

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр» за
спеціальністю 121 «Інженерія програмного забезпечення» (освітня
програма «Інженерія програмного забезпечення»)
на тему:

**«Оптимізація процесів управління завантаженості
лікарні»**

Виконав студент групи ІПЗк-23-1

ЗЕЛІНСЬКИЙ Владислав Анатолійович

Керівник роботи:

САВІЦЬКИЙ Роман Святославович

Рецензент:

ЧИЖМОТРЯ Олексій Володимирович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«14» лютого 2025р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЗЕЛІНСЬКИЙ Владислав Анатолійович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Оптимізація процесів управління завантаженості лікарні»**,
затверджена Наказом закладу вищої освіти від «14» лютого 2025р., №61/с

Вихідні дані для роботи: процес управління завантаженістю лікарні, що
включає облік пацієнтів, розподіл палат і ліжок, планування госпіталізацій,
контроль використання ресурсів та автоматизацію внутрішніх операцій
медичного закладу.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням
розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р.С.	14.02.2025р.	14.02.2025р.
2	Савіцький Р.С.	14.02.2025р.	14.02.2025р.
3	Савіцький Р.С.	14.02.2025р.	14.02.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра на тему «Оптимізація процесів управління завантаженості лікарні» складається з вебдодатку та пояснювальної записки. Пояснювальна записка до випускної роботи містить 93 сторінки, 48 ілюстрацій, 19 таблиць, 7 додатків та 30 використаних літературних джерел.

Метою роботи є створення додатку для спрощення управління лікарняними ресурсами, зокрема ліжковим фондом та пацієнтотоками, для адміністративного та медичного персоналу.

Архітектура вебдодатку побудована за клієнт-серверною моделлю, що забезпечує надійну обробку даних та надає користувачам зручний інтерактивний інтерфейс. Система підтримує різні ролі користувачів (наприклад, адміністратор, лікар) з чітким розмежуванням прав доступу. Централізована база даних зберігає всю необхідну інформацію про структуру лікарні, ліжковий фонд, персонал та пацієнтів, що забезпечує цілісність даних та ефективне управління.

Додаток має інтуїтивно зрозумілий інтерфейс з візуальними панелями моніторингу для відстеження ключових показників завантаженості в реальному часі. Реалізовано функціонал для управління структурою лікарні, ліжковим фондом, персоналом та пацієнтами. Процеси госпіталізації та виписки максимально спрощені для підвищення ефективності роботи персоналу.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТОК, СИСТЕМА УПРАВЛІННЯ, УПРАВЛІННЯ ЛІКАРНЕЮ, ЛІЖКОВИЙ ФОНД, МЕДИЧНА ІНФОРМАЦІЙНА СИСТЕМА, ОПТИМІЗАЦІЯ ПРОЦЕСІВ, ЗАВАНТАЖЕНОСТЬ ЛІКАРНІ.

					ІПЗ.КР.Б – 121- 25 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Зелінський В.А.			Оптимізація процесів управління завантаженості лікарні Пояснювальна записка	Літ.	Арк.	Аркуші
Керівник		Савіцький Р.С.					3	93
		.				Житомирська політехніка, група ІПЗк-23-1		
Рецензент		Чижомотря О.В.						
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's graduation qualification work titled "Optimization of Hospital Workload Management Processes" consists of a web application and an explanatory report. The report comprises 93 pages, 48 illustrations, 19 tables, 7 appendices, and a list of 30 referenced literary sources.

The purpose of this work is to develop an application that simplifies the management of hospital resources—particularly bed capacity and patient flow—for administrative and medical personnel.

The architecture of the web application follows a client-server model, ensuring reliable data processing and providing users with a convenient, interactive interface. The system supports multiple user roles (e.g., administrator, doctor) with clearly defined access rights. A centralized database stores all necessary information about the hospital's structure, bed capacity, staff, and patients, ensuring data integrity and effective resource management.

The application features an intuitive interface with visual monitoring dashboards for real-time tracking of key workload indicators. It includes functionality for managing hospital structure, bed inventory, personnel, and patient records. The processes of admission and discharge have been optimized to enhance the efficiency of the medical staff.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ПЛАТФОРМИ ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕСІВ УПРАВЛІННЯ ЗАВАНТАЖЕНОСТІ ЛІКАРНІ	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту	10
1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	17
Висновки до першого розділу	20
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБДОДАТКУ	22
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	22
2.2 Розробка бази даних системи.....	29
2.3 Проєктування та реалізація алгоритмів роботи системи	33
2.4 Реалізація функціоналу системи платформи.....	35
Висновки до другого розділу.....	44
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ.....	46
3.1 Порядок встановлення та налаштування параметрів платформи.....	46
3.2 Структура інтерфейсу вебдодатку	47
3.3 Тестування роботи вебдодатку.....	60
Висновки до третього розділу	65
ВИСНОВКИ.....	67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	68
ДОДАТКИ.....	71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

CRUD – Create, Read, Update, Delete

HTTP – HyperText Transfer Protocol

JWT – JSON Web Token

MVC – Model-view-controller

ORM – Object-Relational Mapping

SQL – Structured Query Language

WEB – World Wide Web

БД – База даних

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми

Сфера охорони здоров'я активно впроваджує цифрові рішення для підвищення ефективності та якості надання медичної допомоги. Лікарні та медичні заклади стикаються з постійною потребою в оптимізації управління ресурсами, зокрема ліжковим фондом, та координації пацієнтопотоків.

Водночас багато медичних установ продовжують використовувати застарілі або ручні методи обліку, що призводить до неефективного використання ліжок, збільшення часу очікування для пацієнтів та надмірного навантаження на персонал. Рішенням цих проблем є розробка спеціалізованих вебдодатків, що дозволяють централізовано та в реальному часі управляти всіма аспектами завантаженості лікарні.

Метою випускної кваліфікаційної роботи бакалавра є розробка вебдодатку для оптимізації процесів управління завантаженістю лікарні.

Для реалізації поставленої мети потрібно виконати наступні завдання:

1. Провести аналіз існуючих медичних інформаційних систем та додатків для управління лікарняними ресурсами, їх функціональних можливостей та архітектурних рішень.

2. Визначити вимоги до функціональності вебдодатку, з урахуванням потреб користувачів (адміністраторів, лікарів, медичного персоналу).

3. Обрати інструменти та засоби для розробки вебдодатку (мови програмування, фреймворки, СУБД тощо).

4. Спроектувати архітектуру системи, визначити її основні модулі (управління ліжковим фондом, пацієнтами, персоналом) та механізми взаємодії між ними.

5. Реалізувати вебдодаток з функціоналом управління статусами ліжок, госпіталізації та виписки пацієнтів, ведення обліку та звітності.

6. Провести тестування розробленого вебдодатку для перевірки його функціональності, надійності та зручності використання.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

7. Оцінити ефективність розробленого рішення та можливість його подальшого масштабування.

Об'єктом дослідження є процеси автоматизації та цифрового управління завантаженістю лікарні.

Предметом дослідження є методи, моделі та інструменти розробки вебдодатку для ефективного управління лікарняними ресурсами, включаючи управління ліжковим фондом, процесами госпіталізації та виписки пацієнтів, та взаємодію між медичним персоналом.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ПЛАТФОРМИ ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕСІВ УПРАВЛІННЯ ЗАВАНТАЖЕНОСТІ ЛІКАРНІ

1.1 Постановка задачі

Метою кваліфікаційної роботи бакалавра є створення вебдодатку для оптимізації процесів управління завантаженістю лікарні, використовуючи сучасні технології, що дозволить легко користуватись та масштабувати його у майбутньому [1].

Вебдодаток для управління завантаженістю лікарні має представляти собою набір взаємопов'язаних програмних модулів, що забезпечують зручний моніторинг, облік та управління лікарняними ресурсами як для адміністративного, так і для медичного персоналу [2]. Основні модулі включають:

- модуль автентифікації та управління доступом;
- модуль управління структурою лікарні (для адміністраторів);
- модуль управління персоналом (для адміністраторів);
- модуль управління пацієнтами;
- модуль управління госпіталізаціями та ліжковим фондом;
- модуль пошуку та фільтрації.

У результаті має бути створено вебдодаток, який дозволить персоналу лікарні ефективно керувати ліжковим фондом та пацієнтопотоками в онлайн-режимі. Платформа повинна забезпечити просту взаємодію між адміністративним та медичним персоналом, автоматизувавши процеси обліку та звітності.

Основними етапами розробки платформи є:

1. Розробка дизайну вебдодатку:

- створення загальної концепції дизайну додатку;
- створення UI компонентів;
- розробка дизайну сторінок.

2. Побудова архітектури клієнтської частини додатку.
3. Побудова архітектури серверної частини додатку.
4. Розробка системи авторизації та ролей користувачів.
5. Розробка основного функціоналу:
 - створення логіки авторизації користувачів з відповідними ролями;
 - створення віддаленої бази даних для зберігання інформації;
 - імплементація бекенду на клієнтській стороні;
 - організація спілкування серверної частини з клієнтською.

Результатом виконання плану дій є створення платформи, за допомогою якої медичний персонал може відстежувати статуси ліжок, керувати госпіталізацією та випискою пацієнтів. Адміністратори можуть керувати структурою лікарні, персоналом та ліжковим фондом, що полегшить розподіл ресурсів та покращить внутрішню комунікацію в закладі.

1.2 Аналіз аналогів програмного продукту

Для створення ефективної системи управління завантаженістю лікарні було проведено аналіз існуючих рішень у сфері медичного програмного забезпечення [3]. Нижче наведено огляд основних платформ, що частково або повністю реалізують функціональність, схожу на заплановану систему.

Одним із таких рішень є система TeleTracking Health (рис. 1.1).

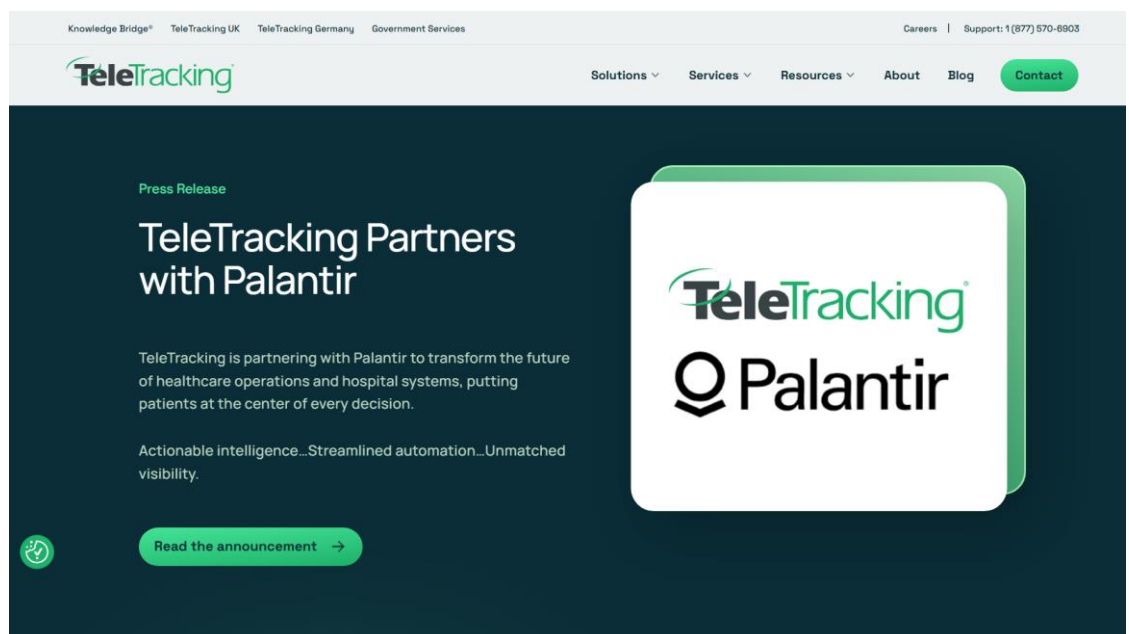


Рис. 1.1 – TeleTracking Health

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

TeleTracking — це провідна платформа для управління лікарняними ресурсами в реальному часі, яка широко використовується у США та Великобританії. Вона забезпечує комплексне управління ліжковим фондом, потоками пацієнтів, а також автоматизацію внутрішніх логістичних процесів у лікарнях.

Сервіс має низку суттєвих переваг, що роблять його ефективним у великих медичних установах. Насамперед варто відзначити потужну систему моніторингу статусу кожного ліжка в реальному часі, яка інтегрується з існуючими клінічними системами. Завдяки цьому персонал може швидко отримувати інформацію про наявність вільних місць, потребу в прибиранні або технічному обслуговуванні. Інтерфейс платформи орієнтований на ефективну комунікацію між персоналом, наприклад, між приймальним відділенням, медсестрами та прибиральниками, що дозволяє мінімізувати затримки у переведенні або виписці пацієнтів. Ще однією важливою перевагою є аналітична система, що дозволяє прогнозувати завантаження, виявляти “вузькі місця” у процесах та оптимізувати маршрути пацієнтів у межах лікарні.

Водночас система має й певні недоліки. По-перше, це комерційна платформа з високою вартістю впровадження, що робить її недоступною для малобюджетних медичних закладів або лікарень у країнах із середнім рівнем доходу. По-друге, налаштування системи потребує тривалої інтеграції з наявною інфраструктурою — включаючи медичні інформаційні системи, внутрішні бази даних, облік кадрів тощо. Також система не має відкритого коду, що виключає можливість самостійної кастомізації без втручання вендора. Крім того, відсутність української локалізації може ускладнити використання платформи в нашому медичному середовищі без додаткових витрат на адаптацію.

Ще одним прикладом є система HospitalRun (рис. 1.2).

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

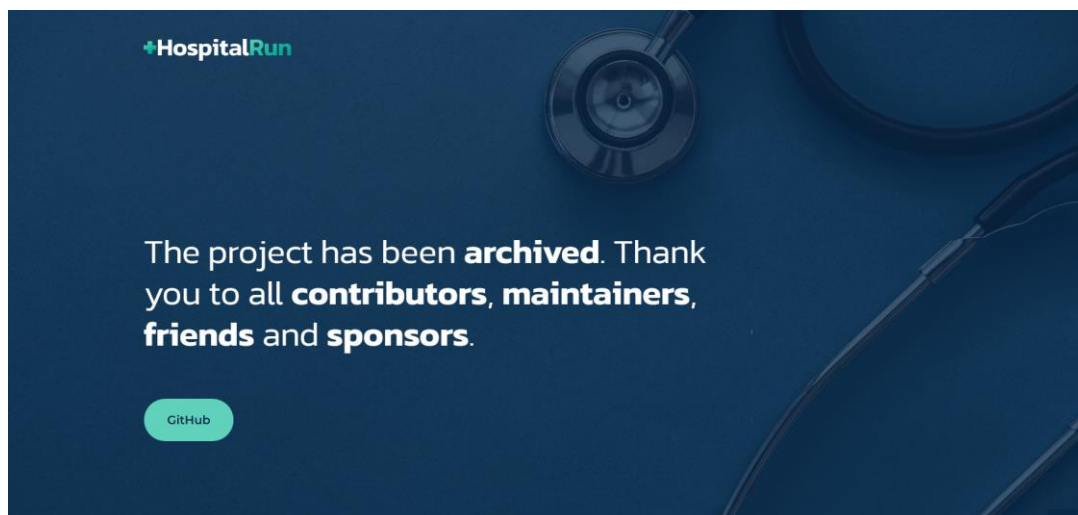


Рис. 1.2 – HospitalRun

HospitalRun — це відкрита система електронного медичного обліку (EMR), орієнтована на медичні заклади з обмеженим доступом до інтернету, зокрема в країнах, що розвиваються. Платформа дозволяє організувати облік пацієнтів, вести реєстрацію медичних процедур, облік медикаментів, а також контролювати наявність ресурсів, включно з ліжками.

Однією з головних переваг HospitalRun є можливість роботи в автономному режимі — усі дані зберігаються локально на пристрої, з подальшою синхронізацією при появі інтернету. Це робить систему надзвичайно зручною для лікарень у сільській місцевості або регіонах із нестабільним зв'язком. Завдяки відкритому вихідному коду платформа може бути повністю кастомізована під конкретні потреби медичного закладу: змінюється інтерфейс, додаються нові модулі, інтегрується локальна мова тощо. Також вона має активну спільноту розробників та волонтерів, які підтримують проєкт, забезпечуючи стабільні оновлення та виправлення помилок.

Проте у HospitalRun є і помітні недоліки. Зокрема, незважаючи на підтримку автономного режиму, функціональність системи обмежена: відсутні складні алгоритми планування ресурсів, аналітики або візуалізації завантаження ліжок у реальному часі. Інтерфейс системи досить простий і не завжди інтуїтивно зрозумілий для користувачів без технічної підготовки. Ще одна проблема — відсутність офіційної підтримки або SLA (service-level agreement), оскільки проєкт базується на волонтерських засадах. Це може бути критичним для

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

великих лікарень, де потрібна оперативна технічна допомога. Крім того, система потребує локального хостингу або самостійного налаштування серверного оточення, що створює додаткове навантаження на ІТ-відділ лікарні.

Варто звернути увагу на систему OpenHospital, яка отримала застосування в низці клінік країн, що розвиваються (рис. 1.3).

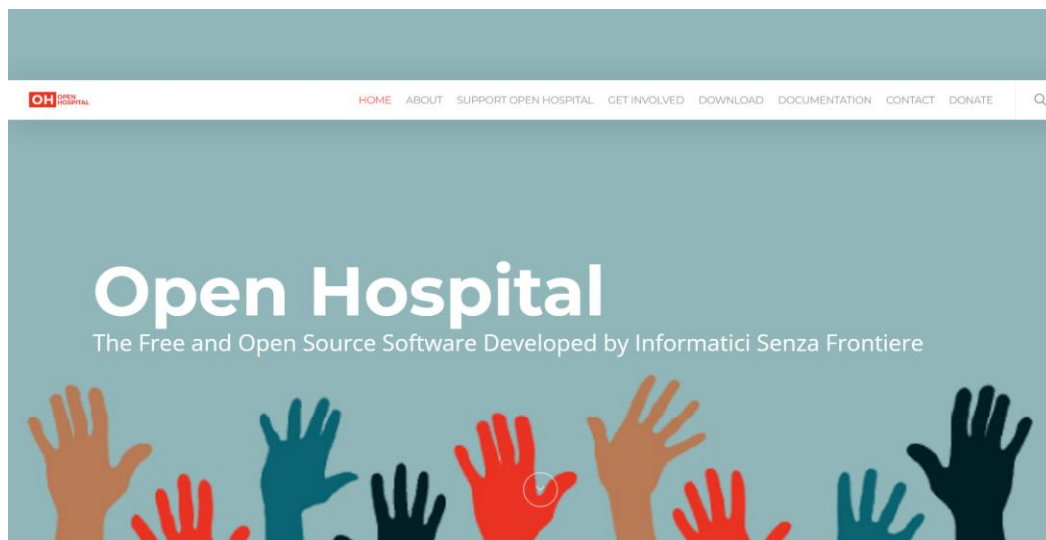


Рис. 1.3 – OpenHospital

OpenHospital — це безкоштовна відкрита медична інформаційна система, розроблена неурядовою організацією Informatici Senza Frontiere. Вона активно використовується в невеликих лікарнях країн Африки та Азії, де потрібна базова, але стабільна система для обліку медичних послуг. Основна функціональність включає ведення медичних записів, госпіталізацій, призначень, лабораторних результатів, облік ліків і базову статистику.

Головною перевагою OpenHospital є її простота — система створена з урахуванням низького рівня технічної підготовки користувачів, а тому має інтерфейс, адаптований до потреб медичного персоналу в умовах обмежених ресурсів. Завдяки відкритому коду можливе гнучке розширення функціоналу та локалізація для конкретної мови чи країни. Програма також працює у локальній мережі або на окремому комп'ютері, без потреби постійного підключення до інтернету. Ще одна важлива перевага — наявність реального впровадження у клініках з позитивним досвідом роботи, що підтверджує життєздатність системи.

Разом з тим, OpenHospital має низку обмежень. Інтерфейс застарілий як

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

візуально, так і з точки зору зручності — робота із записами вимагає великої кількості ручних дій. Платформа не підтримує сучасних вебтехнологій і працює як класичний десктопний додаток, що обмежує її гнучкість. В системі відсутні інтегровані можливості аналітики або візуалізації навантаження, що критично для сучасного підходу до оптимізації ресурсів, зокрема ліжкового фонду. Також бракує гнучкого управління правами доступу, що ускладнює організацію роботи в багатопрофільних медичних установах.

