Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: : «Розробка вебсистеми для моніторингу стану здоров'я

Виконав: студент 4 курсу
групи 4ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Душний Б.О.
(прізвище та ініціали)
Керівник: к.т.н. доц. каф. ПЗ Черноволик Г.О.
(прізвище та ініціали)
Рецензент: к.т.н. доц. каф. ОТ Тарновський М. Г.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ Романюк О. Н. _____
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Душному Богдану Олександровичу

1. Тема роботи – Розробка вебсистеми для моніторингу стану здоров'я.
Керівник роботи: Черноволик Галина Олександрівна, доцент кафедри програмного забезпечення, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.
2. Строк подання студентом роботи 2 червня 2025.
3. Вихідні дані до роботи: середовище розробки – PyCharm, мова розробки – Python; фреймворк Flask, система управління базою даних SQLAlchemy; операційна система – Windows.
4. Зміст текстової частини: вступ; аналіз розвитку вебсистем для моніторингу стану здоров'я; розробка структури, моделі та методу програмного продукту; розробка програмного продукту; тестування програми; висновки; список використаних джерел; додатки.
5. Перелік ілюстративного матеріалу: титульний слайд; актуальність теми; мета, об'єкт, предмет дослідження; новизна отриманих результатів; існуючі аналоги; алгоритм дій користувача; алгоритм додавання запису; розробка вебінтерфейсу; тестування програми; висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Черноволик Г.О. к.т.н., доцент. каф. ПЗ	24.03.2025	03.06.2025

7. Дата видачі завдання 24 березня 2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання та постановка задач дослідження	25.03.2025 – 30.03.2025	Виконан
2	Аналіз вхідних та вихідних даних системи	05.04.2025 – 15.04.2025	Виконан
3	Розробка моделі та методів системи	20.04.2025 – 25.04.2025	Виконан
4	Розробка інтерфейсу користувача	26.04.2025 – 02.05.2025	Виконан
5	Розробка програмних модулів системи	10.05.2025 – 18.05.2025	Виконан
6	Тестування системи	20.05.2025– 24.05.2025	Виконан
7	Оформлення матеріалів до захисту БДР	25.05.2025 – 30.05.2025	Виконан

Студент _____ **Б.О. Душ**
(підпис) (ініціали та прізв

Керівник бакалаврської кваліфікаційної роботи _____ **Г.О.Черновол**
(підпис) (ініціали та прізв

АНОТАЦІЯ

УДК 004.738.5:614.2

Душний Б. О. Розробка вебсистеми для моніторингу стану здоров’я: бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 94 с.

На укр. мові. Бібліогр. : 20 назв ; рис. : 30; табл. 5.

У бакалаврській кваліфікаційній роботі проведено аналіз стану та розвитку програмних застосунків, що використовуються для моніторингу стану здоров’я користувачів. Було проведено огляд існуючих аналогів, визначено їхні переваги та недоліки, що дозволило обґрунтувати доцільність створення власної вебсистеми.

У роботі обґрунтовано вибір мови програмування Python, фреймворку Flask та середовища розробки PyCharm для реалізації проекту. Розроблено вебсистему для моніторингу стану здоров’я, яка дозволяє користувачам зручно переглядати, фіксувати та аналізувати дані про стан здоров’я.

Отримані результати бакалаврської кваліфікаційної роботи можуть бути використані для організації ефективного моніторингу стану здоров’я та полегшення взаємодії між пацієнтами та медичними працівниками.

Ключові слова: вебсистема, моніторинг стану здоров’я, Python, Flask, інтерфейс користувача, PyCharm.

ABSTRACT

UDC 004.738.5:614.2

Dushnyi B. O. Development of a web system for health monitoring: bachelor's qualification work in speciality 121 - software engineering, educational programme - software engineering. Vinnytsia: VNTU, 2025. 94 c.

In Ukrainian. Bibliography: 20 titles; Figures: 30; Table 5.

The bachelor's qualification work analyses the state and development of software applications used to monitor the health of users. A review of existing analogues was conducted, their advantages and disadvantages were identified, which made it possible to justify the feasibility of creating their own web-based system.

The paper substantiates the choice of the Python programming language, the Flask framework and the PyCharm development environment for the project. A web-based health monitoring system has been developed that allows users to conveniently view, record and analyse health data.

The results of the bachelor's thesis can be used to organise effective health monitoring and facilitate interaction between patients and healthcare professionals.

Keywords: web-based system, health monitoring, Python, Flask, user interface, PyCharm.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі цифрові технології дедалі активніше впроваджуються в сферу охорони здоров'я, що відкриває нові можливості для покращення якості медичних послуг [1]. Одним із перспективних напрямів є створення вебсистем для моніторингу стану здоров'я. Такі системи дозволяють здійснювати постійне відстеження фізіологічних параметрів пацієнтів, збирати та аналізувати медичні дані в реальному часі, а також своєчасно реагувати на можливі відхилення від норми [2].

Особливо актуальним це є для людей з хронічними захворюваннями, літніх осіб, а також тих, хто перебуває в реабілітаційний період після лікування. Завдяки автоматизованому збору інформації про стан здоров'я, лікарі отримують змогу ефективніше приймати клінічні рішення, знижуючи ризик ускладнень і забезпечуючи безперервний нагляд без необхідності частих візитів до медичних закладів.

Сучасні технології, такі як хмарні сервіси, мобільні пристрої, сенсори та алгоритми штучного інтелекту, дозволяють реалізовувати надійні, масштабовані та зручні вебрішення для моніторингу здоров'я. Хмарні технології забезпечують безпечне зберігання даних та доступ до них з будь-якої точки світу, а інтуїтивно зрозумілий інтерфейс вебдодатків спрощує користування як для медичних працівників, так і для пацієнтів.

У зв'язку з цим розробка вебсистеми для моніторингу стану здоров'я є актуальним завданням, що спрямоване на підвищення ефективності медичного нагляду, покращення якості життя пацієнтів та сприяння цифровій трансформації сфери охорони здоров'я [3].

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення якості моніторингу стану здоров'я користувачів шляхом впровадження інформаційної системи, що

забезпечує зручний доступ до персональних даних, ефективне збереження та обробку інформації, а також формування аналітики і персоналізованих рекомендацій.

Відповідно до поставленої мети потрібно вирішити такі завдання:

1. Провести аналіз інформаційного забезпечення існуючих систем моніторингу стану здоров’я.

2. Розробити модель та алгоритм функціонування вебсистеми для моніторингу здоров’я.

3. Реалізувати вебінтерфейс для користувачів, який забезпечує реєстрацію та авторизацію облікових записів.

4. Забезпечити можливість введення та перегляду особистих медичних показників, таких як артеріальний тиск, пульс, температура тіла, рівень глюкози тощо.

5. Реалізувати візуалізацію динаміки стану здоров’я користувача у вигляді графіків та статистичних зведень.

6. Створити механізм автоматичного формування порад або повідомлень при виявленні відхилень від нормативних значень.

7. Реалізувати функцію додавання нагадувань про прийом ліків або необхідність проходження обстежень.

8. Створити дешборд із аналітикою активності користувача та загальним оглядом його показників здоров’я.

Об'єктом дослідження виступає процес розробки програмного забезпечення для систем моніторингу стану здоров’я.

Предметом дослідження є алгоритми та методи, що використовуються при розробці вебсистеми моніторингу стану здоров’я, особливості використання мови програмування Python, фреймворку Flask як серверної технології, а також засобів розробки у середовищі PyCharm.

Методи дослідження. У процесі дослідження використовувалися наступні методи:

– методи проєктування вебінтерфейсів з використанням HTML, CSS та JavaScript для забезпечення зручної взаємодії користувача з системою;

– технології розробки серверної частини на базі фреймворку Flask з

використанням мови програмування Python для обробки HTTP-запитів та реалізації логіки системи;

– методи організації баз даних (SQL) для збереження та обробки медичних показників користувача;

– методи тестування функціоналу вебзастосунку з метою перевірки коректності роботи основних модулів;

– застосування інтегрованого середовища розробки PyCharm для написання, налагодження та структурування коду.

Новизна отриманих результатів:

1. Подальшого розвитку набула модель побудови вебсистем моніторингу стану здоров’я, яка враховує специфіку введення, обробки та візуалізації медичних показників у реальному часі. На відміну від існуючих рішень, запропонована модель поєднує наочну послідовність дій користувача із логікою інтеграції даних, що забезпечує підвищення зручності використання та точності взаємодії з системою.

2. Удосконалено підхід до побудови алгоритмічної структури інформаційних систем медичного призначення шляхом розробки універсального механізму обробки різнорідних медичних показників. Запропоновані алгоритми враховують особливості параметрів, що потребують різних форматів введення та дозволяють масштабовано інтегрувати нові типи даних. Це забезпечує гнучкість, адаптивність і точність роботи системи, що є важливим у контексті персоналізованого моніторингу здоров’я.

Практичне значення проєкту полягає в можливості використання розробленого програмного забезпечення для моніторингу здоров’я в медичних установах, лікарями та пацієнтами. Створена вебсистема забезпечує зручний доступ до даних про стан здоров’я пацієнтів, спрощує процес моніторингу, а також дозволяє отримувати персоналізовану аналітику. Це сприятиме підвищенню ефективності медичних послуг та допоможе пацієнтам і медичним фахівцям оперативно реагувати на зміни в стані здоров’я. Програмні модулі можуть бути адаптовані для використання в різних медичних контекстах, включаючи публічні та приватні клініки, що дозволить оптимізувати лікувальні процеси та поліпшити координацію догляду за пацієнтами.

Особистий внесок здобувача. Усі результати, подані в бакалаврській кваліфікаційній роботі, були отримані автором особисто. Всі дані, включаючи таблицю порівняння функціональних характеристик аналогів та перелік розроблених функцій, є результатами власної роботи автора.

Апробація. Результати бакалаврської кваліфікаційної роботи були представлені на конференції «XII Всеукраїнська науково-практична конференція молодих учених, Київ, 2025.» та опубліковані у вигляді тез доповідей на цій конференції [3] .

Публікації. Результати дослідження були опубліковані у матеріалах конференції – «XII Всеукраїнська науково-практична конференція молодих учених, Київ, 2025».

Аналіз. Бакалаврська кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ СТАНУ РОЗВИТКУ ПРОГРАМ ДЛЯ МОНІТОРИНГУ СТАНУ ЗДОРОВ'Я ТА ПОСТАНОВКА ЗАДАЧ РОЗРОБКИ

1.1 Аналіз предметної області

У сучасному світі цифрові технології дедалі активніше інтегруються в різні сфери, зокрема у сферу охорони здоров’я. Зростаюча популярність телемедицини сприяє поширенню рішень, орієнтованих на дистанційний моніторинг фізичного стану людини. Одним із ключових напрямів цього розвитку є створення вебзастосунків, що дозволяють безперервно відслідковувати основні медичні показники та вчасно реагувати на зміни в стані здоров’я.

Сучасні вебсистеми моніторингу здоров’я надають можливість оперативного збору, аналізу та інтерпретації інформації про такі показники, як артеріальний тиск, температура тіла, рівень глюкози в крові, частота серцебиття тощо. Це забезпечує як лікарів, так і самих користувачів зручним інструментом для постійного контролю над фізичним самопочуттям. Такі інструменти не лише підвищують ефективність медичної допомоги, а й дозволяють здійснювати попередню оцінку стану пацієнта без необхідності негайного звернення до фахівця [4].

Навіть за відсутності підключення до носимих пристроїв, подібні системи можуть успішно функціонувати на основі даних, які користувач вводить вручну. Отримана інформація автоматично обробляється програмним забезпеченням, порівнюється з референсними значеннями та за необхідності генерує відповідні повідомлення або рекомендації. Такий підхід суттєво зменшує навантаження на медичний персонал, оскільки базовий аналіз здійснюється системою, а лікарі отримують уже підготовлену інформацію для подальших дій.

На ринку медичних технологій представлено чимало прикладів успішного впровадження подібних рішень. До розробки систем моніторингу активно долучаються як великі технологічні компанії, так і невеликі інноваційні стартапи. Їхні продукти сприяють не лише ранньому виявленню захворювань, а й моніторингу відновлення пацієнтів після хвороб, операцій або тривалого лікування. Це відкриває широкі

можливості для реалізації персоналізованих підходів у медицині – напряму, який нині є надзвичайно актуальним.

Важливою перевагою веборієнтованих рішень у галузі медицини є їхня роль у розвитку превентивної медицини. Вони дозволяють не лише контролювати поточний стан користувача, а й виявляти потенційні ризики ще до появи симптомів. Завдяки цьому лікарі та користувачі мають змогу завчасно коригувати спосіб життя, розпочинати профілактичні заходи або змінювати стратегії лікування. У довгостроковій перспективі це сприяє зниженню навантаження на систему охорони здоров’я та підвищенню рівня загального добробуту [5].

Крім аналітичного модуля, сучасні вебплатформи для моніторингу здоров’я, включають інструменти для візуалізації змін, механізми нагадування про прийом ліків або планові обстеження, а також модулі зі сповіщеннями у разі критичних відхилень. Деякі системи навіть можуть пропонувати рекомендації щодо харчування, активності, відпочинку або контролю рівня стресу. Це дозволяє інтегрувати турботу про здоров’я у щоденне життя користувача.

Таким чином, розробка власної вебсистеми для моніторингу здоров'я є важливим і перспективним кроком у процесі цифровізації медицини. Поєднання інформаційних технологій із медичними знаннями дозволяє створювати інструменти, здатні ефективно підтримувати індивідуальне здоров’я. Урахування сучасних підходів зокрема, використання штучного інтелекту, технологій великих даних і машинного навчання надає розробленим рішенням ще більшої гнучкості, точності й актуальності.

1.2 Аналіз аналогів розроблюваного програмного застосунку

У зв’язку з активним розвитком цифрових технологій та зростанням популярності носимих гаджетів, що призначені для відстеження фізичної активності та загального стану організму, на ринку з’являється дедалі більше інструментів для самостійного моніторингу здоров’я. Такі рішення дають змогу користувачам контролювати широкий спектр параметрів, від якості сну та харчових звичок до частоти серцевих скорочень і рівня стресу. Завдяки можливості підключення до фітнес-браслетів, смарт-годинників та інших смарт-пристроїв, користувачі отримують

актуальну інформацію про свій фізіологічний стан у режимі реального часу. Окрім базового відстеження, більшість таких платформ надають індивідуалізовані рекомендації, які допомагають краще розуміти потреби організму та підтримувати оптимальний рівень фізичного і психоемоційного здоров'я [6].

Персоналізований підхід, який лежить в основі подібних систем, сприяє розробці більш ефективних програм профілактики та підтримки здоров'я з урахуванням особливостей кожного користувача. На сучасному етапі розвитку цієї галузі вже сформувався перелік найпопулярніших рішень, серед яких варто згадати MyFitnessPal, Fitbit, Samsung Health і Garmin Connect. Кожна з цих платформ має свої характерні функціональні особливості та переваги.

MyFitnessPal (див. рисунок 1.1) орієнтований насамперед на моніторинг харчування та контролю споживання калорій. Користувачі мають змогу вести облік раціону, відстежувати щоденну активність і взаємодіяти з базою продуктів, яка автоматично розраховує поживну цінність страв. Додаток підтримує інтеграцію з іншими популярними фітнес-сервісами, що дозволяє синхронізувати дані з різноманітних джерел. Такий функціонал забезпечує комплексний підхід до оцінки способу життя користувача й сприяє ефективнішому досягненню цілей, пов'язаних із здоров'ям [7].

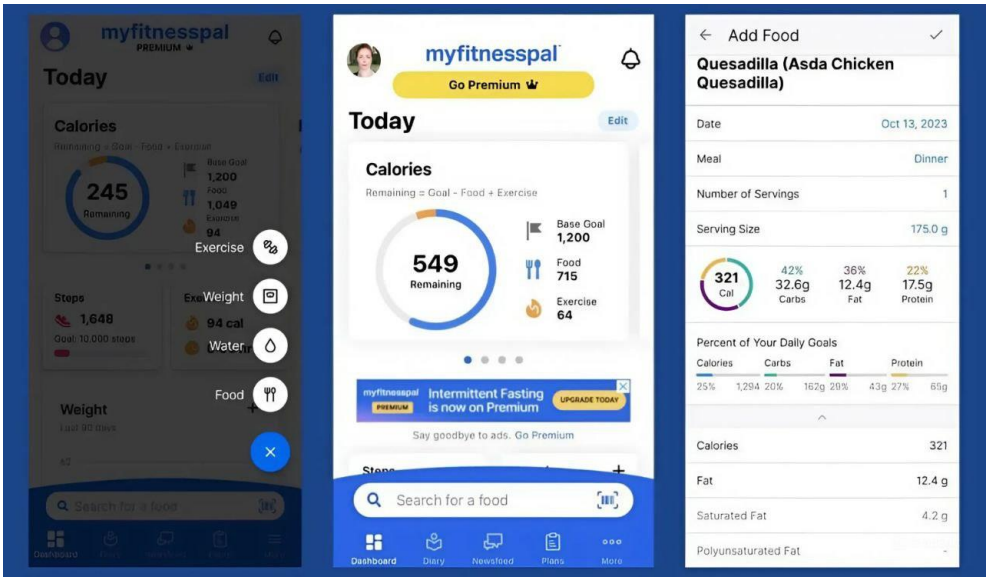

Рисунок 1.1 – Приклад роботи додатку MyFitnessPal

Fitbit (див. рисунок 1.2) – це відомий бренд носимих пристроїв, який пропонує платформу для детального моніторингу фізичної активності користувача. Система дозволяє відстежувати різні показники, зокрема кількість пройдених кроків, витрату калорій, рівень стресу, а також насичення крові киснем. За допомогою пристроїв Fitbit користувачі отримують персоналізовані поради щодо покращення фізичного стану та загального здоров’я. Платформа має інтуїтивно зрозумілий інтерфейс, дає можливість ставити індивідуальні цілі та досягати їх завдяки системі мотиваційних сповіщень і зворотного зв’язку [8].

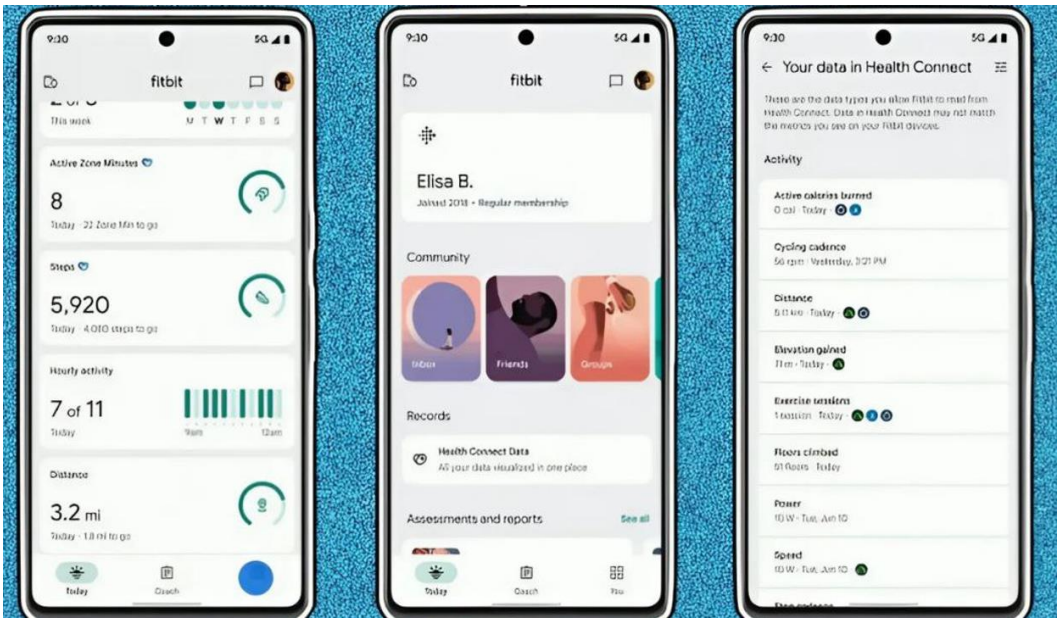

Рисунок 1.2 – Приклад роботи додатку Fitbit

Samsung Health (див. рисунок 1.3) – система для моніторингу здоров'я, яка інтегрується з різними пристроями Samsung і дозволяє відслідковувати активність, сон, рівень стресу та інші фізіологічні показники. Платформа також включає функції для моніторингу харчування, що дозволяє користувачам більш комплексно підходити до управління своїм здоров'ям. Крім того, Samsung Health дозволяє зберігати історію всіх показників здоров’я, що сприяє тривалому моніторингу змін та досягнень. Вона також підтримує синхронізацію з іншими популярними фітнес-додатками, що робить її універсальним інструментом для контролю за фізичним станом [9].

Рисунок 1.3 – Приклад роботи додатку Samsung Health

Garmin Connect (див. рисунок 1.4) – платформа для моніторингу здоров'я та фізичної активності. Вона дозволяє користувачам відстежувати тренування, фізичну активність, сновидіння, пульс, рівень стресу та інші параметри здоров'я. Крім того, Garmin Connect надає можливість аналізувати детальні дані про виконані тренування та прогрес, що дає змогу налаштувати індивідуальні плани для досягнення поставлених цілей. Платформа також підтримує інтеграцію з іншими додатками та пристроями, що дозволяє створювати цілісну картину фізичного стану користувача [10].

Рисунок 1.4 – Приклад роботи додатку Garmin Connect

Зазначені вище застосунки яскраво демонструють значення та перспективи впровадження інформаційних технологій у сферу охорони здоров’я. Вони надають пацієнтам зручні інструменти для організації медичних візитів і контролю за власним станом, що значно спрощує взаємодію з медичними установами. Завдяки таким рішенням забезпечується своєчасне реагування на зміни у стані здоров’я та прийняття необхідних медичних заходів.

Створення подібних систем вимагає глибокого розуміння потреб цільової аудиторії, а також технічних знань для розробки зручних, стабільних і функціональних мобільних додатків. Ці додатки повинні сприяти підвищенню доступності медичних послуг і покращенню їх якості на всіх етапах лікування. У цьому контексті важливими є забезпечення захисту персональних даних, адаптивність інтерфейсу під різні категорії користувачів і впровадження можливостей для індивідуального моніторингу здоров’я. Крім того, значну роль відіграє інтеграція таких систем із медичними структурами для забезпечення точного й оперативного обміну інформацією.

На основі аналізу згаданих аналогів була сформована таблиця 1.1, що містить порівняння їхніх функціональних можливостей із власною розробкою. Це дозволяє виявити ключові переваги й недоліки наявних рішень та визначити вектори подальшого вдосконалення системи.

Таблиця 1.1 – Порівняльні характеристики аналогів

	MyFitness Pal	Fitbit	Samsung Health	Garmin Connect	Власна розробка
Обліковий запис користувача	1	1	1	1	1
Відстеження сну	0	1	1	1	1
Візуалізація даних у вигляді діаграми	1	0	1	0	1
Аналіз пульсу	0	1	1	1	1

Продовження таблиці 1.1

	MyFitness Pal	Fitbit	Samsung Health	Garmin Connect	Власна розробка
Аналіз зібраних даних	0	1	1	1	1
Персоналізовані рекомендації для покращення здоров'я	1	0	0	1	1
Сумарний коефіцієнт	3	4	5	5	6

Відповідно до наведеної таблиці 1.1 порівняльного аналізу, розробка вебсистеми для моніторингу здоров'я, що включає в себе функціонал для відстеження фізичної активності, сну, візуалізації даних у вигляді діаграм та інших параметрів здоров'я, є не лише доцільною, а й необхідною. Запропонована платформа дозволить забезпечити інтегроване рішення для користувачів, що прагнуть підтримувати здоровий спосіб життя, та надасть медичним установам зручні інструменти для моніторингу здоров'я пацієнтів у реальному часі.

