**Software Requirements**

**Specification**

**for**

Software system for medical monitoring and patient care

Version 1.0 approved

Prepared by Serhieienkova Mariia

NURE

20.05.2025

Історія версій

|  |  |  |  |
| --- | --- | --- | --- |
| **Дата** | **Опис** | **Автор** | **Коментар** |
| 20.05.2025 | Версія 1.0 | Сергєєнкова Марія | Створення документу та заповнення усіх позділів |

**Зміст**

[**1 ВСТУП** 5](#_Toc201271131)

[1.1 Огляд продукту 5](#_Toc201271132)

[1.2 Мета 5](#_Toc201271133)

[1.3 Межі 6](#_Toc201271134)

[1.4 Посилання 7](#_Toc201271135)

[1.5 Означення та абревіатури 8](#_Toc201271136)

[**2 ЗАГАЛЬНИЙ ОПИС** 9](#_Toc201271137)

[2.1 Перспективи продукту 9](#_Toc201271138)

[2.2 Функції продукту 10](#_Toc201271139)

[2.3 Характеристики користувачів 11](#_Toc201271140)

[2.4 Загальні обмеження 13](#_Toc201271141)

[2.5 Припущення й залежності 14](#_Toc201271142)

[**3** **КОНКРЕТНІ ВИМОГИ** 16](#_Toc201271143)

[3.1 Вимоги до зовнішніх інтерфейсів 16](#_Toc201271144)

[3.1.1 Інтерфейс користувача 16](#_Toc201271145)

[3.1.2 Апаратний інтерфейс 17](#_Toc201271146)

[3.1.3 Програмний інтерфейс 18](#_Toc201271147)

[3.1.4 Комунікаційний прототип 19](#_Toc201271148)

[3.1.5 Обмеження пам'яті 20](#_Toc201271149)

[3.1.6 Операції 20](#_Toc201271150)

[3.1.7 Функції продукту 22](#_Toc201271151)

[3.1.8 Припущення й залежності 23](#_Toc201271152)

[3.2. Властивості програмного продукту 25](#_Toc201271153)

[3.3. Атрибути програмного коду 26](#_Toc201271154)

[3.4 Вимоги бази даних 27](#_Toc201271155)

[3.5 Інші вимоги 27](#_Toc201271156)

[**4 ДОДАТКОВІ МАТЕРІАЛИ** 29](#_Toc201271157)

# **1 ВСТУП**

## Огляд продукту

В останні роки в українській медичній галузі зростає попит на автоматизовані програмні рішення, які дозволяють ефективно організувати процес моніторингу стану здоров'я пацієнтів, особливо в умовах обмеженого ресурсу персоналу. Потреба в дистанційному нагляді та своєчасному реагуванні на зміни в життєвих показниках пацієнтів постійно зростає.

Сучасні рішення базуються на використанні IoT-пристроїв для збору медичних даних і серверних систем для їх обробки. Саме серверна частина відіграє ключову роль у збереженні, аналізі даних, формуванні звітів, планів лікування та надсиланні сповіщень медперсоналу чи родичам.

Розробка серверної частини для такої системи є важливим кроком у створенні повноцінної платформи для моніторингу здоров’я. Вона відповідає за централізовану обробку та збереження даних, взаємодію з IoT-пристроями, генерацію сповіщень, формування планів лікування та реалізацію механізмів доступу.

Серверна частина даної програмної системи має забезпечувати:

* прийом, обробку та зберігання даних від IoT-пристроїв;
* аналіз отриманих показників та формування рекомендацій;
* організацію сповіщень медичним працівникам та родичам;
* керування планами лікування, електронними щоденниками та звітами;
* реалізацію багаторівневого доступу до даних користувачів.

Технологічно сервер реалізовано мовою Go із застосуванням фреймворку Gin. Для збереження даних використовується PostgreSQL. Обмін повідомленнями здійснюється з використанням відповідних засобів, зокрема SMTP-серверів для email-сповіщень.

## 1.2 Мета

Метою розробки програмної системи є створення серверної частини для автоматизованого медичного моніторингу та підтримки догляду за пацієнтами. Система має забезпечувати збирання, обробку, зберігання та аналіз життєво важливих показників здоров’я (зокрема, пульсу, температури тіла та артеріального тиску), що надходять від IoT-пристроїв, а також надання сервісів, які полегшують взаємодію між лікарем, пацієнтом та родичами пацієнта.

Основні цілі створення системи:

* забезпечити надходження даних з IoT-пристроїв;
* автоматизувати збереження медичних показників пацієнтів у базі даних;
* реалізувати механізми виявлення критичних відхилень та надсилання сповіщень відповідним особам (медичному персоналу, родичам);
* створити інструменти для формування планів лікування, призначення медикаментів і планування візитів;
* надати можливість лікарям та пацієнтам взаємодіяти з персональними медичними даними через API;
* реалізувати систему доступу з урахуванням ролей користувачів (пацієнт, лікар, родич);
* забезпечити ведення електронного щоденника стану здоров’я пацієнта;
* реалізувати можливість надсилання автоматичних email-сповіщень;
* зберігати історію медичних даних і формувати звіти для подальшого аналізу.

Досягнення цих цілей дозволить створити ефективну серверну інфраструктуру, яка може бути застосована як у медичних закладах, так і для персонального використання, наприклад, у реабілітаційних центрах, будинках для літніх людей або вдома.

## 1.3 Межі

У межах даної програмної системи реалізується лише серверна частина для медичного моніторингу та догляду за пацієнтами.