GNU Health – безкоштовна модульна медична інформаційна система з відкритим кодом, орієнтована на громадське здоров'я та лікарняний менеджмент (рис. 1.4).

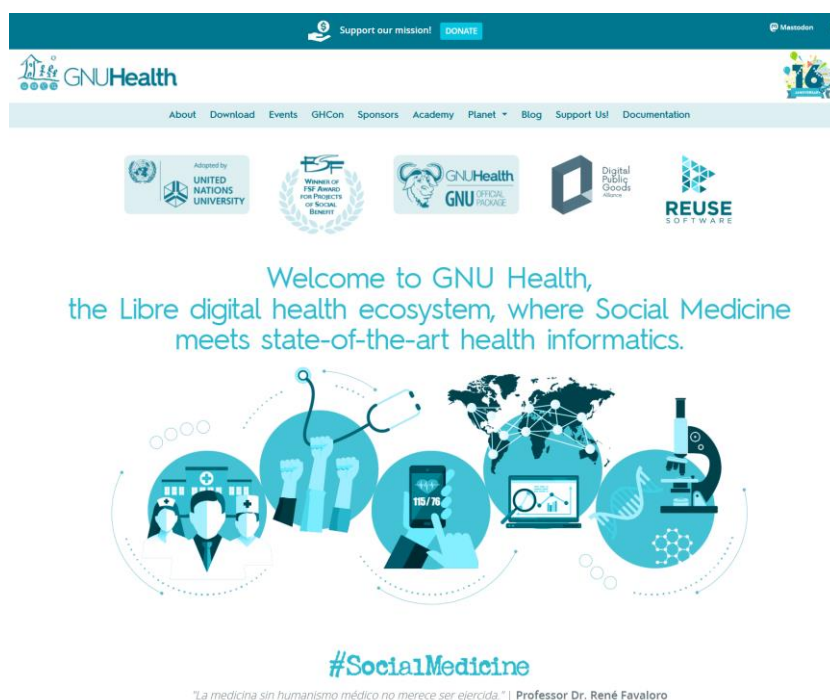


Рис. 1.4 – GNU Health

GNU Health — це потужна платформа, що поєднує функції електронної медичної картки (EMR), системи управління лікарнею (HIS) та платформи для громадського здоров'я. Розроблена у співпраці з міжнародними організаціями, зокрема ВООЗ, вона орієнтована на гнучке впровадження як у великих медичних установах, так і у первинній ланці охорони здоров'я. Платформа дозволяє вести повний цикл медичної документації: від реєстрації пацієнта до формування звітів та інтеграції з лабораторними модулями.

Серед основних переваг GNU Health — розширюваність через модулі,

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

потужна підтримка міжнародних стандартів (HL7, ICD-10, LOINC), а також активна спільнота розробників. Інтеграція з інструментами обробки даних дозволяє аналізувати епідеміологічні тенденції, облік вакцин та хронічних захворювань. Система також підтримує мультиустановну архітектуру — декілька лікарень або клінік можуть працювати через єдину платформу, зберігаючи незалежний контроль над своїми даними.

Водночас GNU Health має недоліки, які стримують її масове впровадження. Інтерфейс користувача складний для нефахівців: налаштування системи потребує значних технічних знань, зокрема розгортання на базі GNU/Linux-серверів. Вебінтерфейс платформи менш функціональний, ніж десктопна версія. Інтеграція зі сторонніми API або мобільними додатками обмежена — це створює бар'єр для швидкої адаптації в закладах, які потребують мобільного доступу або сучасного UI/UX. Крім того, повноцінна підтримка української мови відсутня, що може стати проблемою в умовах локалізації системи під національні потреби.

CareCloud – хмарна медична платформа для управління клініками та оптимізації адміністративних процесів у закладах охорони здоров'я (рис. 1.5).

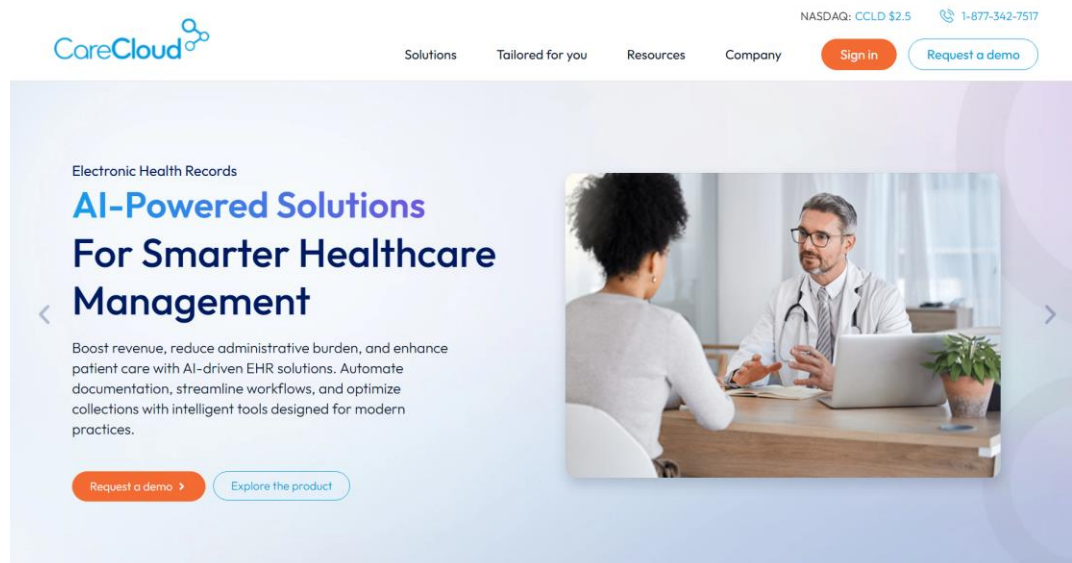


Рис. 1.5 – CareCloud

CareCloud є одним із провідних SaaS-рішень для клінік у США, орієнтованим на амбулаторний сегмент. Платформа поєднує функціональність електронної медичної картки, автоматизацію фінансових операцій, управління

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

записами пацієнтів, оптимізацію розкладів та аналітику. Вона підтримує рольову модель доступу, що дозволяє ефективно розмежовувати права між адміністраторами, лікарями, персоналом рецепції та пацієнтами.

Головна перевага CareCloud — сучасний інтерфейс користувача, адаптований під мобільні пристрої та планшети. Система має вбудовану підтримку телемедицини, зручну інтеграцію з лабораторіями та сторонніми сервісами через RESTful API. Також серед сильних сторін — інструменти бізнес-аналітики, які дозволяють адміністрації клініки відстежувати фінансові показники, навантаження на лікарів та ефективність ресурсів у реальному часі.

Однак CareCloud не позбавлений недоліків. По-перше, це пропрієтарна система з високою абонплатою, що робить її недосяжною для багатьох малих та середніх медичних закладів, особливо в країнах з низьким бюджетом охорони здоров'я. По-друге, через орієнтацію на американський ринок, платформа не підтримує міжнародні стандарти, такі як SNOMED CT чи локалізовані класифікатори (наприклад, МКХ-10 українською). Це ускладнює адаптацію системи до вимог МОЗ України. Крім того, функціональність CareCloud, хоча й потужна, орієнтована більше на приватні практики та мережі клінік, а не на специфіку управління ліжковим фондом чи внутрішньолікарняну логістику.

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1).

Таблиця 1.1

Таблиця порівняння платформ

Характеристика	TeleTracking	HospitalRun	OpenHospital	GNU Health	CareCloud	WardOps
Ліжковий фонд	+	+	±	±	-	+
Госпіталізація і виписка	+	+	+	+	±	+
Візуалізація зайнятості	+	±	±	±	-	+
Управління структурою	+	+	+	+	±	+
Мобільна підтримка	+	±	-	±	+	±
Відкритий код	-	+	+	+	-	+

Проведений аналіз існуючих систем управління завантаженістю лікарень виявив, що на ринку представлені як комерційні, так і відкриті рішення, кожне зі своїми перевагами та недоліками. Комерційні платформи, як правило, забезпечують широкий функціонал і підтримку, але мають високу вартість впровадження та експлуатації, що обмежує їх доступність для багатьох медичних закладів. Відкриті системи є більш гнучкими для кастомізації, але потребують значних зусиль на адаптацію і технічне супроводження.

Аналіз показав, що жоден з наявних продуктів не повністю відповідає специфічним потребам локальних лікарень, зокрема щодо зручності управління ліжковим фондом і інтеграції із внутрішніми процесами медичного закладу. Багато рішень недостатньо орієнтовані на користувача та не враховують особливості локального контексту, що знижує їх ефективність у реальних умовах роботи.

Враховуючи ці фактори, розробка власного вебдодатку, який враховує специфіку українських лікарень і надає простий та інтуїтивний інтерфейс, є доцільною і перспективною. Такий продукт сприятиме оптимізації розподілу палат і ліжок, покращенню комунікації між адміністративним та медичним персоналом, а також підвищить якість медичного обслуговування.

1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для реалізації вебдодатку для оптимізації процесів управління завантаженістю лікарні було обрано сучасний стек технологій, що забезпечує продуктивність, масштабованість та зручність у розробці [4].

ASP.NET Core — це кросплатформовий фреймворк від компанії Microsoft, що використовується для побудови високопродуктивних вебзастосунків [5]. У даному проєкті ця технологія реалізує серверну частину, яка відповідає за обробку запитів, логіку бізнес-процесів та взаємодію з базою даних.

Вибір ASP.NET Core був зумовлений його високою продуктивністю, офіційною підтримкою, широким інструментарієм для роботи з API, а також можливістю інтеграції з Microsoft SQL Server [6]. Окрім цього, фреймворк

дозволяє реалізувати сучасні підходи до безпеки — зокрема, автентифікацію на основі JWT, авторизацію за ролями, логування, валідацію та обробку помилок.

Серед ключових переваг ASP.NET Core варто відзначити підтримку об'єктно-орієнтованого підходу, гнучку систему middleware, модульність і високу масштабованість. Цей фреймворк ідеально підходить для критичних застосунків у сфері охорони здоров'я, де важливі швидкодія, стабільність і захищеність даних. Також важливою є нативна підтримка REST API [7], що дозволяє ефективно взаємодіяти з клієнтом.

Microsoft SQL Server — надійне, потужне та продуктивне рішення від Microsoft, що ідеально інтегрується з ASP.NET Core. У системі воно відповідає за зберігання структурованих даних: інформації про лікарняні ліжка, палати, відділення, пацієнтів, персонал, госпіталізації та інші об'єкти.

Серед переваг SQL Server — підтримка складних транзакцій, масштабованість, гнучка система індексації та оптимізація запитів. Крім цього, SQL Server має вбудовані засоби забезпечення цілісності даних (primary/foreign keys, triggers, constraints), що дозволяє уникати логічних помилок у базі.

Next.js — це сучасний фреймворк для React, який дозволяє реалізовувати вебінтерфейси з підтримкою SSR (server-side rendering), SSG (static site generation) та CSR (client-side rendering) [8]. У поєднанні з мовою TypeScript цей фреймворк використовується для створення клієнтської частини системи [9].

Next.js дозволяє реалізувати інтерфейс із високим рівнем продуктивності та швидким завантаженням сторінок, що особливо важливо в контексті роботи медперсоналу — час реакції системи має бути мінімальним. SSR забезпечує попереднє завантаження сторінок та SEO-оптимізацію, що може бути корисно для відкритих частин застосунку (наприклад, інформаційних панелей чи публічних API-документацій).

Використання TypeScript дозволяє запровадити статичну типізацію, що критично важливо для великих застосунків: вона мінімізує кількість помилок у коді, забезпечує кращу інтеграцію з IDE, пришвидшує розробку та полегшує підтримку.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

Tailwind CSS — це утилітарна CSS-бібліотека, яка дозволяє стилізувати компоненти прямо в HTML/JSX-кодi за допомогою коротких класів. У межах системи управління лікарнею Tailwind CSS використовується для створення адаптивного, чистого та структурованого інтерфейсу [10].

Головною перевагою Tailwind є швидкість розробки та контроль над зовнішнім виглядом. Він дозволяє створювати кастомізовані інтерфейси без необхідності писати окремі CSS-файли. Tailwind також чудово інтегрується з JavaScript-фреймворками і забезпечує добру підтримку мобільної адаптивності.

Tailwind значно пришвидшує створення інтерфейсів, забезпечує чистоту CSS-коду (без повторень і конфліктів) та дозволяє створювати UI, що відповідає сучасним стандартам UX.

shadcn/ui — це бібліотека готових, естетично привабливих UI-компонентів, побудованих на базі Tailwind CSS та Radix UI [11]. У межах даного проєкту вона використовується для реалізації форм, діалогових вікон, таблиць, навігації, кнопок, модальних вікон тощо.

Бібліотека дозволяє пришвидшити розробку інтерфейсу без втрати гнучкості та кастомізації. Компоненти легко стилізуються відповідно до загального дизайну системи, підтримують типізацію з TypeScript і мають хорошу доступність (accessibility), що важливо для медичних додатків, де користувачами можуть бути люди різного віку та навичок.

Завдяки shadcn/ui інтерфейс системи є цілісним, легким для сприйняття, і водночас технічно оптимізованим для використання на різних пристроях.

Вимоги до апаратного забезпечення

На стороні клієнта (користувача вебдодатку):

- операційна система: будь-яка сучасна (Windows, macOS, Linux);
- браузер: Google Chrome, Mozilla Firefox, Safari, Edge (останні версії);
- оперативна пам'ять: від 4 ГБ;
- роздільна здатність екрану: від 1280x720.

На стороні сервера:

- операційна система: Linux (рекомендовано) або Windows Server;

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

- процесор: не менш ніж 4-ядерний, 2.2 GHz;
- оперативна пам'ять: від 8 ГБ;
- місце на жорсткому диску: від 40 ГБ;
- програмне забезпечення:
 - вебсервер (реверс-проксі): Nginx або IIS;
 - .NET 8 Runtime або вище;
 - PostgreSQL 14 або вище;
 - Node.js 18 або вище (для збірки клієнтської частини).

Висновки до першого розділу

У результаті проведеного аналізу предметної області було сформовано чітке бачення функціональних потреб медичних закладів у контексті управління завантаженістю лікарні. Основною метою системи є автоматизація обліку лікарняних ресурсів, зокрема ліжкового фонду, та спрощення процесів госпіталізації, виписки і моніторингу пацієнтів. Система має надати адміністрації інструменти для керування структурою медичного закладу, а медичному персоналу — засоби для оперативного доступу до актуальної інформації про зайнятість палат і ліжок.

У процесі дослідження аналогічних програмних рішень було розглянуто п'ять сучасних платформ: TeleTracking, HospitalRun, OpenHospital, GNU Health та CareCloud. Кожна з них реалізує окремі аспекти управління ресурсами лікарні, однак жодна не забезпечує комплексного підходу, орієнтованого саме на швидке відображення завантаженості ліжок, взаємодію адміністрації з медичним персоналом і простоту використання в українських реаліях. Порівняння технічних рішень, переваг та недоліків дозволило виокремити функції, які мають бути реалізовані у власному проєкті для досягнення максимальної ефективності.

На основі проведеного аналізу було обґрунтовано вибір інструментальних засобів, які найкраще підходять для реалізації поставлених завдань: серверна частина реалізується за допомогою ASP.NET Core (C#), база даних — на основі SQL Server, клієнтська частина — за допомогою Next.js з TypeScript, а інтерфейс

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

користувача створюється за допомогою Tailwind CSS і бібліотеки shadcn/ui. Обрані технології забезпечують високу продуктивність, масштабованість і безпеку системи, що є критично важливим для медичних інформаційних продуктів.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБДОДАТКУ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Вебдодаток для управління завантаженістю лікарні надає інтуїтивно зрозумілий вебінтерфейс, який дозволяє адміністративному та медичному персоналу ефективно керувати ліжковим фондом, госпіталізаціями, пацієнтами та структурою лікарні. Система реалізована як модульна платформа, що дозволяє швидко отримувати доступ до актуальної інформації та виконувати необхідні дії відповідно до ролі користувача [12].

Уся функціональність вебдодатку базується на чіткому розмежуванні прав доступу між двома основними типами користувачів:

- адміністратор — авторизований користувач, який має повний доступ до керування структурою лікарні (відділення, палати, ліжка), персоналом (додавання та деактивація облікових записів), а також може переглядати загальну аналітику та контрольні звіти про стан завантаженості закладу;

- медичний працівник — користувач, який працює з модулями госпіталізації та ліжкового фонду. Має можливість госпіталізувати пацієнтів, змінювати статус ліжка (вільне, зайняте, прибирання), переглядати історію госпіталізацій та здійснювати виписку пацієнтів.

Для кращого розуміння взаємодії між користувачами та системою було побудовано Діаграму варіантів використання, яка відображає основні сценарії роботи в системі (рис. 2.1).

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

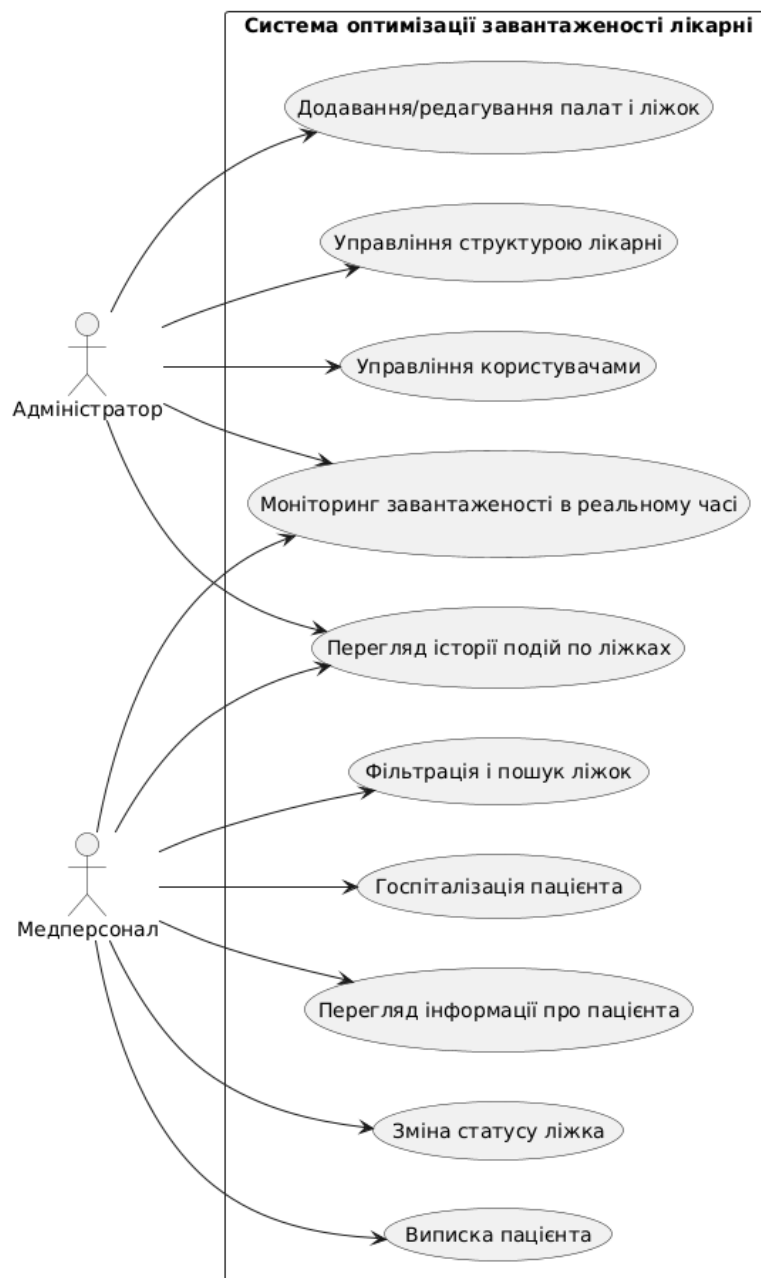


Рис. 2.1 – Діаграма варіантів використання

Опис діаграми варіантів використання наведено в таблицях 2.1 – 2.10:

Таблиця 2.1

Опис прецеденту «Управління структурою лікарні»

Назва варіанту використання	Управління структурою лікарні
Основні актори	Адміністратор
Короткий опис	Користувач з роллю адміністратора має змогу створювати та редагувати елементи структури лікарні: корпуси, поверхи, відділення. Це забезпечує основу для подальшого розміщення палат та ліжок.