Отже, розробка вебсистеми для моніторингу здоров'я є інноваційним кроком у розвитку медичних технологій, який дозволить досягти нових рівнів ефективності у лікуванні та попередженні захворювань, надаючи пацієнтам доступ до необхідної інформації та підтримки для покращення їхнього здоров'я.

1.3 Аналіз методів розв’язання задачі

Розробка вебсистеми для моніторингу здоров'я вимагає вибору найбільш ефективного підходу для забезпечення безпечного і зручного процесу для користувачів. Для цього можна застосувати кілька стратегій, кожна з яких має свої переваги та недоліки, що впливають на загальну ефективність продукту [11]. Вибір правильного підходу є ключовим для досягнення високої якості сервісу, який не лише відповідає потребам користувачів, а й адаптується до швидко змінюваного ринку медичних

технологій.

Один із підходів полягає в розробці універсального вебсервісу, який дозволяє користувачам записувати свої антропометричні та медичні дані, аналізувати їх і отримувати рекомендації на основі введеної інформації. Цей метод забезпечує максимальну універсальність і простоту інтеграції з іншими системами. Він дозволяє охопити широку аудиторію користувачів з різними потребами та дозволяє легко розширювати функціональність платформи в майбутньому. Проте, він може обмежити можливості платформи щодо персоналізації, якщо розробка не враховує специфічні потреби кожного користувача, що може призвести до менш точних рекомендацій.

Альтернативний підхід полягає у створенні інтуїтивно зрозумілого інтерфейсу, що дозволяє користувачам здійснювати самостійний моніторинг різних параметрів здоров'я, таких як фізична активність, тиск, вага тощо [12]. Завдяки простоті використання, цей підхід дає можливість користувачам без труднощів взаємодіяти з платформою. Однак, він потребує постійного оновлення функціоналу для того, щоб відповідати новим вимогам користувачів і тенденціям у сфері здоров'я. Це може вимагати додаткових ресурсів для підтримки та розвитку системи в довгостроковій перспективі.

Після аналізу переваг і недоліків обох методів було вибрано підхід, який передбачає створення зручного та функціонального вебсервісу для моніторингу здоров'я з можливістю самостійного аналізу даних. Цей метод забезпечить гнучкість у використанні та відповідає сучасним вимогам користувачів, що прагнуть отримувати точні дані про своє здоров'я без необхідності взаємодії з носимими пристроями чи медичними фахівцями. Такий підхід дозволяє користувачам підтримувати контроль над своїм здоров'ям, здійснюючи самостійний моніторинг і отримуючи індивідуальні рекомендації, що підвищує ефективність самообслуговування та знижує навантаження на медичні установи.

1.4 Постановка задач дослідження

Після аналізу сильних та слабких сторін існуючих платформ для моніторингу здоров'я були визначені ключові завдання для розробки власної вебсистеми:

- розробити систему збору та аналізу даних про здоров'я користувачів, що дозволить їм моніторити основні параметри, такі як фізична активність, вага, сон, пульс, рівень стресу та інші антропометричні показники;

- створити інтуїтивно зрозумілий та зручний інтерфейс користувача вебсистеми, адаптований під різні пристрої, щоб забезпечити зручний доступ та простоту використання на ПК, смартфонах і планшетах;

- розробити функціонал для введення та зберігання історії показників здоров'я, що дасть користувачам можливість відстежувати динаміку змін, порівнювати показники та встановлювати цілі для покращення здоров'я;

- впровадити систему надання рекомендацій на основі введених даних, яка допоможе користувачам краще розуміти свої показники здоров'я і приймати обґрунтовані рішення щодо змін у своєму способі життя;

- провести всебічне тестування розробленої платформи для забезпечення її надійності, ефективності та зручності використання в реальних умовах, включаючи тестування функціоналу на різних пристроях та з різними веб-браузерами.

Ці завдання були спрямовані на створення вебсистеми, яка не тільки задовольнить потреби сучасних користувачів у самостійному моніторингу здоров'я, але й забезпечить простоту в управлінні та використанні для кожного користувача, незалежно від його технічного досвіду. Ретельно розроблений інтерфейс та функціональні можливості платформи дозволять користувачам з будь-яким рівнем підготовки швидко освоїти систему і використовувати її без зайвих труднощів. Завдяки цьому, користувачі зможуть не тільки ефективно контролювати своє здоров’я, але й отримувати персоналізовану підтримку для покращення свого фізичного стану. Це включає індивідуальні рекомендації та плани дій, адаптовані під конкретні потреби кожного користувача, що значно підвищує мотивацію і допомагає досягати реальних результатів у підтримці здорового способу життя. Тому платформа стане незамінним інструментом для тих, хто прагне покращити своє фізичне та емоційне благополуччя, використовуючи технології для досягнення конкретних цілей.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було проведено комплексний аналіз стану розвитку програм для моніторингу стану здоров’я та визначено задачі розробки програмного забезпечення. Спочатку було досліджено предметну область, де визначено основні вимоги та особливості систем моніторингу здоров’я. Далі проаналізовано існуючі програмні платформи: Fitbit, MyFitnessPal, Samsung Health, Garmin Connect, їхні переваги та недоліки, серед яких обмежена персоналізація, недостатня інтеграція даних та відсутність комплексного підходу. Також розглянуто методи розв’язання задачі моніторингу стану здоров’я та визначено потребу у використанні індивідуалізованого підходу. На завершення було сформульовано основні задачі дослідження, які передбачають створення програмного застосунку для моніторингу стану здоров’я з урахуванням індивідуальних потреб користувачів. Отже, результати аналізу підтвердили доцільність розробки власної системи моніторингу здоров’я, що дозволить забезпечити комплексний підхід до контролю стану користувачів і подолати недоліки сучасних рішень.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ ВЕБСИСТЕМИ

2.1 Аналіз даних

Обробка та аналіз даних є важливими компонентами функціонування вебсистеми моніторингу стану здоров’я. Система отримує дані, які користувач вводить самостійно, після чого ці дані зберігаються, обробляються та візуалізуються для подальшого аналізу. Застосунок побудовано за допомогою мови програмування Python для бекенду, JavaScript для клієнтської частини, а також SQLite як основної системи управління базами даних.

У модулі реєстрації та авторизації користувачі можуть створити особистий обліковий запис, використовуючи ім’я, електронну пошту та пароль. Після успішної реєстрації або входу до системи, користувач отримує доступ до особистої панелі. Вхідними даними є дані форми логін та пароль, а вихідними інтерфейс, що підтверджує авторизацію та відкриває доступ до функцій системи.

Модуль панелі керування відображає основну інформацію про стан здоров’я користувача у вигляді списку останніх записів, ключових показників, цілей і графіків. Вхідними даними є медичні показники, збережені в базі даних, а вихідними інтерфейс із персоналізованою інформацією, доступний після авторизації. Користувач може перейти до інших функцій, таких як додавання нових записів або перегляд аналітики.

У модулі додавання нових показників та аналітики користувач вводить значення, що стосуються стану здоров’я. Ці дані обробляються та візуалізуються у вигляді діаграм за допомогою бібліотеки Chart.js. Вхідні дані це числові значення, введені вручну, а вихідні дані це графіки та таблиці, що демонструють динаміку змін. Модуль також дозволяє встановлювати персоналізовані цілі та порівнювати фактичні результати з бажаними.

Модуль моделей відповідає за взаємодію з базою даних SQLite. Він зберігає всі введені користувачем дані, забезпечуючи їх структуризацію, можливість редагування, видалення та перегляд історії. Вхідними даними є об’єкти, сформовані у форматі Python-класів, а вихідними запити до бази та отримані результати.

Інтерфейс користувача побудовано на основі HTML-шаблонів, які відображають основні сторінки: головну, аналітики, додавання запису, встановлення цілей. Кожна сторінка орієнтована на простоту використання та логічну організацію функцій. Користувач взаємодіє з інтерфейсом через інтуїтивно зрозумілі елементи, а система реагує відповідно до його дій.

Завдяки використанню Python, JavaScript та SQLite, система є легкою в розгортанні, гнучкою у масштабуванні та може адаптуватися до різних груп користувачів. Модульна архітектура забезпечує можливість подальшого розширення функціоналу без необхідності суттєвих змін у вже реалізованих компонентах. В результаті користувач отримує ефективний інструмент для самостійного моніторингу стану здоров’я, аналізу змін та отримання рекомендацій щодо покращення власного самопочуття.

2.2 Розробка моделі вебсистеми для моніторингу стану здоров’я

При створенні програмного забезпечення для вебсистеми, орієнтованої на моніторинг стану здоров’я користувача, важливо детально спланувати всі компоненти та сценарії використання. Це дозволяє забезпечити зрозумілий та ефективний процес взаємодії користувача з системою. У цьому контексті була розроблена UML-діаграма діяльності, що зображено на рисунку 2.1.

UML-діаграма діяльності описує основні етапи взаємодії користувача з системою, включаючи реєстрацію, авторизацію, введення даних про здоров’я, перегляд аналітики та інші функції [13]. Ця діаграма дозволяє чітко визначити послідовність дій та логіку обробки запитів користувача, а також виявити можливі проблеми на етапі розробки, що значно полегшує подальше тестування та вдосконалення системи.

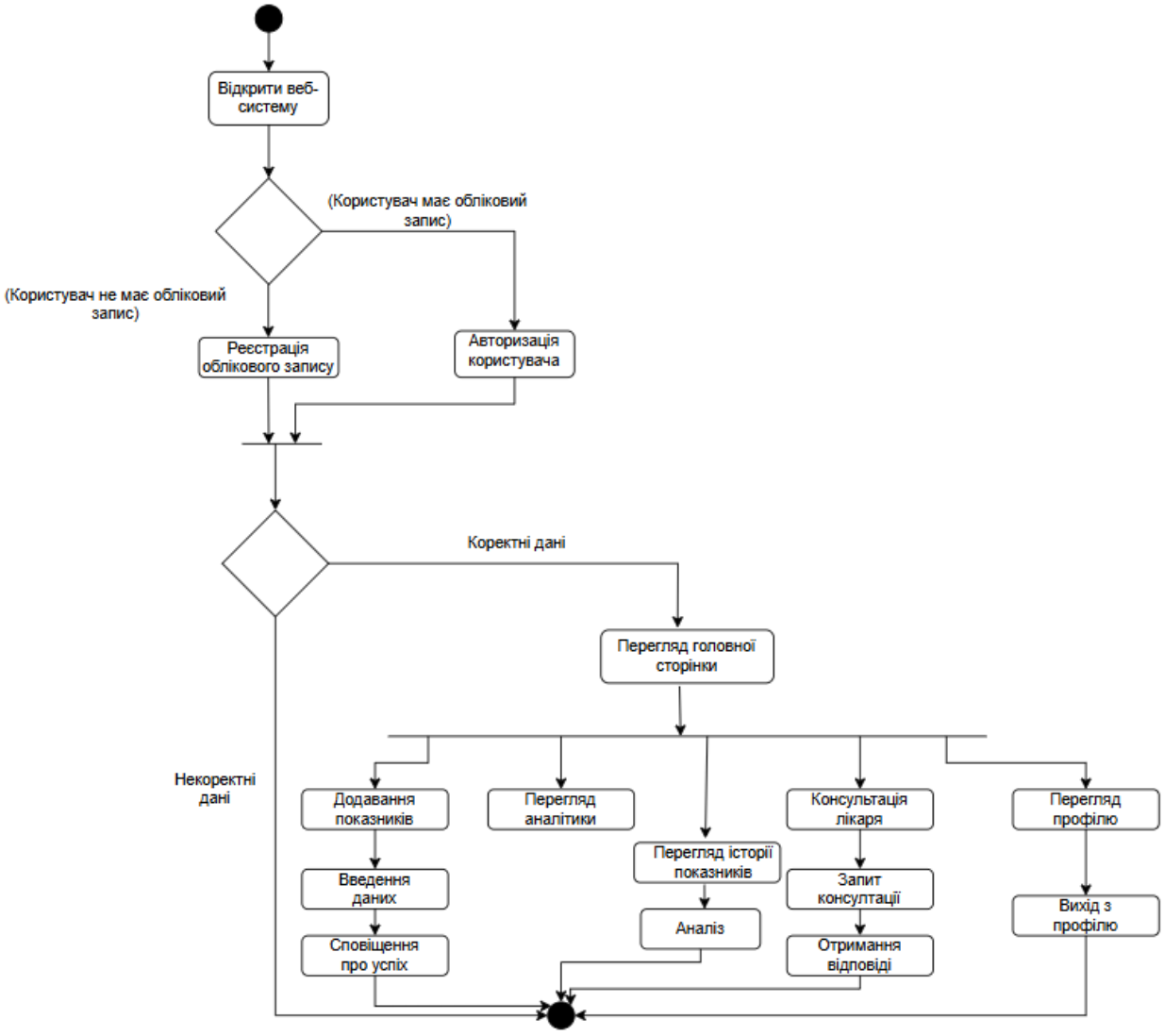

Рисунок 2.1 – Діаграма діяльності вебсистеми

Дана діаграма ілюструє ключові етапи взаємодії користувача із системою, від авторизації до перегляду аналітики та історії записів. Також діаграма дозволяє наочно відобразити логіку роботи користувача з вебінтерфейсом та структуру внутрішніх процесів системи.

На діаграмі діяльності представлено процес взаємодії користувача з вебсистемою для моніторингу стану здоров’я. Процес розпочинається з вибору дії, користувач може зареєструватися або авторизуватися у системі.

Після входу до персонального кабінету користувачу відкривається декілька можливих сценаріїв взаємодії: додавання нового запису про стан здоров’я, перегляд встановлених цілей, аналіз показників у розділі аналітики або перегляд історії записів за обраним параметром.

При додаванні запису користувач обирає тип показника, додає за потреби нотатку, зазначає дату та час вимірювання. Після збереження системою повідомляється про успішне додавання інформації. У випадку перегляду аналітики чи історії, дані виводяться у вигляді графіків або списку записів відповідно до обраного параметра.

Процес завершується можливістю повернутися на головну сторінку або виконати нову дію.

2.3 Розробка алгоритмів моделювання користувацьких дій та обробки даних

У рамках розробки вебсистеми для моніторингу стану здоров’я був реалізований алгоритм дій користувача. Взаємодія з системою починається з вибору, чи зареєстрований користувач. Якщо так, то він переходить на сторінку авторизації, де вводить свої облікові дані. Якщо користувач ще не має акаунту, йому пропонується пройти реєстрацію, ввівши ім’я, електронну пошту та пароль. Після успішної перевірки даних система автоматично переводить користувача до персонального кабінету.

На головній сторінці користувач має можливість скористатися основним функціоналом системи. Зокрема, він може додати новий запис про своє здоров'я. При додаванні нового запису користувач обирає параметр здоров'я, додає нотатку за потреби, вибирає дату і час запису. Після введення та перевірки інформації система генерує повідомлення про успішне збереження даних.

Цей процес забезпечує зручність і точність введення інформації, а також дає користувачу можливість контролювати своє здоров'я з максимальним комфортом.

Дана блок-схема описує загальний алгоритм взаємодії користувача з вебсистемою (див. рисунок 2.2).

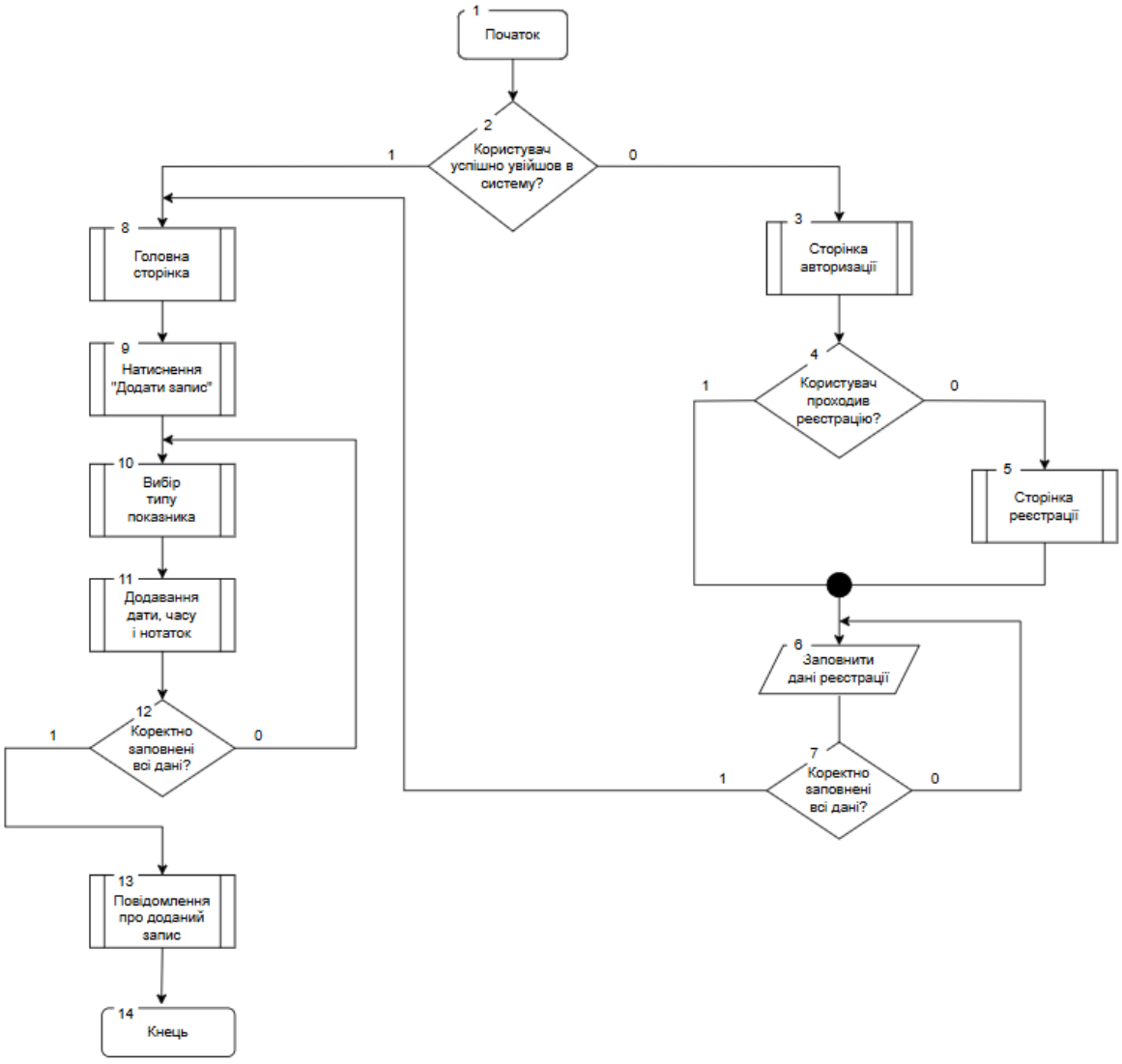

Рисунок 2.2 – Алгоритм дій користувача

Наступна діаграма зображує потік виконання алгоритму додавання запису для вибраного параметру (див. рисунок 2.3). Алгоритм починається з вибору параметру здоров'я, для якого користувач бажає зробити запис. Потім система запитує у користувача введення значення параметру та дату і час вимірювання. Якщо користувач бажає, він також може додати текстову примітку, щоб уточнити обставини або додаткову інформацію щодо стану здоров'я.

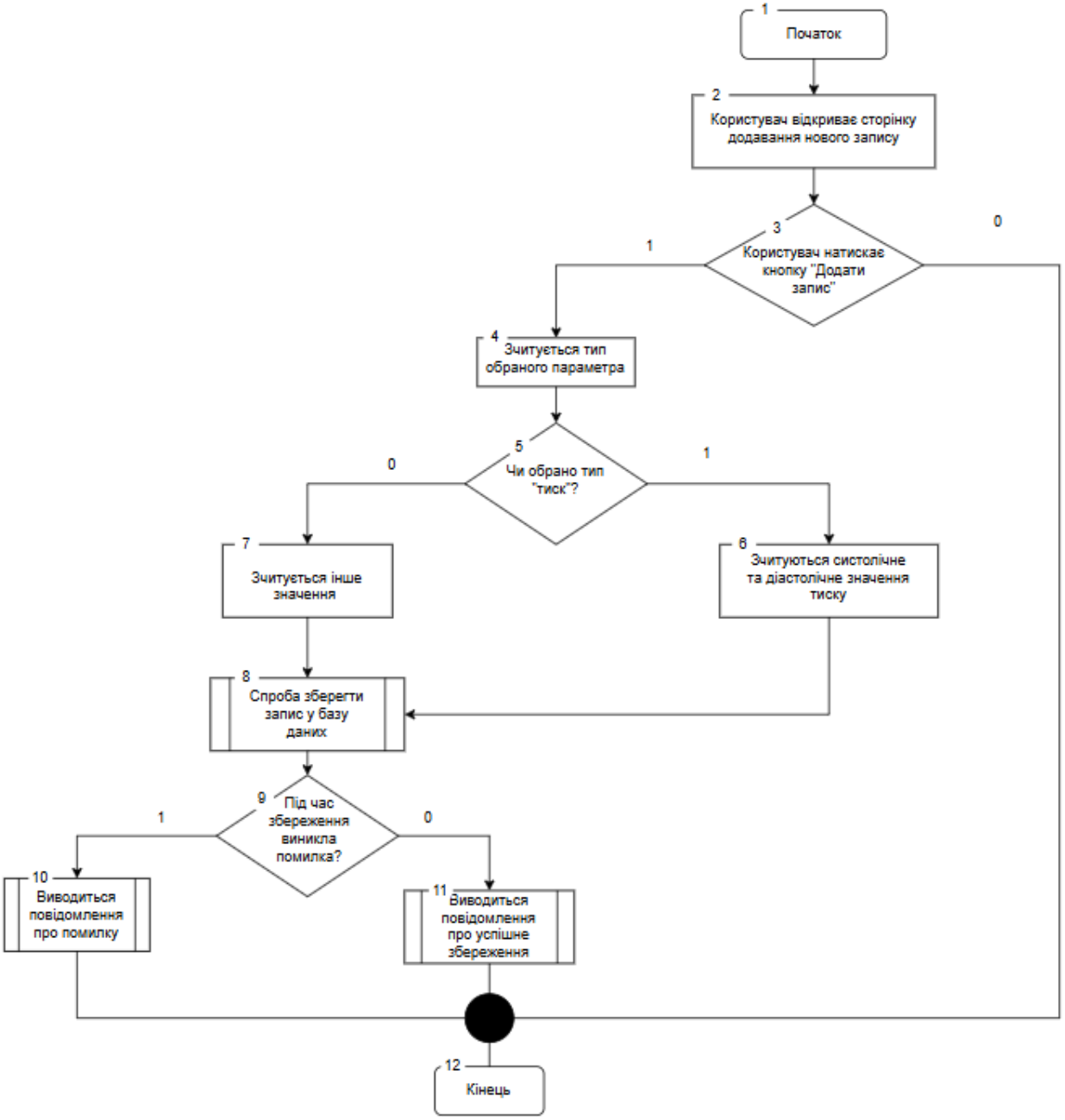

Рисунок 2.3 – Блок-схема алгоритму додавання запису для вибраного параметру

Блок-схема описує наступний алгоритм додавання нового запису:

1. «Початок»: початковий етап, який сигналізує про запуск функції додавання запису користувачем.

2. «Користувач відкриває сторінку додавання нового запису»: при GET-запиті вебінтерфейс завантажує сторінку форми для внесення нових даних. У форму передається поточна дата та час.

3. «Користувач натискає кнопку «Додати запис» у формі»: якщо користувач заповнює форму і надсилає її, система отримує POST-запит.

