МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проєктами

Кафедра програмного забезпечення автоматизованих систем

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма «Інженерія програмного забезпечення»

«Допущений до захисту»

Завідувач кафедри

_____проф. Приходько С.Б.

«_____» _____2026 р.

КВАЛІФІКАЦІЙНАРОБОТА

на здобуття ступеня вищої освіти «бакалавр»

на тему: «Розробка програмного забезпечення для автоматизації облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська»

Виконав: студент групи 4151

_____ Абу Мєхада Х.

(підпис, ПІБ)

Керівник роботи:

Доцент кафедри ПЗАС, к.т.н.

(посада, науковий ступень вчене звання)

_____ Фаріонова Т.А.

(підпис, ПІБ)

Миколаїв – 2026 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проєктами
Кафедра програмного забезпечення автоматизованих систем
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма «Інженерія програмного забезпечення»

«ЗАТВЕРДЖУЮ»
Гарант освітньої програми
_____ доц. Макарова Л.М.
(підпис)
«_07_» ___04___ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття ступеня вищої освіти «бакалавр»

Студенту _____Абу Мєхада Халеду_____
(Прізвище, ім'я, по батькові)

1. Тема роботи: Розробка програмного забезпечення для автоматизації облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська

Керівник роботи доцент кафедри ПЗАС, к.т.н. Фаріонова Тетяна Анатоліївна
Затверджені наказом ректора №275-уч від 07.04.2026 р.
2. Термін подання роботи: 01.06.2026 р.
3. Вихідні дані по роботі:

4. Перелік питань, що належать до розробки (найменування розділів)
Титульний аркуш; Завдання на кваліфікаційну роботу; Анотація (українською та англійською мовами); Зміст; Перелік умовних позначень, символів, одиниць та термінів (при необхідності); Вступ; Аналіз практичної задачі інженерії програмного забезпечення; Проєкт програмного забезпечення; Результати розробки програмного забезпечення; Розділ з охорони праці; Висновки; Список використаних джерел; Додатки (технічне завдання, текст програми, опис програми, інструкція користувача, програма та методика випробувань програмного забезпечення).

5. Перелік презентаційних матеріалів 1.Титульний слайд, 2.Мета та завдання кваліфікаційної роботи, 3.Аналоги ПЗ, 4.Аналіз вимог (Діаграма варіантів використання ПЗ), 5.Архітектура програмного забезпечення (Діаграма компонентів), 6.Проєкт програмного забезпечення (Діаграма розгортання ПЗ), 7-8. Ескізний проєкт (Діаграми діяльності «Запис на прийом», «Ведення електронної медичної картки», «Виписка електронного рецепта»), 9.Ескізний проєкт (логічна модель даних), 10.Ескізний проєкт (прототипи екранних форм), 11-12.Технічний проєкт (діаграма класів та діаграма класів-сутностей), 13-14.Технічний проєкт (діаграма послідовності прецедентів «Запис на прийом» та «Виписка електронного рецепта»), 15.Робочий проєкт. Вибір засобів розробки, 16.Результати розробки ПЗ (екранні форми), 17.Висновки, 18. Заключний слайд

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
4	Гурець Н.В., ст. викладач		

7. Дата видачі завдання 07.04.2026 р.

КАЛЕНДАРНИЙ ПЛАН

Номер	Назва етапів роботи	Терміни виконання	Примітка
1.	Підготовка розділу ВСТУП	30.03.2026	ПП*
2.	Аналіз практичної задачі інженерії ПЗ	06.04.2026	ПП*
3.	Аналіз вимог до ПЗ	13.04.2026	ПП*
4.	Розробка архітектури ПЗ	20.04.2026	
5.	Розробка проекту ПЗ	04.05.2026	
6.	Реалізація та тестування ПЗ	11.05.2026	
7.	Підготовка розділу Результати розробки ПЗ	15.05.2026	
8.	Підготовка розділу з охорони праці	18.05.2026	ПП*
9.	Підготовка розділу ВИСНОВКИ	22.05.2026	
10.	Оформлення списку використаних джерел та додатків	25.05.2026	
11.	Подання на кафедру ПЗАС тексту остаточного варіанту роботи, підписаного її керівником, у роздрукованому та електронному форматі разом із заявами щодо самостійності виконання роботи та ідентичності друкованої та електронної версій роботи (Додатки 1 і 2 «Порядку здійснення заходів з перевірки робіт на наявність текстових збігів/ідентичності/схожості із використанням програмно-технічних засобів», який введений в дію наказом ректора НУК за №20 від 20.01.2020 р.)	01.06.2026	не пізніше, ніж за 14 діб до захисту (згідно п.4.1 зазначеного Порядку
12.	Підготовка презентації та доповіді	05.06.2026	
13.	Попередній захист роботи на засіданні кафедри ПЗАС	08.06.2026	
14.	Подання на кафедру ПЗАС електронних версій наступних документів у форматі pdf: кваліфікаційної роботи; файлу-опису кваліфікаційної роботи (згідно Додатку до наказу ректора НУК за №287-уч від 19.05.2020 р.); презентації доповіді, а також Авторського договору з додатком	15.06.2026	

* - за результатами переддипломної практики (ПП), яка була з 02.02.2026 до 22.03.2026 р.