Таблиця 2.2

Опис прецеденту «Додавання/редагування палат і ліжок»

Назва варіанту використання	Додавання/редагування палат і ліжок
Основні актори	Адміністратор
Короткий опис	Адміністратор може створювати нові палати в межах відділень, а також додавати ліжка з конкретними характеристиками. Можливе редагування наявних записів — зміна кількості, номерів, типів та параметрів.

Таблиця 2.3

Опис прецеденту «Управління користувачами»

Назва варіанту використання	Управління користувачами
Основні актори	Адміністратор
Короткий опис	Адміністратор може створювати облікові записи користувачів, призначати їм ролі та вказувати посаду, а також змінювати або видаляти наявних користувачів у системі.

Таблиця 2.4

Опис прецеденту «Моніторинг завантаженості в реальному часі»

Назва варіанту використання	Моніторинг завантаженості в реальному часі
Основні актори	Адміністратор, Медперсонал
Короткий опис	Усі користувачі мають доступ до дашборду, що відображає поточний стан ліжкового фонду в реальному часі.

Таблиця 2.5

Опис прецеденту «Госпіталізація пацієнта»

Назва варіанту використання	Госпіталізація пацієнта
Основні актори	Медперсонал
Короткий опис	Персонал додає нового пацієнта в систему та закріплює за ним доступне ліжко. Дані зберігаються, а статус ліжка автоматично змінюється на «Зайнято».

Таблиця 2.6

Опис прецеденту «Виписка пацієнта»

Назва варіанту використання	Виписка пацієнта
Основні актори	Медперсонал
Короткий опис	Після завершення лікування пацієнта персонал виконує випику, в результаті чого статус ліжка змінюється (наприклад, на «На прибиранні»). Подія фіксується в системі.

Таблиця 2.7

Опис прецеденту «Зміна статусу ліжка»

Назва варіанту використання	Зміна статусу ліжка
Основні актори	Медперсонал
Короткий опис	Користувач вручну змінює статус ліжка (наприклад, із «На прибиранні» на «Вільне»), що дозволяє відображати актуальний стан для подальшого використання.

Таблиця 2.8

Опис прецеденту «Перегляд інформації про пацієнта»

Назва варіанту використання	Перегляд інформації про пацієнта
Основні актори	Медперсонал
Короткий опис	Персонал має змогу переглядати інформацію про пацієнта, що перебуває на конкретному ліжку, зокрема ПІБ, дату госпіталізації, діагноз та інші дані.

Таблиця 2.9

Опис прецеденту «Перегляд історії подій по ліжках»

Назва варіанту використання	Перегляд історії подій по ліжках
Основні актори	Адміністратор, Медперсонал
Короткий опис	Система зберігає журнал змін по кожному ліжку: зміни статусів, події госпіталізації та виписки. Користувачі можуть переглядати цю історію для аналізу та аудиту.

Таблиця 2.10

Опис прецеденту «Фільтрація і пошук ліжок»

Назва варіанту використання	Фільтрація і пошук ліжок
Основні актори	Медперсонал
Короткий опис	Персонал може шукати вільні ліжка за заданими параметрами — тип палати, гендерна політика, статус тощо. Це дозволяє швидше знаходити підходяще місце для госпіталізації.

Основна мета створення вебдодатку для оптимізації процесів управління завантаженістю лікарні полягає у впровадженні сучасної цифрової системи, яка дозволяє організовано та прозоро відображати, контролювати та редагувати завантаження лікарняних відділень. Йдеться про автоматизацію ключових адміністративних процесів, зокрема моніторинг наявності вільних ліжок, призначення пацієнтів, зміну статусів місць, ведення історії подій, формування

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

структури закладу тощо. Такий підхід дозволяє зменшити навантаження на персонал, усунути людський фактор у веденні обліку та забезпечити своєчасну реакцію на зміну ситуації в лікарні.

Розроблювана система передбачає підтримку чіткої ієрархії об'єктів – від корпусів і поверхів до відділень, палат і ліжок. Кожне ліжко може мати статус: вільне, зайняте, на прибиранні, зарезервоване або недоступне. Завдяки цьому медичний персонал зможе швидко орієнтуватися в ситуації та приймати обґрунтовані рішення. При госпіталізації працівник знаходить відповідне ліжко, заповнює дані пацієнта, і система фіксує цю подію, змінюючи статус відповідного ресурсу.

Вебдодаток має інтуїтивно зрозумілий інтерфейс, що забезпечує доступність для користувачів із різним рівнем підготовки. Підтримується рольова модель доступу. Адміністратор має змогу керувати користувачами, структурою лікарні та переглядати агреговану статистику, а персонал — виконувати щоденні операції госпіталізації, виписки, перегляду інформації та зміни статусів.

Система повинна бути стабільною, масштабованою та продуктивною. Інтерфейс адаптивний — підтримує роботу як на настільних комп'ютерах, так і на мобільних пристроях. Усі дані зберігаються централізовано, що дозволяє забезпечити цілісність і безперервність роботи установи.

Аналіз функціональних вимог дозволив виділити наступні сутності, що забезпечать програмну реалізацію додатку. На (рис. 2.2.) наведено діаграму класів, яка використовується для його роботи.

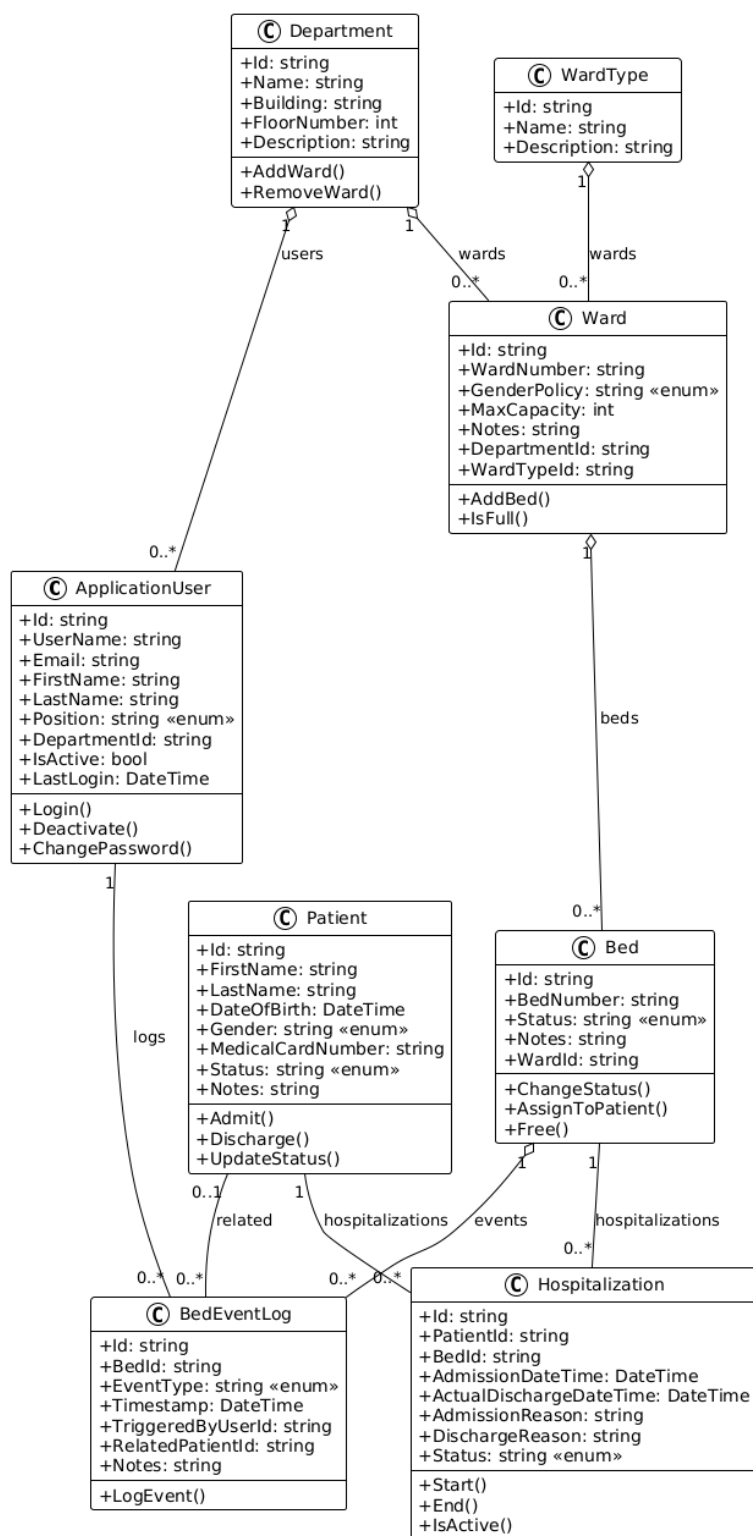


Рис. 2.2 – Діаграма класів

Опишемо діаграму класів (рис. 2.2).

Система управління завантаженістю лікарні побудована на основі об'єктно-орієнтованого підходу [13]. Центральним елементом є клас ApplicationUser, що розширює стандартний механізм автентифікації ASP.NET Core Identity. Він містить додаткові поля, зокрема FirstName, LastName, Position,

DepartmentId, а також інформацію про активність користувача — дату останнього входу (LastLogin) та статус облікового запису (IsActive). Користувачі системи можуть мати різні ролі, такі як адміністратор або медичний працівник, що визначається через поле Position. Цей клас реалізує методи Login(), Deactivate() та ChangePassword() для управління обліковим записом.

Клас Department описує організаційні підрозділи лікарні. Він включає назву відділення, поверх, будівлю та опис. До кожного відділення може належати кілька користувачів (ApplicationUser) та кілька палат (Ward). Клас реалізує методи AddWard() і RemoveWard() для керування структурою лікарні.

Палати описуються класом Ward, який містить номер палати (WardNumber), політику щодо розподілу за статтю (GenderPolicy), максимальну кількість ліжок (MaxCapacity) та додаткові нотатки. Палати належать певному відділенню (Department) та можуть мати визначений тип (WardType). Функціональність класу включає методи AddBed() та IsFull() для керування розміщенням пацієнтів.

Клас Bed репрезентує конкретне ліжко в палаті. Основними атрибутами є номер ліжка (BedNumber), статус (Status, наприклад «вільне», «зайняте», «прибирається») та нотатки. Ліжка прив'язані до палат і можуть мати історію подій. Основні методи: ChangeStatus(), AssignToPatient() і Free() для оперативного управління.

Пацієнти представлені класом Patient, який включає особисту інформацію (ім'я, дата народження, стать), медичний статус (Status) та медичний номер. Пацієнт може мати кілька госпіталізацій (Hospitalization). Методи Admit(), Discharge() і UpdateStatus() дозволяють оновлювати стан пацієнта в процесі лікування.

Госпіталізація відображена класом Hospitalization, який пов'язаний одночасно з пацієнтом і ліжком. Вона включає дати госпіталізації та виписки, причини, статус, а також методи Start(), End() і IsActive() для управління життєвим циклом госпіталізації.

Журнал подій ліжка реалізований класом BedEventLog. Він містить

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

інформацію про тип події (наприклад, зміна статусу), дату й час, користувача, який ініціював зміну, та (опціонально) пацієнта, до якого відноситься подія. Метод LogEvent() дозволяє фіксувати дії персоналу та відстежувати зміни у використанні ліжка.

Завдяки чітким зв'язкам між класами та визначеним методам, система забезпечує повний цикл управління лікарняним ресурсом — від реєстрації користувача й формування структури лікарні до госпіталізації пацієнта та журналювання дій персоналу.

2.2 Розробка бази даних системи

Для збереження структурованих даних та забезпечення цілісності інформації у вебдодатку використовується SQL Server — потужна реляційна система керування базами даних, яка надає інструменти для високопродуктивної обробки транзакцій, складного запитування, захисту даних і масштабованості проєкту [14]. Ця СУБД інтегрується з .NET-середовищем безпосередньо, що забезпечує ефективну взаємодію між застосунком на ASP.NET та базою даних [15].

Архітектура бази даних проєкту сформована на основі принципів нормалізації, що дозволяє уникати надлишковості інформації, зменшити ймовірність логічних помилок та забезпечити гнучкість структури при майбутньому розширенні [16]. Всього реалізовано 8 основних таблиць, які охоплюють ключові сутності системи:

- ApplicationUsers — таблиця, що зберігає дані користувачів (медперсонал, адміністратори);
- Departments — описує структурні підрозділи медичного закладу (відділення);
- Wards — репрезентує палати, які фізично належать до певного відділення;
- Beds — кожне ліжко має статус, розташування і призначення;
- Patients — інформація про пацієнтів, включаючи демографічні та

медичні поля;

– Hospitalizations — фіксує госпіталізації, що зв'язують пацієнтів із ліжками;

– BedEventLogs — журнал змін стану ліжок із позначкою часу та ініціатором дії;

– WardTypes — класифікує палати за функціональним призначенням (реанімація, загальна тощо).

Опис кожної з таблиць бази даних наведений у таблицях(2.11 – 2.18):

Таблиця 2.11

Опис таблиці ApplicationUsers

Поле	Тип	Призначення
Id	uniqueidentifier	Унікальний ідентифікатор користувача
FirstName	nvarchar	Ім'я користувача
LastName	nvarchar	Прізвище користувача
Email	nvarchar	Електронна пошта користувача
PasswordHash	nvarchar	Хеш пароля користувача
Position	int	Посада (enum: лікар, медсестра, адміністратор)
DepartmentId	uniqueidentifier	Зовнішній ключ на таблицю Departments
LastLogin	datetime	Дата та час останнього входу
IsActive	bit	Статус активності користувача

Таблиця 2.12

Опис таблиці Departments

Поле	Тип	Призначення
Id	uniqueidentifier	Унікальний ідентифікатор відділення
Name	nvarchar	Назва відділення
Building	nvarchar	Назва або номер корпусу
FloorNumber	int	Поверх
Description	nvarchar	Додатковий опис

Таблиця 2.13

Опис таблиці Wards

Поле	Тип	Призначення
Id	uniqueidentifier	Унікальний ідентифікатор палати
WardNumber	nvarchar	Номер палати
GenderPolicy	int	Політика розміщення за статтю (enum)
MaxCapacity	int	Максимальна кількість пацієнтів
DepartmentId	uniqueidentifier	Зовнішній ключ на таблицю Departments
WardTypeId	uniqueidentifier	Зовнішній ключ на таблицю WardTypes
Notes	nvarchar	Коментарі або особливі умови

Таблиця 2.14

Опис таблиці Beds

Поле	Тип	Призначення
Id	uniqueidentifier	Унікальний ідентифікатор ліжка
BedNumber	nvarchar	Номер ліжка
WardId	uniqueidentifier	Зовнішній ключ на таблицю Wards
Status	int	Статус ліжка (enum: вільне, зайняте)
Notes	nvarchar	Додаткові коментарі

Таблиця 2.15

Опис таблиці Patients

Поле	Тип	Призначення
Id	uniqueidentifier	Унікальний ідентифікатор пацієнта
FirstName	nvarchar	Ім'я пацієнта
LastName	nvarchar	Прізвище пацієнта
DateOfBirth	date	Дата народження
Gender	int	Стать (enum: чоловіча, жіноча)
MedicalCardNumber	nvarchar	Номер медичної картки
Status	int	Поточний статус (enum: активний, виписаний)
Notes	nvarchar	Додаткова медична інформація

Таблиця 2.16

Опис таблиці Hospitalizations

Поле	Тип	Призначення
Id	uniqueidentifier	Унікальний ідентифікатор госпіталізації
PatientId	uniqueidentifier	Зовнішній ключ на таблицю Patients
BedId	uniqueidentifier	Зовнішній ключ на таблицю Beds
AdmissionDateTime	datetime	Дата і час госпіталізації
ActualDischargeDateTime	datetime	Дата виписки (може бути NULL)
AdmissionReason	nvarchar	Причина госпіталізації
DischargeReason	nvarchar	Причина виписки
Status	int	Статус перебування (enum)

Таблиця 2.17

Опис таблиці BedEventLogs

Поле	Тип	Призначення
Id	uniqueidentifier	Унікальний ідентифікатор запису
BedId	uniqueidentifier	Зовнішній ключ на таблицю Beds
PatientId	uniqueidentifier	Зовнішній ключ на таблицю Patients (може бути NULL)
TriggeredBy	uniqueidentifier	Зовнішній ключ на ApplicationUsers
EventType	int	Тип події (enum)
Timestamp	datetime	Час виконання події
Notes	nvarchar	Додаткові коментарі

Опис таблиці WardTypes

Поле	Тип	Призначення
Id	uniqueidentifier	Унікальний ідентифікатор типу палати
Name	nvarchar	Назва типу (наприклад: інтенсивна терапія)
Description	nvarchar	Опис функціоналу чи призначення

Ключовою ознакою моделі є реляційність: наприклад, Hospitalizations пов'язує таблиці Beds і Patients, встановлюючи фактичну прив'язку пацієнта до ліжка в межах певного періоду. У свою чергу, Beds мають зовнішній ключ на Wards, а Wards — на Departments. Це дозволяє чітко відстежити, в якій палаті, на якому поверсі й у якому відділенні перебуває пацієнт у конкретний момент.

Таблиця ApplicationUsers містить додаткові атрибути, такі як Position, IsActive та LastLogin, що дозволяють відслідковувати активність персоналу, реалізувати доступ на основі ролей і керувати дозволами. Для уникнення дублювання акаунтів впроваджено унікальні обмеження на поля Email та Username, а також контроль заповнення обов'язкових полів (NOT NULL).

Таблиця BedEventLogs має зв'язки одразу з кількома сутностями — зокрема, з ApplicationUser (хто ініціював зміну) і з Patient (якщо подія стосувалась конкретного пацієнта). Це дозволяє вести докладний аудит усіх змін у системі, включаючи вивільнення, блокування чи зайняття ліжка, що є критично важливим для забезпечення прозорості обліку ресурсів лікарні.

Для підтримки коректної класифікації даних в системі використовуються перерахування (enum), наприклад: PositionType, WardGenderPolicy, BedStatus, PatientStatus, HospitalizationStatus, BedEventType. Вони дозволяють зберігати значення у вигляді зрозумілих кодів, уникати введення некоректних або довільних значень, а також забезпечують зручність при обробці даних у клієнтській частині додатку.

Система також підтримує рольову модель доступу, яка визначається полем Position у таблиці користувачів. Вона може бути легко масштабована для додавання ролей типу «медсестра», «лікар», «завідувач» тощо, кожна з яких

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

матиме свій набір доступних операцій у межах застосунку.

У частині контролю даних у часі реалізовано поля типу CreatedAt, AdmissionDateTime, ActualDischargeDateTime, Timestamp, що дозволяє будувати історичну хронологію змін, аналізувати завантаженість відділень у різні періоди та формувати звітність.

Створена модель бази даних легко розширюється. У подальшому її можливо доповнити такими таблицями, як Notifications (система сповіщень для персоналу), PatientHistory (історія хвороби), Permissions (деталізація доступу), а також Reports для збереження агрегованих статистичних звітів щодо завантаженості по відділеннях.

Таким чином, розроблена структура бази даних забезпечує повноцінне логічне представлення внутрішніх процесів у лікарні, підтримує масштабованість, узгодженість даних і дозволяє надійно зберігати інформацію про госпіталізації, пацієнтів і ресурси в умовах багатокористувацького середовища.

2.3 Проєктування та реалізація алгоритмів роботи системи

Для ефективного функціонування вебдодатку управління завантаженістю лікарні було розроблено набір алгоритмів, які охоплюють ключові функціональні процеси: автентифікацію користувачів, створення та редагування структури лікарні, реєстрацію пацієнтів, госпіталізацію, управління палатами та ліжками, а також логування подій [17]. Основна логіка реалізована на стороні сервера за допомогою технології ASP.NET Core Web API, з використанням C# як основної мови програмування [18]. Збереження даних здійснюється через SQL Server, що забезпечує надійність та узгодженість у роботі з великою кількістю пов'язаних сутностей [19].

Кожна дія користувача ініціює відповідний HTTP-запит до API [20]. На сервері обробку запитів забезпечують контролери, які здійснюють перевірку прав доступу (на основі JWT-токена), валідацію вхідних даних (через механізм Data Annotations і FluentValidation), а також взаємодію з базою даних через ORM Entity Framework Core. Усі зміни з боку клієнта супроводжуються аудитом:

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

записується, хто, коли і яку дію виконав. Було реалізовано захист від несанкціонованого доступу, перевірку відповідності даних (наприклад, перевірка наявності вільних ліжок у палаті перед госпіталізацією), а також обробку типових помилок (404, 400, 403, 500) [21]. Для відображення основних сценаріїв роботи користувача було створено діаграму активності (рис. 2.3), що ілюструють логіку основних процесів.