До меж реалізації системи входить:

* прийом емуляційних даних від IoT-пристроїв щодо життєвих показників пацієнтів (температура, пульс, артеріальний тиск);
* обробка цих даних із метою виявлення відхилень;
* фіксація інформації в базі даних;
* надсилання повідомлень лікарям, пацієнтам або родичам (через email) у випадках перевищення норм;
* ведення журналу стану пацієнтів і щоденника здоров’я;
* формування планів лікування, призначення медикаментів і запис візитів до лікаря;
* контроль доступу до функціоналу системи з урахуванням ролей користувачів (пацієнт, лікар, родич);
* реалізація RESTful API для взаємодії клієнтів з бекендом.

Поза межами реалізації залишаються:

* створення графічного користувацького інтерфейсу (GUI) або мобільного додатку;
* інтеграція з державними або приватними електронними медичними системами;
* підтримка мультимовності;
* функціонал резервного копіювання чи масштабування інфраструктури;
* обробка даних з інших сенсорів (наприклад, рівень кисню, ЕКГ).

Таким чином, розроблений у рамках проєкту програмний продукт зосереджується на створенні стабільної, розширюваної та безпечної серверної частини з повноцінним API, яка може бути використана як основа для майбутньої розробки клієнтських інтерфейсів та інтеграції з іншими системами.

## 1.4 Посилання

У даній специфікації програмного забезпечення використовуються такі джерела та нормативні документи:

* IEEE Std 830-1993 – IEEE Recommended Practice for Software Requirements Specifications;
* Gin Web Framework Documentation – офіційна документація до веб-фреймворку Gin для мови Go. Доступ: <https://gin-gonic.com/en/docs/>;
* Go Programming Language Specification – офіційна специфікація мови програмування Go. Доступ: <https://go.dev/ref/spec>;
* PostgreSQL Documentation – офіційна документація до системи керування базами даних PostgreSQL. Доступ: <https://www.postgresql.org/docs/>;
* MQTT Specification – офіційна специфікація протоколу MQTT для IoT-зв’язку. Доступ: <https://mqtt.org/mqtt-specification/>;
* Postman documentation overview – оглядова документація інструменту Postman для тестування API. Доступ: <https://learning.postman.com/docs/introduction/overview/>;

## 1.5 Означення та абревіатури

У цьому документі використовуються наступні терміни та абревіатури:

* API – Application Programming Interface;
* CORS – Cross-Origin Resource Sharing;
* CRUD – Create, Read, Update, Delete;
* IoT – Internet of Things;
* JSON – JavaScript Object Notation;
* JWT – JSON Web Token;
* REST – Representational State Transfer;
* SMTP – Simple Mail Transfer Protocol;
* SQL – Structured Query Language;
* UML – Unified Modeling Language.

# **2 ЗАГАЛЬНИЙ ОПИС**

## 2.1 Перспективи продукту

Система медичного моніторингу та догляду за пацієнтами є відповіддю на актуальні виклики сучасної медичної сфери, яка активно впроваджує цифрові технології з метою підвищення якості обслуговування пацієнтів. Особливої ваги набувають системи, здатні забезпечити постійний нагляд за фізіологічними показниками хворих — як у медичних закладах, так і в умовах домашнього догляду.

Запропонована система орієнтована на створення інфраструктури, що дозволяє фіксувати показники здоров’я (пульс, артеріальний тиск, температура тіла) за допомогою IoT-пристроїв, обробляти ці дані на сервері, зберігати їх у базі даних та надсилати сповіщення у випадках відхилень. Завдяки цьому можна підвищити ефективність роботи медичного персоналу, забезпечити вчасне реагування на погіршення стану пацієнта, а також підтримати комунікацію між лікарем, пацієнтом та родичами.

З технічної точки зору, система побудована як серверний програмний комплекс, що реалізує RESTful API. Це дозволяє легко підключати до неї клієнтські застосунки, мобільні пристрої чи веб-інтерфейси. Таким чином, у майбутньому можливе розширення функціональності проєкту через реалізацію фронтенд-застосунків для лікарів і пацієнтів, створення мобільного застосунку, а також інтеграція з реальними IoT-пристроями.

Система має модульну архітектуру та використовує сучасні технології, що дозволяє досягти високої швидкодії, масштабованості та надійності. Її впровадження може бути доцільним у лікарнях, будинках для літніх людей, приватних клініках або у рамках особистого моніторингу стану здоров’я.

У перспективі, проєкт може бути доповнений аналітичними модулями для прогнозування стану здоров’я, машинним навчанням для автоматичного виявлення критичних ситуацій, а також механізмами інтеграції з національними або приватними медичними платформами.

## 2.2 Функції продукту

Програмна система для медичного моніторингу та догляду за пацієнтами виконує низку функцій, які забезпечують повноцінне відстеження фізіологічного стану користувачів, своєчасне реагування на зміни у показниках здоров’я, взаємодію між пацієнтами, лікарями та родичами, а також збереження медичної інформації в електронному вигляді. Усі функції реалізовано у межах серверної частини програмного забезпечення.