Студент _____ (підпис) Абу Мєхада Х. (ПІБ)

Керівник роботи _____ (підпис) Фаріонова Т.А. (ПІБ)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра присвячена розробці програмного забезпечення для автоматизації облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська, що відноситься до практичної задачі інженерії програмного забезпечення, характеризується комплексністю та невизначеністю умов.

У кваліфікаційній роботі представлено аналіз практичної задачі інженерії програмного забезпечення, розглянуто існуючі програмні продукти та розроблено постановку задачі на створення програмного забезпечення. У першому розділі проведено аналіз практичної задачі, розглянуто існуючі програмні рішення та сформульовано вимоги до нового ПЗ. У другому розділі описано процес розробки програмного комплексу: проаналізовано вимоги, побудовано модель варіантів використання, виконано етапи проєктування (ескізний, технічний і робочий проєкти). У третьому розділі наведено результати реалізації програмного комплексу. Четвертий розділ присвячено питанням охорони праці: розглянуто нормативно-правову базу, підходи до управління охороною праці у відділі з розробки програмного забезпечення зі зниження впливу шкідливих факторів.

Об’єктом даної роботи є процес розробки програмного забезпечення для автоматизації облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м.Краматорська.

Кваліфікаційна робота викладена на 106 сторінках друкованого тексту та містить 21 рисунок, 14 таблиць, список використаних джерел з 24 найменувань та 5 додатків.

Ключові слова: програмне забезпечення, автоматизація облікової діяльності, сімейна амбулаторія, Java, Spring Framework

ABSTRACT

The qualification work of the bachelor is devoted to the development of software for the automation of accounting activities of the network of family outpatient clinics of the Communal Non-Commercial Enterprise "PHC No. 1" in Kramatorsk, which refers to the practical task of software engineering, is characterized by complexity and uncertainty of conditions.

In the qualification work, the analysis of the practical problem of software engineering is presented, existing software products are considered, and the formulation of the problem for software creation is developed. In the first section, the analysis of the practical problem is carried out, the existing software solutions are considered and the requirements for the new software are formulated. The second section describes the process of developing the software package: the requirements are analyzed, a model of use cases is built, and the design stages (sketch, technical and working designs) are completed. The third section presents the results of the implementation of the software package. The fourth section is devoted to labor protection issues: the regulatory framework, approaches to labor protection management in the department for software development to reduce the impact of harmful factors are considered.

The object of this work is the process of developing software for automating the accounting activities of the network of family outpatient clinics of the Communal Non-Commercial Enterprise "PHC No. 1" in Kramatorsk.

The qualification work is presented on 106 pages of printed text and contains 21 figures, 14 tables, a list of used sources from 24 titles and 5 appendices.

Keywords: software, accounting automation, family clinic, Java, Spring Framework

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – База даних

КНП – Комунальне неприбуткове підприємство

ЦПМСД – Центр первинної медико-санітарної допомоги

MIC – Медичні інформаційні системи

CSS - Каскадні таблиці стилів

ER – Сутність-зв’язок (Entity-Relationship)

JDK – Комплект засобів розробки Java (Java Development Kit)

HTML – Мова гіпертекстової розмітки

SQL – Мова структурованих записів

UI – Користувацький інтерфейс

UML – Універсальна мова модклювання

ВСТУП

Цифрова трансформація системи охорони здоров’я в Україні зумовлює необхідність упровадження сучасних інформаційних рішень для підтримки управлінських і облікових процесів у закладах первинної медичної допомоги. У мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська щоденно обробляється значний обсяг даних, пов’язаних із реєстрацією пацієнтів, веденням облікових документів, формуванням звітності та контролем виконання планових показників. Використання розрізнених інструментів і ручних операцій призводить до дублювання інформації, затримок в обробці, підвищення ризику помилок і ускладнює отримання актуальної аналітики для прийняття рішень.

Практична задача інженерії програмного забезпечення, що розглядається в кваліфікаційній роботі, полягає у розробці програмного забезпечення для автоматизації облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська в умовах, що характеризуються різноманітністю типів облікових операцій, вимогами до цілісності та конфіденційності даних, а також потребою у швидкому формуванні звітних показників. Розробка такого ПЗ передбачає створення надійної, масштабованої та зручної системи, здатної забезпечити узгоджену роботу користувачів з даними та підтримати стандартизовані процеси обліку.

Аналіз існуючих програмних продуктів для автоматизації обліку в медичних закладах показує, що значна частина рішень є або надмірно складною та дорогою у впровадженні, або не враховує специфіку організації облікових процесів у конкретній мережі амбулаторій та особливості локальної документації. Це обґрунтовує доцільність розробки власного програмного забезпечення, адаптованого під потреби КНП «ЦПМСД №1» м. Краматорська та зорієнтованого на підвищення оперативності й точності ведення облікових даних.

Дана кваліфікаційна робота присвячена розв’язанню практичної задачі

інженерії програмного забезпечення, що характеризується комплексністю та невизначеністю умов, а саме розробці програмного забезпечення для автоматизації облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська.