Одним із критично важливих процесів є алгоритм госпіталізації пацієнта, що складається з кількох послідовних етапів [22]. На першому кроці уповноважений працівник (наприклад, заввідділенням або адміністратор) у графічному інтерфейсі Next.js обирає відповідного пацієнта зі списку або створює нового. Далі здійснюється вибір ліжка в конкретній палаті — зазвичай це інтерактивна форма з візуалізацією стану доступності кожного місця. У формі вказується причина госпіталізації, бажана дата і час розміщення та, за потреби, супровідні примітки. Клієнтська частина на базі React + TypeScript виконує валідацію форми: перевіряється заповненість обов'язкових полів, правильність дати (не в минулому), а також узгодженість даних (наприклад, чи вільне обране ліжко) [23].

Після проходження валідації дані надсилаються до API за допомогою POST-запиту. Серверна логіка перевіряє токен доступу, статус пацієнта, наявність вибраного ліжка, а також відсутність активної госпіталізації. Якщо перевірки проходять успішно, створюється новий запис у таблиці Hospitalizations, а також оновлюється статус ліжка в таблиці Beds на «Зайнято». Паралельно до таблиці BedEventLogs записується подія типу ADMISSION, що містить точний час події, ID користувача, який ініціював дію, та зв'язок із відповідним пацієнтом і ліжком.

У разі виникнення помилки (наприклад, ліжко вже зайняте іншим пацієнтом, або вказаний ID пацієнта не існує) API повертає повідомлення про помилку з відповідним HTTP-кодом і деталями. І навпаки — у випадку успішної операції клієнтська частина інформує користувача про успішну госпіталізацію і оновлює стан інтерфейсу, відображаючи нового пацієнта в контексті обраної

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

палати.

Завдяки такому підходу система гарантує надійність і точність роботи з медичними даними, захищає їх від несанкціонованих змін і забезпечує повну відтворюваність дій персоналу. Це, в свою чергу, є критичним аспектом для ефективного управління завантаженістю лікарні та підвищення якості обслуговування пацієнтів.

Діаграма активності: Процес госпіталізації пацієнта

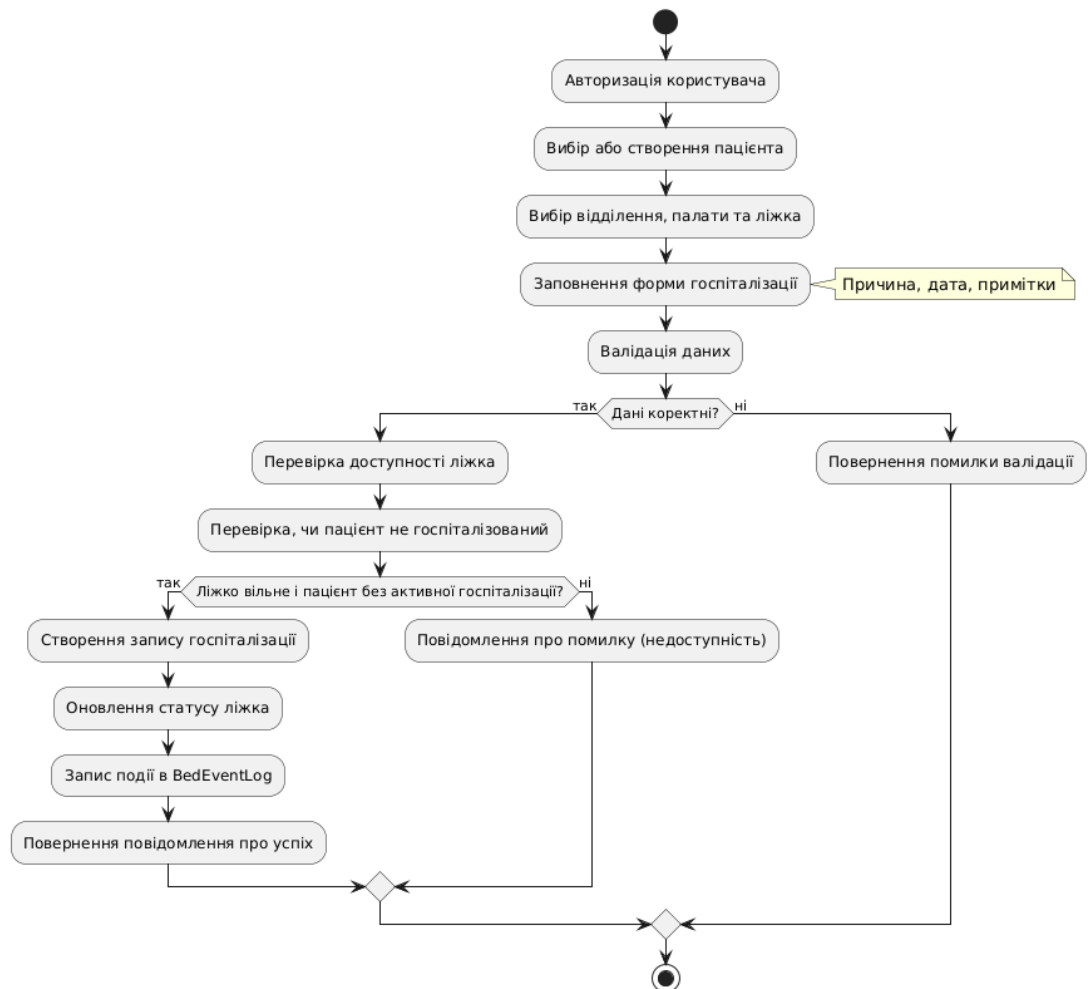


Рис. 2.3 – Діаграма активності

2.4 Реалізація функціоналу системи платформи

Розглянемо основні операції з управління даними в системі, такі як адміністрування персоналу, пацієнтів та їх госпіталізацій.

Функція додавання співробітника (для адміністраторів):

```
public async Task<StaffResponse> Handle(Command request, CancellationToken cancellationToken)
```

Змн.	Арк.	№ докум.	Підпис	Дата

```

    {
        var userExists = await _userManager.FindByEmailAsync(request.Email);
        if (userExists != null)
        {
            throw new InvalidOperationException("Staff with this email
already exists.");
        }

        var department = await _dbContext.Departments.FirstOrDefaultAsync(d
=> d.Id == request.DepartmentId, cancellationToken);
        if (department == null)
        {
            throw new KeyNotFoundException("Department not found.");
        }

        var user = new ApplicationUser
        {
            Email = request.Email,
            UserName = request.Email,
            FirstName = request.FirstName,
            LastName = request.LastName,
            Position = request.Position,
            DepartmentId = request.DepartmentId,
            IsActive = request.IsActive,
            EmailConfirmed = true
        };

        var result = await _userManager.CreateAsync(user, request.Password);
        if (!result.Succeeded)
        {
            throw new InvalidOperationException("Staff registration failed:
" + string.Join(", ", result.Errors.Select(e => e.Description)));
        }

        await _userManager.AddToRoleAsync(user, Roles.Staff);

        return new StaffResponse
        {
            Id = user.Id,
            FirstName = user.FirstName,
            LastName = user.LastName,
            Email = user.Email,
            Position = user.Position,
            DepartmentId = user.DepartmentId,
            DepartmentName = department.Name,
            IsActive = user.IsActive
        };
    }

```

Ця функція дає можливість адміністратору додати нового співробітника до системи. Перед створенням запису виконується валідація: перевіряється, чи всі

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

обов'язкові поля заповнені, чи коректний формат email, та чи не існує вже користувача з такою поштою.

Функція редагування даних співробітника:

```
public async Task<StaffResponse> Handle(Command request, CancellationToken
cancellation_token)
{
    var staff = await _dbContext.Users.FirstOrDefaultAsync(s => s.Id ==
request.Id, cancellation_token);
    if (staff == null)
    {
        throw new KeyNotFoundException("Staff not found.");
    }

    staff.FirstName = request.FirstName;
    staff.LastName = request.LastName;
    staff.Position = request.Position;
    staff.DepartmentId = request.DepartmentId;
    staff.IsActive = request.IsActive;

    _dbContext.Update(staff);
    await _dbContext.SaveChangesAsync(cancellation_token);

    return new StaffResponse
    {
        Id = staff.Id,
        FirstName = staff.FirstName,
        LastName = staff.LastName,
        Email = staff.Email!,
        Position = staff.Position,
        DepartmentId = staff.DepartmentId,
        DepartmentName = staff.Department?.Name,
        IsActive = staff.IsActive,
        LastLogin = staff.LastLogin
    };
}
```

Дана функція дозволяє адміністратору редагувати існуючий профіль співробітника. Система знаходить користувача за його ID та оновлює відповідні поля в базі даних.

Функція додавання пацієнта:

```
public async Task<PatientResponse> Handle(Command request, CancellationToken
cancellation_token)
{
    if (!string.IsNullOrEmpty(request.MedicalCardNumber) &&
        await _dbContext.Patients.AnyAsync(p => p.MedicalCardNumber ==
request.MedicalCardNumber, cancellation_token))
    {
```

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        throw new InvalidOperationException("A patient with the same
medical card number already exists.");
    }

    var patient = new Patient
    {
        Id = Guid.NewGuid(),
        FirstName = request.FirstName,
        LastName = request.LastName,
        DateOfBirth = request.DateOfBirth,
        Gender = request.Gender,
        PhoneNumber = request.PhoneNumber,
        MedicalCardNumber = request.MedicalCardNumber,
        AdmissionDiagnosis = request.AdmissionDiagnosis,
        RequiresIsolation = request.RequiresIsolation,
        Status = PatientStatus.Registered,
        Notes = request.Notes
    };

    _dbContext.Add(patient);
    await _dbContext.SaveChangesAsync(cancellationToken);

    return new PatientResponse
    {
        Id = patient.Id,
        FirstName = patient.FirstName,
        LastName = patient.LastName,
        DateOfBirth = patient.DateOfBirth,
        Gender = patient.Gender,
        PhoneNumber = patient.PhoneNumber,
        MedicalCardNumber = patient.MedicalCardNumber,
        AdmissionDiagnosis = patient.AdmissionDiagnosis,
        RequiresIsolation = patient.RequiresIsolation,
        Status = patient.Status,
        Notes = patient.Notes
    };
}

```

Ця функція надає можливість медичному персоналу додавати нових пацієнтів. Перед збереженням даних виконується валідація на коректність заповнення полів.

Функція госпіталізації пацієнта:

```

public async Task<HospitalizationResponse> Handle(Command request,
Cancellation token cancellationToken)
{
    var patient = await _dbContext.Patients.FirstOrDefaultAsync(p =>
p.Id == request.PatientId, cancellationToken);
    if (patient == null)
    {

```

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        throw new KeyNotFoundException("Patient not found.");
    }

    var bed = await _dbContext.Beds
        .Include(b => b.Ward)
        .ThenInclude(w => w.Department)
        .FirstOrDefaultAsync(b => b.Id == request.BedId,
cancellationTokens);

    if (bed == null)
    {
        throw new KeyNotFoundException("Bed not found.");
    }

    if (bed.Status != BedStatus.Available)
    {
        throw new InvalidOperationException($"Bed is not available.
Current status: {bed.Status}");
    }

    var hospitalization = new Hospitalization
    {
        Id = Guid.NewGuid(),
        PatientId = request.PatientId,
        BedId = request.BedId,
        AdmissionDateTime = request.AdmissionDateTime,
        PlannedDischargeDateTime = request.PlannedDischargeDateTime,
        AdmissionReason = request.AdmissionReason,
        Status = request.Status
    };

    bed.Status = BedStatus.Occupied;
    patient.Status = PatientStatus.Hospitalized;

    await _dbContext.Hospitalizations.AddAsync(hospitalization,
cancellationTokens);
    await _dbContext.SaveChangesAsync(cancellationTokens);

    return new HospitalizationResponse
    {
        Id = hospitalization.Id,
        PatientId = hospitalization.PatientId,
        PatientFullName = $"{patient.FirstName} {patient.LastName}",
        BedId = hospitalization.BedId,
        BedNumber = bed.BedNumber,
        WardNumber = bed.Ward.WardNumber,
        DepartmentName = bed.Ward.Department.Name,
        AdmissionDateTime = hospitalization.AdmissionDateTime,
        PlannedDischargeDateTime =
hospitalization.PlannedDischargeDateTime,

```

					ІІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        ActualDischargeDateTime =
hospitalization.ActualDischargeDateTime,
        AdmissionReason = hospitalization.AdmissionReason,
        DischargeReason = hospitalization.DischargeReason,
        Status = hospitalization.Status
    };
}

```

Ця функція дозволяє госпіталізувати пацієнта. Логіка на сервері перевіряє, чи є обране ліжко вільним (BedStatus.Available). Якщо так, створюється запис про госпіталізацію, статус пацієнта змінюється на Admitted, а статус ліжка — на Occupied. Якщо ліжко зайняте, система поверне помилку.

Функція виписки пацієнта:

```

public async Task<HospitalizationResponse> Handle(Command request,
Cancellation token cancellationToken)
{
    var hospitalization = await _dbContext.Hospitalizations
        .Include(h => h.Patient)
        .Include(h => h.Bed)
        .ThenInclude(b => b.Ward)
        .ThenInclude(w => w.Department)
        .FirstOrDefaultAsync(h => h.PatientId == request.PatientId &&
h.Status == HospitalizationStatus.Active, cancellationToken);

    if (hospitalization == null)
    {
        throw new KeyNotFoundException("Hospitalization not found.");
    }

    hospitalization.Status = HospitalizationStatus.Discharged;
    hospitalization.ActualDischargeDateTime =
request.ActualDischargeDateTime;
    hospitalization.DischargeReason = request.DischargeReason;

    hospitalization.Bed.Status = BedStatus.Cleaning;
    hospitalization.Patient.Status = PatientStatus.Discharged;

    _dbContext.Update(hospitalization);
    await _dbContext.SaveChangesAsync(cancellationToken);

    return new HospitalizationResponse
    {
        Id = hospitalization.Id,
        PatientId = hospitalization.PatientId,
        PatientFullName = $"{hospitalization.Patient.FirstName}
{hospitalization.Patient.LastName}",
        BedId = hospitalization.BedId,
        BedNumber = hospitalization.Bed.BedNumber,
    }
}

```

```

        WardNumber = hospitalization.Bed.Ward.WardNumber,
        DepartmentName = hospitalization.Bed.Ward.Department.Name,
        AdmissionDateTime = hospitalization.AdmissionDateTime,
        PlannedDischargeDateTime =
hospitalization.PlannedDischargeDateTime,
        ActualDischargeDateTime =
hospitalization.ActualDischargeDateTime,
        AdmissionReason = hospitalization.AdmissionReason,
        DischargeReason = hospitalization.DischargeReason,
        Status = hospitalization.Status
    };
}

```

Дана функція дозволяє оформити виписку пацієнта. Система знаходить його активну госпіталізацію, змінює її статус на Discharged, фіксує дату виписки та змінює статус ліжка на Cleaning, щоб сигналізувати про необхідність прибирання.

Функція видалення пацієнта:

```

public async Task<Unit> Handle(Command request, CancellationToken
cancellation_token)
{
    var patient = await _dbContext.Patients
        .Include(p => p.Hospitalizations)
        .FirstOrDefaultAsync(p => p.Id == request.Id,
cancellation_token);

    if (patient == null)
    {
        throw new KeyNotFoundException("Patient not found.");
    }

    if (patient.Hospitalizations.Any())
    {
        throw new InvalidOperationException("Cannot delete patient with
active or past hospitalizations.");
    }

    _dbContext.Remove(patient);
    await _dbContext.SaveChangesAsync(cancellation_token);

    return Unit.Value;
}

```

Дана функція дозволяє видалити пацієнта з системи. Використання цієї функції обмежене і можливе лише у випадках, коли пацієнт не має активних госпіталізацій.

Функція додавання ліжка (для адміністратора):

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

```

public async Task<BedResponse> Handle(Command request, CancellationToken
cancellationTokentoken)
{
    var ward = await _dbContext.Wards.FirstOrDefaultAsync(w => w.Id ==
request.WardId, cancellationTokentoken);
    if (ward == null)
    {
        throw new KeyNotFoundException("Ward not found.");
    }

    if (await _dbContext.Beds.AnyAsync(b =>
b.WardId == request.WardId && b.BedNumber == request.BedNumber,
cancellationTokentoken))
    {
        throw new InvalidOperationException("Bed with the same number
already exists in this ward.");
    }

    var bed = new Bed
    {
        Id = Guid.NewGuid(),
        WardId = request.WardId,
        BedNumber = request.BedNumber,
        Status = request.Status,
        Notes = request.Notes
    };

    _dbContext.Add(bed);
    await _dbContext.SaveChangesAsync(cancellationTokentoken);

    if (bed.Status != BedStatus.Available)
    {
        var bedEventLog = new BedEventLog
        {
            Id = Guid.NewGuid(),
            BedId = bed.Id,
            EventType = BedEventType.StatusManuallyChanged,
            Timestamp = DateTime.UtcNow,
            Notes = $"Initial bed status set to {bed.Status}"
        };

        _dbContext.BedEventLogs.Add(bedEventLog);
        await _dbContext.SaveChangesAsync(cancellationTokentoken);
    }

    return new BedResponse
    {
        Id = bed.Id,
        WardId = bed.WardId,
        WardNumber = ward.WardNumber,
        BedNumber = bed.BedNumber,
    }
}

```

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        Status = bed.Status,
        Notes = bed.Notes
    };
}

```

Ця функція надає адміністратору можливість додавати нові ліжка до структури лікарні, вказуючи їх номер, палату та початковий статус. Доступ до цієї функції обмежений роллю адміністратора.

