
                else:
                    duration = duration + patient_activity.duration
            activities.append(duration)
            start_date = start_date + datetime.timedelta(days=1)
        return render(request, "ivmood/patientactmoods.html", {"moods": moods,
"activities": activities })
    else:
        return render(request, "ivmood/patientactmoods.html", {"moods": [], "activities":
activities })

```

```

def patient_medicines(request, medicine = None, start = None, end = None):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if medicine:
        try:

```

```

        medicine = Medicine.objects.get(id = medicine)
    except Medicine.DoesNotExist:
        medicine = None
    if patient:
        if not start:
            start_date = None
        else:
            start_parts = start.split("-")
            year = int(start_parts[2])
            month = int(start_parts[1] )
            day = int(start_parts[0] )
            start_date = datetime.datetime(year, month, date, 0, 0)
        if not end:
            end_date = None
        else:
            end_parts = end.split("-")
            year = int(start_end[2])
            month = int(start_end[1])
            day = int(start_end[0])
            end_date = datetime.datetime(year, month, date, 23, 59)
        patient_medicines = PatientMedicine.objects.filter(patient = patient)
        if start_date:
            patient_medicines = patient_medicines.filter(date__gte = start_date)
        if end_date:
            patient_medicines = patient_medicines.filter(date__lte = end_date)
        if medicine:
            patient_medicines = patient_medicines.filter(medicine = medicine)
        patient_medicines = patient_medicines.order_by("-date")
        medicines = Medicine.objects.all()
        return render(request, "ivmood/patientmedicines.html", {"patient_medicines":
patient_medicines, "medicines": medicines })
    else:
        medicines = Medicine.objects.all()
        return render(request, "ivmood/patientmedicines.html", {"patient_medicines": [],
"medicines": medicines })

def patient_activities(request, activity = None, activity_variant = None, start = None,
end = None):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:

```

```

    patient = None
if activity:
    try:
        activity = Activity.objects.get(id = activity)
    except Activity.DoesNotExist:
        activity = None
elif activity_variant:
    try:
        activity_variant = Activityvariant.objects.get(id = activity_variant)
    except Activityvariant.DoesNotExist:
        activity_variant = None
if patient:
    if not start:
        start_date = None
    else:
        start_parts = start.split("-")
        year = int(start_parts[2])
        month = int(start_parts[1] )
        day = int(start_parts[0] )
        start_date = datetime.datetime(year, month, date, 0, 0)
    if not end:
        end_date = None
    else:
        end_parts = end.split("-")
        year = int(start_end[2])
        month = int(start_end[1])
        day = int(start_end[0])
        end_date = datetime.datetime(year, month, date, 23, 59)
    if activity_variant:
        activities = PatientActivity.objects.filter(patient = patient, activityvariant =
activity_variant).prefetch_related("answers")
    else:
        activities = PatientActivity.objects.filter(patient = patient,
activity_variant__activity = activity).prefetch_related("answers")
    if start_date:
        activities = activities.filter(done_date__gte = start_date)
    if end_date:
        activities = activities.filter(done_date__lte = end_date)
    activities = activities.order_by("-date")
    all_activities = Activity.objects.all()
    all_activity_variants = Activityvariant.objects.all()

```

```

        return render(request, "ivmood/patientactivities.html", { "patient_activities":
activities, "activities": all_activities, "activity_variants": all_activity_variants })
    else:
        all_activities = Activity.objects.all()
        all_activity_variants = Activityvariant.objects.all()
        return render(request, "ivmood/patientactivities.html", { "patient_activities": [],
"activities": all_activities, "activity_variants": all_activity_variants })

def patient_tests(request, mentaltest = None, start = None, end = None):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if mentaltest:
        try:
            mentaltest = Mentaltest.objects.get(id = mentaltest)
        except Mentaltest.DoesNotExist:
            mentaltest = None
    if patient:
        if not start:
            start_date = None
        else:
            start_parts = start.split("-")
            year = int(start_parts[2])
            month = int(start_parts[1] )
            day = int(start_parts[0] )
            start_date = datetime.datetime(year, month, date, 0, 0)
        if not end:
            end_date = None
        else:
            end_parts = end.split("-")
            year = int(start_end[2])
            month = int(start_end[1])
            day = int(start_end[0])
            end_date = datetime.datetime(year, month, date, 23, 59)
        if mentaltest:
            mentaltests = PatientMentaltest.objects.filter(patient = patient, mentaltest =
mentaltest).prefetch_related("answers")
        else:
            mentaltests = PatientMentaltest.objects.filter(patient =
patient).prefetch_related("answers")

```

```

    if start_date:
        mentaltests = mentaltests.filter(date__gte = start_date)
    if end_date:
        mentaltests = mentaltests.filter(date__lte = end_date)
    mentaltests = mentaltests.order_by("-date")
    tests = Mentaltest.objects.all()
    return render(request, "ivmood/patienttests.html", {"patient_mentaltests":
mentaltests, "mentaltests": tests })
else:
    tests = Mentaltest.objects.all()
    return render(request, "ivmood/patienttests.html", {"patient_mentaltests":
mentaltests, "mentaltests": tests })

```

```

def patient_analyses(request, start = None, end = None):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if patient:
        if not start:
            start_date = None
        else:
            start_parts = start.split("-")
            year = int(start_parts[2])
            month = int(start_parts[1] )
            day = int(start_parts[0] )
            start_date = datetime.datetime(year, month, date, 0, 0)
        if not end:
            end_date = None
        else:
            end_parts = end.split("-")
            year = int(start_end[2])
            month = int(start_end[1])
            day = int(start_end[0])
            end_date = datetime.datetime(year, month, date, 23, 59)
        patient_analyses = Analyse.objects.filter(patient = patient)
        if start_date:
            patient_analyses = patient_analyses.filter(date__gte = start_date)
        if end_date:
            patient_analyses = patient_analyses.filter(date__lte = end_date)
        patient_analyses = patient_analyses.order_by("-date")

```