Метою кваліфікаційної роботи є розробка програмного забезпечення для автоматизації облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська, що дозволить підвищити ефективність роботи за рахунок упорядкування та централізації облікових даних, скорочення часу на їх введення й обробку, забезпечення контролю повноти та коректності записів, а також мінімізації ймовірності виникнення помилок під час виконання облікових операцій.

Упровадження такого рішення також сприятиме підвищенню прозорості облікових процесів, покращенню якості підготовки звітності та оперативності управлінських рішень у закладі.

Для досягнення поставленої мети в ході дипломного проєктування необхідно виконати наступні задачі [1]:

- розглянути особливості облікових процесів у мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська та сучасні підходи до їх автоматизації;
- проаналізувати існуючі програмні засоби для автоматизації обліку та документообігу в закладах первинної медичної допомоги;
- обґрунтувати необхідність розробки власного програмного забезпечення, адаптованого до процесів і регламентів КНП «ЦПМСД №1» м. Краматорська;
- сформулювати постановку задачі та вимоги до програмного забезпечення для автоматизації облікової діяльності мережі амбулаторій;
- виконати основні етапи проєктування програмного забезпечення (ескізні, технічні, робочі) з урахуванням специфіки медичної установи;
- розглянути питання охорони праці у відділі розробки програмного

забезпечення.

- розробити необхідну програмну документацію.

Об’єктом, який розглядається у кваліфікаційній роботі, є процес розробки програмного забезпечення для автоматизації облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м.Краматорська

1 АНАЛІЗ ПРАКТИЧНОЇ ЗАДАЧІ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ОБЛІКОВОЇ ДІЯЛЬНОСТІ МЕРЕЖІ СІМЕЙНИХ АМБУЛАТОРІЙ КНП «ЦПМСД №1» М. КРАМАТОРСЬКА

1.1 Аналіз практичної задачі інженерії програмного забезпечення

Предметною галуззю кваліфікаційної роботи є облікова діяльність закладу первинної медичної допомоги. Мережа сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська забезпечує надання медичних послуг населенню та виконує значний обсяг облікових операцій: реєстрацію пацієнтів і декларацій, ведення обліку звернень та наданих послуг, планування прийому, облік направлень і результатів обстежень, формування внутрішньої та зовнішньої звітності, а також контроль виконання нормативних показників діяльності. Ці процеси охоплюють взаємодію лікарів, медичних сестер, реєстраторів та адміністрації, тому потребують узгодженого інформаційного середовища й чітких правил доступу до даних.

У сучасних умовах заклади охорони здоров’я стикаються з викликами, пов’язаними з фрагментацією даних і значною часткою ручної роботи. Зокрема, ведення обліку в різних журналах/таблицях, дублювання інформації між підрозділами, відсутність єдиних довідників (послуг, лікарів, кабінетів), складність контролю актуальності записів та обмежені можливості оперативного формування звітів призводять до втрат часу й зростання ризику помилок. Додатково актуальними є вимоги до конфіденційності персональних і медичних даних, розмежування доступу за ролями, а також необхідність забезпечення швидкої роботи системи за одночасної роботи багатьох користувачів.

З огляду на зазначені проблеми виникає потреба у розробці та

впровадженні програмного забезпечення, яке забезпечить централізований облік ключових об’єктів діяльності мережі амбулаторій та автоматизує типові операції користувачів. Практична задача, яку розв’язує кваліфікаційна робота, полягає у створенні програмної системи, здатної:

- автоматизувати ведення облікових даних (пацієнти, персонал, кабінети/амбулаторії, довідники послуг) та історію звернень;
- підтримати формування та ведення первинної облікової документації і звітності за встановленими шаблонами;
- мінімізувати кількість помилок під час внесення даних завдяки валідації, довідникам та автоматичному заповненню типових полів;
- забезпечити користувачів (реєстраторів, медичних сестер, лікарів, адміністрацію) зручними інструментами для пошуку, контролю, формування звітів та аналізу показників діяльності;
- забезпечити централізований доступ до актуальної та узгодженої інформації для всіх амбулаторій мережі з розмежуванням прав доступу.

Очікується, що впровадження такого програмного рішення сприятиме підвищенню ефективності роботи мережі амбулаторій, скороченню часу на виконання облікових операцій, покращенню якості та повноти даних, підвищенню прозорості процесів і оперативності управлінських рішень, а також зменшенню ризиків помилок під час підготовки звітності.

1.2 Аналіз існуючого програмного забезпечення та обґрунтування розробки власного ПЗ

Для обґрунтування доцільності розробки програмного забезпечення необхідно проаналізувати існуючі інформаційні системи, що застосовуються в закладах охорони здоров’я для ведення обліку, документообігу та формування звітності. У цьому підрозділі розглядаються типові класи рішень (медичні інформаційні системи, модулі електронного реєстру, системи управління прийомом і звітністю), визначаються їхні переваги та обмеження, а також

обґрунтовується необхідність створення програмного продукту, адаптованого до процесів мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська.