Функція зміни статусу ліжка:

```

public async Task<BedResponse> Handle(Command request, CancellationToken
cancellation_token)
{
    var bed = await _dbContext.Beds
        .Include(b => b.Ward)
        .FirstOrDefaultAsync(b => b.Id == request.Id,
cancellation_token);

    if (bed == null)
    {
        throw new KeyNotFoundException("Bed not found.");
    }

    var previousStatus = bed.Status;

    bed.Status = request.Status;
    bed.Notes = request.Notes;

    _dbContext.Update(bed);

    var bedEventLog = new BedEventLog
    {
        Id = Guid.NewGuid(),
        BedId = bed.Id,
        EventType = request.EventType,
        Timestamp = DateTime.UtcNow,
        UserId = request.UserId,
        Notes = request.Notes ?? $"Status changed from {previousStatus}
to {request.Status}"
    };

    _dbContext.BedEventLogs.Add(bedEventLog);
    await _dbContext.SaveChangesAsync(cancellation_token);

    return new BedResponse
    {
        Id = bed.Id,
        WardId = bed.WardId,
        WardNumber = bed.Ward.WardNumber,
        BedNumber = bed.BedNumber,

```

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        Status = bed.Status,
        Notes = bed.Notes
    };
}

```

Дана функція дозволяє медичному персоналу оперативно змінювати статус ліжка (наприклад, з «Зайнято» на «Прибирання»). Кожна зміна статусу фіксується в журналі подій для подальшого аналізу.

Цей підхід, побудований на чіткому розділенні команд та запитів (CQRS [24]), дозволяє зручно та надійно керувати даними, що суттєво економить час медичного та адміністративного персоналу та покращує координацію процесів у лікарні.

Підсумовуючи, було створено діаграму компонентів (Рис. 2.4):

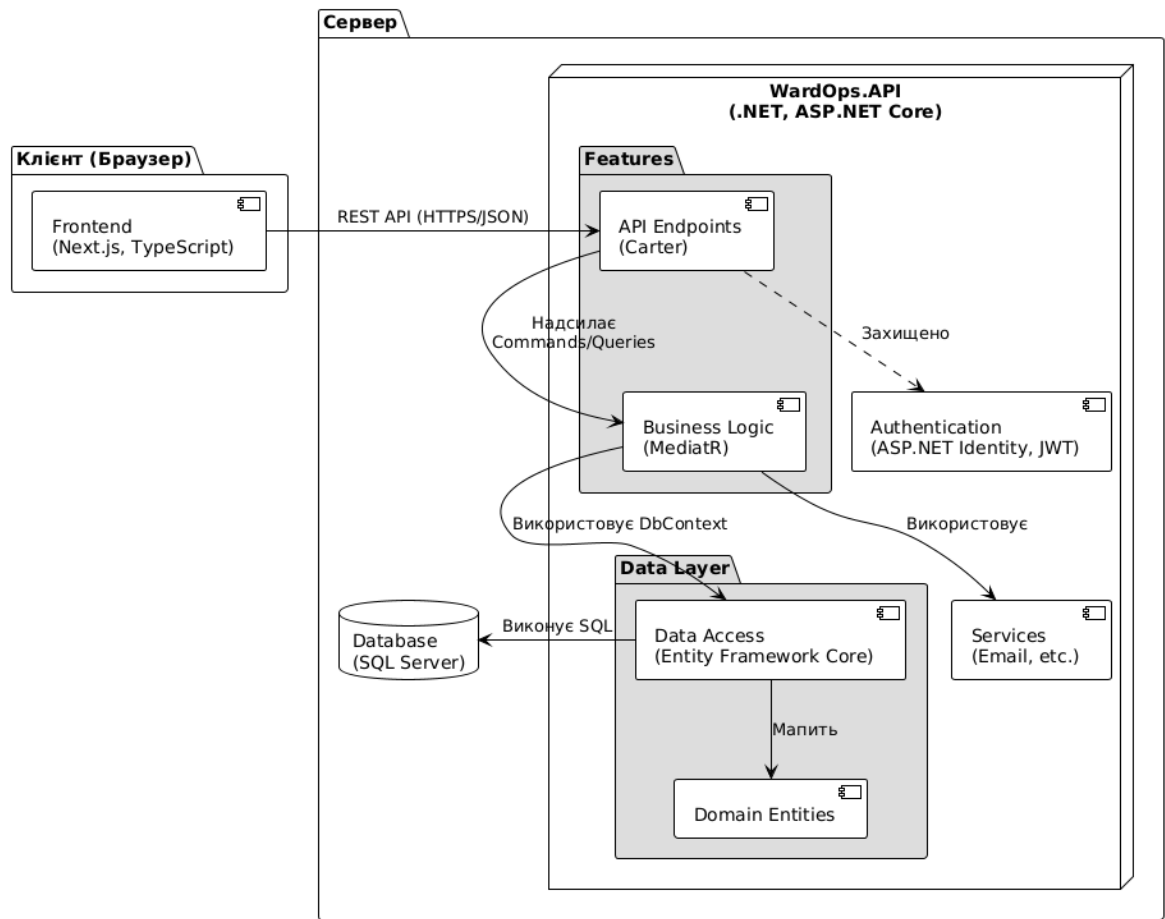


Рис. 2.4 – Діаграма компонентів

Висновки до другого розділу

У процесі проектування інформаційної системи для оптимізації процесів управління завантаженістю лікарні було проаналізовано всі основні сценарії

використання платформи відповідно до функціональних та нефункціональних вимог. На основі цього аналізу сформовано архітектурну модель системи, яка охоплює ключові компоненти: модулі автентифікації, управління структурою лікарні, ліжками, пацієнтами, госпіталізаціями, а також підсистему логування подій.

У межах проєктування було розроблено діаграму класів, яка описує зв'язки між сутностями системи, а також визначено основні методи, що реалізують бізнес-логіку на серверній стороні. Для кращого розуміння взаємодії між елементами системи створено діаграму варіантів використання та діаграму активності, які візуалізують поведінку користувачів і хід обробки основних процесів — таких як госпіталізація, зміна статусу ліжка, виписки пацієнта тощо.

Також було розроблено структуру бази даних, яка враховує потреби різних ролей користувачів системи, забезпечує цілісність і узгодженість інформації про пацієнтів, ліжка, відділення та історію подій. Проєктування бази даних передбачає гнучкість та можливість подальшого розширення, наприклад, за рахунок додавання підсистеми фінансового обліку, звітності або відгуків про якість обслуговування.

У розділі також було описано реалізацію алгоритмів, які забезпечують функціонування основних процесів платформи, включно з перевіркою доступності ліжка, фіксацією логів подій, обробкою госпіталізацій і виписок. Впроваджено механізми валідації, контролю прав доступу та захисту від помилкових дій користувачів, що сприяє стабільності та безпечності роботи системи.

Таким чином, у другому розділі було закладено технічну основу для реалізації інтелектуального вебдодатку, який дозволяє ефективно керувати лікарняною інфраструктурою, забезпечує гнучку підтримку бізнес-процесів і покращує якість прийняття рішень щодо розміщення пацієнтів.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ

3.1 Порядок встановлення та налаштування параметрів платформи

Для опису архітектури платформи було створено діаграму розгортання (рис. 3.1).

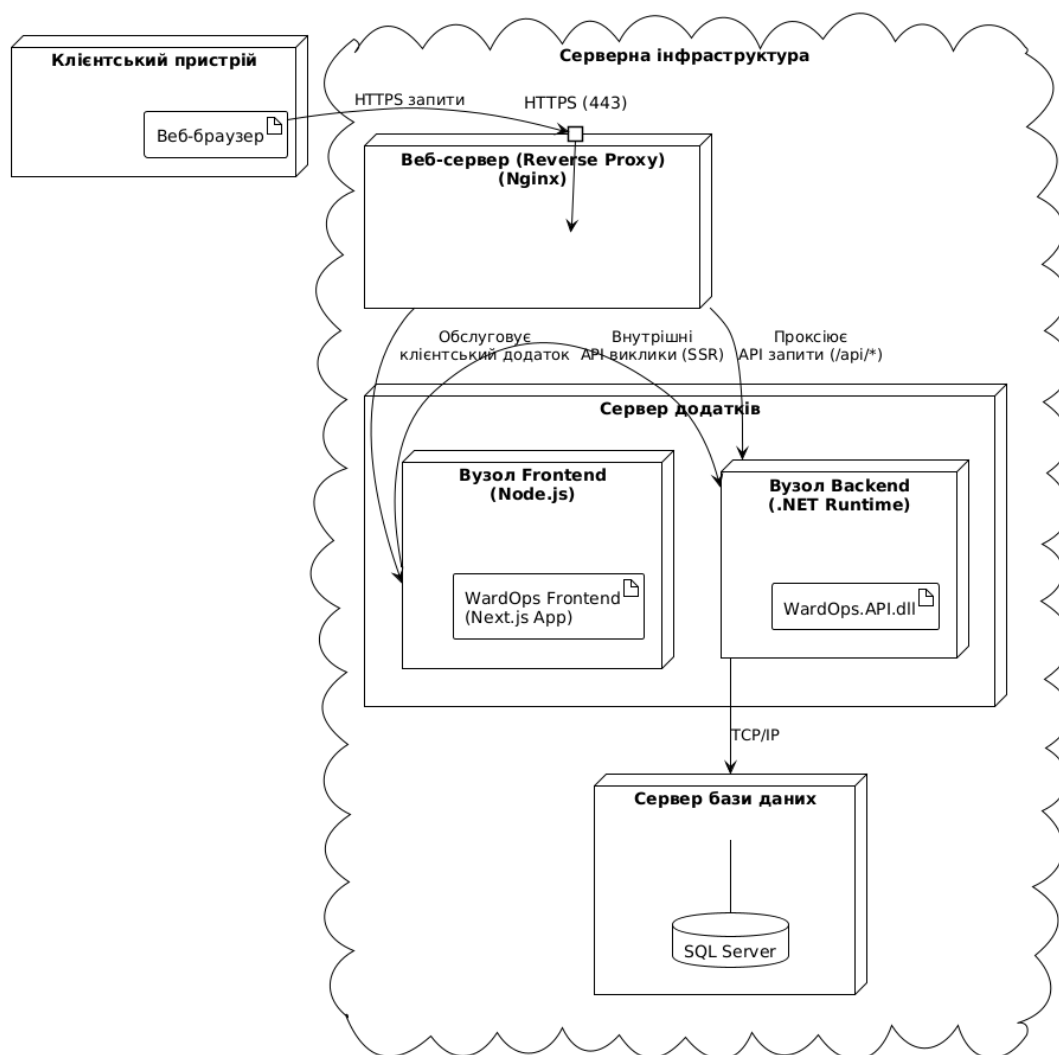


Рис. 3.1 – Діаграма розгортання

Frontend реалізовано з використанням фреймворку Next.js (React) та TypeScript. Встановлення клієнтської частини полягає у збірці проекту (`npm run build`), що створює оптимізований набір статичних файлів та серверних функцій у директорії `.next`. Цей додаток розгортається на сервері з встановленим Node.js. TypeScript забезпечує строгу типізацію, а React — побудову динамічного та компонентного інтерфейсу. Next.js відповідає за маршрутизацію, рендеринг на стороні сервера (SSR) та взаємодію з API бекенду [25].

Backend реалізовано за допомогою .NET (ASP.NET Core). Для його

налаштування необхідно мати встановлений .NET Runtime на сервері [26]. Проект компілюється у набір dll-файлів, які запускаються як самостійний процес. Зазвичай він розміщується за реверс-проксі (наприклад, Nginx), який направляє всі запити, що починаються з /api/, на .NET-додаток. Серверна частина відповідає за обробку бізнес-логіки, автентифікацію, авторизацію за ролями та виконання CRUD-операцій з базою даних [27].

База даних реалізована на основі SQL Server. Після встановлення СУБД потрібно створити базу даних. Її структура розгортається автоматично за допомогою механізму міграцій Entity Framework Core [28], який є частиною бекенд-проекту. Параметри підключення до бази даних налаштовуються у файлі конфігурації appsettings.json на сервері. Таблиці бази зберігають всю інформацію про структуру лікарні, персонал, пацієнтів, ліжка та госпіталізації.

Для повноцінної роботи системи налаштовується реверс-проксі (Nginx), який розподіляє запити між фронтом та бекендом. Конфігурація чутливих даних (рядки підключення до БД, секретні ключі для JWT) здійснюється через змінні середовища або захищені конфігураційні файли на сервері.

3.2 Структура інтерфейсу вебдодатку

Взаємодія користувачів відбувається через вебдодаток. Початкова сторінка - форма авторизації (рис. 3.2), на якій, користувач може авторизуватися в системі, ввівши свої облікові дані.

Рис. 3.2 – Форма логіну

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

На головній сторінці, Дашборді (рис. 3.3), користувач може побачити ключові показники роботи лікарні: загальну кількість пацієнтів, завантаженість ліжкового фонду та кількість відділень. Також тут відображається інформація про поточного користувача.

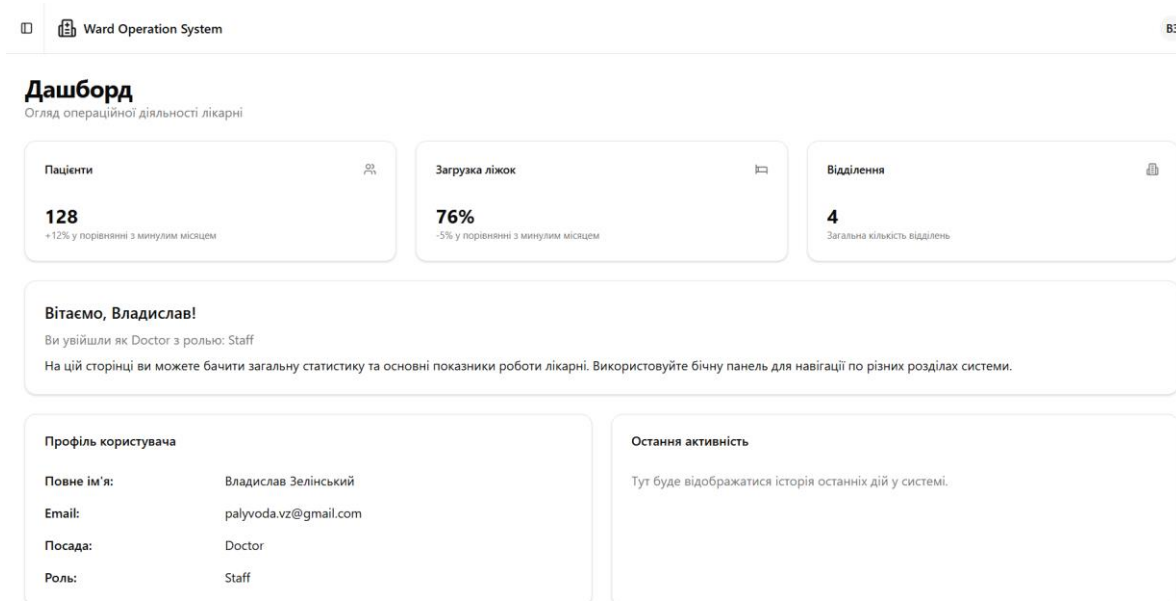


Рис. 3.3 – Дашборд

Для швидкого доступу до основних функцій системи реалізовано бічну панель навігації (рис. 3.4). Вона є ієрархічною та відображає розділи, доступні користувачеві відповідно до його ролі (звичайний користувач або адміністратор).

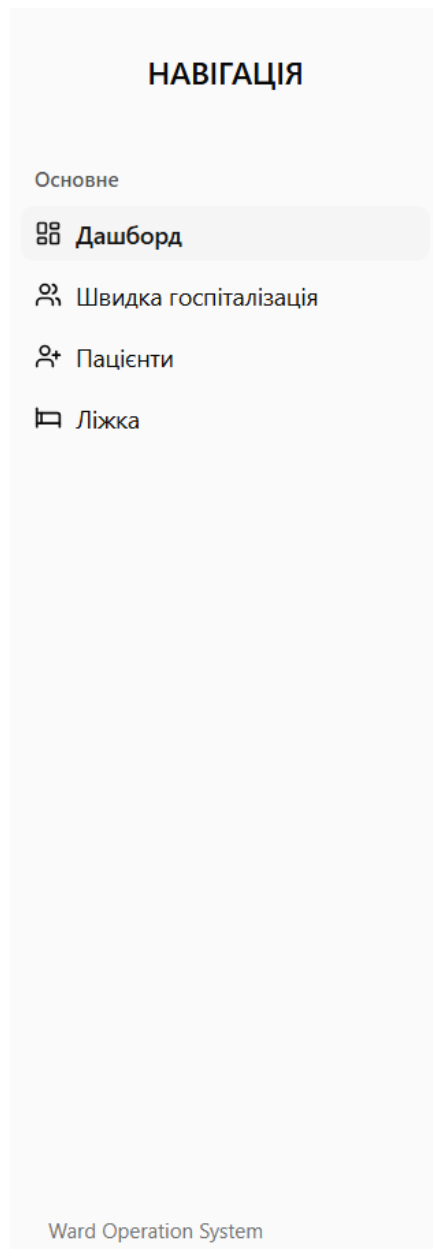


Рис. 3.4 – Навігаційна панель

Користувачі можуть переглядати списки пацієнтів та госпіталізацій у відповідних вкладках на сторінці «Управління пацієнтами» (рис. 3.5). Інтерфейс містить інструменти для пошуку та фільтрації даних.

Ward Operation System

SA

Управління пацієнтами

Реєстрація, госпіталізація та ведення пацієнтів

Пацієнти

Госпіталізації

Виписки

Всі статуси

Пошук пацієнтів...

Додати пацієнта

Список пацієнтів

ПІБ	Вік	Стать	Медкарта	Телефон	Діагноз	Статус	Дії
Софія Яблонська	39 років	Жінка	MC97531	380987766555	Загострення бронхіальної астми	Госпіталізований	...
Григорій Сковорода	52 років	Чоловік	MC13579	380935556677	Негоспітальна пневмонія	Госпіталізований	...
Леся Українка	32 років	Жінка	MC09876	380973322111	Закритий перелом стенової кістки	Госпіталізований	...
Олександр Довженко	45 років	Чоловік	MC67890	380961112233	Гострий апендицит	Госпіталізований	...
Марія Примаченко	66 років	Жінка	MC54321	380957654321	Гіпертонічний криз	Госпіталізований	...
Василь Симоненко	60 років	Чоловік	MC12345	380951234567	Гострий інфаркт міокарда	Госпіталізований	...

Рис. 3.5 – Управління пацієнтами

Натиснувши кнопку «Додати пацієнта», відкриється модальне вікно з формою для реєстрації нового пацієнта в системі (рис. 3.6). Форма містить поля для введення особистих даних, медичної інформації та приміток.

Реєстрація нового пацієнта

Введіть дані пацієнта для реєстрації в системі

Ім'я

Іван

Прізвище

Іванов

Дата народження

02/17/1988

Стать

Чоловік

Телефон

+380982130000

Номер медкарти

MC001234

Діагноз при госпіталізації

Гострий інфаркт міокарда

Статус

Зареєстрований

☒

Потребує ізоляції

Позначте, якщо пацієнт потребує ізольованого розміщення

Примітки

Додаткова інформація...

Скасувати

Зареєструвати

Рис. 3.6 – Форма додавання пацієнта

При натисканні на запис пацієнта відкривається вікно з його детальною інформацією (рис. 3.7). Тут у режимі "тільки для читання" відображаються всі ключові дані: ПІБ, дата народження, стать, номер медичної карти та статус.

Софія Яблонська ×

Детальна інформація про пацієнта

- Статус:** Госпіталізований
- Дата народження:** 12.07.1985
- Стать:** Жіноча
- Телефон:** 380987766555
- Мед. картка №:** MC97531
- Діагноз:** Загострення бронхіальної астми
- Потребує ізоляції:** Ні
- Примітки:** Стан стабільний, продовжувати інгаляційну терапію.

Закрити

Рис. 3.7 – Вікно перегляду деталей про пацієнта

Система надає можливість редагувати дані існуючого пацієнта (рис. 3.8). У цій формі персонал може оновити особисту інформацію, статус або примітки.

Редагування пацієнта ×

Оновіть дані пацієнта

Ім'я: Софія **Прізвище:** Яблонська

Дата народження: 07/11/1985 **Стать:** Жінка

Телефон: 380987766555 **Номер медкарти:** MC97531

Діагноз при госпіталізації: Загострення бронхіальної астми

Статус: Госпіталізований

☐ **Потребує ізоляції**
Позначте, якщо пацієнт потребує ізольованого розміщення

Примітки: Стан стабільний, продовжувати інгаляційну терапію.

Скасувати Оновити

Рис. 3.8 – Форма редагування пацієнта

Можна переглянути усі поточні госпіталізації на вкладці «Госпіталізації» (рис. 3.9.). У таблиці відображається інформація про пацієнта, номер ліжка, дату та причину госпіталізації.

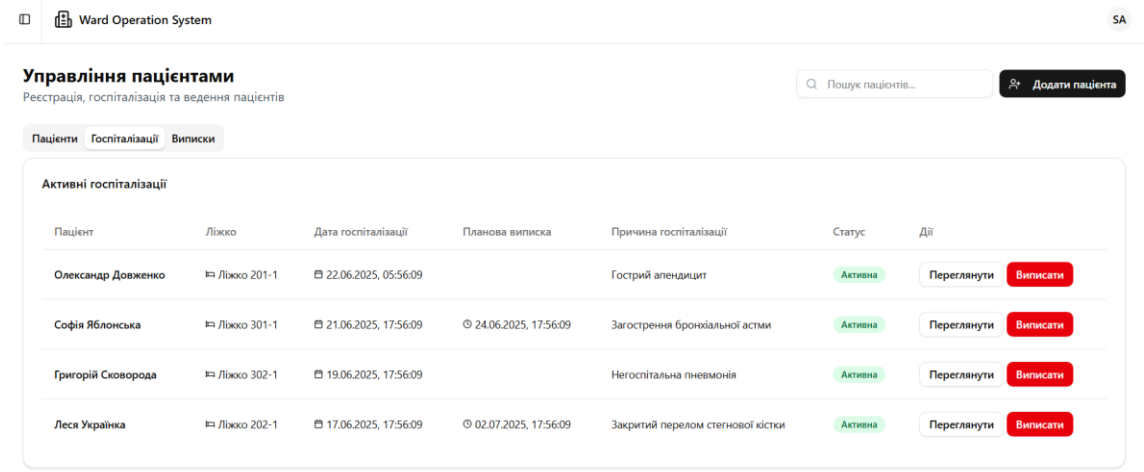


Рис. 3.9 – Вкладка «Госпіталізації»

Для кожної госпіталізації можна переглянути детальну інформацію (рис. 3.10), включаючи дані про пацієнта, ліжко, відділення, а також дати госпіталізації та фактичної виписки.

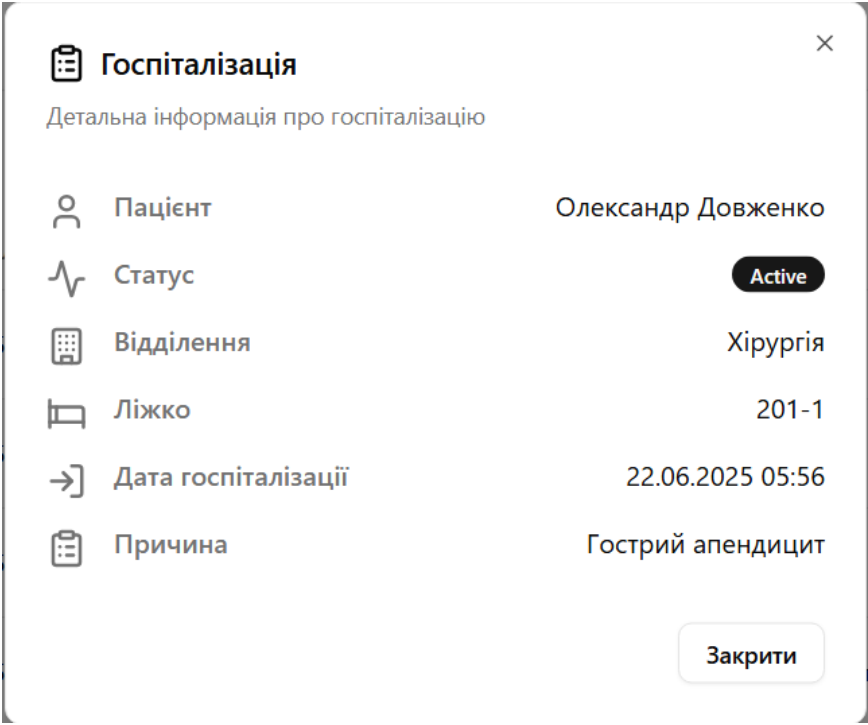


Рис. 3.10 – Вікно перегляду деталей госпіталізації

Для виписки пацієнта передбачена спеціальна форма (рис. 3.11), де необхідно вказати причину та точну дату й час виписки.

Виписка пацієнта

Виписати пацієнта Олександр Довженко з ліжка 201-1

Причина виписки

Пацієнт одужав

Дата та час виписки

06/24/2025 03:17 PM

Скасувати

Виписати пацієнта

Рис. 3.11 – Форма виписки пацієнта

На сторінці «Управління пацієнтами» вкладки «Виписки» (рис. 3.12) персонал може побачити історію виписок пацієнтів.

Ward Operation System

SA

Управління пацієнтами

Регістрація, госпіталізація та ведення пацієнтів

Пошук пацієнтів...

Додати пацієнта

Пацієнти

Госпіталізації

Виписки

Історія виписок

Пацієнт	Дата виписки	Тривалість лікування	Причина виписки	Статус
Григорій Сковорода	22.06.2025, 18:19:00	4 днів	Пацієнт одужав	Виписаний

Рис. 3.12 – Вкладка «Виписки»

На сторінці «Управління статусами ліжок» (рис. 3.13) медичний персонал може в реальному часі відстежувати стан ліжкового фонду свого відділення та оперативно реагувати на зміни.

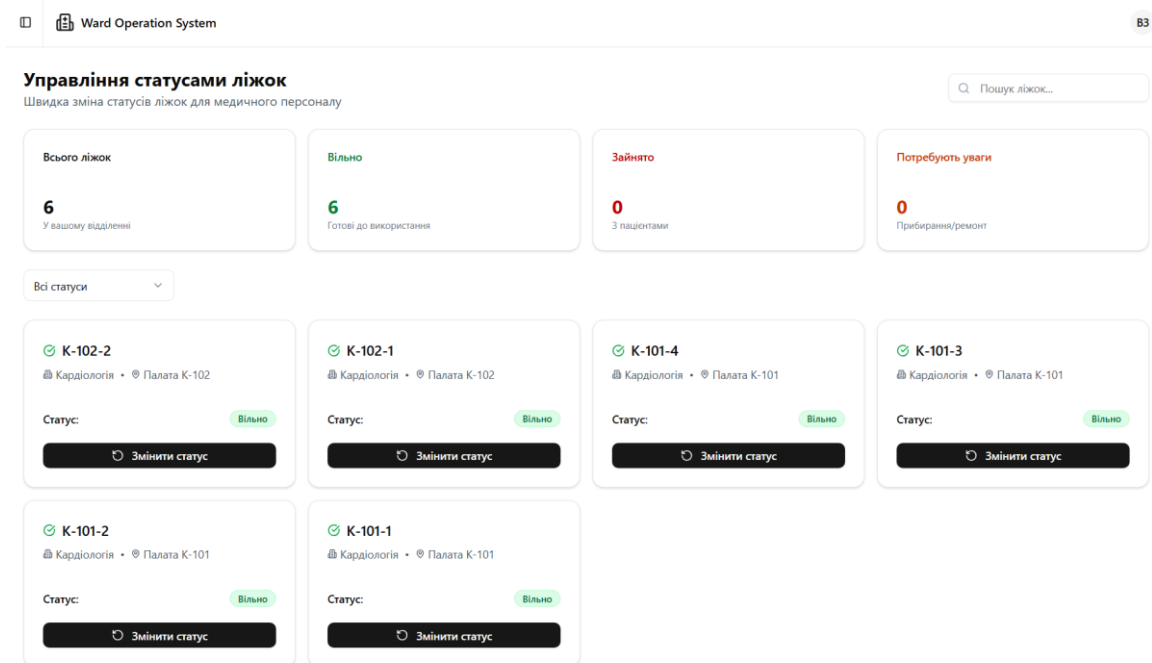


Рис. 3.13 – Управління статусами ліжок

При натисканні на ліжко відкривається форма для швидкої зміни його статусу (рис. 3.14), наприклад, з «Зайнято» на «Прибирання» після виписки пацієнта.

Рис. 3.14 – Форма зміни статусу ліжка

Сторінка «Швидка госпіталізація» (рис. 3.15) дозволяє максимально спростити процес прийому пацієнта, об'єднавши на одному екрані вибір пацієнта та вільного ліжка.

Ward Operation System

83

Швидка госпіталізація

Госпіталізація пацієнта на вільне ліжко

Вибір пацієнта

Існуючий пацієнт

Іван Іванов

MC001234 • 17.02.1988

Діагноз при госпіталізації

Гострий інфаркт міокарда

Потребує ізоляції:

Ні

Причина госпіталізації

Детальний опис причини...

Планова дата виписки

mm/dd/yyyy

Вибір ліжка

Доступні ліжка (6)

K-102-2

Палата K-102

Кардіологія

K-102-1

Палата K-102

Кардіологія

K-101-4

Палата K-101

Кардіологія

✓

Пацієнт обраний

⚠

Оберіть ліжко

Скасувати

Госпіталізувати

Рис. 3.15 – Швидка госпіталізація

Якщо користувач є адміністратором, йому буде доступна адмін-панель з розширеними можливостями (рис. 3.16), яка дозволить проводити операції, що доступні тільки для цієї ролі.

Адміністратор

Повний доступ

Головна

Дашборд

Пацієнти

Адміністрування

Структура лікарні

Управління ліжками

Управління персоналом

Ward Operation System

Рис. 3.16 – Адмін панель

Розділ «Структура лікарні» (рис. 3.17) надає адміністратору інструменти для управління ієрархією закладу: створенням та редагуванням відділень і палат.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

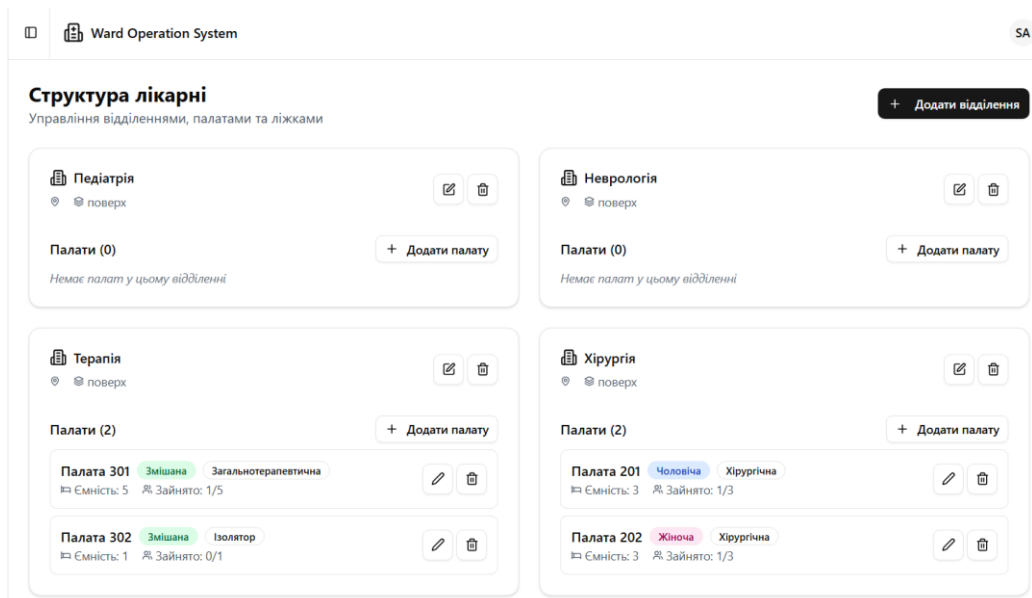


Рис. 3.17 – Структура лікарні

Адміністратор може додати нове відділення, вказавши його назву, корпус та поверх (рис. 3.18).

Створення нового відділення

Введіть дані для нового відділення

Назва відділення

Кардіологія

Корпус

А

Поверх

1

Опис

Опис відділення...

Скасувати

Створити

Рис. 3.18 – Форма додавання відділення

У межах існуючого відділення можна створювати нові палати, визначаючи їх номер, тип, гендерну політику та максимальну місткість (рис. 3.19).

Форма додавання ліжка (рис. 3.21) дозволяє адміністратору створити нове ліжко, прив'язавши його до конкретної палати.

Додавання нового ліжка

Номер ліжка: Наприклад, 101-A

Відділення: Оберіть відділення

Палата: Оберіть палату

Статус: Вільно

Примітки: Додаткова інформація...

Скасувати Додати ліжко

Рис. 3.21 – Форма додавання ліжка

Розділ «Управління персоналом» (рис. 3.22) дозволяє адміністратору керувати обліковими записами всіх співробітників, переглядати їх дані та змінювати статус.

Ward Operation System SA

Управління персоналом
Додавання, редагування та управління співробітниками

Пошук співробітників... Всі відділення Додати співробітника

Співробітники По відділеннях

Список співробітників (7)

Співробітник	Посада	Відділення	Статус	Останній вхід	Дії
ОП Олена Петренко olena.petrenko@wardops.com	Лікар	Кардіологія	Неактивний	Ніколи	✎
Ю Іван Франко ivan.franko@wardops.com	Лікар	Хірургія	Неактивний	Ніколи	✎
МК Михайло Коваленко mykhailo.kovalenko@wardops.com	Лікар	Терапія	Неактивний	Ніколи	✎
ГШ Ганна Шевченко hanna.shevchenko@wardops.com	Медсестра	Кардіологія	Неактивний	Ніколи	✎
СМ Сергій Мельник serhiy.melnyk@wardops.com	Медсестра	Хірургія	Неактивний	Ніколи	✎
КБ Катерина Бондаренко kateryna.bondarenko@wardops.com	Ресстратор	Терапія	Неактивний	Ніколи	✎
ВЗ Владислав Зелінський palyvoda.vz@gmail.com	Лікар	Кардіологія	Активний	Ніколи	✎

Рис. 3.22 – Управління персоналом

Вкладка яка фільтрує персонал по відділеннях (рис. 3.23).

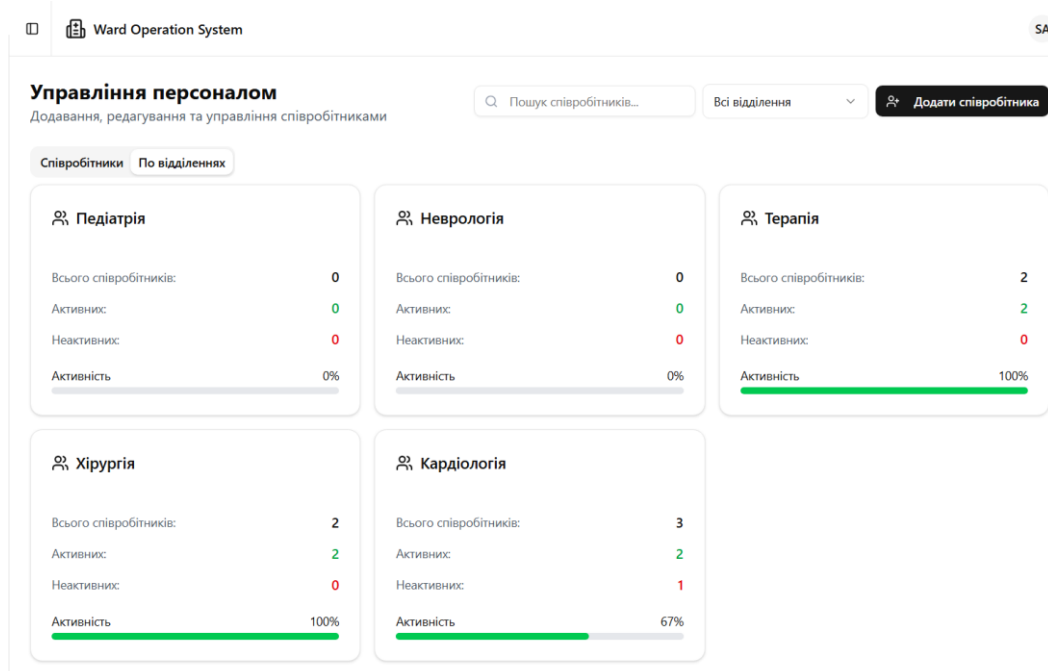


Рис. 3.23 – Статистика по відділеннях

Адміністратор може додати нового співробітника в систему (рис. 3.24), вказавши його особисті дані, email, посаду та відділення.