4. «Зчитується тип обраного параметра»: система визначає, який саме тип даних хоче зберегти користувач.

5. «Чи обрано тип тиск?»: перевіряється, чи дані стосуються артеріального тиску. Якщо так, то обробка йде за одним шляхом, інакше, то за іншим.

6. «Зчитуються систолічне та діастолічне значення тиску»: якщо обрано тип тиск, то зчитуються два значення – систолічний та діастолічний тиск. Потім формується об'єкт запису, який включає ці параметри.

7. «Зчитується інше значення.»: якщо обрано інший тип (не тиск), зчитується одне числове значення та створюється відповідний запис.

8. «Спроба зберегти запис у базу даних»: сформований запис передається до бази даних для збереження. Використовується сесія бази даних для додавання і коміту змін.

9. «Під час збереження виникла помилка?»: система перевіряє, чи сталася помилка під час додавання запису. Якщо так, то переходить до блоку помилки.

10. «Вивести повідомлення про помилку»: виводиться повідомлення для користувача про те, що сталася помилка при збереженні запису.

11. «Вивести повідомлення про успішне збереження та повернення до головної сторінки»: якщо запис успішно збережено, користувач бачить повідомлення про успіх, після чого відбувається перенаправлення на головну сторінку.

12. «Кінець»: завершення роботи алгоритму додавання запису.

2.4 Висновки

У другому розділі було детально розглянуто процес обробки даних, проєктування та реалізації архітектури вебсистеми моніторингу стану здоров’я.

Завдяки використанню сучасних технологій, таких як Python, JavaScript та SQLite, система забезпечує ефективну обробку інформації та зручну взаємодію з користувачем. Побудована UML-діаграма діяльності та блок-схеми алгоритмів дали змогу чітко структурувати логіку функціонування вебсистеми, визначити послідовність дій користувача та внутрішню обробку запитів.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Одним із ключових етапів розробки вебсистеми для моніторингу стану здоров’я користувача є вибір мови програмування, яка забезпечить зручність у розробці, підтримку сучасних вебтехнологій, високу продуктивність та гнучкість. У межах цього проєкту основною мовою реалізації обрано Python**,** що було обґрунтовано порівнянням з іншими популярними мовами: JavaScript, PHP та Java.

Python – це високорівнева інтерпретована мова програмування з простою та зрозумілою синтаксичною структурою, що значно спрощує процес розробки [14]. Вона широко використовується для створення вебдодатків, наукових обчислень, автоматизації, а також у сфері штучного інтелекту й аналізу даних. Для веброзробки Python найчастіше використовується з фреймворком Flask, який дозволяє швидко створювати масштабовані вебсервіси.

JavaScript – основна мова для клієнтської веброзробки. Має широке використання разом з фреймворками, такими як React або Node.js. Проте JavaScript більше орієнтований на фронтенд або повний стек, тоді як для даного проєкту більший акцент робиться на бекенд-логіку та обробку даних.

PHP – традиційно використовувався для створення динамічних вебсайтів, однак поступається Python у гнучкості, читабельності та сучасних можливостях, таких як обробка великих даних або інтеграція з ML-сервісами.

Java – потужна та типобезпечна мова, яка часто використовується в корпоративних системах. Однак розробка на Java вимагає більше ресурсів і часу у порівнянні з Python.

Порівняння функціональних характеристик мов програмування Python, JavaScript, PHP та Java наведено в таблиці 3.1.

Таблиця 3.1 – порівняння функціональних характеристик мов програмування

Характеристика	Python	JavaScript	PHP	Java
Простота синтаксису	1	0	0	0
Підтримка вебфреймворків	1	1	1	1
Гнучкість	1	1	0	1
Підтримка ООП	1	1	0	1
Можливість обробки даних	1	0	0	1
Інтеграція з ML/AI	1	0	0	1
Загальний коефіцієнт	6	3	1	5

На основі аналізу Python отримав найвищий сумарний бал, що підтверджує доцільність його використання для реалізації вебсистеми для моніторингу стану здоров’я.

Наступним кроком стало визначення інтегрованого середовища розробки, яке б найкраще відповідало вимогам до зручної та ефективної розробки програмного забезпечення. Основними критеріями для вибору були: підтримка Python, зручність у використанні, наявність інструментів для налагодження, інтеграція з віртуальним середовищем та системами контролю версій. Було розглянуто декілька популярних середовищ розробки: PyCharm, Visual Studio Code, Sublime Text та Eclipse з плагіном PyDev.

PyCharm – потужне повноцінне IDE, спеціалізоване на Python [15]. Має вбудовану підтримку віртуальних середовищ, систем контролю версій, розширені можливості налагодження, зручну інтеграцію з базами даних, автоматичне доповнення коду та багатий набір інструментів для професійної розробки.

Visual Studio Code – легке, кросплатформне середовище з широкою підтримкою розширень. Добре підходить для Python після встановлення відповідних плагінів Python Extension від Microsoft. Потребує додаткової конфігурації для налаштування налагодження, віртуального середовища та інтеграції з базами даних.

Sublime Text – швидкий та мінімалістичний текстовий редактор із підтримкою

багатьох мов через плагіни. Не є повноцінним IDE і не забезпечує зручної інтеграції з налагоджувачами, системами версій чи віртуальним середовищем без значного ручного налаштування.

Eclipse + PyDev – популярне середовище у світі Java, яке також підтримує Python через плагін PyDev. Хоча має базову підтримку необхідних інструментів, інтерфейс менш інтуїтивний, а налаштування складніше порівняно з іншими IDE.

Порівняння функціональних характеристик середовищ наведено в таблиці 3.2.

Таблиця 3.2 – порівняння функціональних характеристик середовищ програмування

Характеристика	PyCharm	VS Code	Sublime	Eclipse + PyDev
Підтримка Python	1	1	1	1
Вбудована робота з віртуальним середовищем	1	0	0	1
Інтеграція з Git	1	1	0	1
Підтримка налагодження	1	1	0	1
Інструменти для роботи з БД	1	0	0	1
Плагіни та розширення	1	1	1	1
Зручність та повноцінність IDE	1	0	0	0
Загальний коефіцієнт	7	4	2	5

За таблицею найбільш оптимальним середовищем для реалізації вебсистеми виявився PyCharm, оскільки він надає всі необхідні інструменти для ефективної розробки, тестування та підтримки Python-проєктів.

3.2 Розробка модуля реєстрації та авторизації користувача в системі

Головною задачею модуля реєстрації та авторизації в системі моніторингу стану здоров'я є забезпечення контролю доступу та захисту персональних даних користувачів.

Метою також є створення надійної та ефективної системи, яка гарантує безпеку та високу продуктивність при обробці запитів користувачів, забезпечуючи надійний доступ до функціоналу моніторингу здоров'я.

Щоб краще проілюструвати логіку роботи цього модуля, на рисунку 3.1 представлено блок-схему алгоритму авторизації та автентифікації.

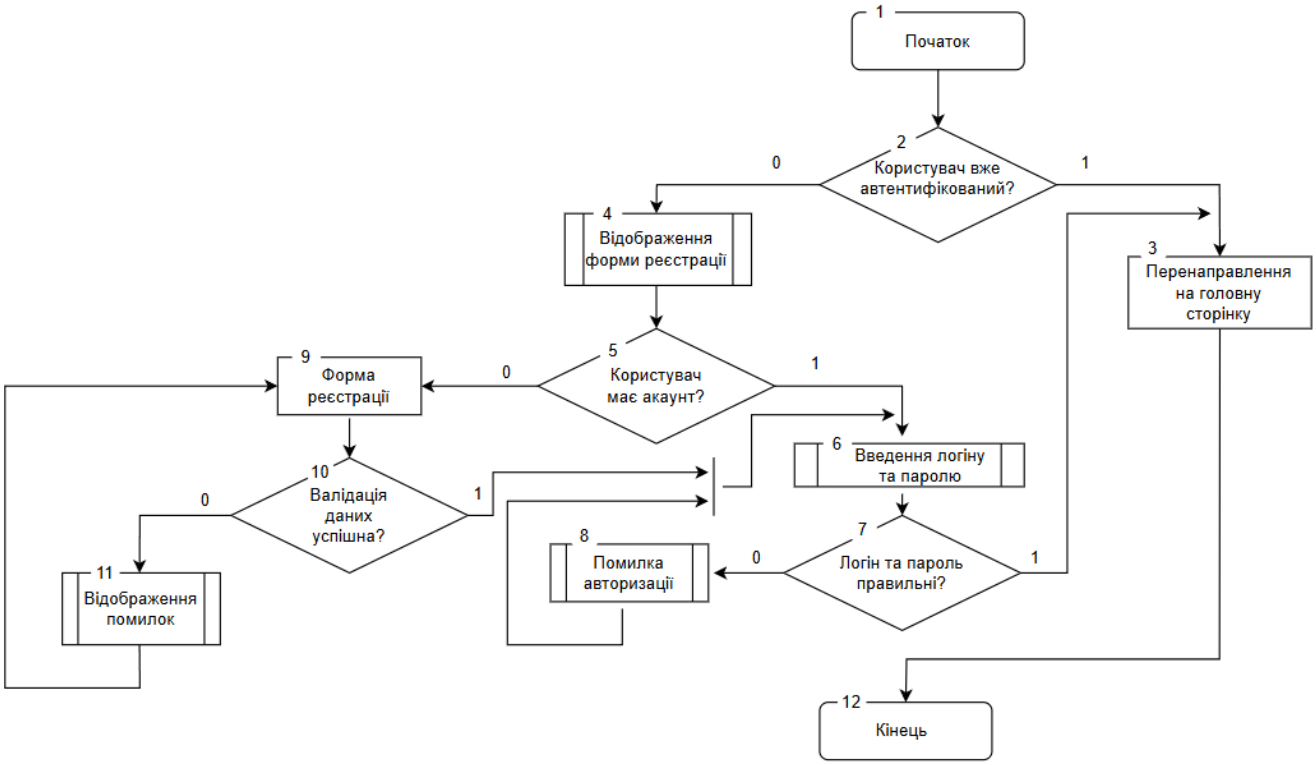

Рисунок 3.1 – Блок-схема алгоритму роботи модуля реєстрації та авторизації

На даній блок-схемі зображено послідовність дій, що виконуються під час автентифікації та авторизації користувача. Вона охоплює етапи введення облікових даних, перевірки їх коректності, визначення прав доступу та реагування системи у разі успішної або невдалої спроби входу. Блок-схема дозволяє наочно уявити логіку прийняття рішень системою безпеки.

На рисунку 3.2 зображено витяг з коду класу LoginForm, який відповідає за побудову форми авторизації в системі.

```
1 usage
class LoginForm(FlaskForm):
    username = StringField('Логін', validators=[DataRequired()])
    password = PasswordField('Пароль', validators=[DataRequired()])
    remember_me = BooleanField('Запам\'ятати мене')
    submit = SubmitField('Увійти')
```

Рисунок 3.2 – Створення класу LoginForm у Flask

Клас LoginForm є підкласом FlaskForm з бібліотеки Flask-WTF, що використовується для створення вебформ із валідацією у Flask-додатках. У цьому класі визначені поля форми: username (логін), password (пароль), remember_me (прапорець для запам'ятовування сесії користувача) та кнопка submit.

Кожне з полів має вбудовані валідатори, наприклад DataRequired(), що гарантує, що поле не буде порожнім при надсиланні. Цей клас інтегрується в маршрут /login, де відбувається перевірка правильності введених даних та авторизація користувача. Такий підхід дозволяє централізовано контролювати логіку входу до системи та гарантувати безпечний доступ до функціоналу моніторингу здоров'я.

Далі розглянемо функцію ініціалізації конфігурації при обробці запиту на вхід до системи моніторингу здоров’я, що зображена на рисунку 3.3.

Ця функція перевіряє, чи користувач уже автентифікований, і в такому разі перенаправляє його до головного інтерфейсу моніторингу. Якщо ні – викликається форма LoginForm, яка відповідає за обробку введених даних користувача.

```
@auth.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('dashboard.index'))
```

Рисунок 3.3 – Початкова перевірка статусу авторизації користувача

Метод login() розпочинає з перевірки, чи користувач уже виконав вхід до системи.

Якщо це так, немає потреби повторно вводити дані, і користувача одразу перенаправляють до основної сторінки – моніторингової панелі.

Наступним етапом розглянемо логіку перевірки автентичності введених даних користувача, що зображено на рисунку 3.4.

У випадку помилки – виводиться повідомлення про неправильний логін або пароль. Якщо облікові дані вірні, користувача авторизують, і йому надається доступ до функціоналу системи.

```
form = LoginForm()
if form.validate_on_submit():
    user = User.query.filter_by(username=form.username.data).first()

    if user is None or not user.check_password(form.password.data):
        flash('Неправильний логін або пароль')
        return redirect(url_for('auth.login'))
```

Рисунок 3.4 – Перевірка логіна та встановлення сесії

Тут перевіряються дані з форми авторизації. Якщо дані правильні – викликається login_user(), яка створює сесію для поточного користувача, зберігаючи його авторизаційний стан.

Отже, розробка модуля авторизації та реєстрації користувачів у системі моніторингу здоров'я забезпечує надійний контроль доступу та захист персональних даних користувачів. Логіка перевірки автентичності введених даних, а також створення сесії користувача, сприяє ефективному управлінню доступом до функціоналу системи. Це дозволяє гарантувати, що лише авторизовані користувачі можуть отримати доступ до конфіденційної інформації та сервісів. Надійність алгоритму та чіткість його реалізації підвищують безпеку системи та покращують загальний користувацький досвід. Такий підхід сприяє захисту персональних даних і відповідає вимогам сучасних стандартів безпеки.

3.3 Розробка модуля панелі керування

Для забезпечення візуалізації основних показників здоров’я, а також швидкого доступу до ключових функцій системи, розроблено спеціальний модуль панелі керування. Цей модуль виконує роль центрального вузла взаємодії користувача з системою моніторингу стану здоров’я. Головна панель управління забезпечує інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам зручно переглядати історію медичних показників, додавати нові записи, а також аналізувати дані за допомогою візуалізацій.

Крім базової функціональності, панель керування також дозволяє відстежувати динаміку змін показників здоров’я, виявляти відхилення, переглядати останні записи та здійснювати навігацію між окремими підмодулями системи. Такий підхід дозволяє зосередити всю взаємодію користувача в одному зручному середовищі.

Модуль панелі керування реалізовано у вигляді окремого компонента Flask під назвою Blueprint [16]. Це дозволяє логічно структурувати маршрути, пов’язані з управлінням станом здоров’я, та відокремити їх від інших частин програми, таких як аутентифікація, база даних або аналітика. Завдяки використанню Blueprint, код програми залишається модульним, масштабованим та легко підтримуваним.

На рисунку 3.5 зображено процес ініціалізації модуля керування, що передбачає створення Blueprint з іменем dashboard, який далі імпортується до основного застосунку Flask.

```
from flask import Blueprint, render_template, request, redirect, url_for, flash, jsonify
from flask_login import login_required, current_user
from models import HealthRecord, HealthGoal, NormalRanges
from extensions import db
from datetime import datetime, timedelta

# Змінюємо назву Blueprint з dashboard_routes на dashboard
dashboard = Blueprint('dashboard', __name__)
```

Рисунок 3.5 – Ініціалізація Blueprint для панелі керування

Основним елементом модуля є головна сторінка панелі керування, яка служить точкою входу до всіх функцій моніторингу. Вона відображає зведену інформацію про ключові показники здоров'я користувача, такі як артеріальний тиск, пульс, рівень

глюкози тощо. Сторінка автоматично витягує дані з бази даних для відповідного користувача та передає їх до шаблону для відображення.

На рисунку 3.6 наведено фрагмент коду функції, що відповідає за обробку запиту на відображення головної сторінки панелі керування. Функція реалізує маршрутизацію, отримання даних та їх підготовку до виводу у веб-інтерфейсі.

```python
@dashboard.route('/')
@login_required
def index():
    # Отримуємо останні записи для відображення на дашборді
    recent_records = {}
    record_types = ['heart_rate', 'blood_pressure', 'weight', 'glucose']

    for record_type in record_types:
        records = HealthRecord.query.filter_by(
            user_id=current_user.id,
            record_type=record_type
        ).order_by(HealthRecord.record_date.desc()).limit(5).all()

        recent_records[record_type] = records

    # Отримуємо активні цілі для відображення на дашборді
    active_goals = HealthGoal.query.filter_by(
        user_id=current_user.id,
        completed=False
    ).all()

    # Отримуємо нормальні діапазони для відображення індикаторів
    normal_ranges = {}
    for nr in NormalRanges.query.all():
        normal_ranges[nr.record_type] = nr

    return render_template('dashboard/index.html',
                           recent_records=recent_records,
                           active_goals=active_goals,
                           normal_ranges=normal_ranges)
```

Рисунок 3.6 – Функція відображення головної сторінки панелі керування

Таким чином, розроблений модуль панелі керування забезпечує користувачам зручний інтерфейс для моніторингу показників здоров’я, встановлення та відстеження цілей, аналізу даних за різні періоди часу. Модуль реалізує принципи безпеки через обов'язкову автентифікацію користувачів і перевірку доступу до даних.

3.4 Розробка модуля додавання записів показників здоров’я

Одним із ключових функціональних компонентів системи моніторингу стану здоров’я є механізм додавання нових записів про показники здоров’я користувача. Цей модуль відіграє важливу роль у забезпеченні точної фіксації медичних даних, що є основою для подальшого аналізу, візуалізації, відстеження динаміки та виявлення потенційних ризиків.

Особливу увагу в реалізації цього механізму приділено гнучкості інтерфейсу введення. Система динамічно адаптує форму відповідно до обраного типу медичного показника, дозволяючи користувачеві вводити лише релевантні дані.

На рисунку 3.7 наведено блок-схему функції, яка відповідає за керування відображенням відповідних полів форми залежно від вибраного типу показника.

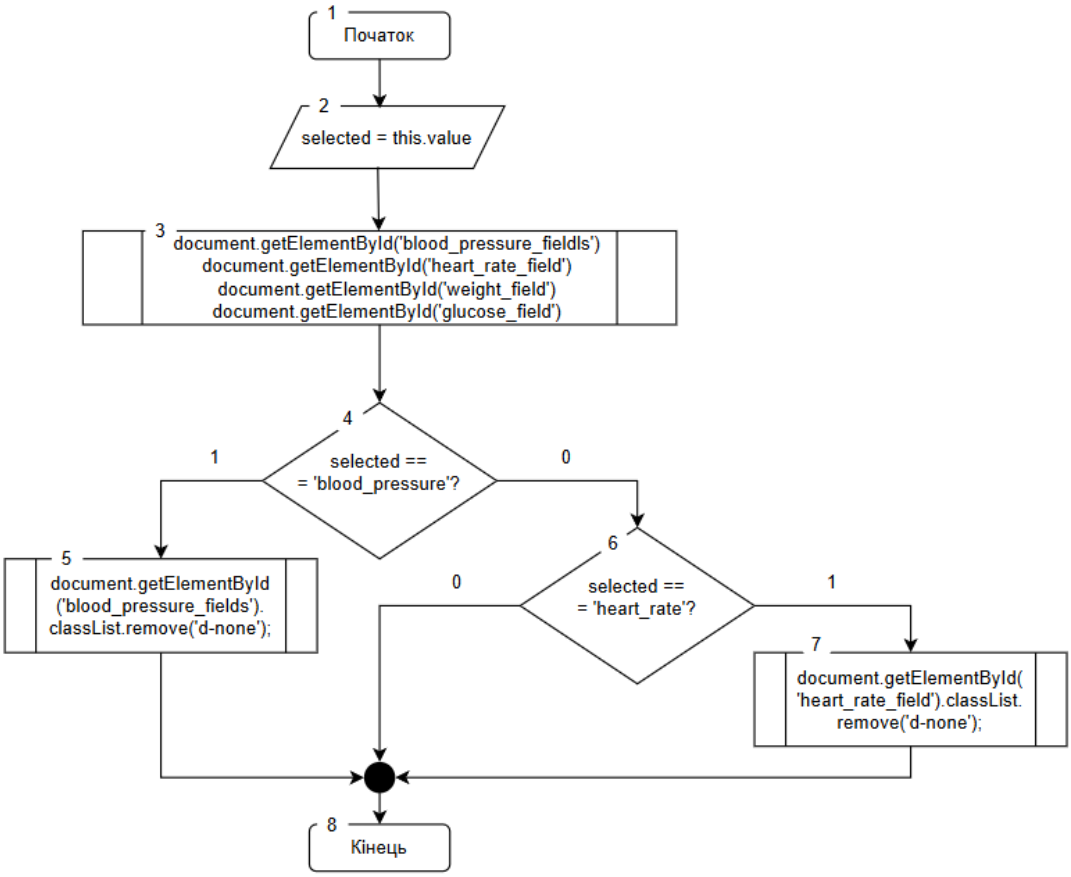

Рисунок 3.7 – Функція для керування відображенням полів форми

1. Функція додається як обробник події change для елемента з id record_type, який є випадаючим списком для вибору типу показника здоров'я.

2. При зміні вибраного типу показника, функція спочатку приховує всі блоки полів введення, додаючи клас d-none до кожного з них.

3. Потім отримується значення вибраного типу показника зі змінної this.value.

4. На основі вибраного типу показника, видаляється клас d-none у відповідного блоку полів, таким чином роблячи його видимим для користувача.

5. Це дозволяє динамічно адаптувати форму під різні типи показників без перезавантаження сторінки, забезпечуючи кращий користувацький досвід.

Крім динамічного відображення полів форми, важливою частиною логіки інтерфейсу є перевірка правильності введених даних перед їх відправленням. Для цього реалізовано функцію, яка обробляє подію submit форми додавання запису. Вона забезпечує контроль за наявністю обов'язкових даних, залежно від типу обраного показника, та підготовку значень для подальшої передачі на сервер. Завдяки цьому система зменшує ризик збереження некоректних або неповних медичних даних.