Медичні інформаційні системи (МІС), інтегровані з eHealth, зазвичай надають базовий набір функцій для реєстрації пацієнтів, ведення декларацій, планування прийому, ведення електронних медичних записів, формування звітів та обміну даними з державними сервісами. Такі рішення можуть суттєво спростити роботу медичного персоналу, однак нерідко потребують адаптації до локальних регламентів обліку, внутрішніх форм документів, структури мережі амбулаторій та специфічних управлінських показників.

Функціональні можливості типових медичних інформаційно-програмних систем:

- надання сервісів для реєстратури та пацієнтів:
  електронна реєстрація пацієнтів та ведення базових даних;
  онлайн-запис на прийом та управління розкладом лікарів;
  SMS/e-mail інформування пацієнтів (нагадування про візит, повідомлення про зміну часу прийому);
- облік медичних послуг, а саме ведення довідників послуг, лікарів, кабінетів та підрозділів (модулі для обліку звернень/візитів, направлень та результатів обстежень);
- звітність та аналітика (автоматичне формування звітів за періодами, підрозділами, лікарями та видами послуг);
- контроль повноти заповнення облікових записів і виявлення розбіжностей у даних.
- аналітичні панелі (дашборди) для моніторингу навантаження та ключових показників.
- інфраструктура та безпека за рахунок розмежування доступу за ролями (лікар, медсестра, реєстратор, адміністратор) та формування журналу дій користувачів);
- підтримка резервного копіювання, експорту звітів (PDF/CSV) та, за

потреби, інтеграцій з зовнішніми сервісами.

Зазначені можливості демонструють, що типові МІС охоплюють значну частину процесів первинної медичної допомоги, проте їх функціонал та інтерфейс можуть не повністю відповідати внутрішнім процедурам ведення обліку в конкретній мережі амбулаторій, а також вимогам щодо форми і змісту локальних звітів (див. рис. 1.1 – 1.3).

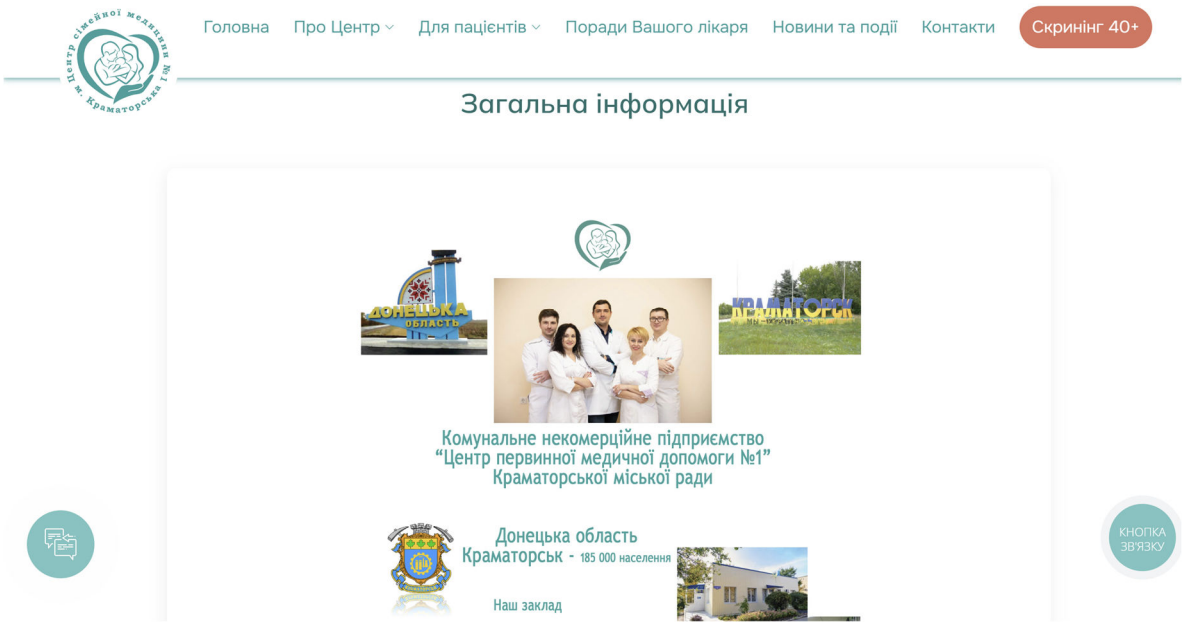

Рисунок 1.1– Головна сторінка сайту [2]