Основні функціональні можливості системи:

1. облік користувачів та ролей доступу:
   1. реєстрація пацієнтів, лікарів, родичів та адміністраторів;
   2. керування правами доступу відповідно до ролі користувача;
   3. автентифікація та авторизація через JWT-токени.
2. робота з IoT-даними:
   1. прийом вимірювань від пристроїв, що фіксують пульс, температуру тіла та артеріальний тиск;
   2. збереження отриманих медичних показників у базі даних;
   3. перевірка даних на предмет відхилень від норми;
   4. формування сповіщень у разі критичних значень.
3. обробка медичних даних:
   1. аналіз тенденцій у змінах показників пацієнта;
   2. генерація звітів для лікарів або пацієнтів;
4. планування лікування:
   1. створення індивідуального плану лікування лікарем;
   2. призначення медичних препаратів із вказаним дозуванням, частотою прийому та періодом лікування;
   3. додавання запланованих візитів до лікаря.
5. система нагадувань:
   1. автоматичне надсилання нагадувань про прийом ліків та заплановані медичні події;
   2. надсилання email-повідомлень лікарям або родичам про критичні стани пацієнта.
6. електронний щоденник пацієнта:
   1. збереження самостійно введених записів про самопочуття пацієнта;
   2. надання доступу лікарям та родичам (за дозволом пацієнта) до цих записів.
7. історія та звітність:
   1. перегляд історії медичних показників;
   2. генерація звітів для лікарів;
   3. ведення журналу подій системи (наприклад, надсилання сповіщень, внесення змін до плану лікування).
8. керування взаємодією з родичами:
   1. можливість пацієнта надавати або обмежувати доступ до своїх даних;
   2. надсилання родичам сповіщень про важливі події;
   3. доступ родичів до щоденника здоров’я та показників пацієнта (якщо надано дозвіл).

## 2.3 Характеристики користувачів

Програмна система для медичного моніторингу та догляду за пацієнтами розробляється з урахуванням взаємодії кількох основних категорій користувачів. Кожен користувач (пацієнт, лікар, родич, адміністратор) взаємодіє з системою в межах своєї ролі, відповідно до рівня доступу.

Основні типи користувачів:

1. лікар (медичний персонал):

це головна категорія користувачів, для якої призначена більшість функцій системи. Користувачі цього типу мають розширений набір можливостей, оскільки відповідають за контроль стану здоров’я пацієнтів, формування планів лікування, аналітику та реагування на критичні ситуації.

Функціональні можливості лікаря:

* перегляд та аналіз даних пацієнтів, закріплених за ними;
* призначення ліків та формування лікувального плану;
* закріплення родичів за конкретним пацієнтом (з можливістю надання або обмеження доступу родичам до особистих медичних даних пацієнта за його проханням);
* взаємодія з діагнозами, ліками, рецептами;
* коригування дозування, частоти прийому або періоду лікування;
* моніторинг динаміки показників здоров’я;
* отримання сповіщень про критичні зміни у стані пацієнта;
* перегляд візитів в цілому або на поточному тижні.

1. пацієнт:

функціональні можливості пацієнта:

* перегляд інформації про закріплених за ним лікарів та родичів;
* доступ до перегляду плану лікування з попередньо прописаними рецептами та назначень діагнозів лікарем;
* ведення електронного щоденника самопочуття;
* отримання сповіщень про необхідність прийому ліків або запланований візит.

1. родич / доглядальник:

дана категорія користувачів має обмежений доступ до медичної інформації пацієнта, відповідно до наданих дозволів. Метою цієї ролі є контроль стану здоров’я близьких осіб та своєчасне реагування на сповіщення про відхилення у показниках.

Функціональні можливості родича:

* перегляд основних показників здоров’я пацієнта (якщо надано доступ);
* доступ до записів у щоденнику самопочуття;
* отримання email-сповіщень у разі критичних змін стану здоров’я;

Родич не має змоги редагувати жодні дані, лише переглядати дозволену інформацію.

1. Адміністратор системи

у рамках системи адміністратор виконує роль технічного користувача, який має контроль над девайсами та закріплює за пацієнтами.

Кожна роль чітко відмежована за правами доступу та взаємодіють із системою через серверний API. Усі дії користувачів відбуваються після успішної автентифікації, а перевірка прав доступу виконується на серверному рівні. Це забезпечує як безпечну обробку персональних медичних даних, так і надійну взаємодію між різними учасниками процесу догляду за пацієнтом.

## 2.4 Загальні обмеження

У ході розробки та впровадження серверної частини програмної системи для медичного моніторингу були враховані ключові технічні, логічні та організаційні обмеження, які впливають на функціональність і сферу застосування продукту.

1. орієнтація виключно на серверну частину:

у межах розробки реалізовано лише серверну частину програмної системи. Клієнтські застосунки (мобільні або веб-інтерфейси) не входять до сфери реалізації. Передбачається, що взаємодія з кінцевими користувачами відбуватиметься через RESTful API, який був розроблений.

1. відсутність функцій діагностики:

система не призначена для автоматичної постановки діагнозів або прийняття медичних рішень без участі лікаря. Усі рекомендації та аналітика, що генеруються системою, мають лише допоміжний характер. Остаточне рішення щодо дій приймає виключно уповноважений медичний працівник.

1. простота емуляції IoT-пристроїв:

у поточній реалізації пристрої, що збирають біологічні показники (температура, пульс, тиск), імітуються, а не фізично використовуються. Це спрощує тестування, але обмежує повноцінне оцінювання продуктивності у реальних умовах. У разі інтеграції з реальними пристроями можуть виникати затримки або інші труднощі, що не були враховані на етапі проєктування.

Ці обмеження слід враховувати під час впровадження та експлуатації системи, а також при плануванні її майбутнього розширення та інтеграції з іншими сервісами або медичними пристроями.

## 2.5 Припущення й залежності

У процесі проєктування та реалізації серверної частини програмної системи для медичного моніторингу та догляду за пацієнтами було сформовано низку припущень і технічних залежностей, дотримання яких є необхідним для забезпечення стабільної, коректної та передбачуваної роботи системи.