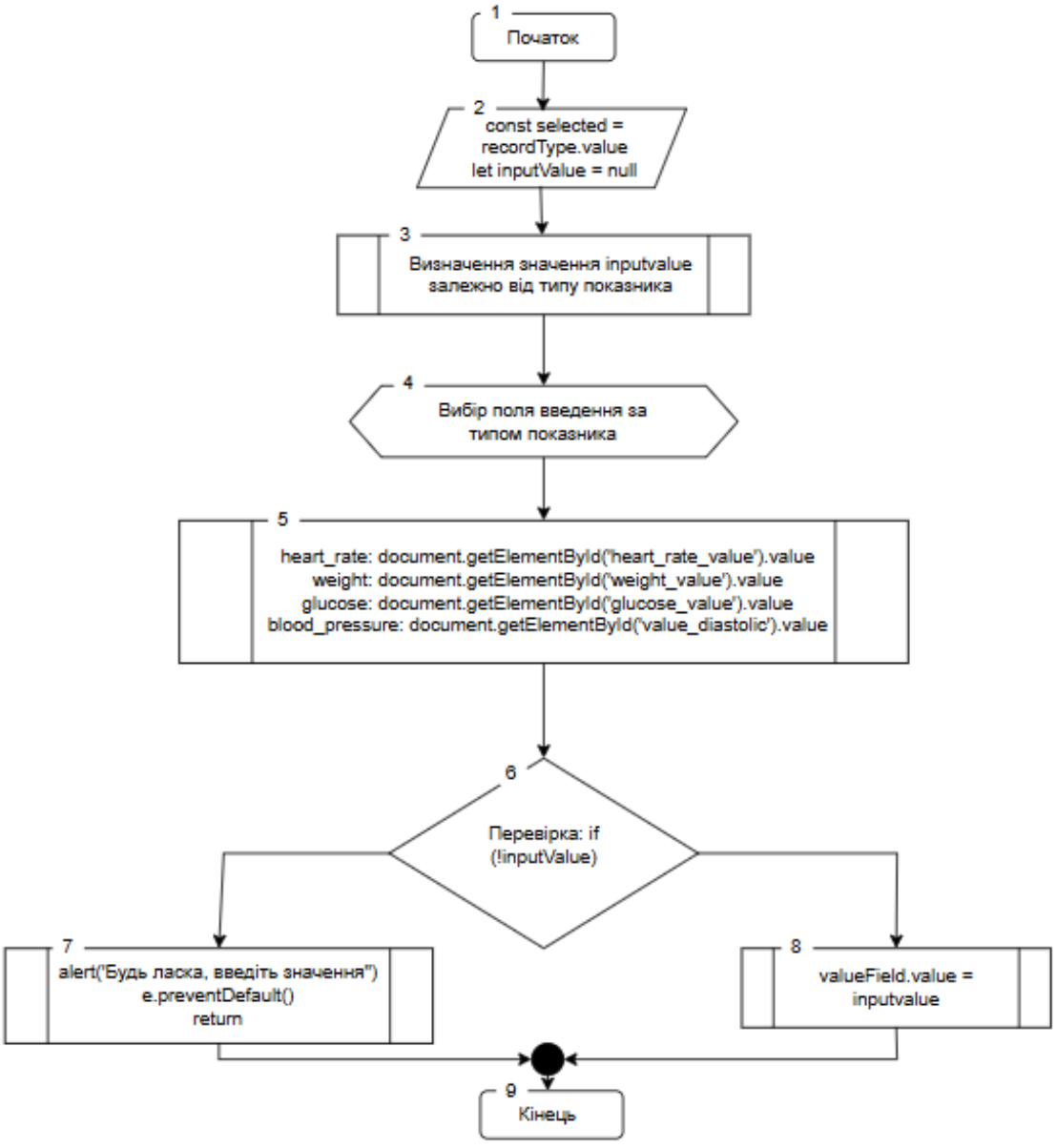

Рисунок 3.8 – Функція обробки відправки форми

1. Функція додається як обробник події submit для елемента форми з id recordForm.

2. При спробі відправки форми, функція спочатку отримує тип вибраного показника зі змінної recordType.value.

3. Залежно від типу показника, функція отримує значення з відповідного поля введення.

4. Функція виконує базову валідацію даних - перевіряє, чи введено значення показника. Якщо значення не введено, виводиться повідомлення про помилку і відправка форми скасовується методом e.preventDefault().

5. Якщо значення введено коректно, функція заповнює приховане поле value цим значенням для відправки на сервер.

Для реалізації функціональності додавання нових медичних записів у системі моніторингу стану здоров’я використовується серверна логіка, побудована на основі популярного вебфреймворку Flask. Цей фреймворк забезпечує ефективну обробку HTTP-запитів, дозволяє організовувати маршрутизацію, взаємодію з шаблонами та інтеграцію з базою даних. Завдяки цьому, розробники мають змогу гнучко реалізувати логіку обробки дій користувача у вебінтерфейсі.

Серверна логіка, що реалізує механізм додавання нових записів, відповідає за прийом даних із клієнтської форми введення. Вона виконує кілька важливих етапів: зчитування вхідних значень із запиту, перевірку їх відповідності очікуваним форматам, фільтрацію некоректних даних, визначення типу медичного показника, формування об’єкта відповідного класу та збереження його у базу даних через ORM-інтерфейс SQLAlchemy [17].

Залежно від типу показника здоров’я (наприклад, тиск, пульс або температура), обробник динамічно визначає, які поля з форми мають бути зчитані. Якщо показник має складну структуру, як-от систолічний та діастолічний тиск, система враховує це під час обробки. Після успішного збереження даних, користувач отримує повідомлення про успіх, а у разі помилки — відповідне попередження.

Блок-схема на рисунку 3.9 наочно ілюструє послідовність дій, які відбуваються під час взаємодії користувача з формою додавання запису: від ініціації події на клієнті

до обробки запиту на сервері та внесення даних у базу. Це дозволяє краще зрозуміти логіку роботи системи й забезпечує наочне подання основних кроків алгоритму.

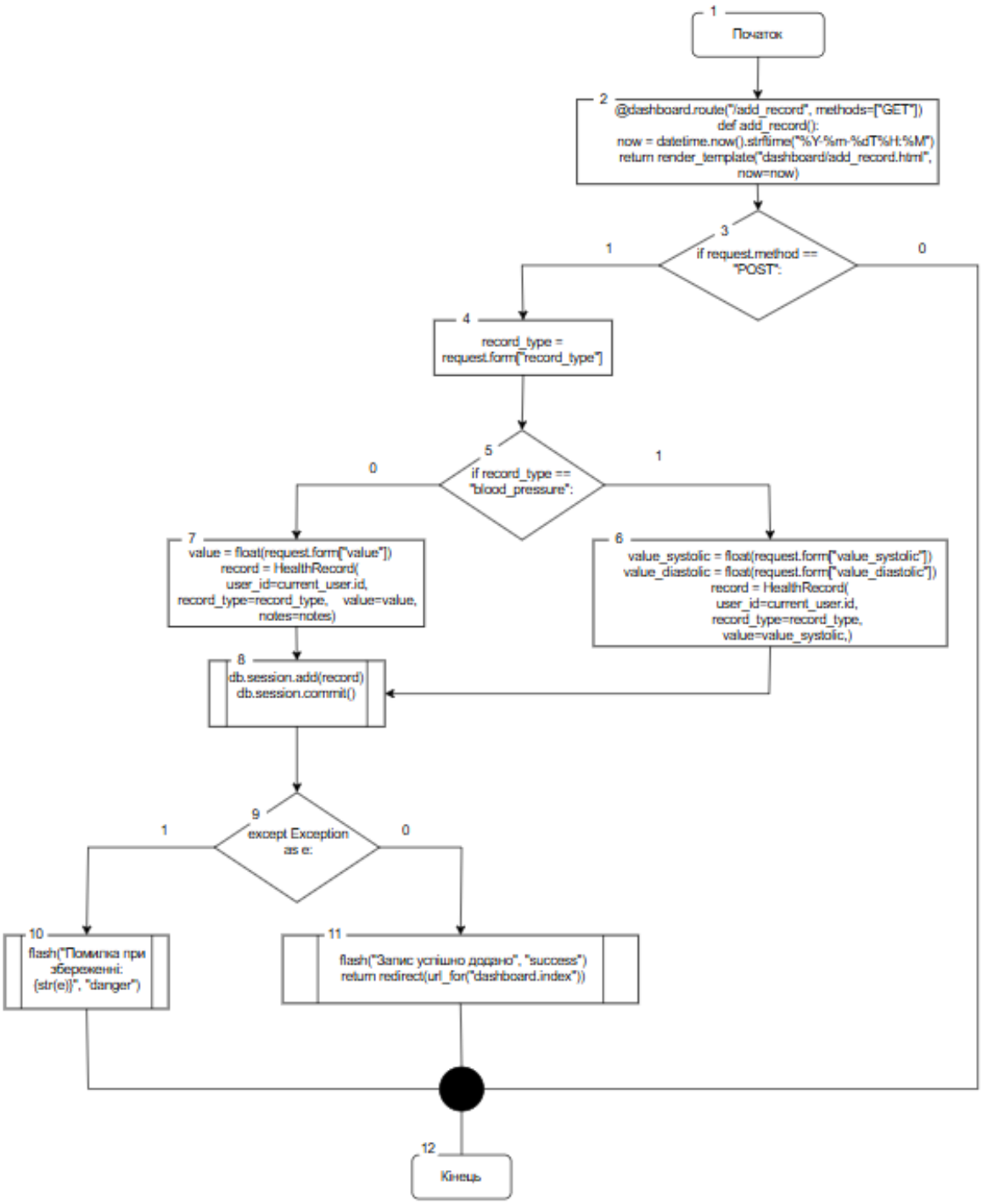

Рисунок 3.9 – Блок-схема алгоритму додавання запису для вибраного параметру

На початку відбувається ініціалізація обробника маршруту Flask через декоратор @dashboard.route, що обробляє як GET, так і POST запити. Якщо запит GET, повертається HTML-шаблон для додавання запису [18].

Якщо запит має метод POST, спочатку отримується тип запису з форми (record_type). Далі логіка розгалужується в залежності від типу: якщо це тиск ("blood_pressure"), зчитуються два значення – систолічний та діастолічний тиск. В іншому випадку зчитується загальне значення (value) з поля форми.

На основі отриманих даних створюється об'єкт HealthRecord, що містить ідентифікатор користувача, тип запису, значення та нотатки. Цей об'єкт зберігається до бази даних за допомогою db.session.add() та db.session.commit() [19].

У випадку успішного збереження користувачу виводиться повідомлення про успіх і здійснюється перенаправлення на головну сторінку. Якщо ж під час збереження виникає помилка, вона перехоплюється, і користувач отримує повідомлення про помилку.

На цьому виконання алгоритму завершується.

3.5 Розробка вебінтерфейсу програмного додатку

У процесі розробки вебсистеми для моніторингу здоров’я особлива увага приділялася створенню зручного та інтуїтивно зрозумілого інтерфейсу користувача.

Першим екраном, з яким взаємодіє користувач, є вікно авторизації, що зображено на рисунку 3.10.

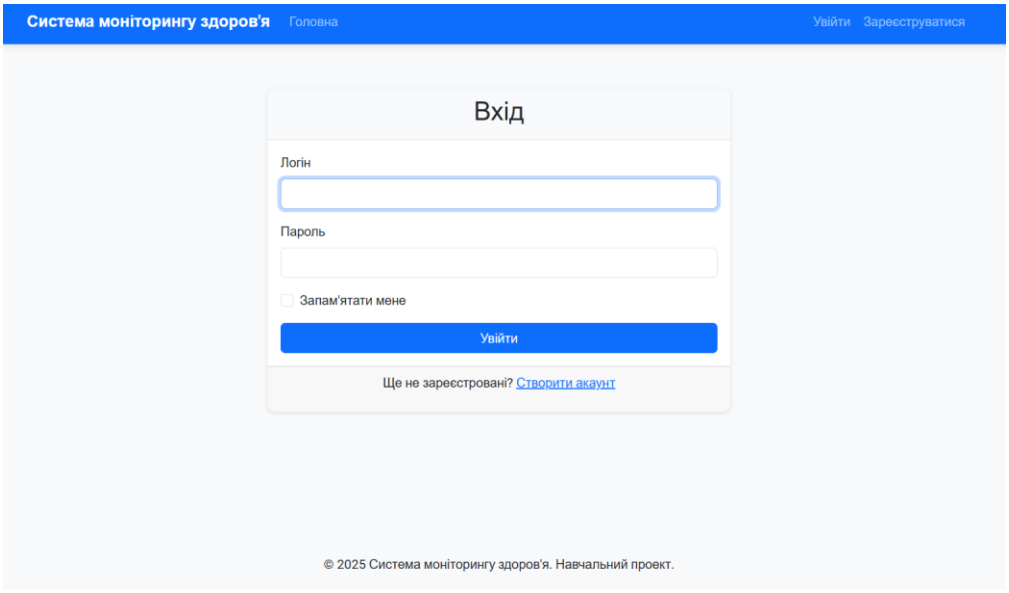

Рисунок 3.10 – Вікно авторизації

Якщо користувач не має облікового запису в системі, він може перейти на екран реєстрації за допомогою кнопки «Зареєструватися» (див. рисунок 3.11). Цей екран має подібне оформлення до вікна авторизації, що забезпечує візуальну єдність дизайну вебсистеми.

На екрані реєстрації користувач вводить свій логін, електронну адресу та пароль для створення персонального облікового запису, який надалі використовується для збереження та моніторингу індивідуальних показників здоров’я.

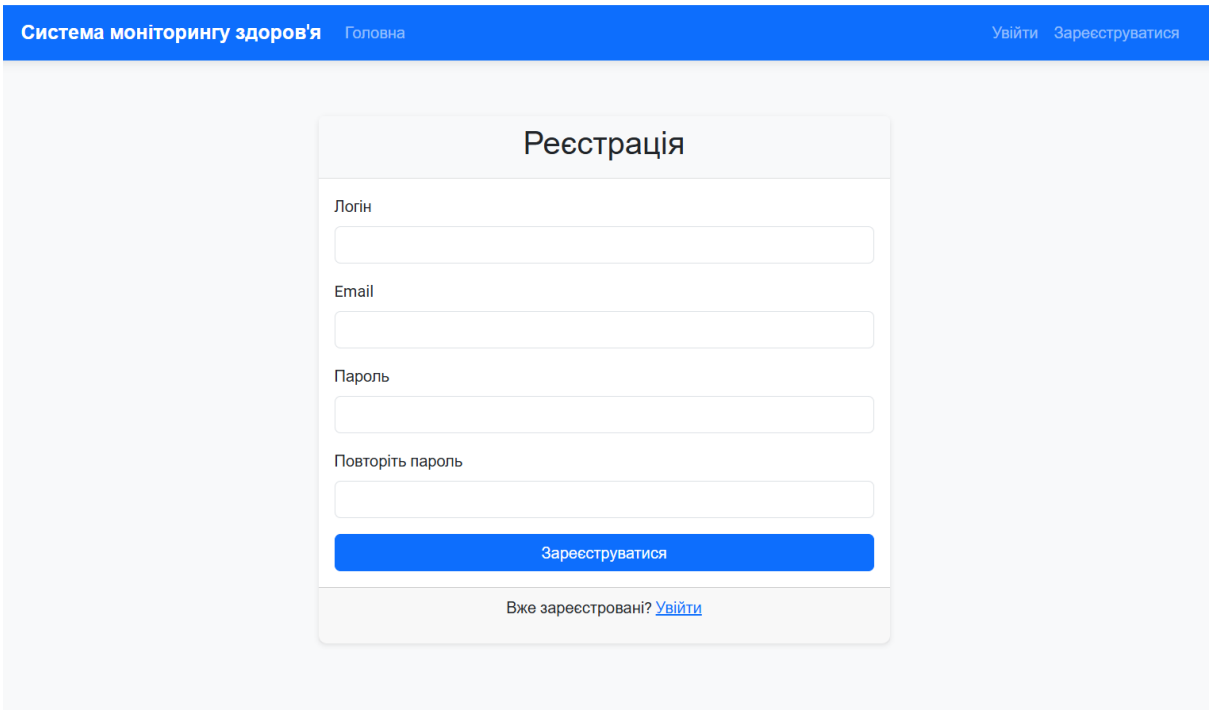

Рисунок 3.11 – Вікно реєстрації

Після успішної реєстрації користувач бачить повідомлення про те, що обліковий запис було створено успішно (див. рисунок 3.12). Для зручності користувача поля електронної адреси та пароля у формі входу заповнюються автоматично.

Крім того, на екрані авторизації доступна опція «Запам’ятати мене», яка дозволяє зберегти дані для подальших входів без повторного введення, що робить користування вебсистемою ще зручнішим.

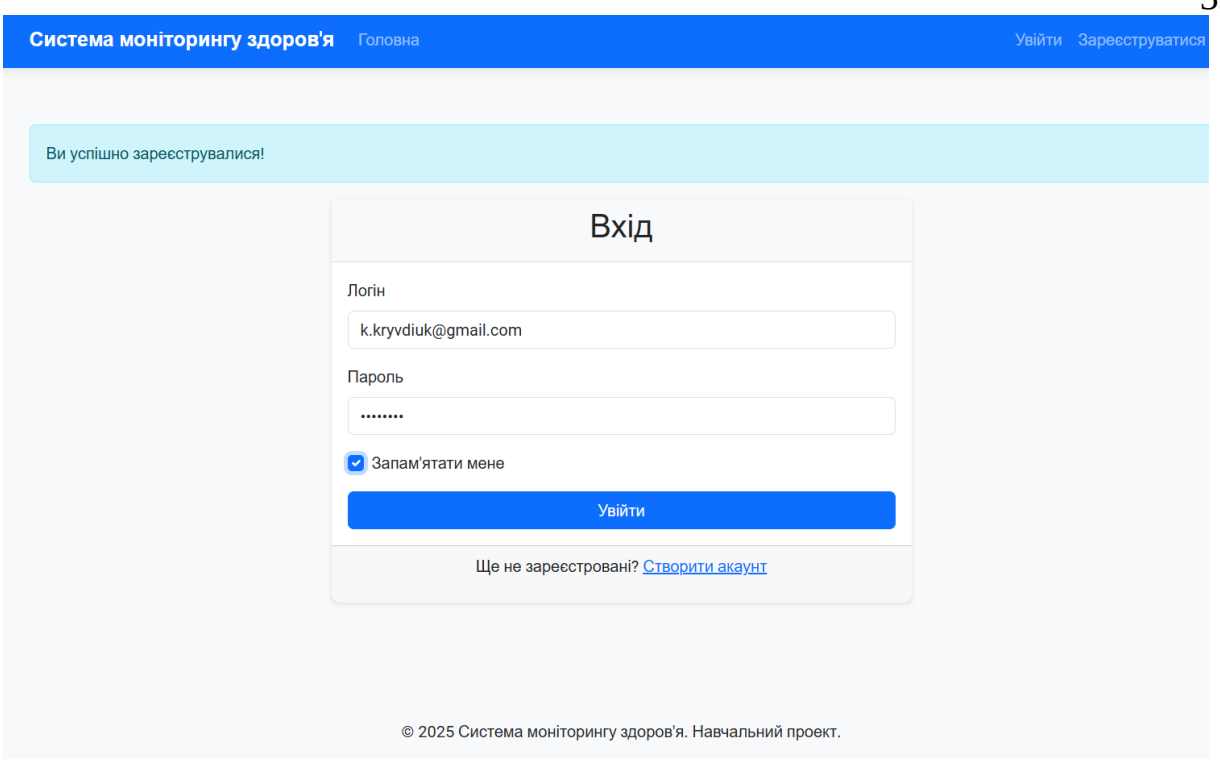

Рисунок 3.12 – Успішна реєстрація

Після успішного входу до персонального кабінету користувача зустрічає вітальне повідомлення та основний функціонал вебсистеми для моніторингу здоров’я (див. рисунок 3.13). На цьому етапі користувач має доступ до ключових розділів, таких як введення та перегляд даних про здоров’я, моніторинг фізичної активності, а також перегляд статистики та рекомендацій. Інтерфейс був спроектований таким чином, щоб забезпечити простоту навігації, що дозволяє швидко знайти необхідну інформацію та ефективно працювати з платформою.

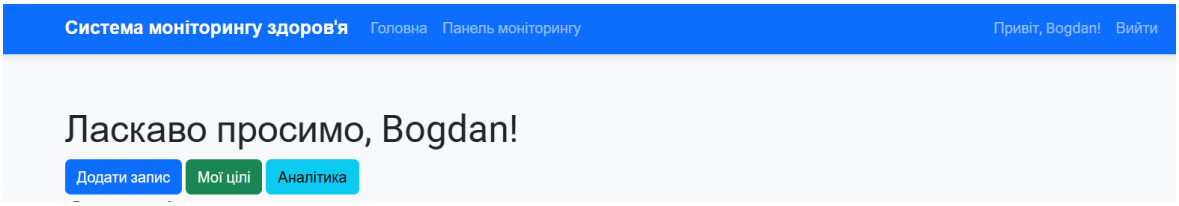

Рисунок 3.13 – Головна сторінка вебсистеми

При натисканні на кнопку «Додати запис» користувач потрапляє на екран із формою введення даних (див. рисунок 3.14). У цій формі необхідно зазначити:

• параметр здоров’я, який користувач бажає додати (тиск, пульс, рівень цукру в крові тощо);

• нотатку, де можна вказати додаткову інформацію, наприклад, що могло вплинути на показник («після тренування», «перед сном» тощо);

• час і дату, коли саме було здійснено вимірювання – для цього використовується інтегрований календар і селектор часу.

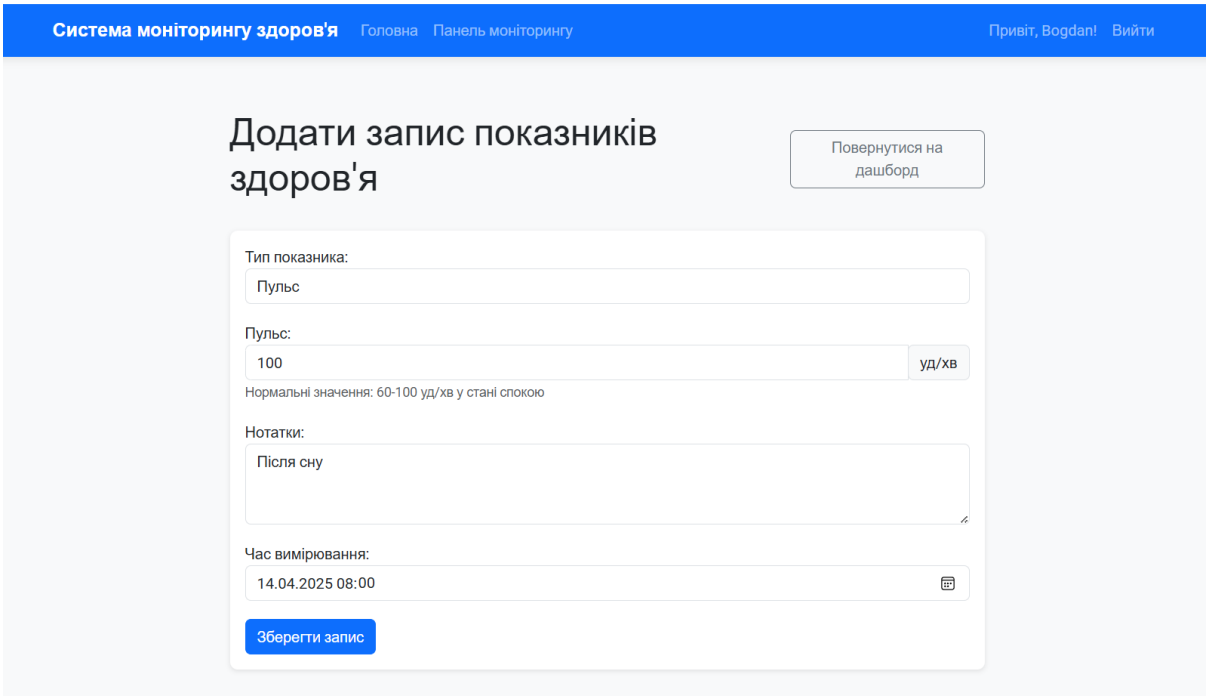

Рисунок 3.14 – Вікно додавання запису

Після заповнення форми та натискання кнопки «Зберегти запис», користувача вітає повідомлення про успішне збереження даних (див. рисунок 3.15). Це підтвердження забезпечує користувача впевненістю, що введена інформація була збережена правильно і доступна для подальшого аналізу.

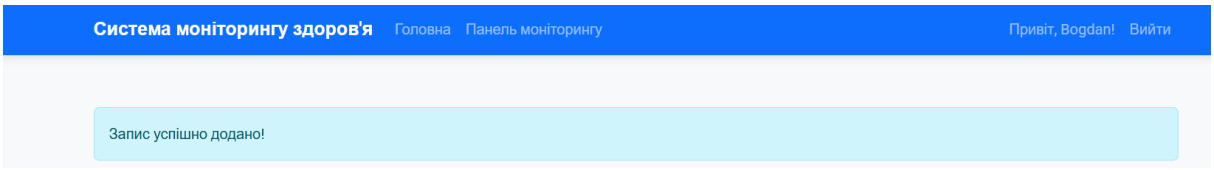

Рисунок 3.15 – Повідомлення про успішно доданий запис

Одразу після цього користувач може переглянути останню історію записів усіх параметрів здоров’я (див. рисунок 3.16). Дані відображаються у вигляді списку з вказанням дати, часу, значення параметра та за потреби – доданої нотатки. Це дозволяє користувачу швидко переглянути свої останні показники без потреби переглядати історію, що значно підвищує зручність користування системою.