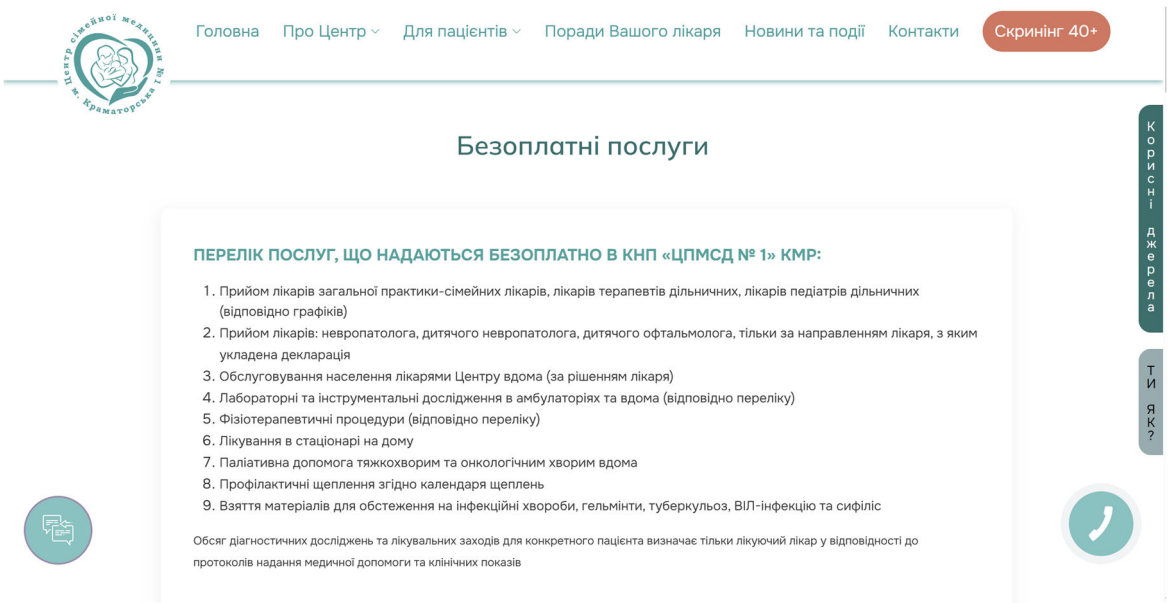

Рисунок 1.2 – Перелік послуг [2]

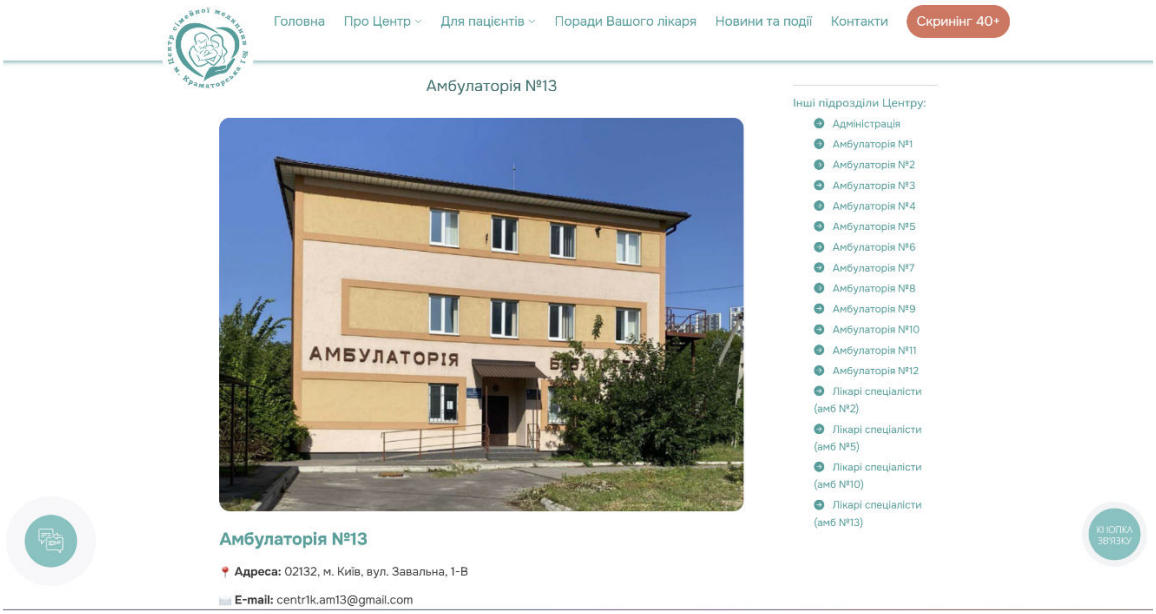

Рисунок 1.3 – Сторінка з контактами [2]

Переваги:

- скорочення часу на облік та оформлення документів за рахунок шаблонів і довідників;
- зменшення людського фактору та кількості помилок завдяки валідації та контролю заповнення;
- централізація даних і прозорість облікових операцій у межах мережі амбулаторій;
- оперативне формування звітності для внутрішнього контролю та зовнішніх потреб (за регламентами);
- можливість масштабування системи на декілька амбулаторій та підтримка ролей користувачів.

Недоліки:

- вартість впровадження та супроводу (ліцензії, сервери, адміністрування, підтримка);
- необхідність навчання персоналу та налаштування процесів роботи із системою;
- ризики простоїв у разі збоїв інфраструктури або відсутності стабільного

інтернет-з’єднання (для хмарних рішень);

- підвищені вимоги до захисту персональних і медичних даних та організації контролю доступу;
- обмежена гнучкість налаштувань у частині специфічних локальних форм обліку та нестандартних звітів.