1. інфраструктурні припущення:
2. стабільне інтернет-з'єднання. Усі користувачі системи, а також IoT-пристрої повинні мати постійний доступ до мережі Інтернет. Втрата з’єднання призводить до тимчасової недоступності окремих функцій (наприклад, передача показників, отримання сповіщень або нагадувань);
3. наявність серверного середовища, здатного розгорнути застосунок, реалізований на мові програмування Go з використанням фреймворку Gin, з підключенням до СКБД PostgreSQL;
4. припущення щодо користувачів системи:
5. різні рівні доступу. Кожен користувач системи (пацієнт, лікар, родич) має відповідну роль, яка регулює обсяг доступної інформації та функцій. Припускається, що авторизація відбувається за заздалегідь створеними обліковими записами, або через попередню реєстрацію;
6. дотримання правил безпеки. Користувачі не повинні передавати свої облікові дані іншим особам і мають використовувати лише надані їм методи автентифікації (email та пароль);
7. залежність від зовнішніх сервісів та технологій:
8. підключення до бази даних PostgreSQL. Уся інформація про користувачів, біопоказники, плани лікування, звіти та історію медичних подій зберігається в реляційній базі даних, з якою взаємодіє сервер через SQL-запити;
9. використання SMTP-сервера для надсилання email-повідомлень користувачам (нагадування, сповіщення про критичні стани тощо);
10. моделі даних у форматі JSON. Усі вхідні та вихідні повідомлення в API реалізовані на основі формату JSON, що вимагає від клієнтів підтримки відповідної серіалізації/десеріалізації.
11. припущення щодо даних і пристроїв:
12. дані, що надходять із пристроїв, є достовірними та відповідають дійсному фізіологічному стану пацієнта. Система не перевіряє правдивість цих даних і не здійснює калібрування;
13. IoT-пристрої налаштовані коректно, підключені до брокера повідомлень (MQTT), а формат їхніх повідомлень відповідає заздалегідь узгодженому шаблону;
14. емуляція пристроїв використовується на етапі тестування. У реальній експлуатації потрібна адаптація до апаратного забезпечення.

# **КОНКРЕТНІ ВИМОГИ**

## 3.1 Вимоги до зовнішніх інтерфейсів

### 3.1.1 Інтерфейс користувача

Оскільки в межах даної програмної системи реалізовано лише серверну частину, інтерфейс користувача безпосередньо не розроблявся. Водночас сервер надає структуроване RESTful API, яке призначене для подальшої інтеграції з фронтенд-клієнтами (веб або мобільними застосунками).

Інтерфейс користувача системи повинен базуватися на наступних принципах, які враховуються під час проєктування API:

* модульність і ролі користувачів. Уся функціональність API розділена відповідно до ролей (пацієнт, лікар, родич, адміністратор), що дозволяє створювати адаптовані інтерфейси для кожного типу користувача;
* авторизація та автентифікація. Для доступу до функцій інтерфейсу передбачена перевірка JWT-токенів, які генеруються під час логіну. Усі запити мають бути автентифіковані;
* інтерфейс очікуваних запитів. Сервер підтримує стандартні HTTP-методи для обробки даних: GET, POST, PUT, DELETE;
* структура відповідей. Усі відповіді API реалізовані у форматі JSON. У випадку успішного запиту повертаються відповідні об’єкти або масиви даних. У разі помилки — стандартизовані повідомлення про причину відмови;
* повідомлення про помилки. Сервер повертає HTTP-коди статусів, що відповідають контексту (200, 201, 400, 401, 403, 404, 500), а також розширену інформацію у форматі JSON.

У наступних етапах розробки система передбачає можливість реалізації UI, який буде:

* для лікаря — відображати призначення, графіки, звіти, сповіщення;
* для пацієнта — показувати історію біопоказників, стан здоров’я, план лікування;
* для родича — надавати обмежений перегляд критичних змін стану пацієнта;
* для адміністратора — дозволяти керування користувачами та ролями.

### 3.1.2 Апаратний інтерфейс

В даній програмній системі апаратна частина (IoT-пристрої) не реалізується. Проте система моделює взаємодію з такими пристроями, і серверна частина містить необхідні механізми для обробки даних, що емулюються як вхідні показники IoT-пристроїв (температура, пульс, тиск).

Згідно з цим, сервер реалізує інтерфейс обробки апаратних даних, що надходять у вигляді HTTP-запитів через REST API. Кожен пристрій має бути зареєстрований у системі і пов’язаний із конкретним пацієнтом. Для передачі даних до серверу пристрої або емулятори повинні надсилати наступні параметри:

* device\_id — унікальний ідентифікатор пристрою;
* timestamp — час вимірювання в unix-форматі;
* pulse — пульс (уд./хв.);
* systolic\_blood\_pressure — систолічний тиск;
* diastolic\_blood\_pressure — діастолічний тиск;
* temperature — температура тіла в градусах Цельсія.

Передача цих даних здійснюється через захищений HTTP POST-запит на визначену кінцеву точку API. Сервер обробляє отриману інформацію, зберігає її до бази даних та виконує додаткову аналітику (виявлення відхилень і створення сповіщень).

Ключові вимоги до взаємодії з апаратними пристроями:

* ідентифікація пристрою — під час кожного запиту пристрій має бути автентифікований (наприклад, за токеном або зареєстрованим device\_id);
* безпека переданих даних — усі запити мають надсилатися через HTTPS;
* стабільність з’єднання — передбачається, що пристрої або емулятори мають стабільний доступ до мережі;
* стандартизовані запити — для забезпечення уніфікованості всі пристрої повинні використовувати узгоджену структуру запиту з JSON-полями.