```

        return render(request, "ivmood/patientanalyses.html", {"patient_analyses ":
patient_analyses })
    else:
        return render(request, "ivmood/patientmoods.html", {"patient_analyses": [] })

def patient_history(request, patient, start = None, end = None):
    try:
        patient = Patient.objects.get(user__username = patient)
    except Patient.DoesNotExist:
        patient = None
    try:
        doctor = Doctor.objects.get(user = request.user)
    except Doctor.DoesNotExist:
        doctor = None
    if not patient or not doctor:
        return render(request, "ivmood/patienthistory.html", {"patient_analyses": [],
"patient_activities": [], "patient_tests": [], "patient_medicines": [], "patient_moods":
patient_moods })
        consults = Consult.objects.filter(doctor = doctor, patient = patient, closed = False)
        consults = consults.exclude(prescription__isnull = False)
        if not consults:
            return render(request, "ivmood/patienthistory.html", {"patient_analyses": [],
"patient_activities": [], "patient_tests": [], "patient_medicines": [], "patient_moods":
patient_moods })
    else:
        if not start:
            start_date = None
        else:
            start_parts = start.split("-")
            year = int(start_parts[2])
            month = int(start_parts[1] )
            day = int(start_parts[0] )
            start_date = datetime.datetime(year, month, date, 0, 0)
        if not end:
            end_date = None
        else:
            end_parts = end.split("-")
            year = int(start_end[2])
            month = int(start_end[1])
            day = int(start_end[0])
            end_date = datetime.datetime(year, month, date, 23, 59)

```

```

    patient_analyses = Analyse.objects.filter(patient = patient)
    patient_tests = PatientMentaltest.objects.filter(patient =
patient).prefetch_related("answers")
    patient_activities = PatientActivity.objects.filter(patient =
patient).prefetch_related("answers")
    patient_medicines = PatientMedicine.objects.filter(patient = patient)
    patient_moods = PatientMood.objects.filter(patient = patient)
    if start_date:
        patient_analyses = patient_analyses.filter(date__gte = start_date)
        patient_tests = patient_tests.filter(date__gte = start_date)
        patient_activities = patient_activities.filter(date__gte = start_date)
        patient_medicines = patient_medicines.filter(date__gte = start_date)
        patient_moods = patient_moods.filter(date__gte = start_date)
    if end_date:
        patient_analyses = patient_analyses.filter(date__lte = end_date)
        patient_tests = patient_tests.filter(date__lte = end_date)
        patient_activities = patient_activities.filter(date__lte = end_date)
        patient_medicines = patient_medicines.filter(date__lte = end_date)
        patient_moods = patient_moods.filter(date__lte = end_date)
    patient_analyses = patient_analyses.order_by("-date")
    patient_tests = patient_tests.order_by("-date")
    patient_activities = patient_activities.order_by("-date")
    patient_medicines = patient_medicines.order_by("-date")
    patient_moods = patient_moods.order_by("date")
    return render(request, "ivmood/patienthistory.html", {"patient_analyses":
patient_analyses, "patient_activities": patient_activities, "patient_tests": patient_tests,
"patient_medicines": patient_medicines, "patient_moods": patient_moods })

```

```

def activities(request):
    activities = Activity.objects.all().prefetch_related("variants")
    return render(request, "ivmood/activities.html", {"activities": activities })

```

```

def mentaltests(request):
    tests = Mentaltest.objects.all().prefetch_related("results")
    return render(request, "ivmood/tests.html", {"tests": tests })

```

```

def patient_communities(request):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None

```

```

    if not patient:
        return render(request, "ivmood/patient_communities.html", {"approved": [],
"new": []})
        communities_aproved = CommunityPatient.objects.filter(patient = patient,
approved = True).distinct("community")
        communities_new = CommunityPatient.objects.filter(patient = patient, approved =
False).distinct("community")
        return render(request, "ivmood/patient_communities.html", {"approved":
communities_aproved, "new": communities_new})

def challenges(request):
    challenges = Challenge.objects.filter(patient =
patient).select_related("typicalchallenge").prefetch_related("typicalchallenge__activityvariants
")
    return render(request, "ivmood/challenges.html", {"challenges": challenges})

def typical_challenges(request):
    challenges = Challenge.objects.filter(patient =
patient).select_related("typicalchallenge").prefetch_related("typicalchallenge__activityvariants
")
    typical_challenges = TypicalChallenge.objects.filter(typical =
True).prefetch_related("activityvariants").order_by("name")
    all_challenges = []
    for typical_challenge in typical_challenges:
        if typical_challenge not in challenges:
            all_challenges.append(typical_challenge)
    return render(request, "ivmood/challenges.html", {"challenges": all_challenges})

def doctor_consults(request):
    try:
        doctor = Doctor.objects.get(user = request.user)
    except Doctor.DoesNotExist:
        doctor = None
    if doctor:
        consults = Consult.objects.filter(doctor = doctor).order_by("-create_date")
    return render(request, "ivmood/consults.html", {"consults": consults })

def patient_consults(request):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:

```

```

        patient = None
    if patient:
        consults = Consult.objects.filter(patient = patient).order_by("-create_date")
    return render(request, "ivmood/patientconsults.html", {"consults": consults })

def doctor_profile(request, doctor):
    try:
        doctor = Doctor.objects.get(user__username =
doctor).select_related("specialization")
    except Doctor.DoesNotExist:
        doctor = None
    return render(request, "ivmood/doctor.html", {"doctor": doctor})

def find_doctor(request, specialization = None, name = None, review = None, years =
None, minyear= None):
    if specialization:
        try:
            specialization = Specialization.objects.get(id = specialization)
        except Specialization.DoesNotExist:
            specialization = None
        doctors = Doctor.objects.all()
        if specialization:
            doctors = doctors.filter(specialization = specialization)
    if name:
        if " " in name:
            name_parts = name.split(" ")
            name_left = name_parts[0]
            name_right = name_parts[1]
            doctors = doctors.filter(Q(user__first_name__icontains = name_left,
user__last_name__icontains = name_right) | Q(user__first_name__icontains = name_right,
user__last_name__icontains = name_left))
        else:
            doctors = doctors.filter(user__last_name__icontains = name)
    if review:
        doctors = doctors.filter(review__gte = review)
    if years:
        now = datetime.datetime.now()
        last_year = now - datetime.timedelta(years = years)
        doctors = doctors.filter(birth_date__gte = last_year)
    if minyear:
        now = datetime.datetime.now()

```