Додавання нового співробітника

Ім'я: Владислав

Прізвище: Зелінський

Email: palyvoda.vz@gmail.com

Посада: Лікар

Відділення: Кардіологія

Пароль:

Підтвердження пароля:

Статус активності: ☒ Співробітник є активним користувачем системи

Скасувати | Додати співробітника

Рис. 3.24 – Форма додавання співробітника

Форма редагування (рис. 3.25) дозволяє оновлювати інформацію про існуючого співробітника, наприклад, змінювати його посаду або статус активності.

Редагування співробітника

Ім'я: Олена

Прізвище: Петренко

Посада: Лікар

Відділення: Кардіологія

Статус активності: ☒ Співробітник є активним користувачем системи

Скасувати Зберегти зміни

Рис. 3.25 – Форма редагування співробітника

3.3 Тестування роботи вебдодатку

Протестуємо пошукову систему на нашій платформі [29]:

Управління персоналом
Додавання, редагування та управління співробітниками

Зелінський Всі відділення Додати співробітника

Співробітники По відділеннях

Список співробітників (1)

Співробітник	Посада	Відділення	Статус	Останній вхід	Дії
ВЗ Владислав Зелінський palyvoda.vz@gmail.com	Лікар	Кардіологія	Активний	Ніколи	✎

Рис. 3.26 – Вводимо прізвище співробітника

Рис. 3.27 – Результат пошуку

Ми можемо бачити, що все працює як треба (рис. 3.27). Адміністратор може знайти потрібного співробітника, ввівши частину його імені, прізвища або email у пошуковий рядок на сторінці «Управління персоналом» [30].

Перевіримо, як працює фільтрація за статусом у нашій системі:

Управління статусами ліжок

Швидка зміна статусів ліжок для медичного персоналу

Пошук ліжок...

Всього ліжок

6

У вашому відділенні

Вільно

2

Готові до використання

Зайнято

1

3 пацієнтами

Потребують уваги

2

Прибирання/ремонт

Всі статуси

Вільно

Зайнято

Прибирання

Ремонт

Зарезервовано

Недоступно

К-102-1

Кардіологія • Палата K-102

Статус: Зайнято

Змінити статус

К-101-4

Палата K-101

Статус: Прибирання

Змінити статус

К-101-3

Кардіологія • Палата K-101

Статус: Вільно

Змінити статус

К-101-2

Кардіологія • Палата K-101

Статус: Вільно

Змінити статус

К-101-1

Палата K-101

Статус: Вільно

Змінити статус

Рис. 3.28 – Обираємо статус, за яким ми хочемо зробити фільтрацію

Управління статусами ліжок

Швидка зміна статусів ліжок для медичного персоналу

Пошук ліжок...

Всього ліжок

2

У вашому відділенні

Вільно

2

Готові до використання

Зайнято

0

3 пацієнтами

Потребують уваги

0

Прибирання/ремонт

Вільно

Вільно

Змінити статус

К-101-3

Кардіологія • Палата K-101

Статус: Вільно

Змінити статус

К-101-2

Кардіологія • Палата K-101

Статус: Вільно

Змінити статус

Рис. 3.29 – Результат фільтрації

У результаті фільтрації ми можемо побачити, що система показала нам всі ліжка з обраним статусом «Вільно» (рис. 3.28 - 3.29). Це дозволяє медичному персоналу швидко знаходити доступні місця.

Спробуємо в ролі медичного працівника швидко госпіталізувати пацієнта:

Ward Operation System

B3

Швидка госпіталізація

Госпіталізація пацієнта на вільне ліжко

Вибір пацієнта

Існуючий пацієнт

Оберіть пацієнта

Причина госпіталізації

Детальний опис причини...

Планова дата виписки

mm/dd/yyyy

Вибір ліжка

Доступні ліжка (2)

K-101-3

Палата K-101

Кардіологія

K-101-2

Палата K-101

Кардіологія

Оберіть пацієнта

Оберіть ліжко

Скасувати

Госпіталізувати

Рис. 3.30 – Форма швидкої госпіталізації

Щоб госпіталізувати пацієнта, користувачу обов’язково потрібно обрати пацієнта та вільне ліжко, а також вказати дату та час госпіталізації (рис. 3.30). Система має вбудовану валідацію, яка не дозволить відправити форму з некоректними даними.

Швидка госпіталізація

Госпіталізація пацієнта на вільне ліжко

Вибір пацієнта

Існуючий пацієнт

Іван Петренко
KRD-20240618-001 • 02.03.1962

Діагноз при госпіталізації

Пацієнт поступив зі скаргами на біль у грудях та задишку.

Потребує ізоляції: Ні

Причина госпіталізації

Підозра на гострий коронарний синдром

Планова дата виписки

07/12/2025

Вибір ліжка

Доступні ліжка (2)

K-101-3

Палата K-101

Кардіологія

K-101-2

Палата K-101

Кардіологія

✓ Пацієнт обраний

✓ Ліжко обране

Скасувати

Госпіталізувати

Рис. 3.31 – Правильно заповнена форма

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

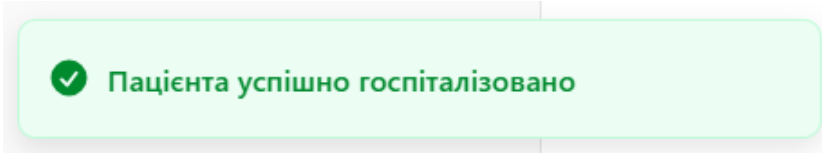


Рис. 3.32 – Успішна госпіталізація пацієнта

Переглянувши (рис. 3.31 - 3.32), можемо бачити, що форма була заповнена правильно, і система повідомила про успішну госпіталізацію пацієнта.

Перейдемо до панелі адміністратора та перевіримо, чи змінився статус ліжка після госпіталізації.

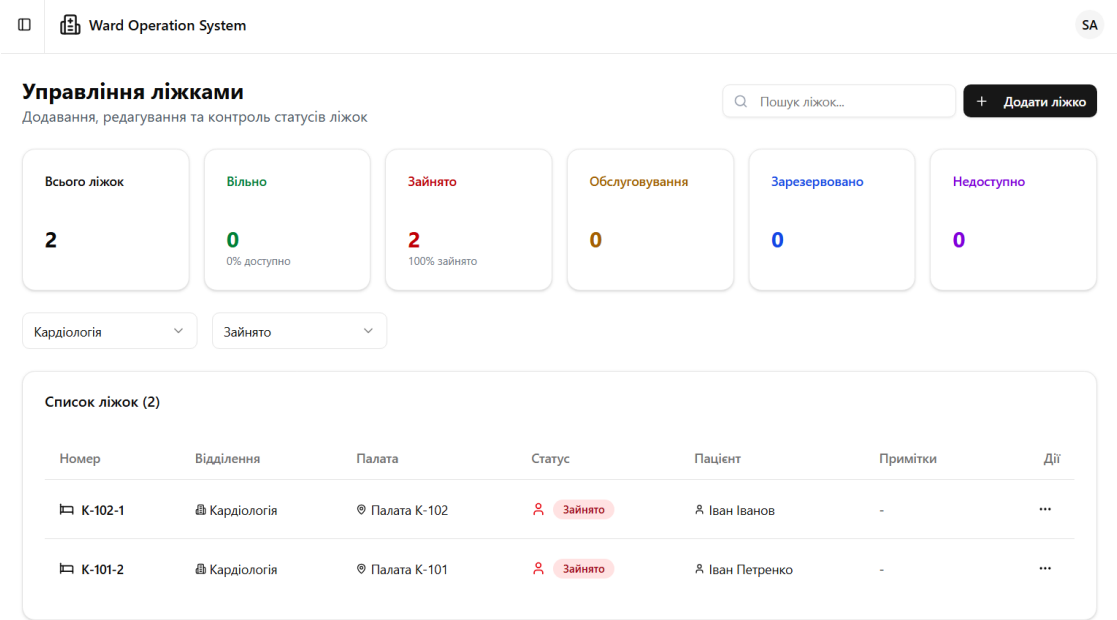


Рис. 3.33 – Статус ліжка змінився на «Зайнято»

Наша госпіталізація успішно відобразилась в системі: ліжко, на яке було госпіталізовано пацієнта, тепер має статус «Зайнято», і в таблиці вказано ім'я пацієнта (рис. 3.33). Адміністратор або медичний персонал може виписати пацієнта, якщо лікування завершено.

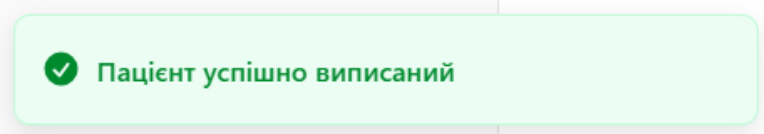


Рис. 3.34 – Повідомлення про успішну виписку пацієнта

Про успішну виписку пацієнта система покаже нам повідомлення (рис. 3.34), а статус ліжка автоматично зміниться на «Прибирання».

Розглянемо правильність роботи додавання нового співробітника адміністратором:

Перейшовши у вкладку «Управління персоналом» та натиснувши «Додати співробітника», нам потрібно заповнити форму. Якщо форма буде заповнена невірно, система покаже помилки валідації під відповідними полями.

The screenshot shows a web form titled "Додавання нового співробітника" (Adding new employee). The form contains several input fields with red borders and red error messages below them:

- Ім'я** (Name): "Введіть ім'я" (Enter name). Error: "Ім'я має містити принаймні 2 символи" (Name must contain at least 2 symbols).
- Прізвище** (Surname): "Введіть прізвище" (Enter surname). Error: "Прізвище має містити принаймні 2 символи" (Surname must contain at least 2 symbols).
- Email**: "email@hospital.ua". Error: "Введіть дійсну електронну адресу" (Enter a valid email address).
- Посада** (Position): A dropdown menu with "Оберіть посаду" (Select position).
- Відділення** (Department): A dropdown menu with "Оберіть відділення" (Select department).
- Пароль** (Password): "Введіть пароль" (Enter password). Error: "Пароль має містити принаймні 8 символів" (Password must contain at least 8 symbols).
- Підтвердження пароля** (Password confirmation): "Підтвердіть пароль" (Confirm password).
- Статус активності** (Activity status): A toggle switch labeled "Співробітник є активним користувачем системи" (Employee is an active system user).

At the bottom of the form are two buttons: "Скасувати" (Cancel) and "Додати співробітника" (Add employee).

Рис. 3.35 – Форма додавання співробітника з валідацією

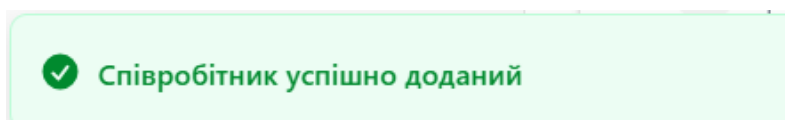


Рис. 3.36 – Повідомлення про успішне додавання співробітника

Заповнивши форму правильно та натиснувши кнопку «Додати», ми бачимо, що наша дія успішно виконана (рис. 3.36), і новий співробітник з'явився у загальному списку (рис. 3.37).

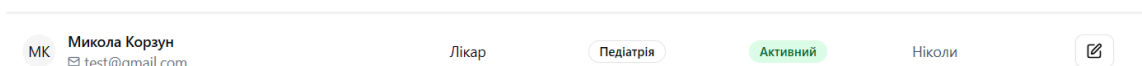


Рис. 3.37 – Відображення нового співробітника у списку

Перевіримо, чи зможе адміністратор видалити відділення лікарні:

Для цього потрібно перейти на сторінку «Структура лікарні» та натиснути кнопку видалення обраного відділення. Після чого з'явиться вікно для підтвердження операції (рис. 3.38).

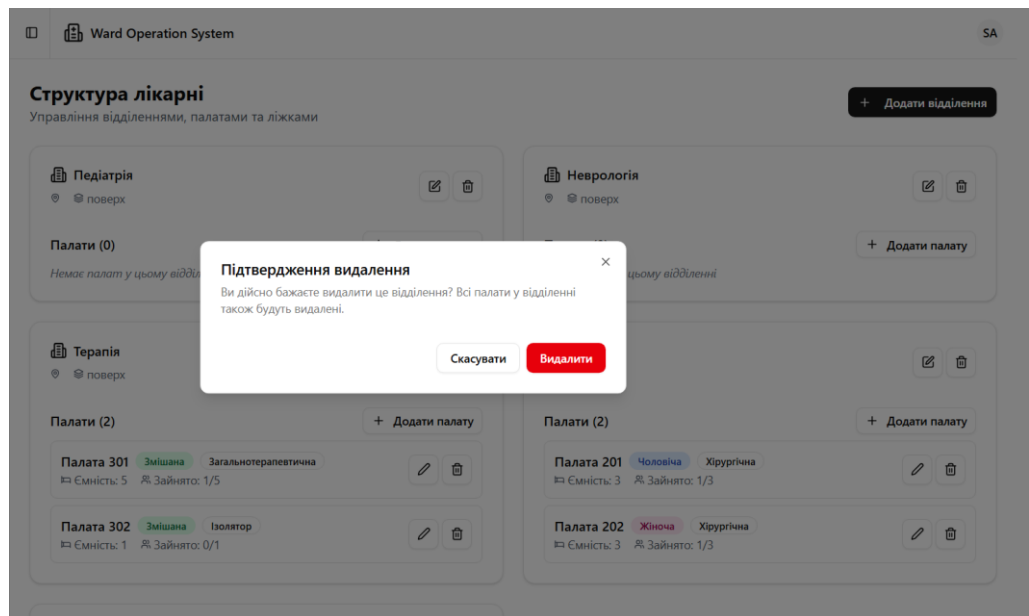


Рис. 3.38 – Видалення відділення

Підтвердивши операцію, натиснувши кнопку «Видалити», ми бачимо успішний результат наших дій.

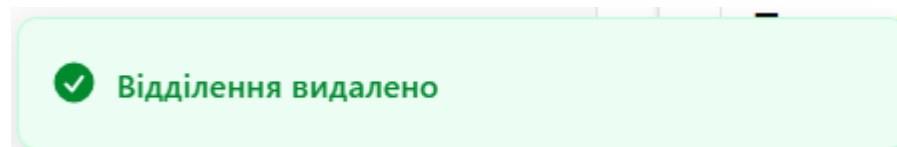


Рис. 3.39 – Повідомлення про успішне видалення відділення

Висновки до третього розділу

У даному розділі було проведено комплексне опрацювання архітектури та функціональної реалізації системи управління завантаженістю лікарні. Було створено детальну діаграму розгортання, яка відобразила ключові компоненти системи та принципи їх взаємодії між собою — від клієнтського застосунку, що реалізований за допомогою Next.js, до серверної частини на базі C# ASP.NET Web API та бази даних SQL Server.

Кожен елемент системи — UI, серверна логіка, API-контролери, бази даних, а також система авторизації — був детально проаналізований, що дозволило сформулювати цілісне уявлення про функціонування платформи в

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дата		

реальному середовищі. Архітектура базується на чіткому поділі відповідальностей та підтримує масштабованість, що дозволяє легко доповнювати функціонал системи в майбутньому.

Особлива увага була приділена реалізації користувацького інтерфейсу для різних ролей — адміністраторів, медичного персоналу та реєстраторів. Відповідно до призначених прав, користувачі отримують доступ лише до відповідного функціоналу: перегляду структури лікарні, керування пацієнтами, госпіталізаціями, пошуку вільних ліжок тощо. Такий підхід забезпечує не тільки ефективність роботи персоналу, а й безпеку даних у системі.

Для адміністраторів реалізована панель управління, яка дозволяє створювати структуру лікарні (відділення, палати, ліжка), додавати та редагувати персонал, керувати користувачами, а також переглядати історію госпіталізацій та статусів завантаженості. Через окремі інтерфейси медперсонал може оформлювати нові госпіталізації, виводити пацієнтів, переглядати вільні місця, а також отримувати зведену інформацію щодо поточного навантаження.

В рамках цього розділу також було проведено тестування розробленого функціоналу в умовах локального середовища. Усі ключові сценарії — створення пацієнтів, реєстрація госпіталізацій, зміна статусу ліжок, доступ до структурного перегляду лікарні — були реалізовані та протестовані без виявлення критичних помилок. Було перевірено обробку граничних випадків, зокрема дублювання даних, некоректні формати введення та перевірку прав доступу.

За результатами проведеного проєктування та реалізації можна зробити висновок, що вебдодаток для управління завантаженістю лікарні є стабільним, функціональним та готовим до подальшого впровадження в тестовому або реальному середовищі медичного закладу.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було розроблено вебдодаток для управління завантаженістю лікарні, який дозволяє медичному персоналу та адміністрації ефективно керувати ліжковим фондом та пацієнтопотоками. Було проведено детальний аналіз предметної області управління медичними закладами, вивчено функціональні можливості аналогічних систем, що дозволило сформулювати перелік основних вимог до майбутнього додатку.

У процесі роботи розроблено архітектуру вебдодатку, яка базується на сучасному стеку технологій з клієнтською частиною на Next.js та серверною на .NET (ASP.NET Core), із застосуванням бази даних SQL Server. Також було спроектовано та реалізовано структуру бази даних, яка включає таблиці для зберігання інформації про персонал, пацієнтів, відділення, палати, ліжка та госпіталізації.

Розглянуто основні сценарії взаємодії користувача з системою, розмежовані за ролями — медичний персонал та адміністратор. Описано їхні ключові можливості: відстеження статусів ліжок, управління пацієнтами, швидка госпіталізація, а для адміністраторів — повне управління структурою лікарні, персоналом та ресурсами через спеціалізовану панель.

Сформовано архітектурні діаграми, зокрема діаграми класів, компонентів та розгортання, що демонструють структуру та логіку роботи додатку. Проведено обґрунтування вибору інструментальних засобів, описано їх переваги та визначено вимоги до апаратного забезпечення для клієнтської та серверної частин.

У результаті виконання роботи було реалізовано повнофункціональний вебдодаток з інтуїтивно зрозумілим інтерфейсом, який спрощує процес взаємодії між адміністративним та медичним персоналом, оптимізує управління лікарняними ресурсами та надає зручні засоби для моніторингу та звітності в реальному часі.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Освітній портал. [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.ztu.edu.ua/course/view.php?id=6649>
2. Пономаренко Т. О. Технології створення веб-додатків. — Київ: КНТ, 2019. — 312 с.
3. Сидоренко А. Ю. Управління IT-проєктами: теорія і практика. — Київ: Кондор-Видавництво, 2019. — 328 с.
4. Чала Н. Д. Архітектура програмного забезпечення. — Харків: ХНУРЕ, 2020. — 280 с.
5. Microsoft Docs: ASP.NET Core Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/>
6. SQL Server Official Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/sql/sql-server/>
7. REST API Tutorial [Електронний ресурс] – Режим доступу до ресурсу: <https://www.restapitutorial.com/>
8. Next.js Official Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs>
9. TypeScript Official Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>
10. Tailwind CSS Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://tailwindcss.com/docs>
11. Shadcn/ui Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://ui.shadcn.com/docs>
12. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. — Санта-Фе: Prentice Hall, 2018. — 420 с.
13. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. — Addison-Wesley Professional, 2003. — 560 с.
14. Іванов О. С. Реляційні бази даних. Теорія і практика. — Київ: Либідь, 2016. — 368 с.

15. Microsoft. SignalR for ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/signalr>
16. Ковальчук І. М. Бази даних: проектування та використання. — Львів: БСВ «Медійна реклама», 2018. — 440 с.
17. Richards M. Microservices vs. Service-Oriented Architecture. — O'Reilly Media, 2016. — 120 с.
18. Skeet J. C# in Depth, Fourth Edition. — Нью-Йорк: Manning Publications, 2019. — 528 с.
19. Троелсен Є., Джепікс Ф. Мова програмування C# 7 та платформа .NET і .NET Core. — Київ: Діалектика, 2020. — 1328 с.
20. Висоцький І. В. Основи проектування програмного забезпечення. — Харків: ХНУРЕ, 2019. — 246 с.
21. Fowler M. Patterns of Enterprise Application Architecture. — Бостон: Addison-Wesley Professional, 2003. — 560 с.
22. React Official Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/>
23. Microsoft Docs: CQRS pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs>
24. W3Schools [Електронний ресурс] – Режим доступу до ресурсу: <https://www.w3schools.com>
25. Microsoft Docs: .NET Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/>
26. Microsoft Docs: Azure DevOps Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/azure/devops/>
27. Lerman J., Miller R. Programming Entity Framework: Code First. — Sebastopol: O'Reilly Media, 2011. — 244 с.
28. Microsoft Docs: Entity Framework Core [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/core/>
29. Глушков А. В. Тестування програмного забезпечення. — Київ: КНТ, 2018. — 310 с.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

30. Swagger OpenAPI Specification [Електронний ресурс] – Режим доступу до ресурсу: <https://swagger.io/specification/>

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

Login.cs