Крім того, кожен запис можна за потреби редагувати або видаляти, що дає користувачеві додаткову гнучкість у керуванні своєю інформацією та моніторингу здоров'я.

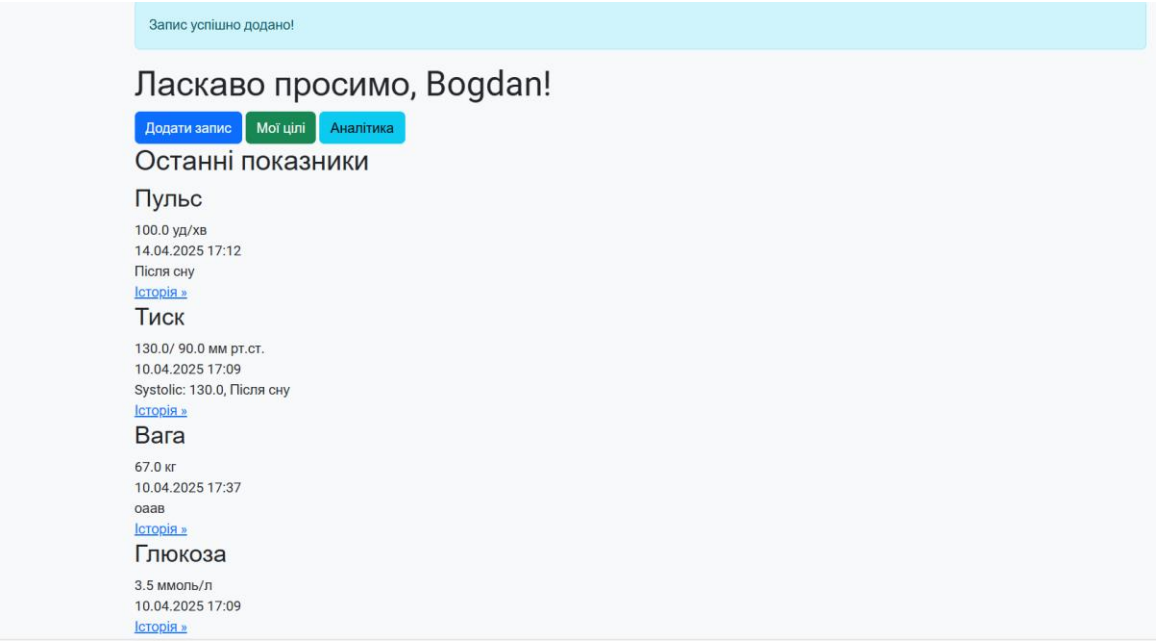

Рисунок 3.16 – Історія останнього запису по кожному параметру

Після натискання на кнопку «Історія» для конкретного показника, користувачеві одразу відображається графік, що ілюструє зміни цього параметра у часі (див. рисунок 3.17).

Графік будується на основі збережених записів, і для кожної позначеної дати виводиться відповідне значення показника. Це дозволяє користувачам легко відстежувати динаміку змін своїх показників здоров'я, порівнювати їх між різними періодами та виявляти потенційні тренди або аномалії. Такий функціонал сприяє більш глибокому аналізу і кращому розумінню стану здоров'я, що є важливим для прийняття обґрунтованих рішень щодо змін у способі життя.

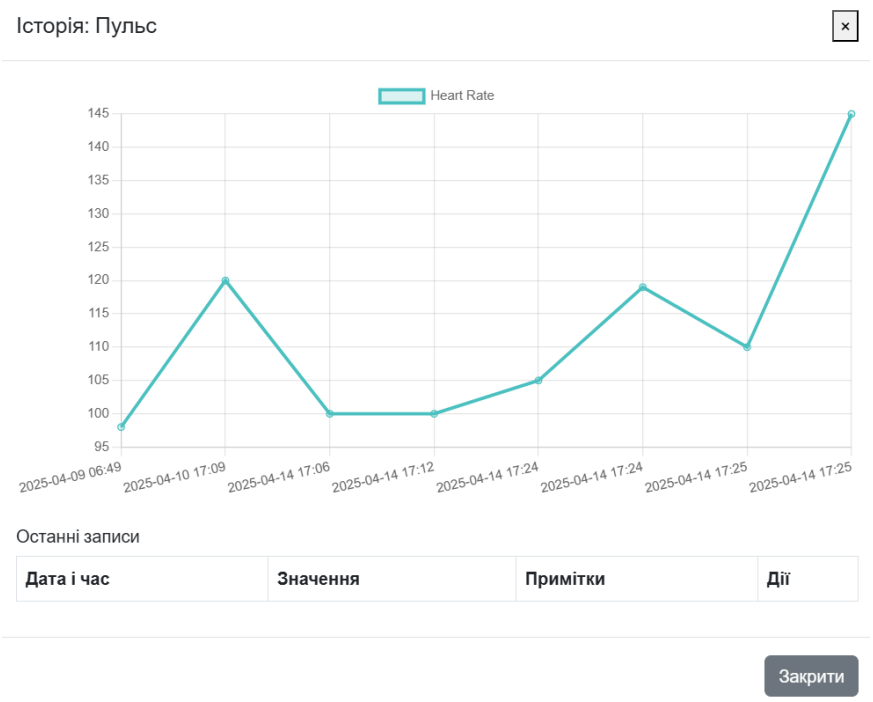

Рисунок 3.17 – Історія для параметру «Пульс»

У вікні «Аналітика» користувач має можливість обрати параметр, за яким бажає відстежувати динаміку своїх показників здоров’я у зручному візуальному форматі (див. рисунок 3.18). Цей інтерфейс дозволяє користувачам легко фільтрувати дані за різними критеріями, такими як період часу (день, тиждень, місяць) або тип показника (вага, пульс, рівень стресу тощо). Користувач також може налаштувати вигляд графіків для зручнішого сприйняття інформації, що дає змогу швидко оцінити зміни в стані здоров'я та приймати відповідні рішення щодо здорового способу життя.

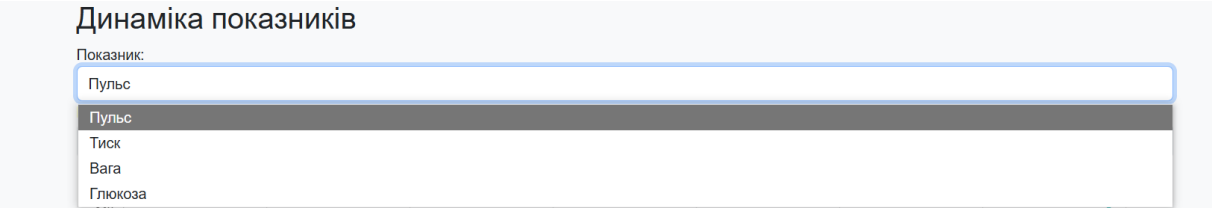

Рисунок 3.18 – Список можливих параметрів відстеження

Після вибору відповідного параметра система автоматично будує графік, що відображає зміни цього показника у часі (див. рисунок 3.19). Цей інструмент дозволяє користувачеві швидко аналізувати стан здоров’я та робити висновки на основі зібраних

даних. Завдяки візуалізації користувач може легко помітити тренди або коливання в своїх показниках, що може бути корисним для виявлення факторів, які впливають на їх здоров'я. Також цей графік допомагає користувачеві краще розуміти, як зміни в їхньому способі життя або дієті можуть впливати на фізичний стан.

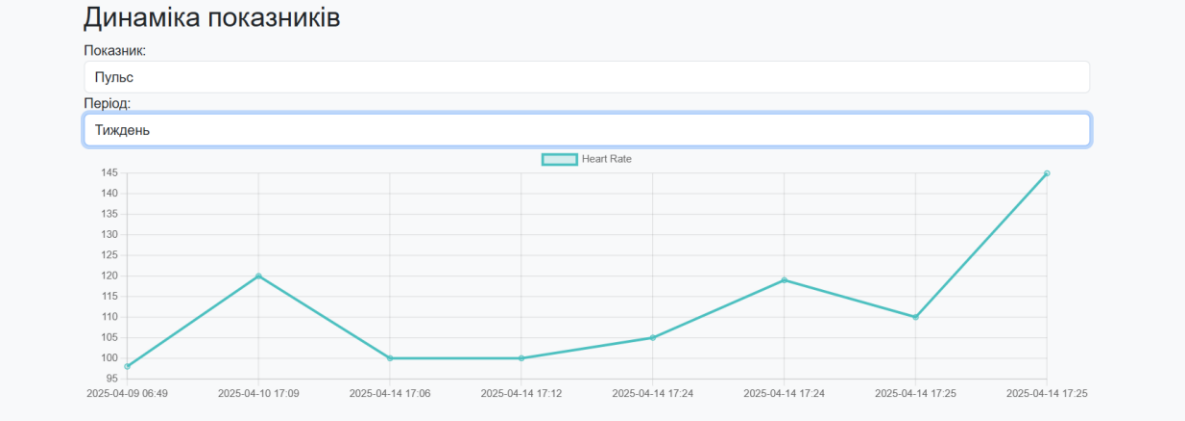

Рисунок 3.19 – Графік змін пульсу за тиждень

Створення візуального інтерфейсу системи моніторингу здоров'я відбулося з використанням HTML, CSS та JavaScript у середовищі розробки PyCharm, що забезпечує гнучкість та ефективність у процесі дизайну. Шаблони вебсторінок розроблені з урахуванням принципів зручності використання та доступності інформації для користувачів системи. Завдяки використанню цих технологій вдалося забезпечити адаптивність інтерфейсу, що дозволяє системі коректно відображатися на різних пристроях, таких як ПК, планшети та смартфони. Крім того, була реалізована інтерактивність елементів інтерфейсу, що підвищує ефективність взаємодії користувачів з вебсистемою, дозволяючи легко здійснювати навігацію та зберігати дані без зайвих труднощів.

3.6 Висновки

У третьому розділі було обґрунтовано вибір програмних засобів для реалізації вебсистеми для моніторингу здоров’я користувачів. З огляду на вимоги до швидкодії, масштабованості та зручності підтримки, як основну технологію серверної частини було обрано фреймворк Flask, що працює з мовою програмування Python. Flask

забезпечує гнучку архітектуру, зручне керування маршрутами та ефективну обробку HTTP-запитів, що є ключовими для вебзастосунків даного типу. Для валідації введених даних та забезпечення базової безпеки було використано стандартні інструменти Flask, зокрема модулі для роботи з формами та сесіями. Таким чином, обраний технологічний стек повністю відповідає потребам розробки функціонального, зручного та розширюваного вебінтерфейсу для моніторингу стану здоров’я користувачів. Також в даному розділі було представлено розроблений вебінтерфейс з детальним його описом.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Аналіз методів тестування програмного забезпечення

Тестування є ключовим етапом життєвого циклу програмного забезпечення, спрямованим на виявлення недоліків, забезпечення стабільності системи та відповідності її функціональним і нефункціональним вимогам [20]. Залежно від мети та стадії розробки використовуються різні методи, що мають свої переваги й обмеження.

Одним із підходів до класифікації методів тестування є розподіл за рівнями. На початкових етапах застосовується тестування окремих компонентів, яке дає змогу ізольовано перевірити логіку конкретних елементів. Далі відбувається перевірка взаємодії між кількома модулями після їх об’єднання в єдину систему. Після інтеграції проводиться загальне тестування готового продукту з точки зору його функціональності та узгодженості із технічним завданням. На завершальному етапі виконується приймальна перевірка, яка оцінює, чи відповідає система очікуванням замовника або кінцевого користувача.

З технічної точки зору розрізняють статичне та динамічне тестування. Статичні методи ґрунтуються на аналізі коду без його виконання, включають рецензування, перевірку документації та автоматизований аналіз джерельного коду на предмет потенційних помилок. Натомість динамічні методи передбачають запуск програми та спостереження за її поведінкою під час виконання.

У межах динамічного підходу існують різні техніки. Один із варіантів передбачає тестування системи без знання її внутрішньої структури, коли перевірка базується лише на зовнішніх функціях та специфікаціях. Інший метод вимагає повного доступу до коду й дозволяє аналізувати логіку програми зсередини. Також використовується комбінований підхід, коли тестувальник має часткову інформацію про внутрішню реалізацію, проте основна увага зосереджена на поведінці системи.

Окремим напрямом є розмежування ручного та автоматизованого тестування. Ручне тестування виконується спеціалістами без залучення додаткових інструментів,

тоді як автоматизовані засоби забезпечують повторюваність, прискорюють перевірку та дозволяють охопити великі обсяги даних при мінімальних витратах часу.

У сучасній практиці також застосовуються поведінкові підходи до тестування, які будуються на основі опису очікуваних сценаріїв взаємодії користувача з системою, а також методи, що враховують ризики і дозволяють зосередитись на найбільш уразливих ділянках програмного забезпечення.

Аналіз наявних підходів показує, що жоден з них не є універсальним. Оптимальний результат досягається шляхом комбінування різних методів залежно від характеристик проєкту, технічних вимог та наявних ресурсів. Виважене застосування тестових стратегій дозволяє значно підвищити якість кінцевого продукту та знизити ймовірність виникнення критичних помилок під час його експлуатації.

4.2 Тестування розробленого програмного застосунку

Тестування вебсистеми – це процес перевірки її функціональності, безпеки та коректності роботи у веббраузерах. Основна мета – переконатися, що система працює стабільно, коректно обробляє дані користувача, відповідає бізнес-логіці та не містить критичних помилок.

Перший етап полягає у розгортанні тестової версії системи на локальному сервері або в тестовому середовищі. Наступним кроком є проведення функціонального тестування, що передбачає перевірку введення та виведення даних, обробку запитів та додавання показників здоров’я, встановлення цілей, а також відображення аналітики. Усі дії перевіряються на відповідність очікуваній логіці поведінки системи.

Розглянемо приклад тестування форми реєстрації (див.рисунок 4.1). У випадку, якщо користувач намагається зареєструватися без заповнення обов’язкового поля (наприклад, логіна), система повинна вивести повідомлення про необхідність заповнити це поле.

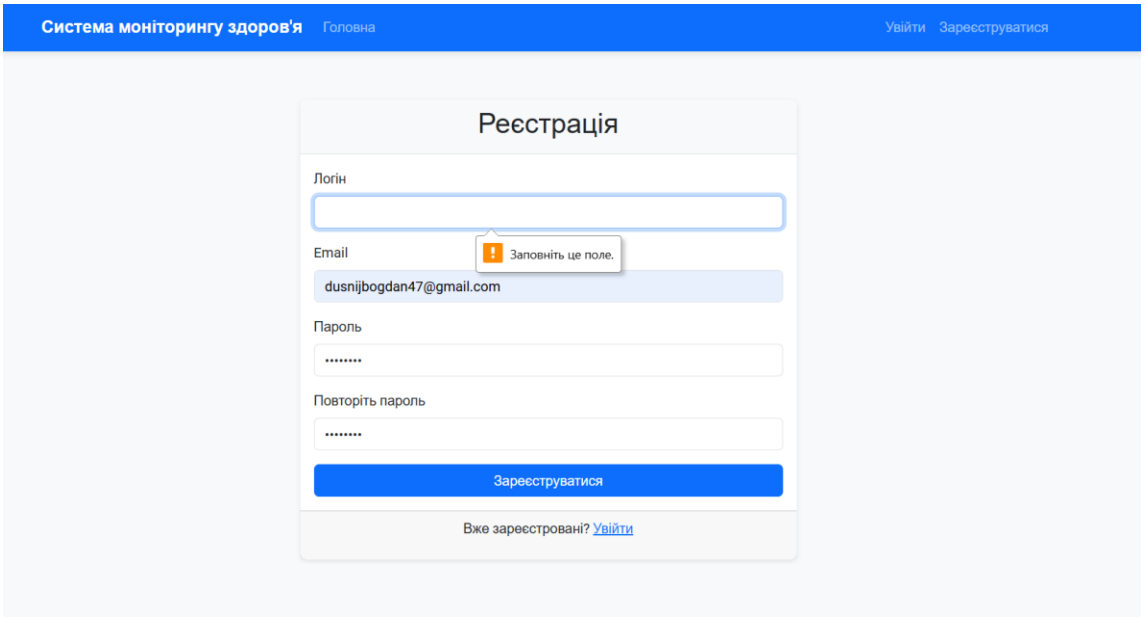

Рисунок 4.1 – Результат реєстрації без заповнення основних полів

Наступним етапом було проведено тестування валідації вхідних даних під час авторизації користувача, що зображено на рисунку 4.2. Зокрема, перевірено ситуацію, коли користувач намагається увійти в акаунт із некоректно заповненими полями, наприклад, з неправильно введеними логіном та паролем. У такому випадку система повинна вивести відповідне повідомлення про помилку та не допустити вхід до облікового запису. Це дозволяє уникнути помилок при введенні даних і запобігає несанкціонованому доступу.

У процесі тестування особливу увагу приділено перевірці реакції системи на різні типи неправильних даних: порожні поля, введення символів замість цифр, використання недопустимих знаків, короткі або надто довгі паролі. У кожному з таких випадків система коректно реагувала на введені помилки, виводячи інформативне попередження для користувача та блокуючи подальшу авторизацію. Це свідчить про надійність реалізованих механізмів валідації.

Такий тип тестування є важливим для забезпечення безпеки вебсистеми, оскільки дозволяє запобігти спробам несанкціонованого доступу, а також покращує зручність для користувача завдяки зрозумілим підказкам щодо помилок. Надійна валідація не лише захищає систему, а й формує довіру користувачів до вебсистеми, адже дає чітке

розуміння, чому введені дані не прийняті. Крім того, це знижує ризик технічних збоїв і полегшує процес усунення помилок на етапі розробки.

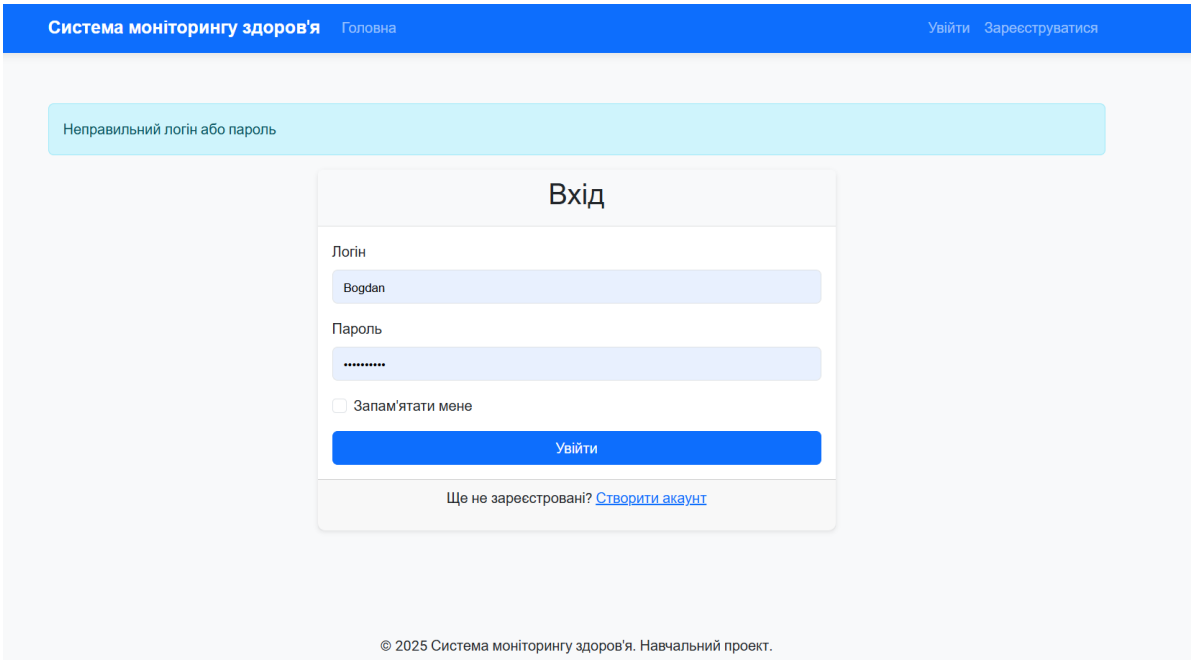

Рисунок 4.2 – Повідомлення про помилку при спробі входу з некоректними даними

Після тестування обробки помилкових даних наступним кроком стала спроба введення коректної інформації у форму входу. Усі обов’язкові поля були заповнені правильно, відповідно до вимог системи. У результаті успішної авторизації вебсистема здійснила перевірку введених даних, надала доступ до облікового запису користувача та перенаправила його на головну сторінку (див.рисунок 4.3).

На головній сторінці користувача зустрічає вітальне повідомлення, що підтверджує успішний вхід до системи та правильну роботу механізму автентифікації.

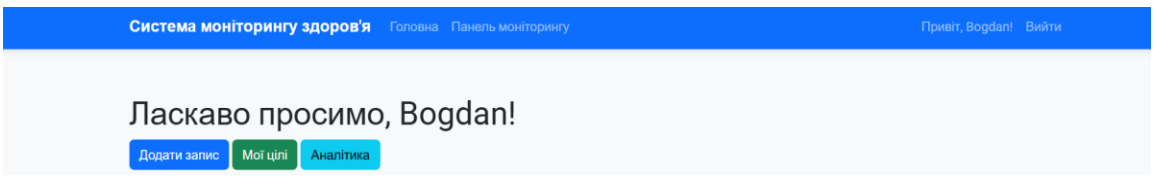

Рисунок 4.3 – Коректно введені дані користувача

Наступним етапом тестування стала перевірка функціональності додавання нового запису про стан здоров’я з некоректно заповненими даними, що зображено на рисунку 4.4. Було змодельовано ситуацію, коли користувач вводить дані у неправильному форматі, наприклад, неможливі та вигадані значення, які не відповідають реальним фізіологічним показникам. Це можуть бути занадто високі або низькі параметри, використання недопустимих символів, або введення тексту замість числових значень.

У таких випадках система повинна виявити всі помилки на етапі валідації, згенерувати відповідне повідомлення для користувача із поясненням проблеми та не допустити збереження помилкових даних до бази. Такий підхід забезпечує цілісність і достовірність зібраної інформації в системі. Крім того, це підвищує надійність програмного продукту та дозволяє уникнути критичних збоїв при подальшій обробці чи аналізі даних.