```

        last_year = now - datetime.timedelta(years = minyear)
        doctors = doctors.filter(birth_date__lte = last_year)
        doctors = doctors.order_by("-review")
        return render(request, "ivmood/finddoctors.html", {"doctors": doctors })

def community_moods(request, community, start = None, end = None):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if not patient:
        return render(request, "ivmood/community_moods.html", {"moods": []})
    try:
        community = Community.objects.get(id = community)
    except Community.DoesNotExist:
        community = None
    if not community:
        return render(request, "ivmood/community_moods.html", {"moods": []})
    check = CommunityPatient.objects.filter(patient = patient, community = community,
approved = True)
    if check:
        patients = CommunityPatient.objects.filter(community = community, approved =
True)
    else:
        return render(request, "ivmood/community_moods.html", {"moods": []})
    start_no = False
    if start:
        start_parts = start.split("-")
        year = int(start_parts[2])
        month = int(start_parts[1] )
        day = int(start_parts[0] )
        start_date = datetime.datetime(year, month, date, 0, 0)
    else:
        start_no = True
    end_no = False
    if end:
        end_parts = end.split("-")
        year = int(start_end[2])
        month = int(start_end[1])
        day = int(start_end[0])
        end_date = datetime.datetime(year, month, date, 23, 59)

```

```

else:
    end_no = True
if end_no and not start_no:
    end_date = start_date + datetime.timedelta(months = 1)
elif not end_no and start_no:
    start_date = end_date - datetime.timedelta(months = 1)
elif end_no and start_no:
    start_date = datetime.datetime.now()
    end_date = start_date + datetime.timedelta(months = 1)
moods = []
for patient in patients:
    patient_moods = {}
    patient_moods["patient"] = patient
    patient_moods["moods"] = PatientMood.objects.filter(patient = patient,
date__gte = start_date, date__lte = end_date, shared = True).order_by("date")
    moods.append(patient_moods)
return render(request, "ivmood/patientmoods.html", {"moods": moods })

def activityvariants_moods(request, start = None, end = None, mood = None):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if not patient:
        return render(request, "ivmood/activityvariants_moods.html", {"activities": []})
    if not start:
        start_date = None
    else:
        start_parts = start.split("-")
        year = int(start_parts[2])
        month = int(start_parts[1] )
        day = int(start_parts[0] )
        start_date = datetime.datetime(year, month, date, 0, 0)
    if not end:
        end_date = None
    else:
        end_parts = end.split("-")
        year = int(start_end[2])
        month = int(start_end[1])
        day = int(start_end[0])
        end_date = datetime.datetime(year, month, date, 23, 59)

```

```

activities = PatientActivity.objects.filter(patient = patient)
moods = PatientMood.objects.filter(patient = patient)
if start_date:
    activities = activities.filter(done_date__gte = start_date)
    moods = moods.filter(date__gte = start_date)
if end_date:
    activities = activities.filter(done_date__gte = end_date)
    moods = moods.filter(date__gte = end_date)
if mood:
    mood_variants = [[mood]]
else:
    mood_variants = [[1,2], [4,5]]
activity_variants = activities.distinct("activityvariant")
activities_calculated = []
for mood in mood_variants:
    curr = []
    for activity in activity_variants:
        info = {}
        info["activity"] = activity
        act_variants = activities.filter(activityvariant = activity)
        res = 0
        for activity_variant in act_variants:
            mood_curr = moods.filter(date__gte =
activity_variant.done_date).order_by("date")[0]
            if mood_curr.mood in mood:
                res = res + 1
            info["situations"] = res
            curr.append(info)
        activities_calculated.append(curr)
    return render(request, "ivmood/activityvariants_moods.html", {"activities":
activities})

```

```

def activity_moods(request, start = None, end = None, mood = None):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if not patient:
        return render(request, "ivmood/activity_moods.html", {"activities": []})
    if not start:
        start_date = None

```

```

else:
    start_parts = start.split("-")
    year = int(start_parts[2])
    month = int(start_parts[1] )
    day = int(start_parts[0] )
    start_date = datetime.datetime(year, month, date, 0, 0)
if not end:
    end_date = None
else:
    end_parts = end.split("-")
    year = int(start_end[2])
    month = int(start_end[1])
    day = int(start_end[0])
    end_date = datetime.datetime(year, month, date, 23, 59)
activities = PatientActivity.objects.filter(patient = patient)
moods = PatientMood.objects.filter(patient = patient)
if start_date:
    activities = activities.filter(done_date__gte = start_date)
    moods = moods.filter(date__gte = start_date)
if end_date:
    activities = activities.filter(done_date__gte = end_date)
    moods = moods.filter(date__gte = end_date)
if mood:
    mood_variants = [[mood]]
else:
    mood_variants = [[1,2], [4,5]]
activity_variants = activities.distinct("activity")
activities_calculated = []
for mood in mood_variants:
    curr = []
    for activity in activity_variants:
        info = {}
        info["activity"] = activity
        act_variants = activities.filter(activityvariant__activity = activity)
        res = 0
        for activity_variant in act_variants:
            mood_curr = moods.filter(date__gte =
activity_variant.done_date).order_by("date")[0]
            if mood_curr.mood in mood:
                res = res + 1
            info["situations"] = res

```

```

        curr.append(info)
        activities_calculated.append(curr)
    return render(request, "ivmood/activity_moods.html", {"activities": activities})

def patient_goals(request):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if not patient:
        return render(request, "ivmood/patient_goals.html", {"goals": []})
    goals = Dailygoal.objects.filter(patient = patient, state =
True).prefetch_related("activityvariant").prefetch_related("activityvariant__activity")
    return render(request, "ivmood/activity_moods.html", {"goals": goals })

def control_mood(request):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if not patient:
        return render(request, "ivmood/control_mood.html", {"form": None})
    if request.user.is_active:
        if request.method == "POST":
            mood_form = PatientMoodForm(request.POST)
            if mood_form.is_valid():
                patient_mood = mood_form.save(commit=False)
                patient_mood.patient = patient
                patient_mood.save()
                return redirect("index")
            else:
                mood_form = PatientMoodForm()
        else:
            mood_form = None
    return render(request, "ivmood/control_mood.html", {"form": mood_form })

def ask_consult(request):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None

```