```

public static class Login
{
    public class Command : IRequest<AuthResponse>
    {
        public string Email { get; set; } = string.Empty;
        public string Password { get; set; } = string.Empty;
    }

    internal sealed class Handler : IRequestHandler<Command, AuthResponse>
    {
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly ITokenService _tokenService;

        public Handler(UserManager<ApplicationUser> userManager, ITokenService
tokenService)
        {
            _userManager = userManager;
            _tokenService = tokenService;
        }

        public async Task<AuthResponse> Handle(Command request, CancellationToken
cancellationToken)
        {
            var user = await _userManager.FindByEmailAsync(request.Email);
            if (user == null)
            {
                throw new InvalidOperationException("Invalid email or password");
            }

            var isPasswordValid = await _userManager.CheckPasswordAsync(user,
request.Password);
            if (!isPasswordValid)
            {
                throw new InvalidOperationException("Invalid email or password");
            }

            var token = await _tokenService.GenerateTokenAsync(user);
            var roles = await _userManager.GetRolesAsync(user);

            return new AuthResponse
            {
                Token = token,
                Id = user.Id,
                Email = user.Email,
                FirstName = user.FirstName,
                LastName = user.LastName,
                Position = user.Position,
                Role = roles.FirstOrDefault() ?? ""
            };
        }
    }
}

public class LoginEndpoint : ICarterModule
{
    public void AddRoutes(IEndpointRouteBuilder app)
    {

```

```

        app.MapPost("api/auth/login", async (LoginRequest request, ISender sender,
IValidator<LoginRequest> validator) =>
        {
            var validationResult = await validator.ValidateAsync(request);
            if (!validationResult.IsValid)
            {
                return Results.ValidationProblem(validationResult.ToDictionary());
            }

            var command = new Login.Command
            {
                Email = request.Email,
                Password = request.Password
            };

            var result = await sender.Send(command);

            return Results.Ok(result);
        })
        .WithTags("Authentication")
        .WithName("Login")
        .WithDescription("Authenticates a user and returns a JWT token")
        .Produces<AuthResponse>(StatusCodes.Status200OK)
        .Produces(StatusCodes.Status400BadRequest)
        .Produces(StatusCodes.Status500InternalServerError);
    }
}

```

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

CreateDepartment.cs

```

public static class CreateDepartment
{
    public class Command : IRequest<DepartmentResponse>
    {
        public string Name { get; set; } = string.Empty;
        public string? Building { get; set; }
        public int? FloorNumber { get; set; }
        public string? Description { get; set; }
    }

    internal sealed class Handler : IRequestHandler<Command, DepartmentResponse>
    {
        private readonly ApplicationDbContext _dbContext;

        public Handler(ApplicationDbContext dbContext)
        {
            _dbContext = dbContext;
        }

        public async Task<DepartmentResponse> Handle(Command request,
            CancellationToken cancellationToken)
        {
            if (await _dbContext.Departments.AnyAsync(d => d.Name == request.Name,
                cancellationToken))
            {
                throw new InvalidOperationException("Department with the same name
                already exists.");
            }

            var department = new Department
            {
                Id = Guid.NewGuid(),
                Name = request.Name,
                Building = request.Building,
                FloorNumber = request.FloorNumber,
                Description = request.Description
            };

            _dbContext.Add(department);
            await _dbContext.SaveChangesAsync(cancellationToken);

            return new DepartmentResponse()
            {
                Id = department.Id,
                Name = department.Name,
                Building = department.Building,
                FloorNumber = department.FloorNumber,
                Description = department.Description
            };
        }
    }
}

public class CreateDepartmentEndpoint : ICarterModule
{
    public void AddRoutes(IEndpointRouteBuilder app)
    {

```

Змн.	Арк.	№ докум.	Підпис	Дата

```

    app.MapPost("api/departments", async (CreateDepartmentRequest request,
ISender sender, IValidator<CreateDepartmentRequest> validator) =>
    {
        var validationResult = await validator.ValidateAsync(request);
        if (!validationResult.IsValid)
        {
            return Results.ValidationProblem(validationResult.ToDictionary());
        }

        var command = new CreateDepartment.Command
        {
            Name = request.Name,
            Building = request.Building,
            FloorNumber = request.FloorNumber,
            Description = request.Description
        };

        var result = await sender.Send(command);

        return Results.Created($"/api/departments/{result.Id}", result);
    })
    .RequireAuthorization(AuthorizationPolicies.AdminPolicy)
    .WithTags("Departments")
    .WithName("CreateDepartment")
    .WithDescription("Creates a new department")
    .Produces<DepartmentResponse>(StatusCodes.Status201Created)
    .Produces(StatusCodes.Status400BadRequest)
    .Produces(StatusCodes.Status500InternalServerError);
}
}

```

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

ApplicationDbContext.cs

```

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) :
    base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);

        builder.ApplyConfigurationsFromAssembly(typeof(ApplicationDbContext).Assembly);

        var adminRoleId = "a18be9c0-aa65-4af8-bd17-00bd9344e575";
        var staffRoleId = "136bf839-a5ce-47e0-8a7c-506c8c18254c";

        builder.Entity<IdentityRole>().HasData(
            new IdentityRole { Id = adminRoleId, Name = "Admin", NormalizedName =
"ADMIN" },
            new IdentityRole { Id = staffRoleId, Name = "Staff", NormalizedName =
"STAFF" }
        );
    }

    public DbSet<Department> Departments { get; set; }
    public DbSet<WardType> WardTypes { get; set; }
    public DbSet<Ward> Wards { get; set; }
    public DbSet<Bed> Beds { get; set; }
    public DbSet<Patient> Patients { get; set; }
    public DbSet<Hospitalization> Hospitalizations { get; set; }
    public DbSet<BedEventLog> BedEventLogs { get; set; }
}

```

patients/page.tsx

```

interface DialogState {
  isAddPatientOpen: boolean;
  isEditPatientOpen: boolean;
  isViewPatientOpen: boolean;
  isConfirmDeleteOpen: boolean;
  isAssignHospitalizationOpen: boolean;
  isViewHospitalizationOpen: boolean;
  isDischargePatientOpen: boolean;
  selectedPatient: Patient | null;
  selectedHospitalization: Hospitalization | null;
}

export default function PatientManagement() {
  const { patients, addPatient, updatePatient, deletePatient, isLoading } =
    usePatient();
  const { hospitalizations, dischargePatient } = useHospitalization();

  const { form, setPatientData, resetForm } = usePatientForm();
  const { dischargeForm, resetForm: resetDischargeForm } =
    useHospitalizationForm();

  const [patientFormDialogMode, setPatientFormDialogMode] = useState<
    "add" | "edit"
  >("add");

  console.log("Patients:", patients);

  const [activeTab, setActiveTab] = useState("patients");
  const [searchTerm, setSearchTerm] = useState("");
  const [dialogState, setDialogState] = useState<DialogState>({
    isAddPatientOpen: false,
    isEditPatientOpen: false,
    isViewPatientOpen: false,
    isConfirmDeleteOpen: false,
    isAssignHospitalizationOpen: false,
    isViewHospitalizationOpen: false,
    isDischargePatientOpen: false,
    selectedPatient: null,
    selectedHospitalization: null,
  });

  const onAddPatient = () => {
    resetForm();
    setPatientFormDialogMode("add");
    setDialogState((prev) => ({ ...prev, isAddPatientOpen: true }));
  };

```