Висновок. Типові МІС здатні суттєво покращити організацію обліку та документообігу в закладах первинної медичної допомоги, однак для мережі сімейних амбулаторій важливо мати рішення, яке враховує конкретні регламенти, структуру підрозділів, локальні шаблони документів і управлінські потреби. Саме тому розробка програмного забезпечення, адаптованого під КНП «ЦПМСД №1» м. Краматорська, є доцільною.

Універсальні системи документообігу та управління ресурсами (ERP/ECM) можуть застосовуватися у медичних закладах для обліку персоналу, договорів, наказів, фінансових документів та внутрішніх заявок. Вони забезпечують гнучкі механізми погодження, зберігання та пошуку документів, але не завжди містять предметно-орієнтовані модулі для обліку прийому пацієнтів, медичних послуг і специфічної звітності первинної ланки. (див. рис. 1.4 – 1.6).

Функціональні можливості:

- електронний документообіг: реєстрація, маршрутизація та погодження внутрішніх документів;
- облік ресурсів і довідників: співробітники, підрозділи, контрагенти, шаблони документів;
- аналітика та звітність: формування зведених звітів за документами і процесами погодження;
- веб-інтерфейс та контроль доступу: підтримка ролей, журналювання, політики безпеки. Такі системи корисні для загального документообігу, однак потребують інтеграції або доопрацювання для медичного обліку.

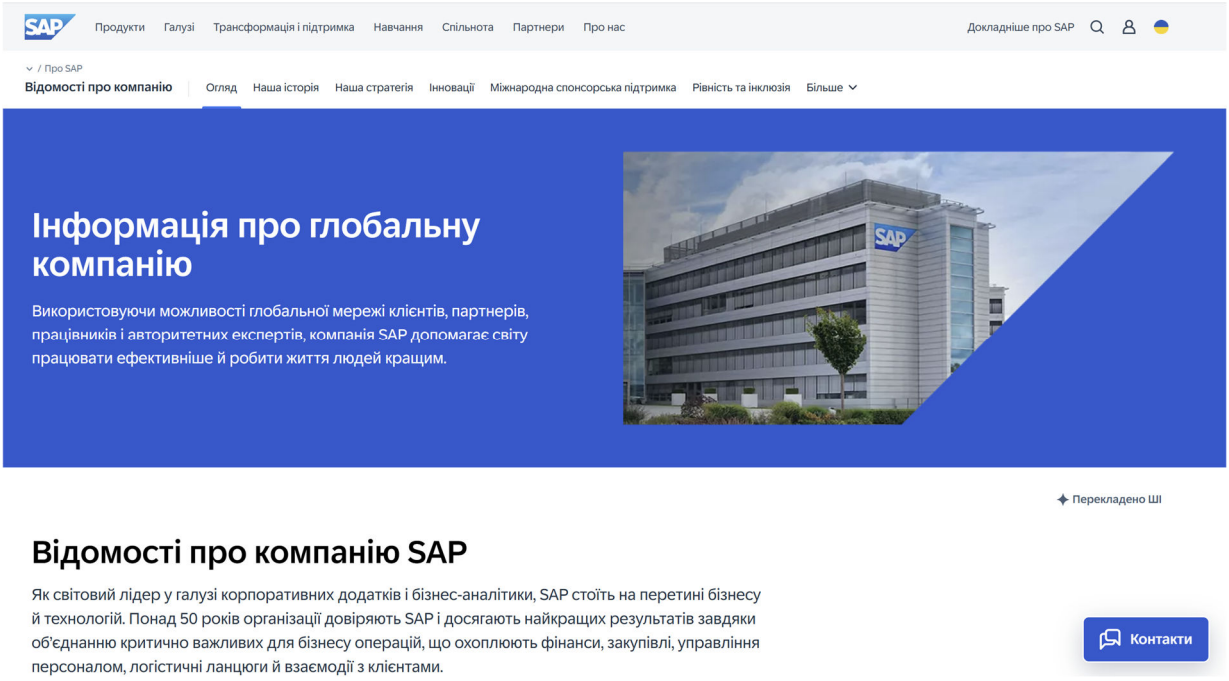

Рисунок 1.4 – Головна сторінка сайту [3]