```

if not patient:
    return render(request, "ivmood/ask_consult.html", {"form": None})
if request.user.is_active:
    if request.method == "POST":
        consult_form = ConsultForm(request.POST)
        if consult_form.is_valid():
            consult = consult_form.save(commit=False)
            consult.patient = patient
            consult.save()
            return redirect("consults")
        else:
            consult_form = ConsultForm()
    else:
        consult_form = None
return render(request, "ivmood/ask_consult.html", {"form": consult_form })

def create_community(request):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if not patient:
        return render(request, "ivmood/ask_consult.html", {"form": None})
    if request.user.is_active:
        if request.method == "POST":
            community_form = CommunityForm(request.POST)
            if community_form.is_valid():
                community = community_form.save(commit=False)
                community.save()
                patient_include = CommunityPatient(community = community, patient =
patient, approved = True)
                patient_include.save()
                return redirect("communities")
            else:
                community_form = CommunityForm()
        else:
            community_form = None
    return render(request, "ivmood/ask_consult.html", {"form": consult_form })

def fix_activity(request, activity, step = None):
    challenges_done = []

```

```

try:
    patient = Patient.objects.get(user = request.user)
except Patient.DoesNotExist:
    patient = None
if not patient:
    return render(request, "ivmood/fix_activity.html", {"activity_form": None,
"step_form": None, "activity": None, "challenges": [] })
try:
    activity = Activityvariant.objects.get(id = activity).prefetch_related("steps")
except Activityvariant.DoesNotExist:
    activity = None
if not activity:
    return render(request, "ivmood/fix_activity.html", {"activity_form": None,
"step_form": None, "activity": None, "challenges": []})
if step:
    try:
        step = Activityvariantstep.objects.get(activityvariant =
activity).oder_by("id")[step]
    except Activityvariantstep.DoesNotExist:
        step = None
if request.user.is_active:
    if request.method == "POST":
        done = False
        if not step:
            activity_form = PatientActivityForm(request.POST)
            if activity_form.is_valid():
                ractivity = activity_form.save(commit=False)
                ractivity.activityvariant = activity
                ractivity.patient = patient
                ractivity.save()
                challenges = Challenge.objects.filter(patient =
patient).select_related("typicalchallenge").prefetch_filter("typicalchallenge__activityvariants")
                now = datetime.datetime.now()
                start = datetime.datetime(now.year, now.month, now.day, 0, 0)
                end = datetime.datetime(now.year, now.month, now.day, 23, 59)
                for challenge in challenges:
                    variants = challenge.typicalchallenge.activityvariants
                    res = 0
                    for variant in variants:
                        activities = PatientActivity.objects.filter(patient = patient,
activityvariant = variant, done_date__gte = start, done_date__lte = end)

```

```

        if activities:
            res = res + 1
            number = variants.values("id").annotate(Count("id"))
            if res == number:
                challenges_done.append(challenge)
            step = Activityvariantstep.objects.get(activityvariant =
activity).oder_by("id")[0]
            done = True
        if step.answered == True:
            activitystep_form = PatientActivitystepForm(request.POST)
            if activitystep_form.is_valid():
                activitystep = activitystep_form.save(commit=False)
                activitystep.patient = patient
                activitystep.activityvariantstep = step
                activitystep.save()
                done = True
        if done:
            return redirect("activityhistory")
    else:
        if not step:
            activity_form = PatientActivityForm(request.POST)
            step = Activityvariantstep.objects.get(activityvariant =
activity).oder_by("id")[0]
            if step.answered == True:
                step_form = PatientActivitystepForm(request.POST)
            else:
                activity_form = None
                step_form = None
            return render(request, "ivmood/fix_activity.html", {"activity_form": activity_form,
"step_form": step_form, "activity": activity, "challenges": challenges_done })

def fix_medicine(request):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if not patient:
        return render(request, "ivmood/fix_medicine.html", {"form": None})
    if request.user.is_active:
        if request.method == "POST":
            medicine_form = PatientMedicineForm(request.POST)

```

```

        if medicine_form.is_valid():
            medicine = medicine_form.save(commit=False)
            medicine.patient = patient
            medicine.save()
            return redirect("medicines")
        else:
            medicine_form = PatientMedicineForm()
    else:
        medicine_form = None
    return render(request, "ivmood/fix_medicine.html", {"form": medicine_form })

def fix_analyse(request):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if not patient:
        return render(request, "ivmood/fix_analyse.html", {"form": None})
    if request.user.is_active:
        if request.method == "POST":
            analyse_form = AnalyseForm(request.POST)
            if analyse_form.is_valid():
                analyse = analyse_form.save(commit=False)
                analyse.patient = patient
                analyse.save()
                analyse.save_m2m()
                return redirect("analyses")
            else:
                analyse_form = AnalyseForm ()
        else:
            analyse_form = None
    return render(request, "ivmood/fix_analyse.html", {"form": analyse_form })

def create_typical_challenge(request):
    if request.user.is_staff:
        if request.method == "POST":
            tchallenge_form = TypicalChallengeForm(request.POST)
            if tchallenge_form.is_valid():
                tchallenge = tchallenge_form.save(commit=False)
                tchallenge.save()
                tchallenge.save_m2m()

```

```

        return redirect("typicalchallenges")
    else:
        tchallenge_form = TypicalChallengeForm()
    else:
        tchallenge_form = None
    return render(request, "ivmood/typicalchallenge.html", {"form": tchallenge_form })

```

```

def create_challenge(request):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if not patient:
        return render(request, "ivmood/challenge.html", {"form": None})
    if request.user.is_active:
        if request.method == "POST":
            challenge_form = ChallengeForm(request.POST)
            if challenge_form.is_valid():
                challenge = challenge_form.save(commit=False)
                challenge.patient = patient
                challenge.save()
                return redirect("index")
            else:
                challenge_form = ChallengeForm()
        else:
            challenge_form = None
    return render(request, "ivmood/challenge.html", {"form": challenge_form })

```

```

def add_activity(request, activity):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if not patient:
        return render(request, "ivmood/addactivity.html", {"form": None})
    try:
        activityvariant = Activityvariant.objects.get(id = activity)
    except Activityvariant.DoesNotExist:
        activityvariant = None
    if not patient:
        return render(request, "ivmood/addactivity.html", {"form": None})

```