Таким чином, навіть без фізичної реалізації апаратного інтерфейсу, система повністю готова до роботи з реальними IoT-пристроями в майбутньому.

### 3.1.3 Програмний інтерфейс

Програмний інтерфейс реалізовано у вигляді RESTful API, який забезпечує взаємодію клієнтських застосунків (наприклад, веб або мобільного інтерфейсу, IoT-пристроїв) із серверною частиною системи.

Програмний інтерфейс розроблено за допомогою мови Go та фреймворку Gin. API обробляє всі основні запити до серверу, включаючи створення, отримання, оновлення та видалення даних, що стосуються користувачів, показників здоров’я, планів лікування, сповіщень тощо.

Основні характеристики програмного інтерфейсу:

* формат обміну даними: JSON;
* методи запитів: GET, POST, PUT, DELETE;
* авторизація: реалізовано через JWT (JSON Web Token);
* маршрути: згруповані за ролями користувачів — пацієнт, лікар, родич, адміністратор.

Особливості реалізації:

* всі ендпоінти перевіряють авторизацію користувача на основі JWT-токену, що передається в заголовках запиту;
* обмеження доступу реалізовано за допомогою middleware, який ідентифікує роль користувача і дозволяє або блокує доступ до ресурсу;
* обробка помилок здійснюється централізовано — при виникненні виняткових ситуацій API повертає структуровану відповідь з кодом помилки і текстом повідомлення;
* додаткові перевірки на коректність даних виконуються за допомогою тегів binding і валідаційних функцій Gin.

Безпека:

* всі запити повинні виконуватись лише через HTTPS-з’єднання;
* паролі користувачів зберігаються в базі в захешованому вигляді;
* кожен користувач отримує унікальний JWT-токен, що містить зашифровану інформацію про його роль та ідентифікатор.

Таким чином, програмний інтерфейс є центральним компонентом взаємодії всіх зовнішніх клієнтів із системою та забезпечує надійну, безпечну й масштабовану структуру для обробки запитів у медичному середовищі.

### 3.1.4 Комунікаційний прототип

Комунікаційна модель у системі побудована за класичною архітектурною схемою «клієнт-сервер», де всі клієнти (веб-додаток, мобільний застосунок або IoT-пристрої — навіть у вигляді емуляторів) взаємодіють із центральною серверною частиною через програмний інтерфейс REST API.

Основні принципи організації комунікацій:

* протокол передавання даних: HTTP/HTTPS;
* формат даних: JSON — універсальний, легкий формат обміну, підтримуваний усіма сучасними платформами;
* модель взаємодії: синхронна — кожен запит клієнта очікує відповідь від сервера;
* безпечне з’єднання: всі запити повинні надсилатися по протоколу HTTPS для захисту конфіденційної інформації (наприклад, даних про стан здоров’я).

Взаємодія системи з клієнтами:

* ініціалізація: клієнт надсилає запит авторизації на сервер, передаючи логін і пароль;
* отримання токену: після успішної автентифікації сервер генерує JWT-токен, який надсилається у відповідь;
* використання токену: клієнт додає токен у заголовок Authorization всіх наступних запитів для ідентифікації;
* виконання дій: сервер обробляє запити відповідно до ролі користувача (пацієнт, лікар, родич) і повертає відповідні дані або повідомлення про помилки.

Обробка збоїв і надійність:

* у випадку недійсного або відсутнього токену сервер повертає код помилки 401 (Unauthorized);
* якщо запит містить некоректні або невалідні дані, повертається 400 (Bad Request);
* усі важливі дії логуються в системі для подальшого аудиту.

Переваги обраного підходу:

* масштабованість — REST дозволяє обслуговувати численні клієнти одночасно;
* простота реалізації — завдяки використанню стандартних HTTP-методів;
* безпечність — передбачає використання HTTPS і JWT-токенів;
* гнучкість — комунікаційна модель легко адаптується до майбутньої інтеграції з фронтендом або IoT.

### 3.1.5 Обмеження пам'яті

У межах реалізації серверної частини програмної системи були враховані обмеження, пов’язані з використанням оперативної пам’яті та обсягами переданих і збережених даних. Це особливо важливо для забезпечення стабільної роботи сервісу в умовах обмежених ресурсів, а також при масштабуванні або розгортанні на хмарній інфраструктурі.

### 3.1.6 Операції

Серверна частина системи реалізує ключову бізнес-логіку, забезпечуючи виконання всіх основних операцій, пов’язаних з обробкою медичних даних, керуванням користувачами та взаємодією з базою даних. Усі операції реалізовані за допомогою RESTful API з відповідними маршрутами для кожної ролі користувача.