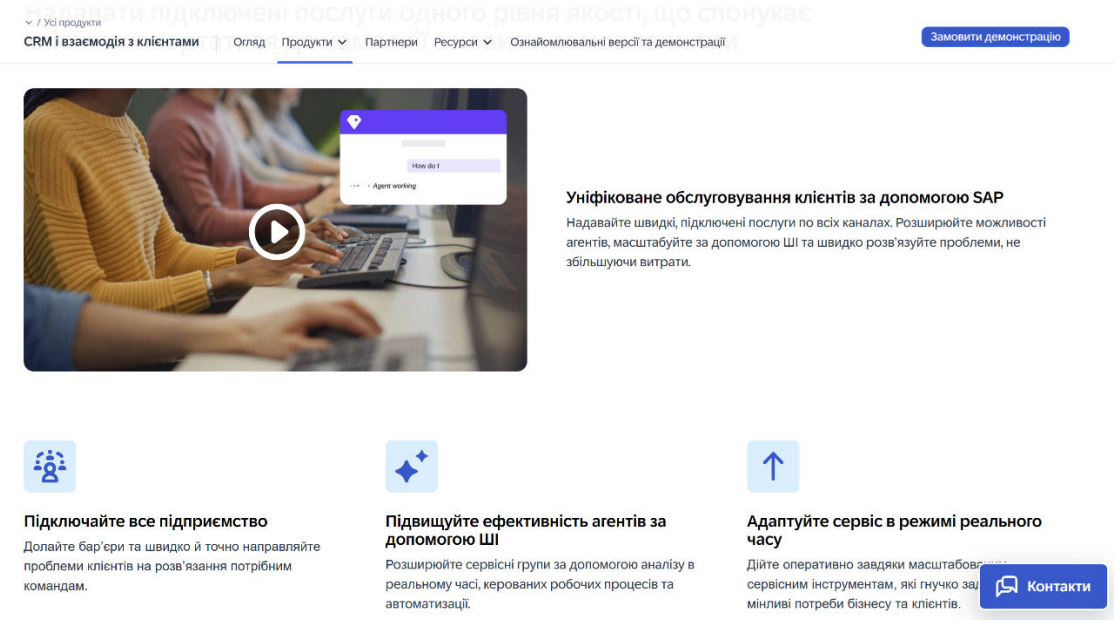

Рисунок 1.5 – Сторінка управління взаємодією з клієнтами [3]

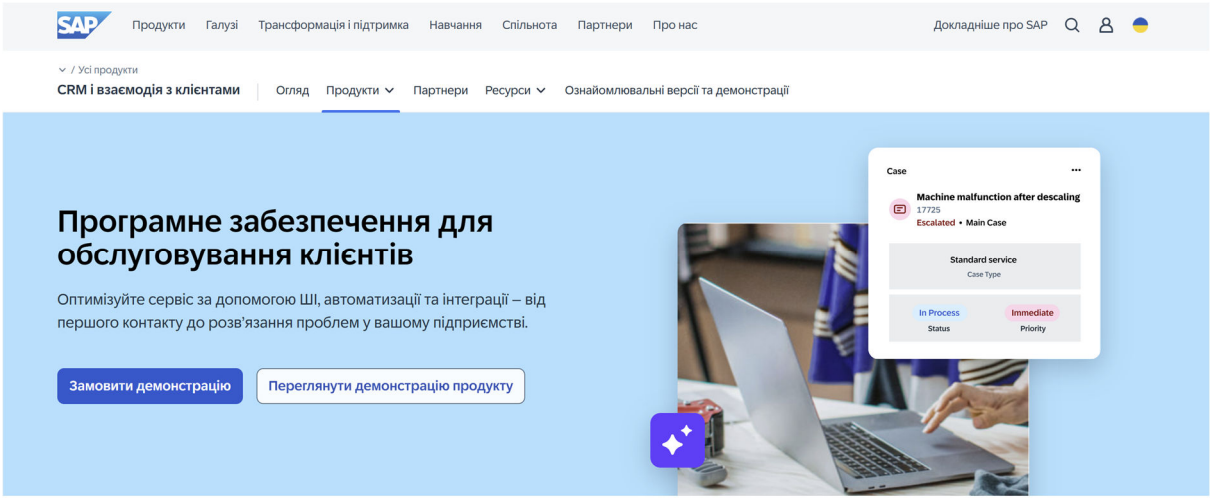

Рисунок 1.6 – Сторінка взаємодії з клієнтами [3]

Переваги:

- гнучке налаштування бізнес-процесів погодження та зберігання документів;
- централізоване зберігання документів і швидкий пошук за реквізитами;
- наявність механізмів контролю доступу, ведення журналів та політик інформаційної безпеки;
- можливість інтеграції з іншими підсистемами закладу (кадри, фінанси, архів).

Недоліки:

- висока складність впровадження та потреба у тривалому налаштуванні процесів;
- необхідність залучення фахівців для інтеграції та супроводу;
- недостатня предметна спеціалізація щодо медичного обліку первинної ланки без додаткових модулів;
- підвищені вимоги до серверних ресурсів та адміністрування у разі масштабного розгортання.

Висновок. Універсальні ECM/ERP-рішення є корисними для організації загального документообігу та управління ресурсами, проте для автоматизації

облікової діяльності мережі сімейних амбулаторій потрібна спеціалізація під медичні процеси (облік візитів, послуг, звітності) та можливість швидко адаптувати форми й правила ведення даних. Це підсилює обґрунтування розробки власного програмного забезпечення для потреб КНП «ЦПМСД №1» м. Краматорська.

1.3 Визначення вимог до програмного забезпечення

Ґрунтуючись на результатах проведеного аналізу практичної задачі інженерії програмного забезпечення у сфері первинної медичної допомоги, було прийнято рішення розробити власне програмне забезпечення для автоматизації облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська. Виходячи з потреб користувачів та обмежень готових рішень, сформулюємо постановку задачі: потрібно розробити застосунок, який забезпечить централізований облік даних, підтримку типових операцій і формування звітності та задовольнятиме наступним вимогам.