```

if request.user.is_active:
    if request.method == "POST":
        goal_form = DailygoalForm(request.POST)
        if goal_form.is_valid():
            goal = goal_form.save(commit=False)
            goal.patient = patient
            goal.activityvariant = activityvariant
            goal.save()
            return redirect("index")
        else:
            goal_form = DailygoalForm()
    else:
        goal_form = None
return render(request, "ivmood/addactivity.html", {"form": goal_form })

def create_activityvariant(request):
    if request.user.is_staff:
        if request.method == "POST":
            activityvariant_form = ActivityvariantForm(request.POST)
            if activityvariant_form.is_valid():
                activityvariant = activityvariant_form.save(commit=False)
                activityvariant.save()
                activityvariant.save_m2m()
                return redirect("index")
            else:
                activityvariant_form = ActivityvariantForm()
        else:
            activityvariant_form = None
    return render(request, "ivmood/createactivityvariant.html", {"form":
activityvariant_form })

def edit_activityvariant(request, id):
    try:
        activityvariant = Activityvariant.objects.get(id = id)
    except Activityvariant.DoesNotExist:
        patient = None
    if not patient:
        return render(request, "ivmood/createactivityvariant.html", {"form": None})
    if request.user.is_staff:
        if request.method == "POST":

```

```

        activityvariant_form = ActivityvariantForm(request.POST, instance =
activityvariant)
        if activityvariant_form.is_valid():
            activityvariant_form.save()
            return redirect("index")
        else:
            activityvariant_form = ActivityvariantForm (None, instance = activityvariant)
        else:
            activityvariant_form = None
    return render(request, "ivmood/createactivityvariant.html", {"form":
activityvariant_form })

```

```

def edit_consult(request, id):
    try:
        doctor = Doctor.objects.get(user = request.user)
    except Doctor.DoesNotExist:
        doctor = None
    try:
        consult = Consult.objects.get(id = id)
    except Consult.DoesNotExist:
        consult = None
    if not doctor or not consult or consult.doctor != doctor:
        return render(request, "ivmood/consulting.html", {"form": None})
    if request.method == "POST":
        consult_form = DoctorConsultForm(request.POST, instance = consult)
        if consult_form.is_valid():
            consult_form.save()
            return redirect("doctorconsults")
        else:
            consult_form = DoctorConsultForm(None, instance = consult)
    else:
        consult_form = None
    return render(request, "ivmood/consulting.html", {"form": consult_form })

```

```

def assess(request, consult, point):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    try:
        consult = Consult.objects.get(id = consult).prefetch_related("doctor")

```

```

except Consult.DoesNotExist:
    consult = None
if not patient or not consult or consult.patient != patient:
    return redirect("consults")
if point >= 1 and point <=5:
    consult.review = point
    consult.save()
    consults = Consult.objects.filter(doctor = consult.doctor, review__gte = 1,
review__lte = 5)
    sum = 0
    q = 0
    doctor = consult.doctor
    for cons in consults:
        sum = sum + cons.review
        q = q + 1
    doctor.review = sum / q
    doctor.save()
return redirect("consults")

```

```

def delete_goal(request, goal):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if patient:
        try:
            goal = Dailygoal.objects.get(id = goal, patient = patient)
        except Dailygoal.DoesNotExist:
            goal = None
    if goal:
        goal.delete()
    return redirect("index")

```

```

def delete_challenger(request, challenge):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if patient:
        try:
            challenge = Challenge.objects.get(id = challenge, patient = patient)

```

```

except Challenge.DoesNotExist:
    challenge = None
if challenge:
    challenge.delete()
return redirect("index")

def delete_community(request, community, patient = None):
    patient2 = None
    try:
        if not patient:
            patient = Patient.objects.get(user = request.user)
        else:
            patient = Patient.objects.get(user__username = patient)
            patient2 = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    if patient:
        try:
            if not patient2:
                community = CommunityPatient.objects.get(community__id = community,
patient = patient)
            else:
                community = CommunityPatient.objects.get(community__id = community,
patient = patient2)
            if community:
                community = CommunityPatient.objects.get(community__id = community,
patient = patient)
            else:
                community = None
        except CommunityPatient.DoesNotExist:
            community = None
    if community:
        community.delete()
    return redirect("index")

def add_to_community(request, community, patient):
    try:
        patient2 = Patient.objects.get(patient = patient)
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None

```

```

try:
    community = Community.objects.get(id = community)
except Consult.DoesNotExist:
    community = None
    communitypatient = CommunityPatient.objects.filter(community = community,
patient = patient)
    communitypatient2 = CommunityPatient.objects.filter(community = community,
patient = patient2)
    if not patient or not community or not community or not communitypatient or
community2:
        return redirect("index")
    communitypatient = CommunityPatient(community = community, patient =
patient2, approved = False)
    communitypatient.save()
    return redirect("index")

def create_test(request):
    if request.user.is_staff:
        if request.method == "POST":
            test_form = MentaltestForm(request.POST)
            if test_form.is_valid():
                test = test_form.save()
                test.save_m2m()
                return redirect("tests")
            else:
                test_form = MentaltestForm ()
        else:
            test_form = None
    return render(request, "ivmood/creatementaltest.html", {"form": test_form })

def edit_test(request, id):
    try:
        mentaltest = Mentaltest.objects.get(id =
id).prefetch_related("questions").prefetch_related("questions_answers")
    except Mentaltest.DoesNotExist:
        mentaltest = None
    if not mentaltest:
        return render(request, "ivmood/creatementaltest.html", {"form": None})
    if request.user.is_staff:
        if request.method == "POST":
            mentaltest_form = MentaltestForm(request.POST, instance = mentaltest)

```

```

        if mentaltest_form.is_valid():
            mentaltest_form.save()
            return redirect("test", mentaltest.id)
        else:
            mentaltest_form = MentaltestForm (None, instance = mentaltest)
    else:
        mentaltest_form = None
    return render(request, "ivmood/creatementaltest.html", {"form": mentaltest_form
})

```

```

def create_test_question(request):
    if request.user.is_staff:
        if request.method == "POST":
            test_form = MentaltestQuestionForm(request.POST)
            if test_form.is_valid():
                test = test_form.save()
                return redirect("test", test.id)
            else:
                test_form = MentaltestQuestionForm()
        else:
            test_form = None
    return render(request, "ivmood/createtestquestion.html", {"form": test_form })

```

```

def edit_testresult(request, id):
    try:
        mentaltestresult = MentaltestResult.objects.get(id = id)
    except MentaltestResult.DoesNotExist:
        mentaltestresult = None
    if not mentaltestresult:
        return render(request, "ivmood/creatementaltestresult.html", {"form": None})
    if request.user.is_staff:
        if request.method == "POST":
            mentaltestresult_form = MentaltestResultForm(request.POST, instance =
mentaltestresult)
            if mentaltestresult_form.is_valid():
                mentaltestresult_form.save()
                return redirect("test", mentaltestresult.mentaltest.id)
            else:
                mentaltestresult_form = MentaltestResultForm(None, instance =
mentaltestresult)
        else:

```