```

const onViewPatient = (patient: Patient) => {
  setDialogState((prev) => ({
    ...prev,
    selectedPatient: patient,
    isViewPatientOpen: true,
  }));
};

const onEditPatient = (patient: Patient) => {
  setPatientFormDialogMode("edit");
  setPatientData(patient);
  setDialogState((prev) => ({
    ...prev,
    selectedPatient: patient,
    isEditPatientOpen: true,
  }));
};

const onDeletePatient = (patient: Patient) => {
  setDialogState((prev) => ({
    ...prev,
    selectedPatient: patient,
    isConfirmDeleteOpen: true,
  }));
};

const onViewHospitalization = (hospitalization: Hospitalization) => {
  setDialogState((prev) => ({
    ...prev,
    selectedHospitalization: hospitalization,
    isViewHospitalizationOpen: true,
  }));
};

const onDischargePatient = (hospitalization: Hospitalization) => {
  resetDischargeForm();
  setDialogState((prev) => ({
    ...prev,
    selectedHospitalization: hospitalization,
    isDischargePatientOpen: true,
  }));
};

const handleClosePatientFormDialog = () => {
  setDialogState((prev) => ({
    ...prev,
    isAddPatientOpen: false,
    isEditPatientOpen: false,
    selectedPatient: null,
  }));
  resetForm();
};

```

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

```

};

const handleAddPatient = async (patientData: PatientFormValues) => {
  try {
    await addPatient(patientData);
    setDialogState((prev) => ({ ...prev, isAddPatientOpen: false }));
    resetForm();
  } catch (error) {
    console.error("Failed to add patient:", error);
    toast.error("Помилка при додаванні пацієнта");
  }
};

const handleEditPatient = async (patientData: PatientFormValues) => {
  if (!dialogState.selectedPatient) return;

  try {
    await updatePatient(dialogState.selectedPatient.id, patientData);
    setDialogState((prev) => ({
      ...prev,
      isEditPatientOpen: false,
      selectedPatient: null,
    }));
    resetForm();
    toast.success("Пацієнт успішно оновлений");
  } catch (error) {
    console.error("Failed to update patient:", error);
    toast.error("Помилка при оновленні пацієнта");
  }
};

const handleDeletePatient = async () => {
  if (!dialogState.selectedPatient) return;

  try {
    await deletePatient(dialogState.selectedPatient.id);
    setDialogState((prev) => ({
      ...prev,
      isConfirmDeleteOpen: false,
      selectedPatient: null,
    }));
  } catch (error) {
    console.error("Failed to delete patient:", error);
    toast.error("Помилка при видаленні пацієнта");
  }
};

const handleDischargePatient = async (
  dischargeData: DischargePatientFormValues,
) => {
  try {

```

```

    await dischargePatient(
      dialogState.selectedHospitalization!.patientId!,
      dischargeData,
    );
    setDialogState((prev) => ({
      ...prev,
      isDischargePatientOpen: false,
      selectedHospitalization: null,
    }));
    resetDischargeForm();
    toast.success("Пацієнт успішно виписаний");
  } catch (error) {
    console.error("Failed to discharge patient:", error);
    toast.error("Помилка при виписці пацієнта");
  }
};

return (
  <div className="space-y-6">
    <div className="flex items-center justify-between">
      <div>
        <h2 className="text-2xl font-bold">Управління пацієнтами</h2>
        <p className="text-gray-600">
          Реєстрація, госпіталізація та ведення пацієнтів
        </p>
      </div>
      <div className="flex items-center gap-2">
        <div className="relative">
          <Search className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-400 h-4 w-4" />
          <Input
            placeholder="Пошук пацієнтів..."
            value={searchTerm}
            onChange={(e) => setSearchTerm(e.target.value)}
            className="pl-10 w-64"
          />
        </div>
        <Button onClick={onAddPatient} variant="default">
          <UserPlus className="h-4 w-4 mr-2" />
          Додати пацієнта
        </Button>
      </div>
    </div>

    <Tabs value={activeTab} onValueChange={setActiveTab}>
      <TabsList>
        <TabsTrigger value="patients">Пацієнти</TabsTrigger>
        <TabsTrigger value="hospitalizations">Госпіталізації</TabsTrigger>
        <TabsTrigger value="discharges">Виписки</TabsTrigger>
      </TabsList>

```

```

<TabsContent value="patients" className="space-y-4">
  <PatientList
    patients={patients}
    searchTerm={searchTerm}
    onViewPatient={onViewPatient}
    onEditPatient={onEditPatient}
    onDeletePatient={onDeletePatient}
  />
</TabsContent>

<TabsContent value="hospitalizations" className="space-y-4">
  <HospitalizationList
    hospitalizations={hospitalizations}
    onViewHospitalization={onViewHospitalization}
    onDischargePatient={onDischargePatient}
  />
</TabsContent>

<TabsContent value="discharges" className="space-y-4">
  <DischargeList hospitalizations={hospitalizations} />
</TabsContent>
</Tabs>

<PatientFormDialog
  isOpen={dialogState.isAddPatientOpen || dialogState.isEditPatientOpen}
  onClose={handleClosePatientFormDialog}
  onSubmit={
    patientFormDialogMode === "add" ? handleAddPatient : handleEditPatient
  }
  form={form}
  isLoading={isLoading}
  mode={patientFormDialogMode}
/>

<ConfirmDeleteDialog
  isOpen={dialogState.isConfirmDeleteOpen}
  onClose={() =>
    setDialogState((prev) => ({
      ...prev,
      isConfirmDeleteOpen: false,
      selectedPatient: null,
    }))
  }
  onConfirm={handleDeletePatient}
  isLoading={isLoading}
  entityName={`націєнтра "${dialogState.selectedPatient?.firstName}
${dialogState.selectedPatient?.lastName}"`}
/>

<PatientViewDialog
  isOpen={dialogState.isViewPatientOpen}

```

```

        onClose={() =>
            setDialogState((prev) => ({
                ...prev,
                isViewPatientOpen: false,
                selectedPatient: null,
            }))
        }
        patient={dialogState.selectedPatient}
    />

<DischargePatientFormDialog
    isOpen={dialogState.isDischargePatientOpen}
    onClose={() =>
        setDialogState((prev) => ({
            ...prev,
            isDischargePatientOpen: false,
            selectedHospitalization: null,
        }))
    }
    onSubmit={handleDischargePatient}
    form={dischargeForm}
    selectedHospitalization={dialogState.selectedHospitalization}
    isLoading={isLoading}
/>

<HospitalizationViewDialog
    isOpen={dialogState.isViewHospitalizationOpen}
    onClose={() =>
        setDialogState((prev) => ({
            ...prev,
            isViewHospitalizationOpen: false,
            selectedHospitalization: null,
        }))
    }
    hospitalization={dialogState.selectedHospitalization}
/>
</div>
);
}

```

bed-status/page.tsx

```

const userRole = "Staff";

export default function BedStatusManagement() {
  const { beds, isLoading, changeBedStatus } = useBed();
  const { departments } = useDepartment();

  const [searchTerm, setSearchTerm] = useState("");
  const [selectedDepartment, setSelectedDepartment] = useState("all");
  const [selectedStatus, setSelectedStatus] = useState("all");

  const [isChangeStatusOpen, setIsChangeStatusOpen] = useState(false);
  const [selectedBed, setSelectedBed] = useState<BedResponse | null>(null);

  const form = useForm<ChangeBedStatusFormValues>({
    resolver: zodResolver(changeBedStatusSchema),
    defaultValues: {
      status: undefined,
      notes: "",
    },
  });

  const filteredBeds = beds.filter((bed) => {
    const searchMatch =
      bed.bedNumber.toLowerCase().includes(searchTerm.toLowerCase()) ||
      bed.departmentName.toLowerCase().includes(searchTerm.toLowerCase()) ||
      (bed.patientName &&
        bed.patientName.toLowerCase().includes(searchTerm.toLowerCase())) ||
      (bed.notes && bed.notes.toLowerCase().includes(searchTerm.toLowerCase()));

    const departmentMatch =
      selectedDepartment === "all" || bed.departmentId === selectedDepartment;
    const statusMatch =
      selectedStatus === "all" || bed.status === selectedStatus;

    return searchMatch && departmentMatch && statusMatch;
  });

  const handleChangeStatus = (bed: BedResponse) => {
    setSelectedBed(bed);
    form.reset({
      status: bed.status,
      notes: bed.notes || "",
    });
    setIsChangeStatusOpen(true);
  };

  const handleStatusChange = async (data: ChangeBedStatusFormValues) => {

```

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						83
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    try {
      await changeBedStatus(selectedBed!.id, data);
      setIsChangeStatusOpen(false);
      setSelectedBed(null);
      form.reset();
      toast.success("Статус ліжка успішно змінено");
    } catch (error) {
      console.error("Error changing bed status:", error);
      toast.error("Не вдалося змінити статус ліжка");
    }
  };

  return (
    <div className="space-y-6">
      <div className="flex flex-col sm:flex-row sm:items-center justify-between gap-4">
        <div>
          <h2 className="text-2xl font-bold">Управління статусами ліжок</h2>
          <p className="text-gray-600">
            Швидка зміна статусів ліжок для медичного персоналу
          </p>
        </div>
        <div className="flex items-center gap-2">
          <div className="relative">
            <Search className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-400 h-4 w-4" />
            <Input
              placeholder="Пошук ліжок..."
              value={searchTerm}
              onChange={(e) => setSearchTerm(e.target.value)}
              className="pl-10 w-64"
            />
          </div>
        </div>
      </div>

      <BedStatusStats filteredBeds={filteredBeds} />

      <BedStatusFilters
        departments={departments ?? []}
        userRole={userRole}
        setSelectedDepartment={setSelectedDepartment}
        selectedDepartment={selectedDepartment}
        selectedStatus={selectedStatus}
        setSelectedStatus={setSelectedStatus}
      />

      <BedStatusList
        filteredBeds={filteredBeds}
        handleChangeStatus={handleChangeStatus}
      />
    </div>
  );

```

```

    <ChangeBedStatusFormDialog
      isOpen={isChangeStatusOpen}
      onClose={() => {
        setIsChangeStatusOpen(false);
        setSelectedBed(null);
        form.reset();
      }}
      onSubmit={handleStatusChange}
      form={form}
      userRole={userRole}
      selectedBed={selectedBed}
      isLoading={isLoading}
    />
  </div>
);
}

```

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						85
Змн.	Арк.	№ докум.	Підпис	Дата		

quick-hospitalization/page.tsx

```

export default function QuickHospitalization() {
  const { patients, refetch: fetchPatients } = usePatient();
  const { beds, fetchBeds } = useBed();

  const form = useForm<CreateHospitalizationFormValues>({
    resolver: zodResolver(createHospitalizationSchema),
    defaultValues: {
      patientId: "",
      bedId: "",
      admissionDateTime: new Date(),
      admissionReason: "",
    },
  });

  const filteredPatients = patients.filter(
    (patient) => patient.status === "Registered",
  );

  const availableBeds = beds.filter((bed) => bed.status === "Available");

  const onSubmit = async (data: CreateHospitalizationFormValues) => {
    try {
      await HospitalizationService.createHospitalization(data);
      toast.success("Пацієнта успішно госпіталізовано");
      form.reset();
    } catch (error) {
      console.error("Error during hospitalization:", error);
      toast.error("Помилка при госпіталізації пацієнта");
    } finally {
      fetchPatients();
      fetchBeds();
    }
  };

  const selectedPatient =
    patients.find((p) => p.id === form.watch("patientId")) || null;
  const selectedBed = beds.find((b) => b.id === form.watch("bedId")) || null;

  return (
    <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-6">
      <div className="space-y-6">
        <div className="flex items-center gap-3">
          <UserPlus className="h-6 w-6 text-blue-600" />
          <div>
            <h2 className="text-2xl font-bold">Швидка госпіталізація</h2>
            <p className="text-gray-600">
              Госпіталізація пацієнта на вільне ліжко
            </p>
          </div>
        </div>
      </div>
    </form>
  );
}

```

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						86
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    </p>
  </div>
</div>

<div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
  <Card>
    <CardHeader>
      <CardTitle className="flex items-center gap-2">
        <User className="h-5 w-5" />
        Вибір пацієнта
      </CardTitle>
    </CardHeader>
    <CardContent className="space-y-4">
      <div className="space-y-2">
        <Label htmlFor="existingPatient">Існуючий пацієнт</Label>
        <Select
          value={form.watch("patientId")}
          onChange={(val) => form.setValue("patientId", val)}
        >
          <SelectTrigger>
            <SelectValue placeholder="Оберіть пацієнта" />
          </SelectTrigger>
          <SelectContent>
            {filteredPatients.map((patient) => (
              <SelectItem key={patient.id} value={patient.id}>
                <div className="flex flex-col">
                  <span>{patient.fullName}</span>
                  <span className="text-xs text-gray-500">
                    {patient.medicalCardNumber} • {" "}
                    {new Date(patient.dateOfBirth).toLocaleDateString(
                      "uk-UA",
                    )}
                  </span>
                </div>
              </SelectItem>
            ))}
          </SelectContent>
        </Select>
        {form.formState.errors.patientId && (
          <span className="text-xs text-red-500">
            {form.formState.errors.patientId.message}
          </span>
        )}
      </div>

      {selectedPatient && (
        <div className="space-y-2">
          <Label>Діагноз при госпіталізації</Label>
          <div className="bg-gray-50 rounded p-3 text-sm text-gray-800
border">
            {selectedPatient.admissionDiagnosis || "Діагноз не вказано"}
          </div>
        </div>
      )}
    </CardContent>
  </Card>
</div>

```

```

</div>
<div className="flex items-center gap-2">
  <span className="text-xs text-gray-600">
    Потребує ізоляції:
  </span>
  {selectedPatient.requiresIsolation ? (
    <span className="font-semibold">Так</span>
  ) : (
    <span className="font-semibold">Ні</span>
  )}
</div>
</div>
)}

<div className="space-y-2">
  <Label htmlFor="admissionReason">Причина госпіталізації</Label>
  <Textarea
    id="admissionReason"
    placeholder="Детальний опис причини..."
    {...form.register("admissionReason")}
  />
  {form.formState.errors.admissionReason && (
    <span className="text-xs text-red-500">
      {form.formState.errors.admissionReason.message}
    </span>
  )}
</div>

<div className="space-y-2">
  <Label htmlFor="plannedDischarge">Планова дата виписки</Label>
  <Input
    id="plannedDischarge"
    type="date"
    {...form.register("plannedDischargeDateTime", {
      valueAsDate: true,
    })}
  />
  {form.formState.errors.plannedDischargeDateTime && (
    <span className="text-xs text-red-500">
      {form.formState.errors.plannedDischargeDateTime.message}
    </span>
  )}
</div>
</CardContent>
</Card>

<Card>
  <CardHeader>
    <CardTitle className="flex items-center gap-2">
      <Bed className="h-5 w-5" />
      Вибір ліжка

```

```

        </CardTitle>
      </CardHeader>
      <CardContent className="space-y-4">
        <div className="space-y-2">
          <Label>Доступні ліжка ({availableBeds.length})</Label>
          <div className="space-y-2 max-h-64 overflow-y-auto">
            {availableBeds.map((bed) => (
              <div
                key={bed.id}
                className={`p-3 border rounded-lg cursor-pointer
transition-all ${
                  form.watch("bedId") === bed.id
                    ? "border-blue-500 bg-blue-50"
                    : "border-gray-200 hover:border-gray-300"
                }`}
                onClick={() => form.setValue("bedId", bed.id)}
              >
                <div className="flex items-center justify-between mb-2">
                  <span className="font-medium">{bed.bedNumber}</span>
                  <Badge variant="outline">{bed.departmentName}</Badge>
                </div>
                <div className="text-sm text-gray-600 mb-2">
                  Палата {bed.wardNumber}
                </div>
              </div>
            ))}
          </div>
          {form.formState.errors.bedId && (
            <span className="text-xs text-red-500">
              {form.formState.errors.bedId.message}
            </span>
          )}
        </div>
      </CardContent>
    </Card>
  </div>

  <Card>
    <CardContent className="pt-6">
      <div className="flex items-center justify-between">
        <div className="flex items-center gap-4">
          {selectedPatient ? (
            <div className="flex items-center gap-2 text-green-600">
              <CheckCircle className="h-4 w-4" />
              <span className="text-sm">Пацієнт обраний</span>
            </div>
          ) : (
            <div className="flex items-center gap-2 text-gray-500">
              <AlertTriangle className="h-4 w-4" />
              <span className="text-sm">Оберіть пацієнта</span>
            </div>
          )}
        </div>
      </div>
    </CardContent>
  </Card>

```

```

    })

    {selectedBed ? (
      <div className="flex items-center gap-2 text-green-600">
        <CheckCircle className="h-4 w-4" />
        <span className="text-sm">Ліжко обране</span>
      </div>
    ) : (
      <div className="flex items-center gap-2 text-gray-500">
        <AlertTriangle className="h-4 w-4" />
        <span className="text-sm">Оберіть ліжко</span>
      </div>
    )}
  </div>

  <div className="flex gap-2">
    <Button
      variant="outline"
      type="button"
      onClick={() => form.reset()}
    >
      Скасувати
    </Button>
    <Button
      type="submit"
      disabled={!form.watch("patientId") || !form.watch("bedId")}
    >
      Госпіталізувати
    </Button>
  </div>
</div>
</CardContent>
</Card>
</div>
</form>
);
}

```

login/page.tsx

```

export default function LoginPage() {
  const { login } = useAuth();
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");
  const [isLoading, setIsLoading] = useState(false);

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setError("");
    setIsLoading(true);

    if (!email.trim()) {
      setError("Email required");
      setIsLoading(false);
      return;
    }

    if (!password) {
      setError("Password required");
      setIsLoading(false);
      return;
    }

    try {
      await login(email, password);
    } catch (error) {
      console.error("Login error:", error);
      setError(
        error instanceof Error
          ? error.message
          : "Login failed. Check your credentials and try again.",
      );
    } finally {
      setIsLoading(false);
    }
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-background
theme-transition p-4">
      <div className="w-full max-w-md space-y-6">
        <div className="text-center space-y-2">
          <div className="flex justify-center">
            <div className="p-3 bg-primary/10 rounded-full">
              <Hospital className="h-8 w-8 text-primary" />
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}

```

```

</div>
<h1 className="text-2xl font-bold text-foreground">
  Ward Operation System
</h1>
<p className="text-muted-foreground">
  Увійдіть до свого облікового запису
</p>
</div>

<Card className="theme-transition">
  <CardContent>
    <form onSubmit={handleSubmit} className="space-y-4">
      <div className="space-y-2">
        <Label htmlFor="email">Email</Label>
        <div className="relative">
          <Mail className="absolute left-3 top-1/2 transform -translate-
y-1/2 text-muted-foreground h-4 w-4" />
          <Input
            id="email"
            type="email"
            placeholder="your@email.com"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="pl-10"
            required
          />
        </div>
      </div>

      <div className="space-y-2">
        <Label htmlFor="password">Пароль</Label>
        <div className="relative">
          <Lock className="absolute left-3 top-1/2 transform -translate-
y-1/2 text-muted-foreground h-4 w-4" />
          <Input
            id="password"
            type={"password"}
            placeholder="Введіть пароль"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            className="pl-10"
            required
          />
        </div>
      </div>

      {error && (
        <Alert variant="destructive">
          <AlertTitle>Помилка входу</AlertTitle>
          <AlertDescription>{error}</AlertDescription>
        </Alert>
      )}
    </form>
  </CardContent>
</Card>

```

```

    }}

    <Button type="submit" className="w-full" disabled={isLoading}>
      {isLoading ? "Вхід..." : "Увійти"}
    </Button>
  </form>
</CardContent>
</Card>
</div>
</div>
);
}

```

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						93
Змн.	Арк.	№ докум.	Підпис	Дата		