Основні операції:

1. авторизація та автентифікація:
   1. користувачі проходять авторизацію за допомогою email та пароля;
   2. у разі успішної авторизації генерується JWT-токен, який надалі передається в кожному запиті для підтвердження прав доступу;
   3. токени мають термін дії, після якого необхідна повторна авторизація.
2. реєстрація користувачів (пацієнтів, лікарів, родичів):
   1. створення нового облікового запису з відповідною роллю;
   2. прив’язка родича до пацієнта і навпаки та прив’язка лікаря до пацієнта;
   3. перевірка унікальності email та інших ідентифікаційних параметрів.
3. прийом медичних показників із пристроїв (емуляція IoT):
   1. прийом запиту від пристрою: пульс, тиск, температура;
   2. збереження отриманих даних у таблиці indicators\_stamps;
   3. автоматична перевірка на допустимі межі та створення повідомлень про критичні стани (таблиця indicators\_notifications).
4. формування плану лікування лікарем:
   1. лікар створює план лікування, додає візити, призначає медикаменти;
   2. кожне призначення містить дозування, частоту, тривалість прийому;
   3. генеруються нагадування та звіти для пацієнта.
5. отримання нагадувань та сповіщень пацієнтом, пацієнт за допомогою свого профілю отримує:
   * 1. сповіщення про прийом ліків;
     2. нагадування про візит;
6. ведення електронного щоденника:
   1. пацієнт додає записи про самопочуття;
   2. лікар і родич можуть переглядати ці записи за наявності дозволу;
   3. збереження нотаток в таблиці patient\_notes.
7. управління рівнями доступу:
   1. пацієнт може надати або відкликати дозвіл на доступ до медичних даних для родича;
   2. кожен запит перевіряється middleware на відповідність правам доступу.

Технічні характеристики реалізації операцій:

* усі операції реалізовані у вигляді окремих обробників (handlers) у фреймворку Gin;
* перевірка доступу здійснюється на основі ролей, вказаних у JWT-токені;
* дані передаються у форматі JSON;
* серіалізація/десеріалізація запитів і відповідей — через Go-структури з тегами json, db, binding.

Надійність операцій:

* для критичних дій реалізовано перевірку коректності даних (валідацію);
* логування кожного запиту дозволяє виявляти помилки в роботі API.

Таким чином, серверна частина системи надає повний набір REST API-операцій, необхідних для підтримки логіки медичного моніторингу, взаємодії між користувачами та обробки біомедичних показників.

### 3.1.7 Функції продукту

Функціональні можливості програмної системи поділяються відповідно до ролей користувачів: пацієнт, лікар, родич та адміністратор. Серверна частина реалізує всі основні функції шляхом створення REST API маршрутів, які обслуговують потреби кожного типу користувача відповідно до його рівня доступу. Функціональність охоплює всі аспекти медичного моніторингу — від збору даних до формування планів лікування та інформування користувачів.

1. функції пацієнта:
2. F1.2 Ведення електронного щоденника самопочуття (додавання нотаток);
3. F1.3 Отримання нагадувань про прийом ліків та заплановані візити;
4. F1.4 Перегляд призначених ліків, графіку лікування та плану візитів;
5. функції лікаря:
6. F2.1 Перегляд детальної інформації про пацієнтів;
7. F2.2 Закріплення родича за конкретним пацієнтом та надання доступу перегляду даних за дозволу пацієнта;
8. F2.3 Закріплення діагнозу за пацієнтом та призначення лікування: створення плану, додавання медикаментів, візитів;
9. F2.4 Отримання повідомлень про критичні зміни у біомедичних показниках;
10. F2.5 Перегляд запланованих візитів;
11. функції родича:
12. F3.1 Перегляд повної інформації щодо лікування закріпленого пацієнта (якщо надано доступ) ;
13. F3.2 Отримання email-сповіщень про критичні показники закріпленого пацієнта;
14. F3.3 Читання електронного щоденника пацієнта (якщо надано доступ);
15. функції адміністратора:
16. F4.1 Реєстрація нового девайсу;
17. F4.2 Закріплення девайсу за пацієнтом.
18. системні функції:
19. F5.1 Прийом біомедичних даних (пульс, тиск, температура) від емуляторів IoT-пристроїв;
20. F5.2 Збереження отриманих показників у базі даних та логування;
21. F5.3 Автоматичний аналіз показників та створення повідомлень про відхилення;
22. F5.4 Надсилання email-повідомлень у разі порушення нормальних значень;

### 3.1.8 Припущення й залежності

Для коректного функціонування серверної частини програмної системи для медичного моніторингу та догляду за пацієнтами передбачаються наступні технічні та логічні припущення, а також зовнішні залежності, без яких повна реалізація функціоналу є неможливою:

1. припущення:
2. клієнтські додатки (веб або мобільні інтерфейси, які не входять до обсягу системи) мають бути розроблені відповідно до структури REST API, реалізованої на сервері;
3. користувачі системи (лікарі, пацієнти, родичі, адміністратори) мають змогу авторизуватись у системі через email та пароль;
4. медичні IoT-пристрої (або їх програмні емулятори) стабільно передають показники (пульс, тиск, температура) у встановленому форматі;
5. усі пристрої мають стабільне підключення до Інтернету, щоб забезпечити своєчасну доставку даних та повідомлень;
6. всі дані, що зберігаються в базі даних, відповідають актуальному стану пацієнта і вносяться кваліфікованими медичними працівниками;
7. технічні залежності:
8. мова програмування: Go (Golang) — забезпечує високу продуктивність і надійність серверної частини;
9. веб-фреймворк: Gin — використовується для побудови HTTP-обробників, маршрутизації, middleware;
10. система керування базами даних: PostgreSQL — реляційна БД для зберігання даних користувачів, показників здоров’я, планів лікування, повідомлень тощо;
11. протокол обміну з IoT: MQTT (якщо емуляція пристроїв реалізована через брокер) ;
12. електронна пошта: використовується SMTP-сервер для відправки повідомлень користувачам;
13. інструменти тестування: Postman — застосовується для ручної перевірки REST API;
14. логічні залежності:
15. користувачі повинні мати унікальні облікові записи, паролі яких зберігаються в хешованому вигляді;
16. база даних повинна бути ініціалізована перед запуском серверного застосунку;
17. серверна частина має бути попередньо налаштована з валідними адресами email для відправки повідомлень;
18. дані, що надходять з пристроїв, повинні відповідати заданому формату та межам медичних норм.