```

        mentaltestresult_form = None
        return render(request, "ivmood/creatementaltestresult.html", {"form":
mentaltestresult_form })

def edit_testquestion(request, id):
    try:
        mentaltestquestion = MentaltestQuestion.objects.get(id =
id).prefetch_related("answers")
    except MentaltestQuestion.DoesNotExist:
        mentaltestquestion = None
    if not mentaltestquestion:
        return render(request, "ivmood/creatementaltestquestion.html", {"form": None})
    if request.user.is_staff:
        if request.method == "POST":
            mentaltestquestion_form = MentaltestQuestionForm(request.POST, instance =
mentaltestquestion)
            if mentaltestquestion_form.is_valid():
                mentaltestquestion_form.save()
                return redirect("test", mentaltestquestion.mentaltest.id)
            else:
                mentaltestquestion_form = MentaltestQuestionForm(None, instance =
mentaltestquestion)
            else:
                mentaltestquestion_form = None
        return render(request, "ivmood/creatementaltestquestion.html", {"form":
mentaltestquestion_form })

def create_test_answer(request):
    if request.user.is_staff:
        if request.method == "POST":
            test_form = MentaltestQuestionAnswerForm(request.POST)
            if test_form.is_valid():
                test = test_form.save()
                return redirect("test", test.mentaltestquestion.id)
            else:
                test_form = MentaltestQuestionAnswerForm()
        else:
            test_form = None
        return render(request, "ivmood/createtestquestionanswer.html", {"form":
test_form })
    def edit_testanswer(request, id):

```

```

try:
    mentaltestanswer = MentaltestQuestionAnswer.objects.get(id = id)
except MentaltestQuestionAnswer.DoesNotExist:
    mentaltestanswer = None
if not mentaltestanswer:
    return render(request, "ivmood/creatementaltestanswer.html", {"form": None})
if request.user.is_staff:
    if request.method == "POST":
        mentaltestanswer_form = MentaltestQuestionAnswerForm(request.POST,
instance = mentaltestanswer)
        if mentaltestanswer_form.is_valid():
            mentaltestanswer_form.save()
            return redirect("test", mentaltestanswer.mentaltestquestion.mentaltest.id)
        else:
            mentaltestanswer_form = MentaltestQuestionAnswerForm(None, instance =
mentaltestanswer)
    else:
        mentaltestanswer_form = None
    return render(request, "ivmood/creatementaltestasnwser.html", {"form":
mentaltestanswer_form })

def save_test_answer(request, mentaltest):
    try:
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    try:
        mentaltest = Mentaltest.objects.get(id = mentaltest)
    except Mentaltest.DoesNotExist:
        mentaltest = None
    if patient and mentaltest:
        current_test = PatientMentaltest.objects.filter(patient = patient, mentaltest =
mentaltest)
        if not current_test:
            current_test = PatientMentaltest(patient = patient, mentaltest = mentaltest,
result = 0)
            current_test.save()
        if request.method == "POST":
            answer_form = PatientMentaltestQuestionAnswerForm(request.POST)
            if answer_form.is_valid():
                answer = answer_form.save(commit = False)

```

```

        now = datetime.datetime.now()
        start = datetime.datetime(now.year, now.month, now.day, 0, 0)
        end = datetime.datetime(now.year, now.month, now.day, 23, 59)
        answer_old =
PatientMentaltestQuestionAnswer.objects.filter(patientmentaltest = answer.patientmentaltest,
answer__mentaltestquestion = answer.mentaltestquestion, patientmentaltest__date__gte =
start, patientmentaltest__date__lte = end)
        if answer_old:
            answer_old.answer = answer.answer
            answer_old.save()
        else:
            answer.save()
        answers = PatientMentaltestQuestionAnswer.objects.filter(patientmentaltest
= current_test)
        res = 0
        for ans in answers:
            res = res + ans.answer.result
        current_test.result = res
        current_test.save()
        return redirect("test", test.mentaltestquestion.id)
    else:
        answer_form = PatientMentaltestQuestionAnswer()
    else:
        answer_form = None
    return render(request, "ivmood/testquestionanswer.html", {"form": answer_form })

def full_test(request, id):
    try:
        mentaltest = Mentaltest.objects.get(id = id)
    except Mentaltest.DoesNotExist:
        mentaltest = None
    if not mentaltest or not request.user.is_staff:
        return render(request, "ivmood/fulltest.html", {"test": None, "questions": [],
"results": []})
    results = MentaltestResult.objects.filter(mentaltest = mentaltest)
    questions = MentaltestQuestion.objects.filter(mentaltest =
mentaltest).prefetch_related("answers")
    return render(request, "ivmood/fulltest.html", {"test": mentaltest, "questions":
questions, "results": results})
    def approve_community(request, community):
        try:

```