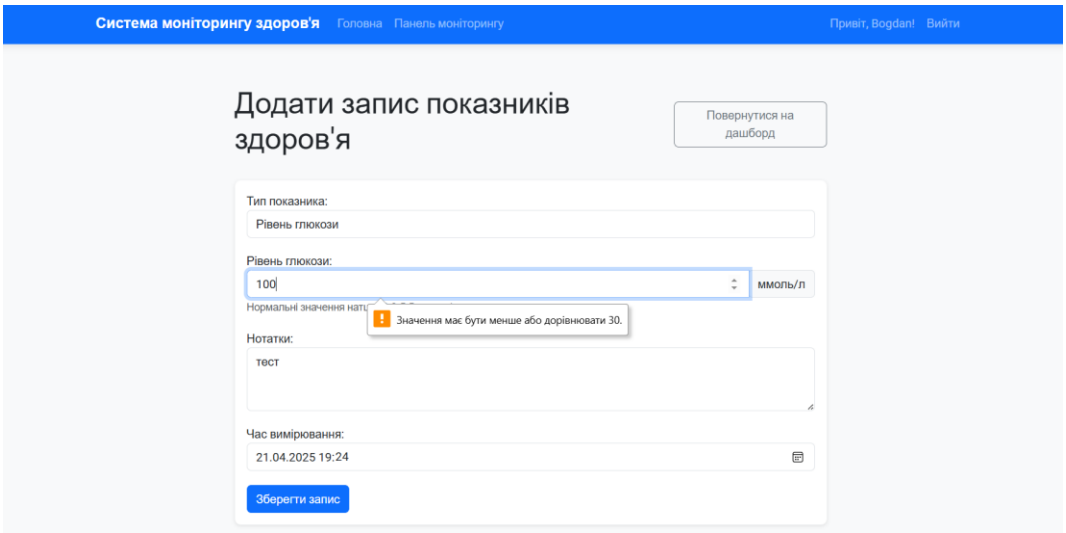

Рисунок 4.4 – Повідомлення про помилку при спробі додати запис із некоректними даними

Після перевірки ситуацій із некоректним введенням даних було здійснено тестування додавання запису про стан здоров’я з правильно заповненою формою. Усі обов’язкові поля були введені у відповідному форматі, що дозволило системі успішно обробити запит.

У результаті система зберегла новий запис у базі даних і відобразила повідомлення про успішне додавання (див.рисунок 4.5), що свідчить про правильну роботу механізму валідації та обробки даних. Користувач був автоматично перенаправлений на оновлену сторінку з відображенням нового запису в загальному списку.

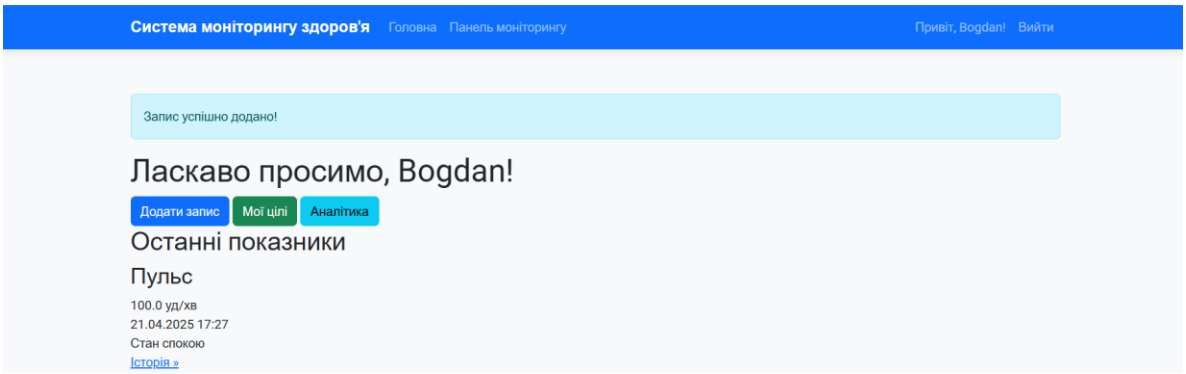

Рисунок 4.5 – Повідомлення про успішне додавання запису про стан здоров’я

Отже, проведене тестування вебсистеми моніторингу стану здоров’я підтвердило її працездатність у різних сценаріях взаємодії з користувачем. Система коректно обробляє як помилкове, так і правильне введення даних, надає зворотний зв’язок через повідомлення, забезпечує базову валідацію форм та збереження інформації в базі даних. Це свідчить про стабільну роботу основних функціональних компонентів системи та готовність до подальшого використання.

4.3 Розробка інструкції користувача вебсистеми

Інструкція користувача містить основні рекомендації щодо використання вебсистеми моніторингу стану здоров’я. Перед початком роботи користувач повинен переконатися, що його пристрій відповідає технічним вимогам для стабільної роботи вебсистеми. Детальні технічні вимоги подано в таблицях 4.1 та 4.2. Крім того, перед використанням системи рекомендується ознайомитися з основними функціями платформи, щоб швидше освоїти її можливості та отримати максимальну користь від моніторингу показників здоров’я.

Інтерфейс системи побудовано так, щоб кожен користувач міг інтуїтивно розібратися у функціях, а також отримати корисні поради на кожному етапі взаємодії з платформою.

Таблиця 4.1 – Мінімальні технічні вимоги:

Параметр	Значення
Тип процесора	Двоядерний, 1.6 GHz і вище
Об’єм оперативної пам’яті	2 ГБ
Місце на диску	200 МБ (для кешу браузера)
Операційна система	Windows 7 / macOS 10.13 / Linux
Веббраузер	Google Chrome 90+, Firefox 88+
Доступ до Інтернету	Мінімум 2 Мбіт/с

Таблиця 4.2 – Рекомендована технічні вимоги:

Параметр	Значення
Тип процесора	Чотириядерний, 2.4 GHz і вище
Об’єм оперативної пам’яті	4 ГБ і більше
Місце на диску	500 МБ
Операційна система	Windows 10 / macOS 12 / Linux
Веббраузер	Остання версія Chrome, Firefox або Edge

Після відкриття вебсистеми користувач потрапляє на головну сторінку, де пропонується авторизуватись або зареєструвати новий обліковий запис. У разі успішної авторизації користувач отримує доступ до персонального кабінету з основним функціоналом системи. У персональному кабінеті користувач може додавати записи про стан здоров’я, переглядати попередньо збережені показники, встановлювати цілі, а також аналізувати прогрес за допомогою графіків та статистики. Якщо користувач не

авторизований, доступ до функцій додавання, редагування даних або перегляду аналітики буде обмежений – система запропонує увійти або зареєструватись.

Форма додавання запису дозволяє вказати тип показника здоров’я, його значення, дату, за потреби – систолічний тиск та текстові примітки. У разі успішного збереження нового запису система виводить повідомлення про успішну дію та оновлює список записів. Користувач також має можливість редагувати або видаляти записи, що дає додаткову гнучкість у використанні системи.

Крім цього, користувач може переглядати динаміку своїх показників здоров’я у вигляді графіків, що візуалізують зміни за обраний період. Це дозволяє краще контролювати стан свого здоров’я та вчасно реагувати на відхилення. Аналіз змін у показниках на графіках допомагає користувачам відслідковувати тренди та своєчасно коригувати спосіб життя або звертатися до медичних спеціалістів.

Усі дії користувача, пов’язані з введенням та переглядом даних, відображаються у внутрішній історії дій, що формує своєрідний електронний щоденник здоров’я. Ця історія дає змогу користувачеві переглядати всі попередні записи та відслідковувати загальний прогрес за певний період часу, що сприяє кращому розумінню змін у стані здоров’я.

4.4 Висновки

У четвертому розділі кваліфікаційної роботи, було проведено комплексне тестування програмних модулів вебсистеми моніторингу стану здоров’я. Основну увагу приділено перевірці коректності обробки користувацьких дій: валідації введених даних під час реєстрації, авторизації та додавання записів про стан здоров’я. Тестування охоплювало функції введення, збереження, редагування даних і побудови графіків.

Крім того, створено інструкцію користувача з описом технічних вимог та основних дій у системі – від реєстрації до перегляду аналітики, що спрощує адаптацію користувачів і сприяє ефективному використанню вебсистеми.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено вебсистему для моніторингу стану здоров’я користувачів. Система включає такі основні функціональні модулі:

– модуль реєстрації та авторизації користувачів, що забезпечує безпечний доступ до персонального кабінету;

– модуль панелі керування, через який користувач може керувати своїми даними та переглядати історію внесених показників;

– модуль додавання записів показників здоров’я, який дозволяє фіксувати, зберігати й переглядати динаміку фізіологічних параметрів.

Для реалізації системи було спроєктовано архітектуру, розроблено логіку взаємодії між модулями, а також реалізовано алгоритми обробки та збереження даних. У якості мови програмування використано Python у поєднанні з фреймворком Flask. Розробка здійснювалася в середовищі PyCharm.

У першому розділі роботи проведено огляд сучасних рішень у сфері цифрового моніторингу здоров’я, визначено їхні переваги та недоліки, що дало змогу сформувати вимоги до власної системи. У другому розділі описано методику побудови вебзастосунку, зокрема модель роботи системи та принципи обробки користувацьких даних. Третій розділ присвячено технічній реалізації: обґрунтовано вибір мови програмування, фреймворку та середовища розробки, створено окремі модулі й інтерфейс користувача. У четвертому розділі проведено тестування функціональних можливостей вебсистеми, що підтвердило її коректну роботу відповідно до заданих функціональних вимог.

Система є зручною у використанні, придатною для подальшого масштабування, зокрема для інтеграції з пристроями для автоматичного зчитування показників або з медичними системами для віддаленого моніторингу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Цифрові трансформації системи охорони здоров’я [Електронний ресурс] – Режим доступу до ресурсу: https://is.gd/FFeb8Z (дата звернення: 03.03.2025)

2. Голян В., Самойленко Д. Створення архітектури програмної системи моніторингу інформації про стан здоров’я людини [Електронний ресурс] – Режим доступу до ресурсу: https://europub.co.uk/articles/stvorennia-arxitekturi-programnoyi-sistemi-monitoringu-informaciyi-pro-stan-zdorovia-liudini-A-477489.

3. Душний Б.О. Вебсистема для моніторингу стану здоров'я./ XII Всеукраїнська науково-практична конференція молодих учених, Київ, 2025. [Електронний ресурс] – Режим доступу до ресурсу: https://zcit.kubg.edu.ua/index.php/journal/issue/view/13/23/3.

4. Цифрові технології та телемедицина [Електронний ресурс] – Режим доступу до ресурсу: https://www.who.int/news-room/fact-sheets (дата звернення: 07.03.2025)

5. Концепця превентивної медицини [Електронний ресурс] – Режим доступу до ресурсу: https://is.gd/gWyKq5 (дата звернення: 07.03.2025)

6. Носимі пристрої для моніторингу здоров’я [Електронний ресурс] – Режим доступу до ресурсу: https://itc.ua/articles/yak-nosimi-gadzheti-dopomagayut-stezhiti-za-zdorov-yam/ (дата звернення: 07.03.2025)

7. MyFitnessPal [Електронний ресурс] – Режим доступу до ресурсу: https://icoola.ua/blog/my-fitness-pal-app (дата звернення: 09.03.2025)

8. Fitbit [Електронний ресурс] – Режим доступу до ресурсу: https://surli.cc/gibgns (дата звернення: 09.03.2025)

9. Samsung Health [Електронний ресурс] – Режим доступу до ресурсу: https://v.gd/Zh9f4M (дата звернення: 09.03.2025)

10. Garmin Connect [Електронний ресурс] – Режим доступу до ресурсу: https://connect.garmin.com/ (дата звернення: 09.03.2025)

11. Технології моніторингу здоров’я [Електронний ресурс] – Режим доступу до

ресурсу: https://medtech.ua/articles /monitoring-zdorovya-texnologii/ (дата звернення: 15.03.2025)

12. Hypersense Software. Designing User-Friendly Interfaces for Healthcare Applications [Електронний ресурс] – Режим доступу до ресурсу: https://v.gd/9KvrJr (дата звернення: 18.03.2025)

13. UML Activity Diagram [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/40575/ (дата звернення: 21.03.2025)

14. Python [Електронний ресурс] – Режим доступу до ресурсу: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-jazik-programmirovanija-python/ (дата звернення: 25.03.2025)

15. Середовище розробки PyCharm [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/pycharm/ (дата звернення: 25.03.2025)

16. Flask Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/stable/ (дата звернення: 25.03.2025)

17. SQLAlchemy Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.sqlalchemy.org/ (дата звернення: 01.04.2025)

18. Flask-Login [Електронний ресурс] – Режим доступу до ресурсу: https://flasklogin.readthedocs.io/en/latest/ (дата звернення: 01.04.2025)

19. Werkzeug Security Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://werkzeug.palletsprojects.com/en/3.0.x/utils/#module-werkzeug.security (дата звернення: 01.04.2025)

20. Що таке тестування ПЗ [Електронний ресурс] – Режим доступу до ресурсу: https://university.sigma.software/what-is-software-testing/ (дата звернення: 10.04.2025)

Додатки

57

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
_____ О. Н. Романюк
«24» березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на бакалаврську кваліфікаційну роботу «Вебсистема для моніторингу стану здоров'я»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____ доц. каф. ПЗ Черноволик Г.О.
«24» 03 2025р.

Виконав:
_____ студент гр. 4ПІ-21Б Душний Б.О.
«24» 03 2025р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка вебсистеми для моніторингу стану здоров’я».

Галузь застосування – сфера охорони здоров’я: медичні заклади.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від «20» березня 2025 р. ректора ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою роботи є підвищення якості моніторингу стану здоров’я користувачів шляхом впровадження інформаційної системи, що забезпечує зручний доступ до персональних даних, ефективне збереження та обробку інформації, а також формування аналітики і персоналізованих рекомендацій.

Призначення роботи – створення веборієнтованої системи, яка дозволяє користувачам фіксувати, переглядати та аналізувати дані про стан здоров’я за допомогою інтерактивного інтерфейсу.

4. Вихідні дані для проведення НДР

1. Цифрові трансформації системи охорони здоров’я [Електронний ресурс] – Режим доступу до ресурсу: https://is.gd/FFeb8Z (дата звернення: 03.03.2025)

2. Голян В., Самойленко Д. Створення архітектури програмної системи моніторингу інформації про стан здоров’я людини [Електронний ресурс] – Режим доступу до ресурсу: https://europub.co.uk/articles/stvorennia-arxitekturi-programnoyi-sistemi-monitoringu-informaciyi-pro-stan-zdorovia-liudini-A-477489.

3. Цифрові технології та телемедицина [Електронний ресурс] – Режим доступу до ресурсу: https://www.who.int/news-room/fact-sheets (дата звернення: 07.03.2025)

5. Технічні вимоги

Комп’ютер з 64-розрядним (x64) процесором та тактовою частотою 2 ГГц. Об’єм оперативної пам’яті 8 ГБ, розмір жорсткого диску 256 ГБ. Операційна система Windows 10. З клієнтського боку: мобільний пристрій з ОС Android; ПК з будь-якою операційною системою та будь-який браузер.

6. Конструктивні вимоги.

Вебистема повинна відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

- пояснювальна записка до бакалаврської кваліфікаційної роботи;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку програм для комунікацій в системах телемедицини та постановка задач розробки	25.03.2025 – 30.03.2025
2	Розробка методів, моделі та алгоритмів роботи програми	05.04.2025 – 15.04.2025
3	Розробка програмного забезпечення	20.04.2025 – 25.04.2025
4	Тестування програми	26.04.2025 – 02.05.2025
5	Оформлення матеріалів до захисту БКР	10.05.2025 – 18.05.2025

10. Порядок контролю та прийняття.

Всі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додадот Б. Перевірка на плагіат

61

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка вебсистеми для моніторингу стану здоров'я»

Тип роботи: Бакалаврська кваліфікаційна робота

(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: кафедра програмного забезпечення , ФІТКІ

(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 9,4 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____

(прізвище, ініціали, посада) (підпис)

_____ _____

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Черноволик Г. О.

(підпис) (прізвище, ініціали, посада)

Здобувач _____ Душний Б.О.

(підпис) (прізвище, ініціали)