## 3.2. Властивості програмного продукту

Розроблена серверна частина програмної системи для медичного моніторингу та догляду за пацієнтами повинна відповідати низці важливих характеристик, які гарантують її надійність, стабільність, безпеку й ефективність у роботі. Враховуючи специфіку медичної галузі, програмний продукт повинен мати високий рівень відповідальності за обробку критично важливих даних.

1. Функціональність

Серверна частина забезпечує реалізацію основного функціоналу системи, зокрема:

* прийом та зберігання медичних показників (пульс, температура, тиск);
* автоматичну обробку й аналіз показників;
* формування та збереження планів лікування;
* призначення медикаментів;
* надсилання сповіщень у випадку виявлення критичних змін;
* ведення медичного щоденника пацієнта;
* розмежування доступу відповідно до ролей (пацієнт, лікар, родич, адміністратор);
* надсилання email-сповіщень через SMTP.

Функціональність реалізована у вигляді REST API, що дозволяє інтегрувати сервер з майбутніми фронтенд- або мобільними інтерфейсами.

1. Надійність

Сервер має обробляти запити без збоїв, навіть за умов високого навантаження або часткової недоступності клієнтських компонентів. Обробка помилок реалізована на всіх ключових етапах, включаючи:

* перевірку вхідних даних;
* обробку виключень під час взаємодії з базою даних;
* перевірку авторизації та автентифікації;
* логування подій.

Механізм повторних спроб і обробки критичних ситуацій дозволяє уникнути втрати даних.

1. Безпека

Безпека є пріоритетною властивістю системи, оскільки вона працює з конфіденційною медичною інформацією. Реалізовані такі заходи:

* хешування паролів користувачів;
* авторизація через JWT-токени;
* middleware, що контролює доступ до маршрутів відповідно до ролей;
* перевірка валідності токену при кожному запиті;
* HTTPS-зʼєднання (передбачається в умовах розгортання).

1. Супроводжуваність

Архітектура серверної частини є модульною:

* моделі зберігаються окремо (структури даних);
* для кожної сутності створено репозиторій;
* бізнес-логіка винесена в сервіси;
* маршрути згруповані й керуються з окремого компонента.

Таке структурування дозволяє легко розширювати функціональність або вносити зміни без порушення роботи всієї системи.

1. Масштабованість

Завдяки використанню мови Go, яка добре підходить для побудови високонавантажених сервісів, система легко масштабується. Всі обробники запитів є асинхронними, а база даних PostgreSQL дозволяє обробляти велику кількість одночасних транзакцій.

## 3.3. Атрибути програмного коду

У процесі створення серверної частини програмної системи для медичного моніторингу та догляду за пацієнтами були враховані ключові нефункціональні характеристики, які визначають якість, стійкість та ефективність її роботи.

## 3.4 Вимоги бази даних

Програмна система для медичного моніторингу та догляду за пацієнтами повинна використовувати реляційну систему керування базами даних PostgreSQL як основне сховище інформації. У базі даних зберігаються дані про користувачів, пацієнтів, родичів, лікарів, медичні показники (температура, пульс, тиск), плани лікування, повідомлення, історію змін стану здоров’я.

1. база даних має забезпечувати:
2. цілісність і збереження даних у довгостроковій перспективі;
3. логічну побудову зв’язків між сутностями;
4. безпечний доступ до персональних і медичних даних;
5. підтримку фільтрації та аналітики медичних записів.
6. модель бази даних повинна передбачати:
7. нормалізацію до третьої нормальної форми (3НФ);
8. наявність первинних і зовнішніх ключів для забезпечення цілісності;
9. зберігання паролів у зашифрованому вигляді (bcrypt);
10. унікальність ідентифікаторів користувачів та інших важливих полів (наприклад, email);
11. чітке розмежування між різними типами користувачів (пацієнт, лікар, адміністратор, родич).

## 3.5 Інші вимоги

Серверна частина програмної системи має відповідати додатковим вимогам, що сприятимуть її стабільному функціонуванню та супроводженню в медичному середовищі.

Система повинна:

* підтримувати відокремлення логіки доступу для кожного типу користувача (лікар, пацієнт, родич, адміністратор);
* бути документованою, з описом API та базовими сценаріями використання;
* забезпечувати простоту масштабування, з урахуванням збільшення кількості користувачів і обсягу даних;
* бути реалізованою у вигляді REST API, придатного для використання сторонніми клієнтами;
* мати можливість інтеграції з зовнішніми сервісами повідомлень, наприклад SMTP або іншими засобами оповіщення;
* бути реалізованою з урахуванням модульної архітектури, що полегшує підтримку та розширення функціональності.

# **4 ДОДАТКОВІ МАТЕРІАЛИ**

У цьому розділі наведено візуальні та структурні моделі, які доповнюють текстову частину специфікації програмного забезпечення та деталізують архітектурні й функціональні аспекти розробленої системи.

У межах розробки серверної частини програмної системи для медичного моніторингу та догляду за пацієнтами було створено набір діаграм, які відображають ключові аспекти архітектури, структури даних і логіки функціонування. Ці діаграми допомагають краще зрозуміти взаємодію компонентів, структуру коду та особливості обробки інформації в системі.

4.1 UML-діаграма прецедентів

Діаграма прецедентів (див.рис. 4.1) дозволяє описати базові сценарії взаємодії користувачів із системою. Вона включає основні ролі: адміністратор, лікар, пацієнт та родич. Для кожного з акторів показано відповідні дії, наприклад, перегляд щоденника, призначення лікування, отримання сповіщень тощо.