```
        patient = Patient.objects.get(user = request.user)
    except Patient.DoesNotExist:
        patient = None
    try:
        community = Community.objects.get(id = community)
    except Consult.DoesNotExist:
        community = None
    if patient and community:
        try:
            community = CommunityPatient.objects.get(community = community, patient =
patient)
        except Community.DoesNotExist:
            community = None
    if community:
        community.approved = True
        community.save()
    return redirect("index")
```

ДОДАТОК В
Слайди презентації



Рисунок В.1 – Титульний слайд

Мета роботи

- Об'єктом роботи є процес моніторингу психічного здоров'я людини.
- Предметом роботи є програмне забезпечення моніторингу психічного здоров'я.
- Мета роботи – розробити програмне забезпечення моніторингу психічного здоров'я для надання допомоги людині у виявленні та розв'язанні психологічних проблем.

Рисунок В.2 – Мета роботи

Завдання роботи

- аналіз програмних засобів моніторингу психічного здоров'я і визначення концепції програми;
- проектування програмного забезпечення моніторингу психічного здоров'я;
- реалізація програмного забезпечення моніторингу психічного здоров'я.

3

Рисунок В.3 – Завдання роботи

Порівняння програмних засобів моніторингу психічного здоров'я

Критерій	Moodfit	Daylio	IVMood
Фіксування настроїв	+	+	+
Внесення виконаної діяльності	+	+	+
Інструкції з психологічних практик	+	–	+
Наявність платних функцій	+	+	–
Аналіз залежності зміни настроїв від діяльності	Платний	Платний	Безкоштовний
Відслідковування змін настроїв у спільноті	–	–	+
Консультування лікарями	–	–	+

4

Рисунок В.4 – Порівняння програмних засобів моніторингу психічного здоров'я

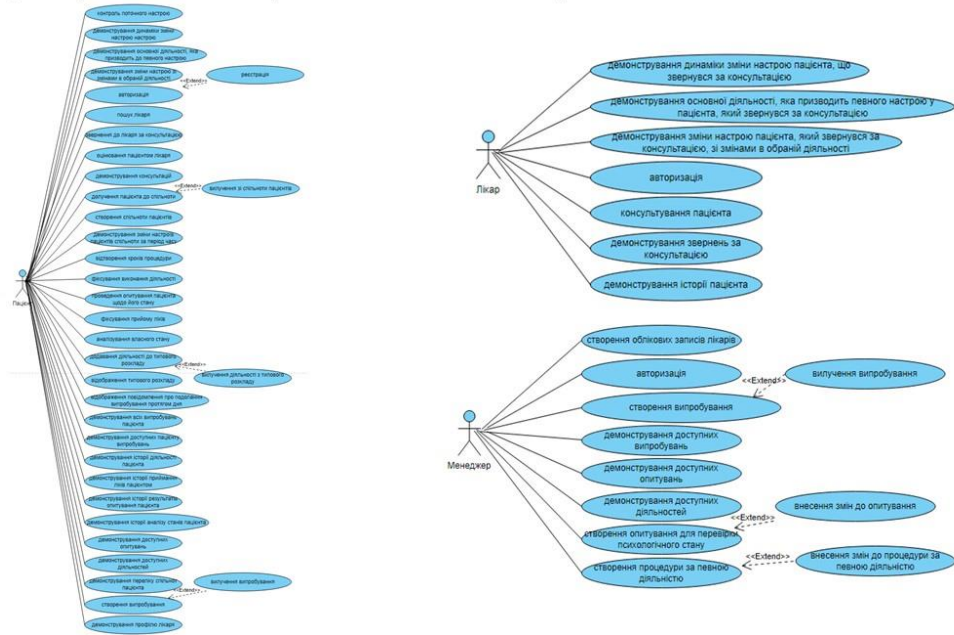
Порівняння мов C# і Python для розробки

Критерій	C#	Python
Використовувана парадигма програмування	Об'єктно-орієнтована	Мультипарадигмальна