Додаток В. Лістинг програми

```
# app.py
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager

# Ініціалізація додатку
app = Flask(name)
app.config.from_object('config.Config')

# Ініціалізація бази даних
db = SQLAlchemy(app)

# Ініціалізація менеджера авторизації
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'auth.login'

# Імпорт моделей даних
from models import User, HealthRecord

# Створення таблиць бази даних
with app.app_context():
    db.create_all()

# Реєстрація маршрутів
from routes.auth import auth as auth_blueprint
from routes.dashboard import dashboard as dashboard_blueprint

app.register_blueprint(auth_blueprint)
app.register_blueprint(dashboard_blueprint)

@app.route('/')
def home():
    return render_template('home.html')

if name == 'main':
    app.run(debug=True)

# config.py
import os

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'this-is-a-dev-key'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or
'sqlite:///health_monitor.db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False

# models.py
```

```
from app import db, login_manager
from flask_login import UserMixin
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime

@login_manager.user_loader
def load_user(id):
    return User.query.get(int(id))

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), index=True, unique=True)
    email = db.Column(db.String(120), index=True, unique=True)
    password_hash = db.Column(db.String(128))
    health_records = db.relationship('HealthRecord', backref='owner', lazy='dynamic')

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)

    def repr(self):
        return f'<User {self.username}>'

class HealthRecord(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    record_date = db.Column(db.DateTime, default=datetime.utcnow)
    record_type = db.Column(db.String(64))  # e.g., 'heart_rate', 'blood_pressure', 'weight'
    value = db.Column(db.Float)
    notes = db.Column(db.Text, nullable=True)

    def repr(self):
        return f'<HealthRecord {self.record_type}: {self.value}>'

# routes/__init__.py
# Порожній файл для ініціалізації пакету routes

# routes/auth.py
from flask import Blueprint, render_template, redirect, url_for, flash, request
from flask_login import login_user, logout_user, login_required, current_user
from werkzeug.urls import url_parse
from app import db
from models import User
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, BooleanField, SubmitField
from wtforms.validators import DataRequired, Email, EqualTo, ValidationError

auth = Blueprint('auth', name)

class LoginForm(FlaskForm):
```

```
        username = StringField('Логін', validators=[DataRequired()])
        password = PasswordField('Пароль', validators=[DataRequired()])
        remember_me = BooleanField('Запам\'ятати мене')
        submit = SubmitField('Увійти')

    Богдан Душний, [08.04.2025 22:17]
    class RegistrationForm(FlaskForm):
        username = StringField('Логін', validators=[DataRequired()])
        email = StringField('Email', validators=[DataRequired(), Email()])
        password = PasswordField('Пароль', validators=[DataRequired()])
        password2 = PasswordField('Повторіть пароль', validators=[DataRequired(),
EqualTo('password')])
        submit = SubmitField('Зареєструватися')

        def validate_username(self, username):
            user = User.query.filter_by(username=username.data).first()
            if user is not None:
                raise ValidationError('Цей логін вже використовується.')

        def validate_email(self, email):
            user = User.query.filter_by(email=email.data).first()
            if user is not None:
                raise ValidationError('Цей email вже використовується.')

    @auth.route('/login', methods=['GET', 'POST'])
    def login():
        if current_user.is_authenticated:
            return redirect(url_for('dashboard.index'))

        form = LoginForm()
        if form.validate_on_submit():
            user = User.query.filter_by(username=form.username.data).first()

            if user is None or not user.check_password(form.password.data):
                flash('Неправильний логін або пароль')
                return redirect(url_for('auth.login'))

            login_user(user, remember=form.remember_me.data)
            next_page = request.args.get('next')

            if not next_page or url_parse(next_page).netloc != '':
                next_page = url_for('dashboard.index')

            return redirect(next_page)

        return render_template('auth/login.html', title='Вхід', form=form)

    @auth.route('/logout')
    def logout():
        logout_user()
        return redirect(url_for('home'))
```

```
@auth.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('dashboard.index'))

    form = RegistrationForm()
    if form.validate_on_submit():
        user = User(username=form.username.data, email=form.email.data)
        user.set_password(form.password.data)
        db.session.add(user)
        db.session.commit()

        flash('Ви успішно зареєструвалися!')
        return redirect(url_for('auth.login'))

    return render_template('auth/register.html', title='Реєстрація', form=form)

# routes/dashboard.py
from flask import Blueprint, render_template, redirect, url_for, flash, request
from flask_login import login_required, current_user
from app import db
from models import HealthRecord
from flask_wtf import FlaskForm
from wtforms import StringField, FloatField, TextAreaField, SelectField, SubmitField
from wtforms.validators import DataRequired

dashboard = Blueprint('dashboard', name)

class RecordForm(FlaskForm):
    record_type = SelectField('Тип запису', choices=[
        ('heart_rate', 'Пульс'),
        ('blood_pressure', 'Тиск'),
        ('weight', 'Вага'),
        ('temperature', 'Температура'),
        ('glucose', 'Рівень цукру')
    ])
    value = FloatField('Значення', validators=[DataRequired()])
    notes = TextAreaField('Нотатки')
    submit = SubmitField('Зберегти')

@dashboard.route('/dashboard')
@login_required
def index():
    records =
HealthRecord.query.filter_by(user_id=current_user.id).order_by(HealthRecord.record_date.desc()).all()
    return render_template('dashboard/index.html', title='Панель моніторингу', records=records)

@dashboard.route('/dashboard/add', methods=['GET', 'POST'])
@login_required
def add_record():
    form = RecordForm()
    if form.validate_on_submit():
```

```
            record = HealthRecord(
                user_id=current_user.id,
                record_type=form.record_type.data,
                value=form.value.data,
                notes=form.notes.data
            )
            db.session.add(record)
            db.session.commit()
            flash('Запис успішно додано!')
            return redirect(url_for('dashboard.index'))
        return render_template('dashboard/add_record.html', title='Додати запис', form=form)
    # templates/base.html
    <!DOCTYPE html>
    <html lang="uk">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>{% block title %}Система моніторингу здоров'я{% endblock %}</title>
        <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/css/bootstrap.min.css">
        <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
        {% block styles %}{% endblock %}
    </head>
    <body>
        <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
            <div class="container">
                <a class="navbar-brand" href="{{ url_for('home') }}">Система моніторингу здоров'я</a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="collapse navbar-collapse" id="navbarNav">
                    <ul class="navbar-nav me-auto">
                        <li class="nav-item">
                            <a class="nav-link" href="{{ url_for('home') }}">Головна</a>
                        </li>
                        {% if current_user.is_authenticated %}
                        <li class="nav-item">
                            <a class="nav-link" href="{{ url_for('dashboard.index') }}">Панель
моніторингу</a>
                        </li>
                        {% endif %}
                    </ul>
                    <ul class="navbar-nav">
                        {% if current_user.is_authenticated %}
                        <li class="nav-item">
                            <span class="nav-link">Привіт, {{ current_user.username }}!</span>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" href="{{ url_for('auth.logout') }}">Вийти</a>
                        </li>
                        {% else %}
```

```
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url_for('auth.login') }}">Увійти</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url_for('auth.register') }}">Зареєструватися</a>
                    </li>
                    {% endif %}
                </ul>
            </div>
        </div>
    </nav>

    <main class="container mt-4">
        {% with messages = get_flashed_messages() %}
        {% if messages %}
        <div class="row">
            <div class="col">
                {% for message in messages %}
                <div class="alert alert-info">{{ message }}</div>
                {% endfor %}
            </div>
        </div>
        {% endif %}
        {% endwith %}

        {% block content %}{% endblock %}
    </main>

    <footer class="mt-5 py-3 bg-light">
        <div class="container text-center">
            <p>© 2025 Система моніторингу здоров'я. Навчальний проект.</p>
        </div>
    </footer>

    <script
src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/js/bootstrap.bundle.min.js"></script>
    <script src="{{ url_for('static', filename='js/main.js') }}"></script>
    {% block scripts %}{% endblock %}
</body>
</html>


from extensions import db, login_manager
from flask_login import UserMixin
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime


class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), index=True, unique=True)
    email = db.Column(db.String(120), index=True, unique=True)
```

```
    password_hash = db.Column(db.String(128))
    health_records = db.relationship('HealthRecord', backref='owner', lazy='dynamic')
    health_goals = db.relationship('HealthGoal', backref='owner', lazy='dynamic')

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)

    def __repr__(self):
        return f'<User {self.username}>'



class HealthRecord(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    record_date = db.Column(db.DateTime, default=datetime.utcnow)
    record_type = db.Column(db.String(64))  # e.g., 'heart_rate', 'blood_pressure', 'weight', 'glucose'
    value = db.Column(db.Float)
    value_systolic = db.Column(db.Float, nullable=True)  # Для систолічного тиску
    notes = db.Column(db.Text, nullable=True)

    def __repr__(self):
        return f'<HealthRecord {self.record_type}: {self.value}>'


class HealthGoal(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    goal_type = db.Column(db.String(64))  # e.g., 'weight', 'blood_pressure', 'heart_rate'
    target_value = db.Column(db.Float)
    target_value_systolic = db.Column(db.Float, nullable=True)  # Для цілей систолічного тиску
    start_date = db.Column(db.DateTime, default=datetime.utcnow)
    target_date = db.Column(db.DateTime, nullable=False)
    description = db.Column(db.String(200))
    completed = db.Column(db.Boolean, default=False)

    def __repr__(self):
        return f'<HealthGoal {self.goal_type}: {self.target_value} by {self.target_date}>'


class NormalRanges(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    record_type = db.Column(db.String(64), unique=True)
    min_value = db.Column(db.Float)
    max_value = db.Column(db.Float)
    min_value_systolic = db.Column(db.Float, nullable=True)  # Для систолічного тиску
    max_value_systolic = db.Column(db.Float, nullable=True)  # Для систолічного тиску

    def __repr__(self):
```

```
        return f'<NormalRange {self.record_type}: {self.min_value}-{self.max_value}>'


@login_manager.user_loader
def load_user(id):
    return User.query.get(int(id))

from flask import Blueprint, render_template, request, redirect, url_for, flash, jsonify
from flask_login import login_required, current_user
from models import HealthRecord, HealthGoal, NormalRanges
from extensions import db
from datetime import datetime, timedelta

# Змінюємо назву Blueprint з dashboard_routes на dashboard
dashboard = Blueprint('dashboard', __name__)

@dashboard.route('/')
@login_required
def index():
    # Отримуємо останні записи для відображення на дашборді
    recent_records = {}
    record_types = ['heart_rate', 'blood_pressure', 'weight', 'glucose']

    for record_type in record_types:
        records = HealthRecord.query.filter_by(
            user_id=current_user.id,
            record_type=record_type
        ).order_by(HealthRecord.record_date.desc()).limit(5).all()

        recent_records[record_type] = records

    # Отримуємо активні цілі для відображення на дашборді
    active_goals = HealthGoal.query.filter_by(
        user_id=current_user.id,
        completed=False
    ).all()

    # Отримуємо нормальні діапазони для відображення індикаторів
    normal_ranges = {}
    for nr in NormalRanges.query.all():
        normal_ranges[nr.record_type] = nr

    return render_template('dashboard/index.html',
                           recent_records=recent_records,
                           active_goals=active_goals,
                           normal_ranges=normal_ranges)


@dashboard.route('/add_record', methods=['GET', 'POST'])
@login_required
def add_record():
```

```
    if request.method == 'POST':
        record_type = request.form['record_type']
        notes = request.form.get('notes', '')

        try:
            # Для кров'яного тиску обробляємо окремо
            if record_type == 'blood_pressure':
                # Перевіряємо поля для кров'яного тиску
                if 'value_systolic' not in request.form or not request.form['value_systolic']:
                    flash('Будь ласка, введіть значення для систолічного тиску', 'danger')
                    return render_template('dashboard/add_record.html', now=datetime.now().strftime('%Y-%m-
%dT%H:%M'))

                if 'value' not in request.form or not request.form['value']:
                    flash('Будь ласка, введіть значення для діастолічного тиску', 'danger')
                    return render_template('dashboard/add_record.html', now=datetime.now().strftime('%Y-%m-
%dT%H:%M'))

                value = float(request.form['value'])  # Діастолічний тиск
                value_systolic = float(request.form['value_systolic'])

                # Зберігаємо систолічний тиск в примітках і у відповідному полі
                notes = f"Systolic: {value_systolic}, " + notes

                record = HealthRecord(
                    user_id=current_user.id,
                    record_type=record_type,
                    value=value,  # Діастолічний
                    value_systolic=value_systolic,  # Систолічний
                    notes=notes
                )
            else:
                # Для інших типів показників
                if 'value' not in request.form or not request.form['value'].strip():
                    flash('Будь ласка, введіть значення для вибраного показника', 'danger')
                    return render_template('dashboard/add_record.html', now=datetime.now().strftime('%Y-%m-
%dT%H:%M'))

                value = float(request.form['value'])

                record = HealthRecord(
                    user_id=current_user.id,
                    record_type=record_type,
                    value=value,
                    notes=notes
                )

            db.session.add(record)
            db.session.commit()

            flash('Запис успішно додано!', 'success')
```

```
            return redirect(url_for('dashboard.index'))

        except ValueError as e:
            flash(f'Некоректне значення. Будь ласка, введіть числове значення. Деталі: {str(e)}', 'danger')
            return render_template('dashboard/add_record.html', now=datetime.now().strftime('%Y-%m-
%dT%H:%M'))

    # Передаємо поточну дату і час для поля вимірювання
    now = datetime.now().strftime('%Y-%m-%dT%H:%M')
    return render_template('dashboard/add_record.html', now=now)

    # Передаємо поточну дату і час для поля вимірювання
    now = datetime.now().strftime('%Y-%m-%dT%H:%M')
    return render_template('dashboard/add_record.html', now=now)


@dashboard.route('/goals', methods=['GET', 'POST'])
@login_required
def goals():
    if request.method == 'POST':
        goal_type = request.form['goal_type']
        target_value = float(request.form['target_value'])
        target_date_str = request.form['target_date']
        description = request.form['description']

        # Для кров'яного тиску зберігаємо обидва значення в description
        if goal_type == 'blood_pressure':
            target_value_systolic = request.form['target_value_systolic']
            description = f"Systolic Target: {target_value_systolic}, " + description

        target_date = datetime.strptime(target_date_str, '%Y-%m-%d')

        goal = HealthGoal(
            user_id=current_user.id,
            goal_type=goal_type,
            target_value=target_value,
            target_date=target_date,
            description=description
        )

        db.session.add(goal)
        db.session.commit()

        flash('Ціль успішно додано!', 'success')
        return redirect(url_for('dashboard.goals'))

    # Отримуємо всі цілі користувача
    active_goals = HealthGoal.query.filter_by(
        user_id=current_user.id,
        completed=False
    ).all()
```

```
    completed_goals = HealthGoal.query.filter_by(
        user_id=current_user.id,
        completed=True
    ).all()

    return render_template('dashboard/goals.html',
                           active_goals=active_goals,
                           completed_goals=completed_goals)


@dashboard.route('/analytics')
@login_required
def analytics():
    # Визначаємо проміжок часу для аналізу
    period = request.args.get('period', 'month')

    if period == 'week':
        start_date = datetime.utcnow() - timedelta(days=7)
    elif period == 'month':
        start_date = datetime.utcnow() - timedelta(days=30)
    elif period == 'year':
        start_date = datetime.utcnow() - timedelta(days=365)
    else:
        start_date = datetime.utcnow() - timedelta(days=30)  # default: month

    # Отримуємо записи за вказаний період
    records = HealthRecord.query.filter(
        HealthRecord.user_id == current_user.id,
        HealthRecord.record_date >= start_date
    ).order_by(HealthRecord.record_date).all()

    # Організовуємо записи за типами
    records_by_type = {}
    for record in records:
        if record.record_type not in records_by_type:
            records_by_type[record.record_type] = []
        records_by_type[record.record_type].append(record)

    # Обчислюємо статистику для кожного типу показників
    stats = {}
    for record_type, type_records in records_by_type.items():
        if record_type == 'blood_pressure':
            # Для кров'яного тиску - тільки діастолічний
            diastolic_values = [r.value for r in type_records]

            if diastolic_values:
                stats[record_type] = {
                    'avg': sum(diastolic_values) / len(diastolic_values),
                    'min': min(diastolic_values),
                    'max': max(diastolic_values)
```

```
            }
        else:
            values = [r.value for r in type_records]
            if values:
                stats[record_type] = {
                    'avg': sum(values) / len(values),
                    'min': min(values),
                    'max': max(values)
                }

    # Отримуємо нормальні діапазони
    normal_ranges = {}
    for nr in NormalRanges.query.all():
        normal_ranges[nr.record_type] = nr

    return render_template('dashboard/analytics.html',
                           records_by_type=records_by_type,
                           stats=stats,
                           normal_ranges=normal_ranges,
                           period=period)


@dashboard.route('/virtual_doctor')
@login_required
def virtual_doctor():
    return render_template('dashboard/virtual_doctor.html')


@dashboard.route('/api/chart_data/<record_type>')
@login_required
def chart_data(record_type):
    # Визначаємо проміжок часу для даних графіка
    period = request.args.get('period', 'month')

    if period == 'week':
        start_date = datetime.utcnow() - timedelta(days=7)
    elif period == 'month':
        start_date = datetime.utcnow() - timedelta(days=30)
    elif period == 'year':
        start_date = datetime.utcnow() - timedelta(days=365)
    else:
        start_date = datetime.utcnow() - timedelta(days=30)  # default: month

    # Отримуємо записи вказаного типу за вказаний період
    records = HealthRecord.query.filter(
        HealthRecord.user_id == current_user.id,
        HealthRecord.record_type == record_type,
        HealthRecord.record_date >= start_date
    ).order_by(HealthRecord.record_date).all()

    # Формуємо дані для графіка
```

```
    chart_data = {
        'labels': [],
        'datasets': []
    }

    if record_type == 'blood_pressure':
        # Для кров'яного тиску - тільки діастолічний тиск
        diastolic_data = []

        for record in records:
            date_str = record.record_date.strftime('%Y-%m-%d %H:%M')
            chart_data['labels'].append(date_str)
            diastolic_data.append(record.value)

        chart_data['datasets'] = [
            {
                'label': 'Діастолічний',
                'data': diastolic_data,
                'borderColor': 'rgba(54, 162, 235, 1)',
                'backgroundColor': 'rgba(54, 162, 235, 0.2)',
            }
        ]
    else:
        # Для інших типів будуємо один набір даних
        values = []

        for record in records:
            date_str = record.record_date.strftime('%Y-%m-%d %H:%M')
            chart_data['labels'].append(date_str)
            values.append(record.value)

        chart_data['datasets'] = [
            {
                'label': record_type.replace('_', ' ').title(),
                'data': values,
                'borderColor': 'rgba(75, 192, 192, 1)',
                'backgroundColor': 'rgba(75, 192, 192, 0.2)',
            }
        ]

    return jsonify(chart_data)


@dashboard.route('/api/goal/<int:goal_id>/complete', methods=['POST'])
@login_required
def complete_goal(goal_id):

    # Перевіряємо, чи ціль належить поточному користувачу
    # Має бути додана ця строка перед перевіркою user_id
    goal = HealthGoal.query.get_or_404(goal_id)
    if goal.user_id != current_user.id:
```

```
        return jsonify({'success': False, 'message': 'Доступ заборонено'}), 403

    goal.completed = True
    db.session.commit()

    return jsonify({'success': True})


@dashboard.route('/api/goal/<int:goal_id>/delete', methods=['POST'])
@login_required
def delete_goal(goal_id):
    goal = HealthGoal.query.get_or_404(goal_id)

    # Перевіряємо, чи ціль належить поточному користувачу
    if goal.user_id != current_user.id:
        return jsonify({'success': False, 'message': 'Доступ заборонено'}), 403

    db.session.delete(goal)
    db.session.commit()

    return jsonify({'success': True})

    @dashboard.route('/api/goal/<int:goal_id>/delete', methods=['POST'])
    @login_required
    def delete_goal(goal_id):
        goal = HealthGoal.query.get_or_404(goal_id)

        if goal.user_id != current_user.id:
            return jsonify({'success': False, 'message': 'Доступ заборонено'}), 403

        db.session.delete(goal)
        db.session.commit()

        return jsonify({'success': True})

    @dashboard.route('/api/goal/<int:goal_id>/progress', methods=['GET'])
    @login_required
    def goal_progress(goal_id):
        goal = HealthGoal.query.get_or_404(goal_id)

        if goal.user_id != current_user.id:
            return jsonify({'success': False, 'message': 'Доступ заборонено'}), 403

        latest_record = HealthRecord.query.filter_by(
            user_id=current_user.id,
            record_type=goal.goal_type
        ).order_by(HealthRecord.record_date.desc()).first()

        start_record = HealthRecord.query.filter(
            HealthRecord.user_id == current_user.id,
            HealthRecord.record_type == goal.goal_type,
```

```
    HealthRecord.record_date >= goal.start_date
).order_by(HealthRecord.record_date).first()

if not latest_record or not start_record:
    return jsonify({
        'success': False,
        'message': 'Недостатньо даних для відстеження прогресу'
    }), 400

progress_data = {}

if goal.goal_type == 'blood_pressure':
    start_diastolic = start_record.value
    current_diastolic = latest_record.value
    target_diastolic = goal.target_value

    if start_diastolic > target_diastolic:
        total_diff = start_diastolic - target_diastolic
        current_diff = start_diastolic - current_diastolic
        diastolic_progress = min(100, max(0, (current_diff / total_diff) * 100))
    else:
        total_diff = target_diastolic - start_diastolic
        current_diff = current_diastolic - start_diastolic
        diastolic_progress = min(100, max(0, (current_diff / total_diff) * 100))

    progress_data = {
        'start': start_diastolic,
        'current': current_diastolic,
        'target': target_diastolic,
        'progress': diastolic_progress,
        'overall': diastolic_progress
    }
else:
    start_value = start_record.value
    current_value = latest_record.value
    target_value = goal.target_value

    if start_value > target_value:
        total_diff = start_value - target_value
        current_diff = start_value - current_value
        progress = min(100, max(0, (current_diff / total_diff) * 100))
    else:
        total_diff = target_value - start_value
        current_diff = current_value - start_value
        progress = min(100, max(0, (current_diff / total_diff) * 100))

    progress_data = {
        'start': start_value,
        'current': current_value,
        'target': target_value,
        'progress': progress,
```

```
            'overall': progress
        }

    return jsonify({
        'success': True,
        'data': progress_data
    })

@dashboard.route('/api/record/<int:record_id>/update_notes', methods=['POST'])
@login_required
def update_record_notes(record_id):
    record = HealthRecord.query.get_or_404(record_id)

    if record.user_id != current_user.id:
        return jsonify({'success': False, 'message': 'Доступ заборонено'}), 403

    data = request.get_json()
    record.notes = data.get('notes', '')

    db.session.commit()

    return jsonify({'success': True})

@dashboard.route('/api/record/<int:record_id>/delete', methods=['POST'])
@login_required
def delete_record(record_id):
    record = HealthRecord.query.get_or_404(record_id)

    if record.user_id != current_user.id:
        return jsonify({'success': False, 'message': 'Доступ заборонено'}), 403

    db.session.delete(record)
    db.session.commit()

    return jsonify({'success': True})
```

```
from flask import Blueprint, render_template, redirect, url_for, flash, request
from flask_login import login_user, logout_user, login_required, current_user
from werkzeug.urls import url_parse
from extensions import db
from models import User
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, BooleanField, SubmitField
from wtforms.validators import DataRequired, Email, EqualTo, ValidationError

auth = Blueprint('auth', __name__)


class LoginForm(FlaskForm):
    username = StringField('Логін', validators=[DataRequired()])
    password = PasswordField('Пароль', validators=[DataRequired()])
```

```
    remember_me = BooleanField('Запам\'ятати мене')
    submit = SubmitField('Увійти')


class RegistrationForm(FlaskForm):
    username = StringField('Логін', validators=[DataRequired()])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Пароль', validators=[DataRequired()])
    password2 = PasswordField('Повторіть пароль', validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Зареєструватися')

    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user is not None:
            raise ValidationError('Цей логін вже використовується.')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user is not None:
            raise ValidationError('Цей email вже використовується.')


@auth.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('dashboard.index'))

    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()

        if user is None or not user.check_password(form.password.data):
            flash('Неправильний логін або пароль')
            return redirect(url_for('auth.login'))

        login_user(user, remember=form.remember_me.data)
        next_page = request.args.get('next')

        if not next_page or url_parse(next_page).netloc != '':
            next_page = url_for('dashboard.index')

        return redirect(next_page)

    return render_template('auth/login.html', title='Вхід', form=form)


@auth.route('/logout')
def logout():
    logout_user()
    return redirect(url_for('home'))
```

```
@auth.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('dashboard.index'))

    form = RegistrationForm()
    if form.validate_on_submit():
        user = User(username=form.username.data, email=form.email.data)
        user.set_password(form.password.data)
        db.session.add(user)
        db.session.commit()

        flash('Ви успішно зареєструвалися!')
        return redirect(url_for('auth.login'))

    return render_template('auth/register.html', title='Реєстрація', form=form)
```