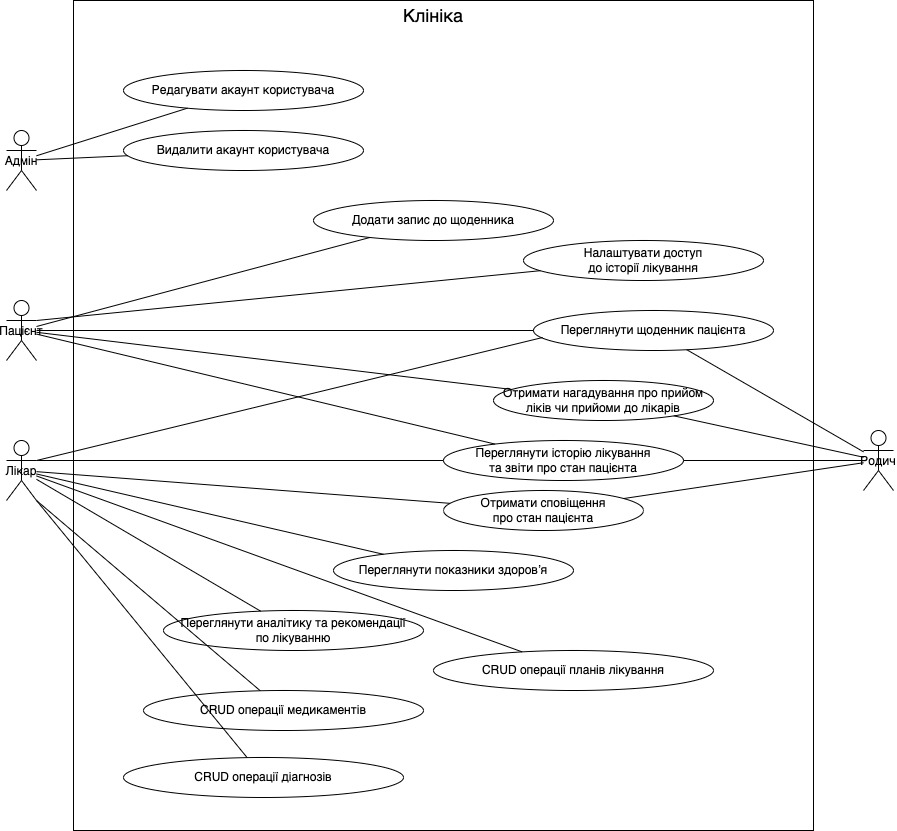


Рисунок 4.1 – UML-діаграма прецедентів серверної частини програмної системи

4.2 UML-діаграма пакетів

Діаграма пакетів (див.рис. 4.2) демонструє логічну структуру серверного застосунку, розбиту на пакети згідно з функціональними зонами.

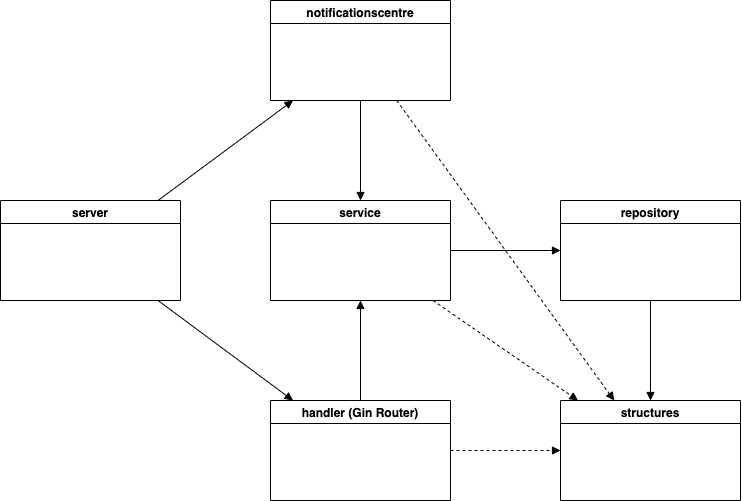
****

Рисунок 4.2 – UML-діаграма пакетів серверної частини

4.3 UML-діаграма компонентів

Діаграма компонентів (див.рис. 4.3) ілюструє, як окремі модулі системи взаємодіють один з одним. Вона відображає такі складові, як HTTP-сервер, сервіс авторизації, сервіс обробки медичних даних, модулі сповіщень, модуль звітності, база даних PostgreSQL тощо.

Зображення, що містить схема, текст, План, Креслення

Вміст на основі ШІ може бути неправильним.

Рисунок 4.3 – UML-діаграма компонентів системи

4.4 UML-діаграма розгортання

Для відображення архітектури розміщення програмного забезпечення створено діаграму розгортання (див.рис. 4.4). На ній показано сервер додатку, базу даних, емулятор IoT-пристрою та користувацькі клієнти (браузер або інші зовнішні інтерфейси), які взаємодіють із сервером через REST API.

Зображення, що містить текст, схема, План, Креслення

Вміст на основі ШІ може бути неправильним.

Рисунок 4.4 – UML-діаграма розгортання системи

4.5 ER-діаграма бази даних

ER-діаграма (див.рис. 4.5) демонструє логічну структуру даних, які зберігаються у базі.

Зображення, що містить текст, схема, План, Паралель

Вміст на основі ШІ може бути неправильним.

Рисунок 4.5 – ER-діаграма структури бази даних