Варіант типізації	Статична	Динамічна
Порівняний вплив на тривалість розробки	Базовий	Зменшує
Вплив на швидкість роботи програми	Зменшує	Базовий
Бібліотеки розширень	Базова	Переважає за кількістю

5

Рисунок В.5 – Порівняння мов C# і Python для розробки

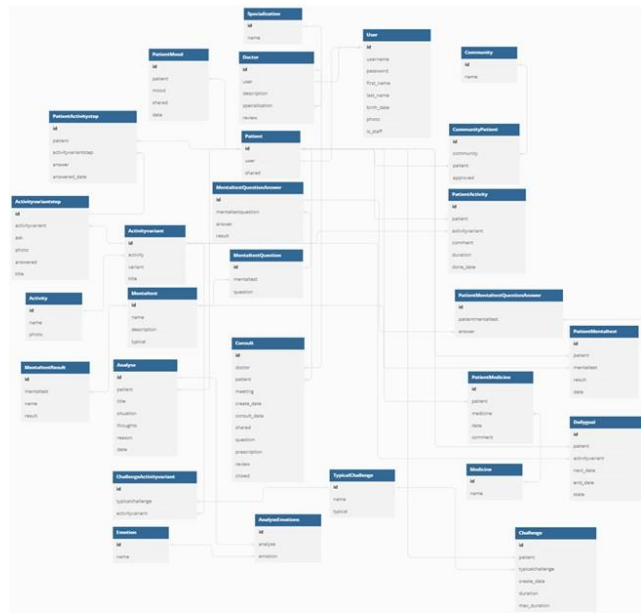
Діаграми варіантів використання системи



6

Рисунок В.6 – Діаграми варіантів використання системи

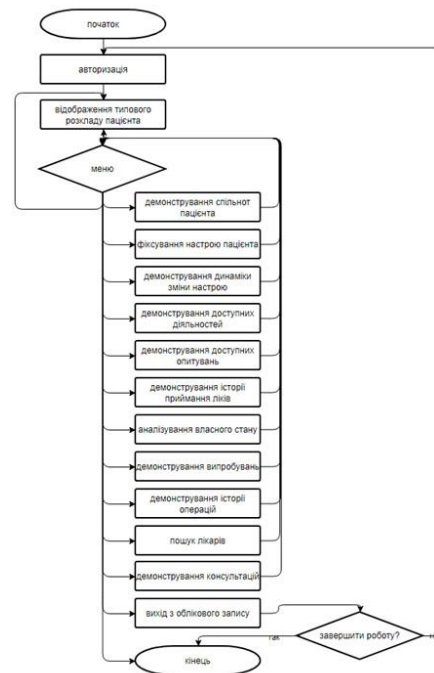
Схема бази даних



7

Рисунок В.7 – Схема БД

Схема функціонування програми для пацієнта



8

Рисунок В.8 – Схема функціонування програми для пацієнта

Фіксування настрою і типовий розклад

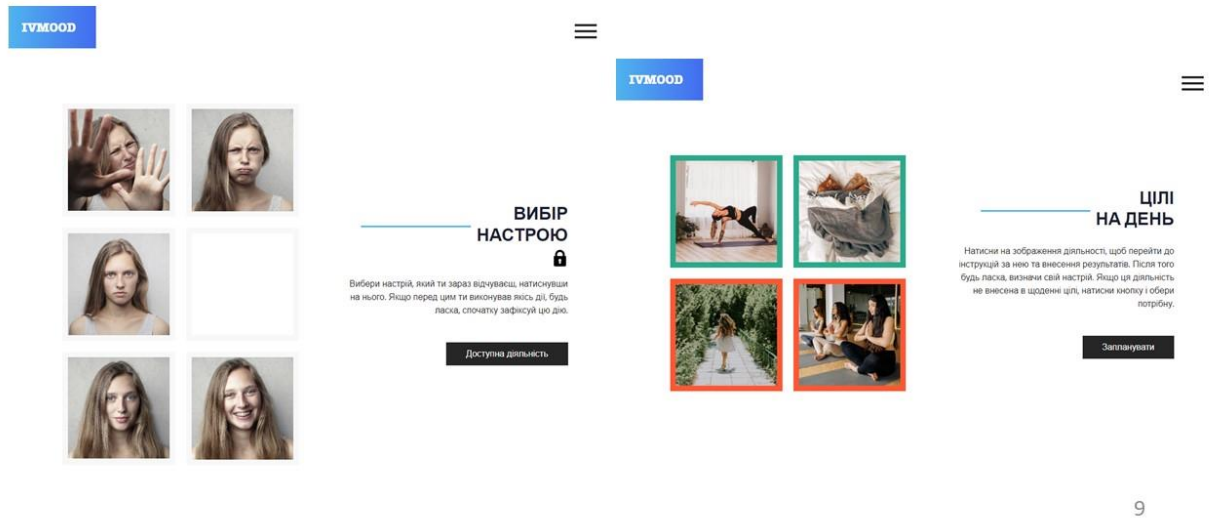


Рисунок В.9 – Фіксування настрою і типовий розклад

Робота пацієнта і лікаря з програмою

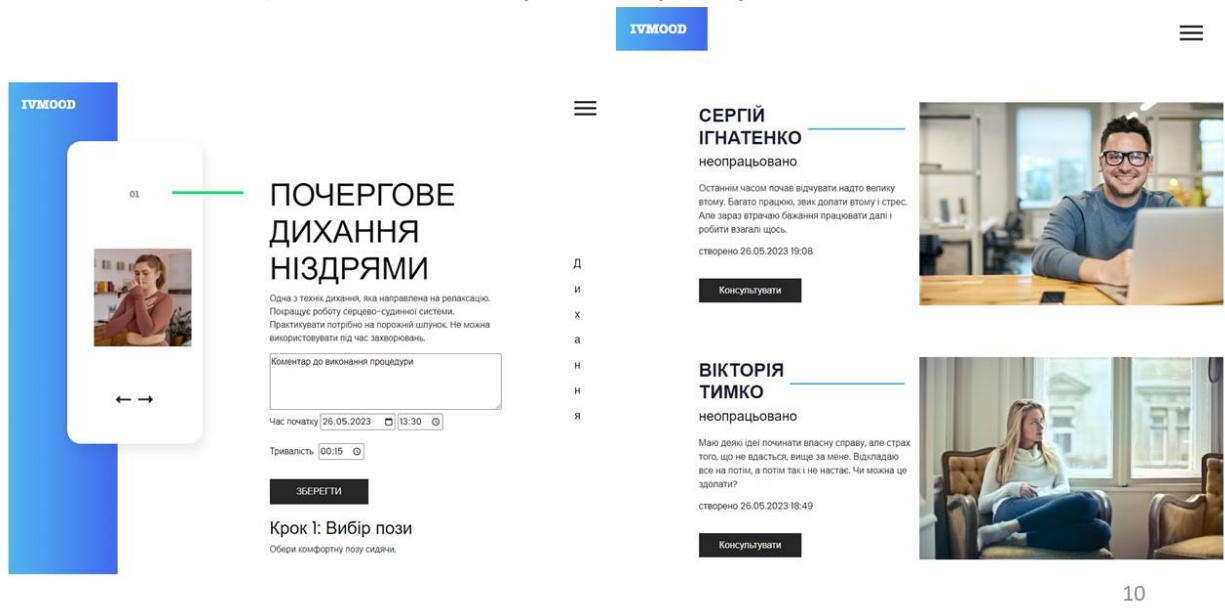
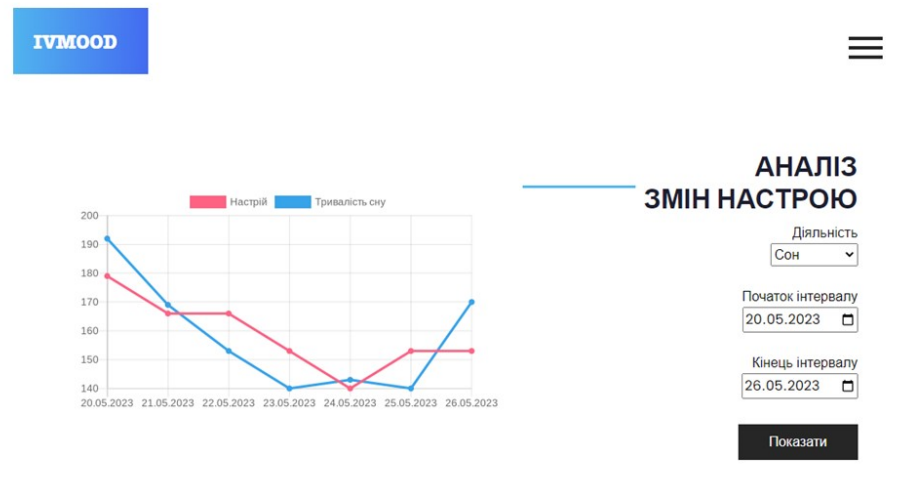


Рисунок В.10 – Робота пацієнта і лікаря з програмою

Аналіз зміни настрою за діяльністю



11

Рисунок В.11 – Аналіз зміни настрою за діяльністю

Висновки

- Програмну систему було розроблено мовами Python, HTML, CSS, JavaScript з використанням системи керування базою даних MySQL. До результатів роботи слід віднести і саму реалізовану програмну систему, і результати аналізу предметної області, де було визначено функції подібних програм і визначено відсутність взаємодії всередині спільноти, наявність платних засобів для аналізу, та результати проєктування програми.
- Розроблена програма направлена на підтримку фіксації настрою, станів, діяльності пацієнта, аналіз зміни настрою, власного стану індивідуально самим пацієнтом, моніторингу настроїв у спільноті та на базову підтримку людей, які потребують допомоги, лікарями. Програма орієнтована на те, щоб люди, які зіштовхуються з психологічними проблемами, могли їх вчасно виявити, на ранніх стадіях, та зрозуміли, що вони потребують професійної підтримки.

12

Рисунок В.12 – Висновки