Додаток Г. Ілюстративна частина

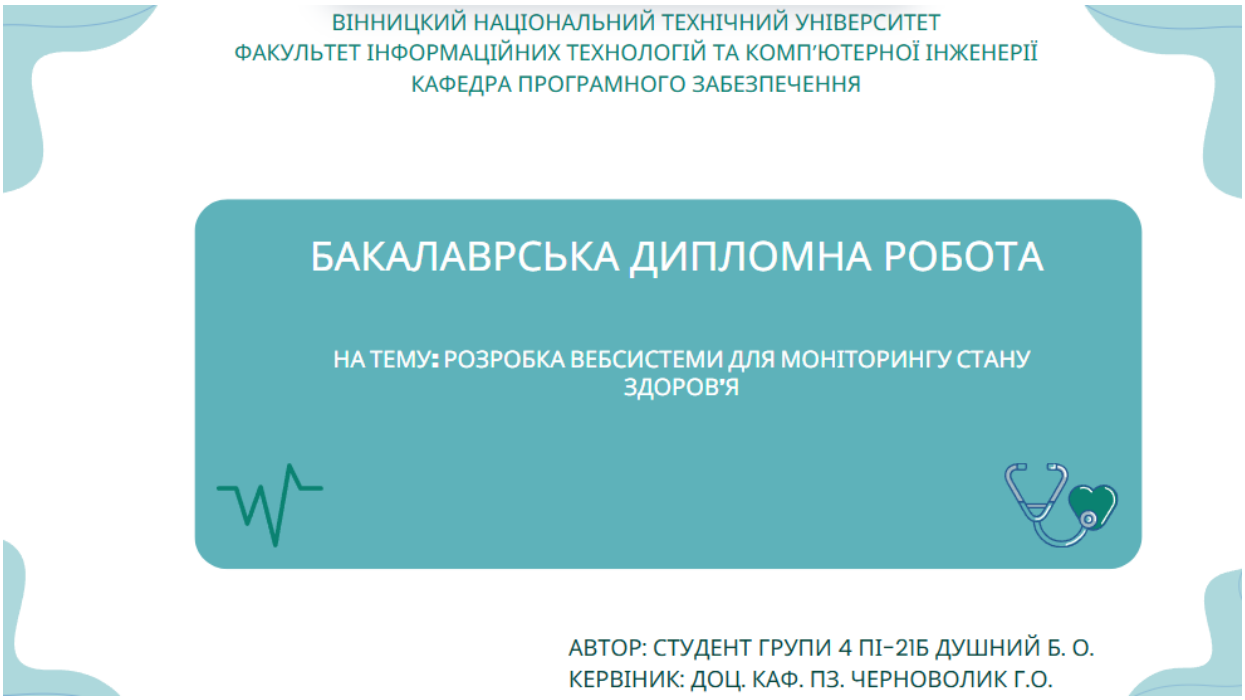

Рисунок Г.1 – Титульний слайд

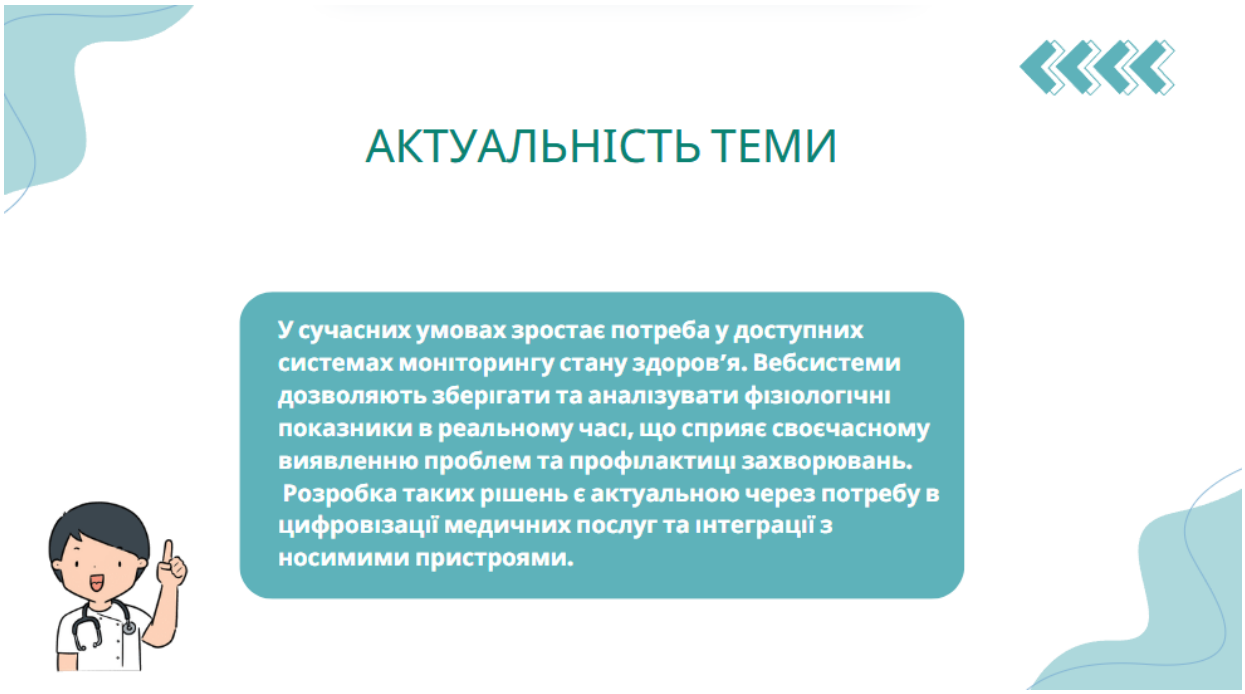

Рисунок Г.2 – Актуальність розробки

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

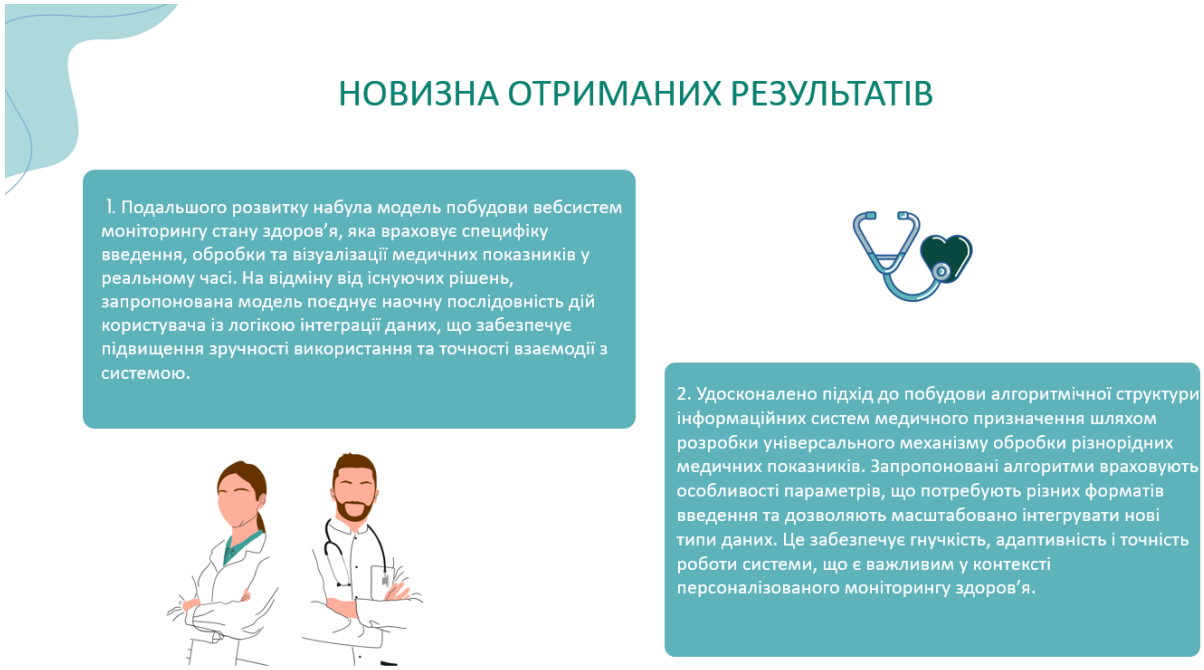

Рисунок Г.4 – Новизна отриманих результатів

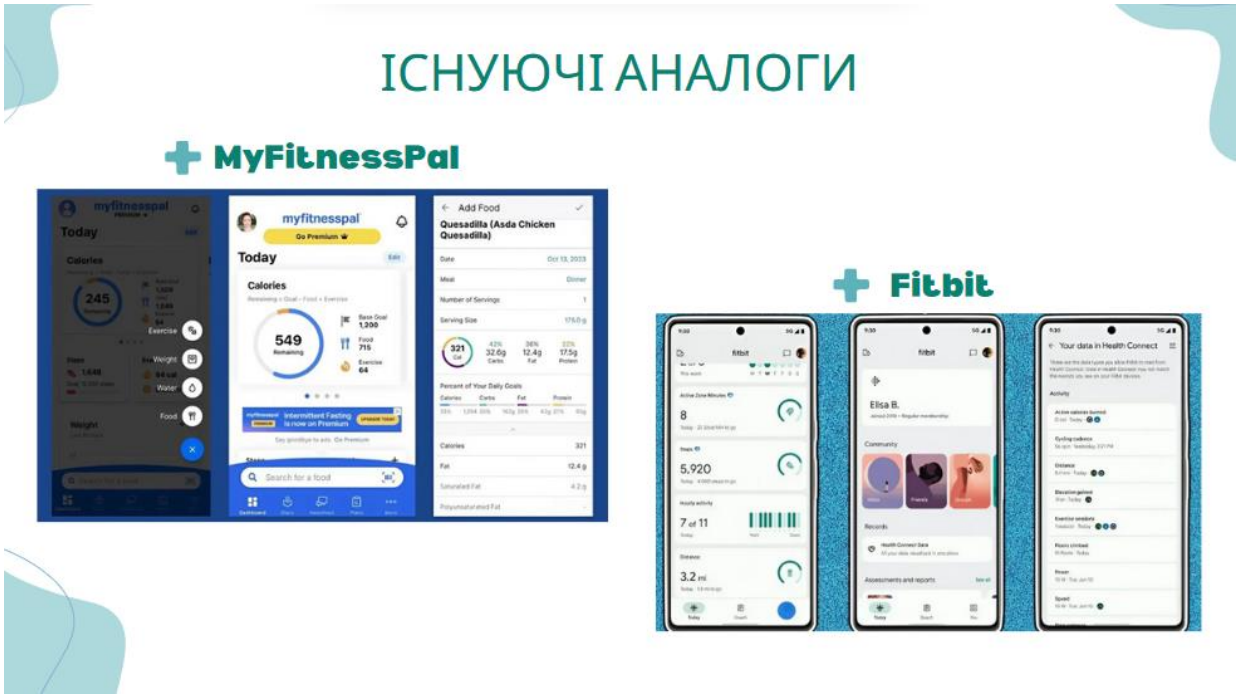

Рисунок Г.5 – Існуючі аналоги

Рисунок Г.6 – Існуючі аналоги

Рисунок Г.7 – Алгоритм дій користувача

Рисунок Г.8 – Алгоритм додавання запису

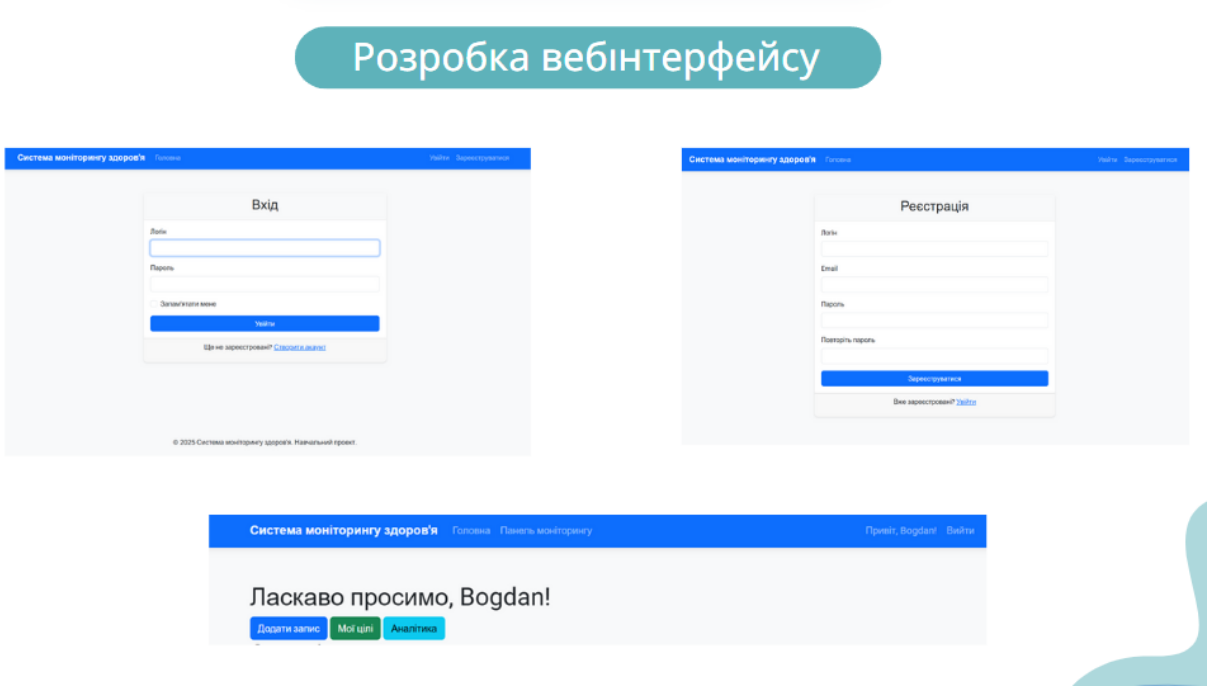

Рисунок Г.9 – Розробка вебінтерфейсу

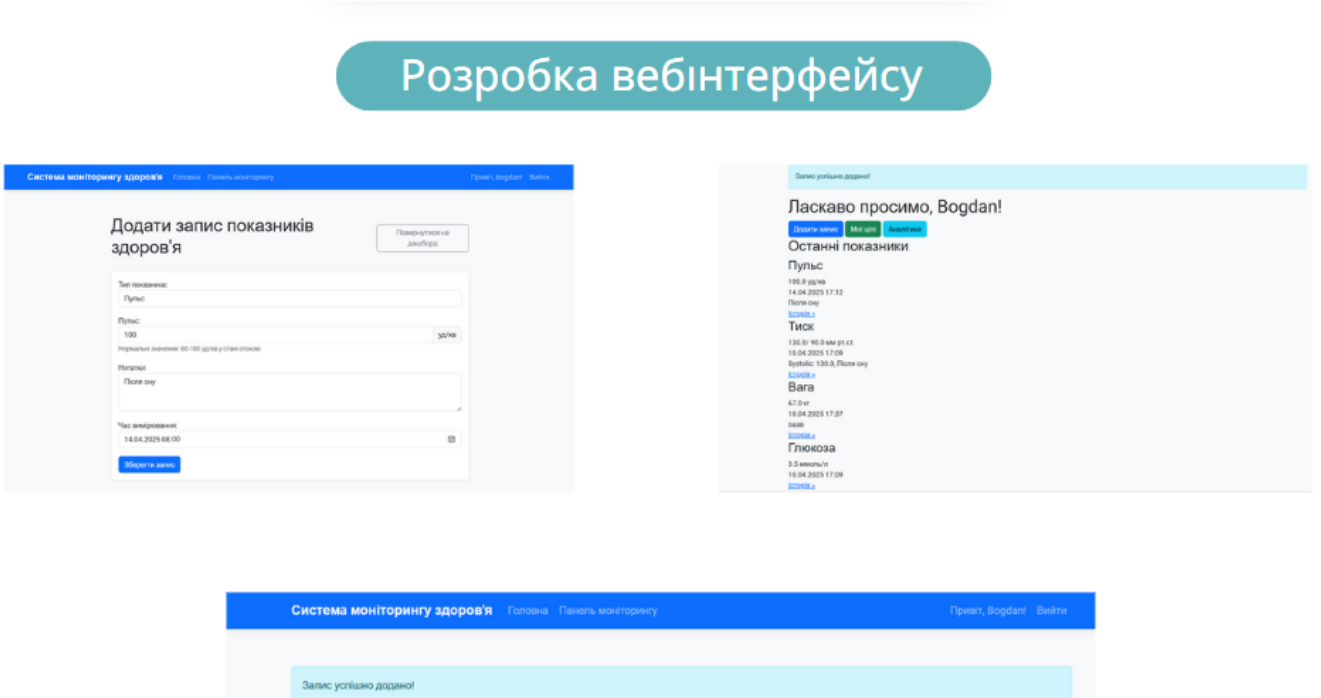

Рисунок Г.10 –Розробка вебінтерфейсу

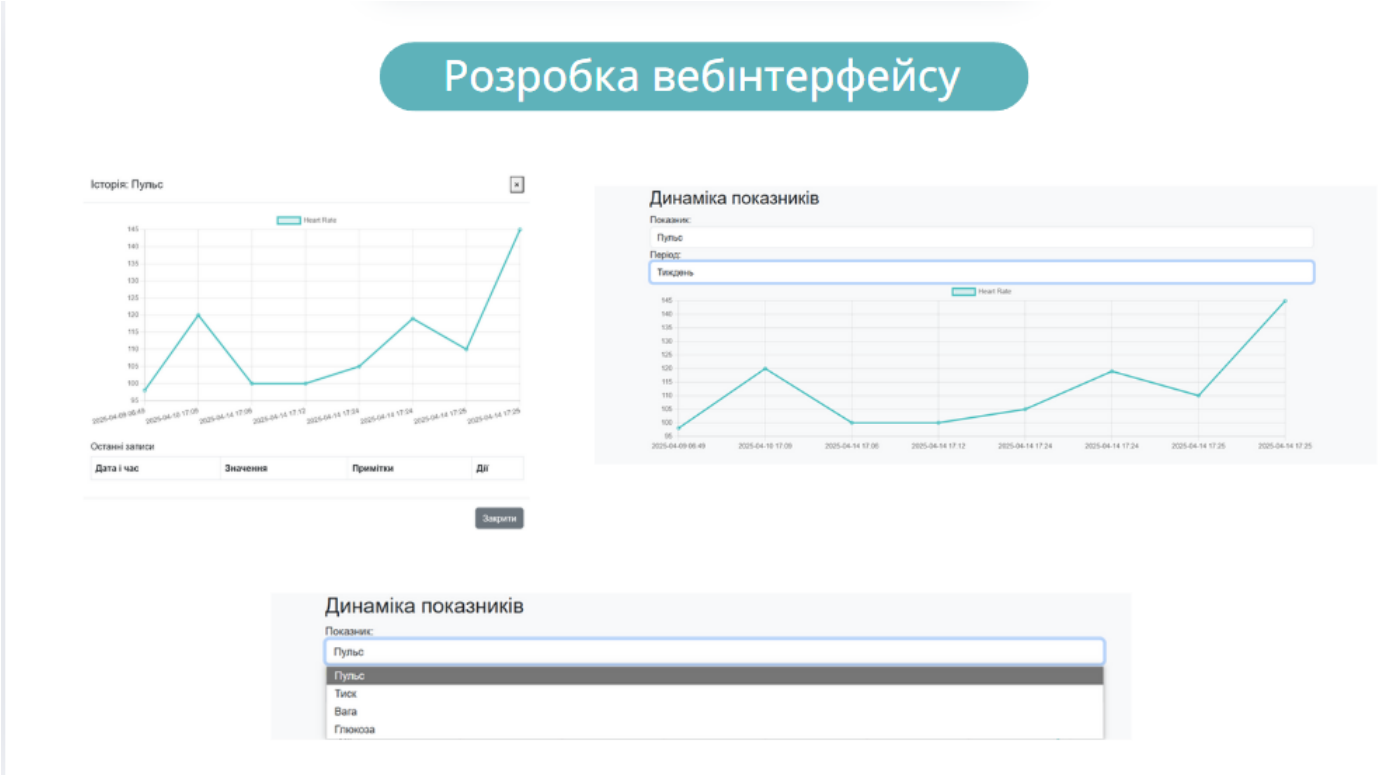

Рисунок Г.11 – Розробка вебінтерфейсу

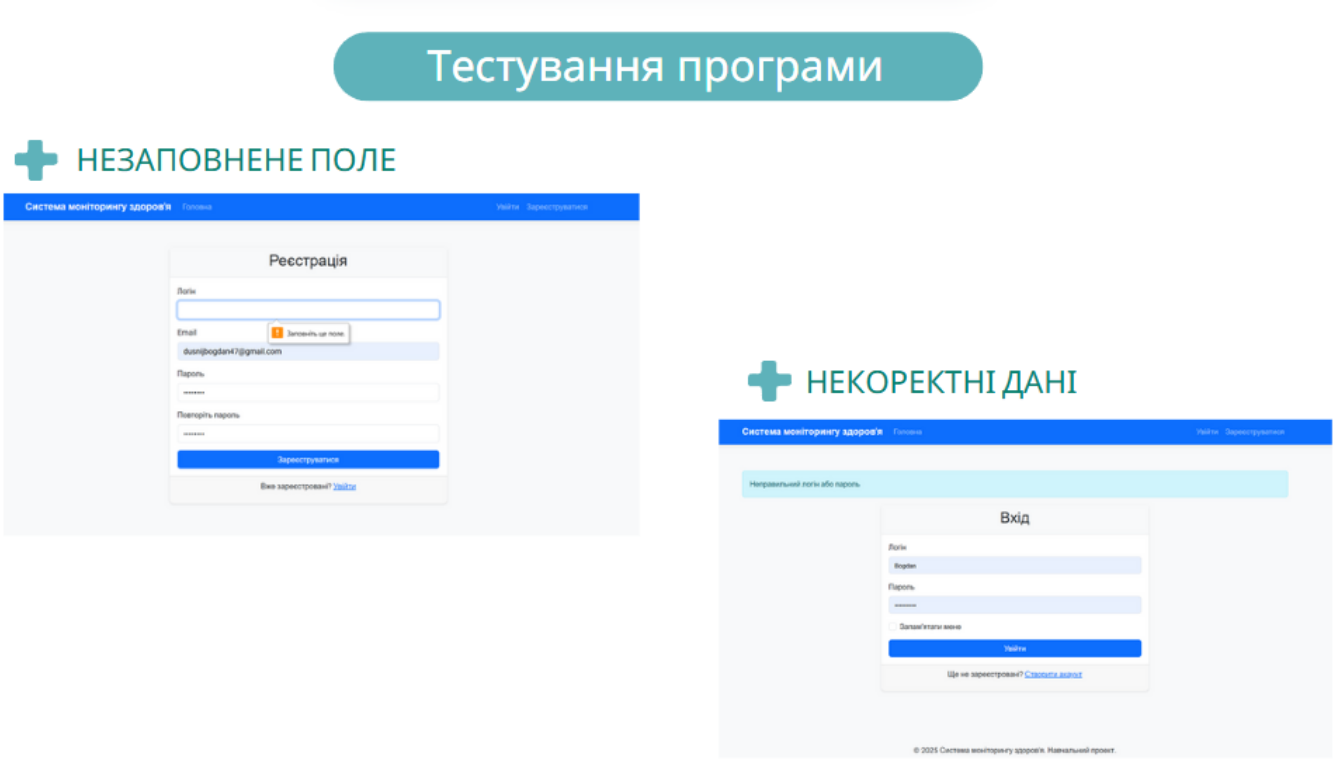

Рисунок Г.12 – Тестування програми

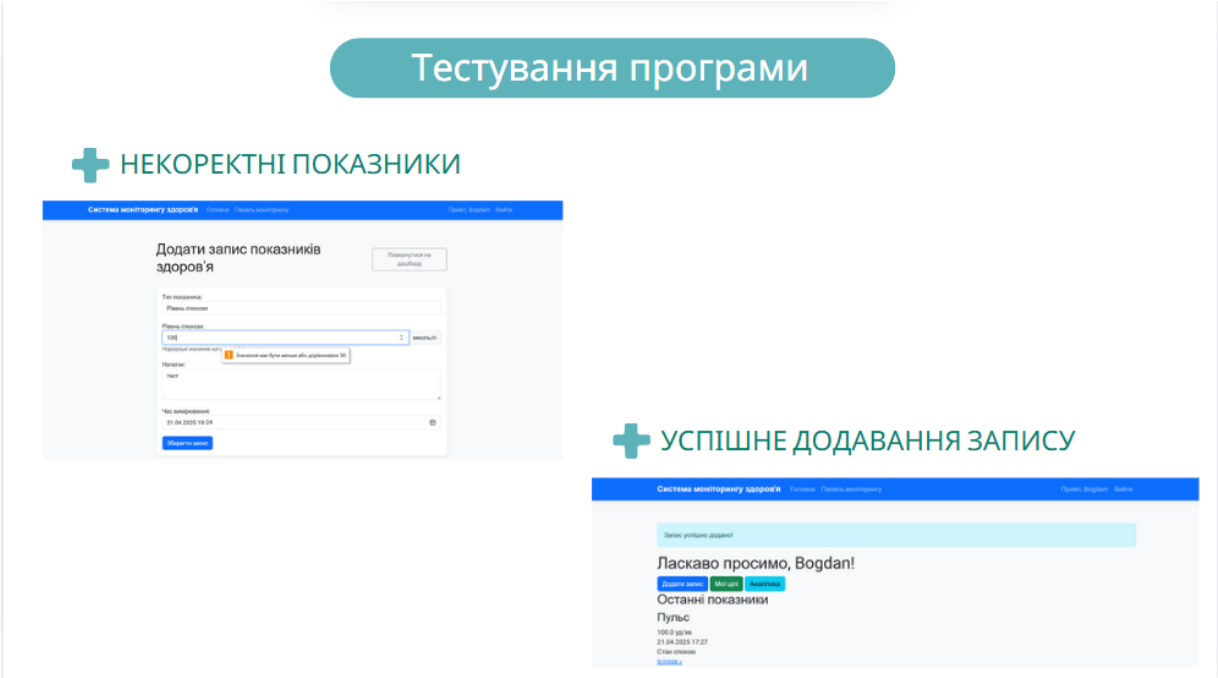

Рисунок Г.13 – Схеми інтерфейсу

ВИСНОВКИ

У результаті виконання бакалаврської дипломної роботи було розроблено вебсистему для моніторингу стану здоров'я користувачів. Система включає такі основні функціональні модулі:

- модуль реєстрації та авторизації користувачів, що забезпечує безпечний доступ до персонального кабінету;
- модуль панелі керування, через який користувач може керувати своїми даними та переглядати історію внесених показників;
- модуль додавання записів показників здоров'я, який дозволяє фіксувати, зберігати й переглядати динаміку фізіологічних параметрів.

Для реалізації системи було спроєктовано архітектуру, розроблено логіку взаємодії між модулями, а також реалізовано алгоритми обробки та збереження даних. У якості мови програмування використано **Python** у поєднанні з фреймворком **Flask**. Розробка здійснювалася в середовищі **PyCharm**.

У першому розділі роботи проведено огляд сучасних рішень у сфері цифрового моніторингу здоров'я, визначено їхні переваги та недоліки, що дало змогу сформувати вимоги до власної системи. У другому розділі описано методику побудови вебзастосунку, зокрема модель роботи системи та принципи обробки користувацьких даних. Третій розділ присвячено технічній реалізації: обґрунтовано вибір мови програмування, фреймворку та середовища розробки, створено окремі модулі й інтерфейс користувача. У четвертому розділі проведено тестування функціональних можливостей вебсистеми, що підтвердило її коректну роботу відповідно до заданих функціональних вимог.

Система є зручною у використанні, придатною для подальшого масштабування, зокрема для інтеграції з пристроями для автоматичного зчитування показників або з медичними системами для віддаленого моніторингу.

Рисунок Г.14 – Висновки

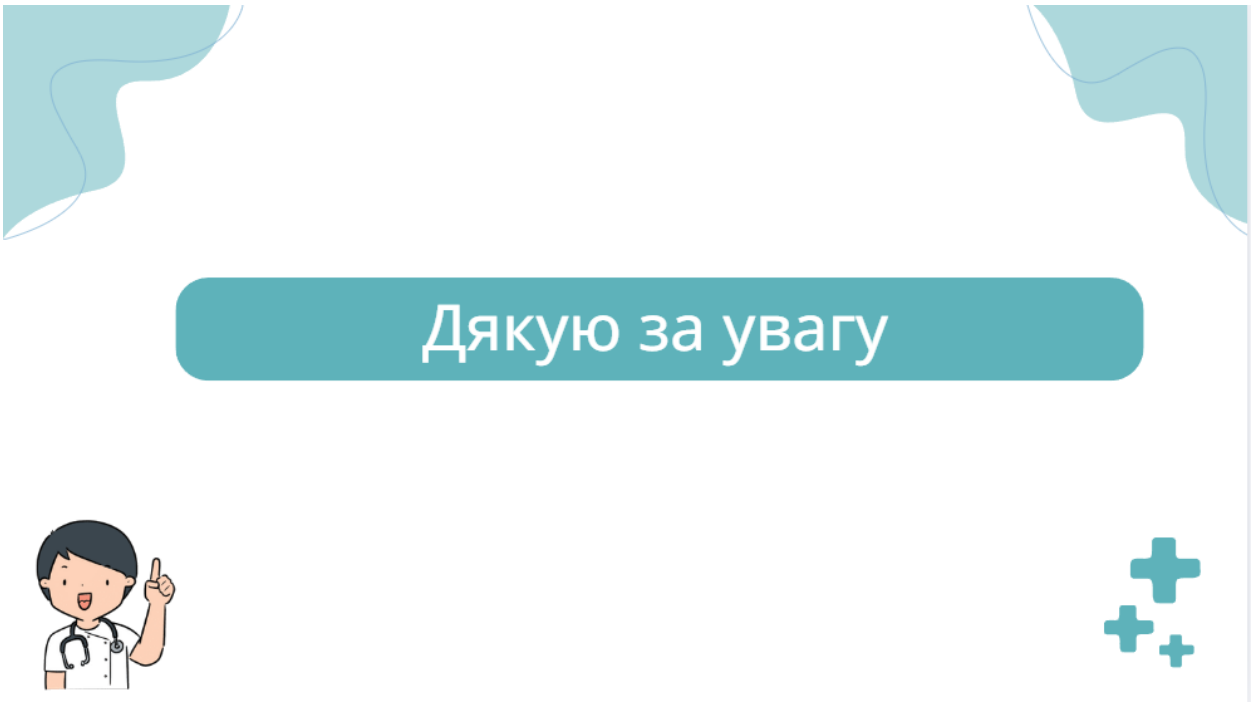

Рисунок Г.15 – Фінальний слайд