Функціональні вимоги до програмного забезпечення:

- додавання, редагування та видалення інформації про пацієнтів, працівників, амбулаторії/кабінети та довідники медичних послуг;
- реєстрація звернень/візитів пацієнтів та ведення історії облікових записів за прийомами;
- формування звітів за обліковими даними (з фільтрами за періодом, амбулаторією, лікарем, видом послуги тощо);
- підтримка друку та/або експорту облікових форм і звітів у PDF/CSV;
- розмежування прав доступу до даних за ролями та ведення журналу дій користувачів.

Нефункціональні вимоги програмного забезпечення:

- зручний та інтуїтивно зрозумілий інтерфейс для користувачів з різним рівнем підготовки;

- підтримка української мови та уніфікованих довідників/термінів у межах системи;
- час відповіді системи не перевищує 1 секунди при стандартних операціях пошуку/збереження записів за нормального навантаження.

Вимоги до складу й параметрів технічних засобів:

- мінімальні ресурси клієнтського ПК: 4 ГБ ОЗП, 2-ядерний процесор 2 ГГц або вище, SSD з вільним простором не менше, ніж 10 Гб;
- відеокарта, вбудована у процесор, або дискретна – обсяг відеопам'яті не менше ніж 512 Мб з відеовиходом HDMI;
- монітор Full HD – 1920x1080 з можливістю підключення до HDMI порту відеокарти;
- мережева карта з підтримкою швидкості передавання даних не менше ніж 100 Мбіт/с;
- можливість підключення до локального принтера для друку облікових документів.

Вимоги до інформаційної та програмної сумісності:

- можливість формування та надсилання звітів електронною поштою (за потреби);
- при використанні веб-інтерфейсу – сумісність із сучасними браузерами (Chrome, Firefox, Edge);
- ОС: Windows 11 або вище (для клієнтського робочого місця);
- СУБД: MySQL для мережевого розгортання;
- підтримка експорту звітів у PDF та CSV.

Детальні вимоги до програмного забезпечення наведені у Додатку А – Технічне завдання.

Висновки за розділом 1

1 У розділі 1 виконано аналіз практичної задачі автоматизації облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська

та визначено ключові проблеми, пов’язані з фрагментацією даних, дублюванням записів і високою часткою ручних операцій.

2 Розглянуто типові програмні рішення (МІС, ECM/ERP) і встановлено, що вони не завжди враховують локальні регламенти, структуру підрозділів і специфіку звітності первинної ланки.

3 На підставі отриманих результатів обґрунтовано доцільність розробки власного вебзастосунку та сформовано вимоги до системи: централізація довідників і даних, підтримка запису на прийом, ведення електронної медичної картки та рецептів, формування звітів, а також забезпечення ролей доступу, валідації та сумісності з типовим робочим середовищем.

2 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ОБЛІКОВОЇ ДІЯЛЬНОСТІ МЕРЕЖІ СІМЕЙНИХ АМБУЛАТОРІЙ

2.1 Аналіз вимог до програмного забезпечення

Побудова моделі варіантів використання

Для побудови моделі варіантів використання програмного забезпечення мережі сімейних амбулаторій насамперед потрібно визначити основних акторів, які взаємодіятимуть із системою. Актори – це зовнішні по відношенню до системи ролі, що ініціюють різні процеси. У таблиці 2.1 подано стислий опис основних акторів, які братимуть участь у роботі з програмним комплексом.

Таблиця 2.1 – Актори програмного комплексу

Актор	Короткий опис
Пацієнт	Може записуватися на прийом, переглядати власну медичну інформацію, змінювати або скасовувати записи, а також отримувати консультації дистанційно.
Лікар	Має доступ до електронних медичних карт пацієнтів, проводить прийоми, оформлює медичні записи, призначає лікування, створює електронні рецепти та веде статистику.
Адміністратор	Керує розкладом лікарів, реєструє пацієнтів, налаштовує доступи, веде облік ресурсу амбулаторії, формує звіти та статистику для керівництва.

Після визначення акторів було сформовано варіанти використання, що відображають основні сценарії роботи системи. Для кожного з варіантів використання указано, які актори беруть участь у ньому. Згідно із вимогами замовника, медичні сестри сімейних амбулаторій є представниками лікарів, тому окремої сутності – актора. У таблиці 2.2 наведено перелік визначених варіантів за користувачами та їх короткий опис.

Таблиця 2.2 – Варіанти використання програмного комплексу

Актор	Назва	Короткий опис
Пацієнт, Лікар, Адміністратор	Запис на прийом	Пацієнти можуть записатися на прийом до лікаря, лікарі та адміністратори можуть додавати записи вручну або через електронний журнал.
Пацієнт, Лікар, Адміністратор	Скасування/редагування запису	Усі користувачі можуть скасовувати або редагувати записи на прийом, якщо це необхідно.
Пацієнт	Редагування/видалення особистих даних	Пацієнти можуть змінювати чи видаляти власні персональні дані у системі.
Лікар, Адміністратор	Ведення електронної медичної картки	Лікарі та адміністратори можуть створювати, оновлювати та видаляти записи в електронних медичних картках пацієнтів.
Лікар	Виписка електронних рецептів	Лікарі можуть створювати та надсилати електронні рецепти пацієнтам.
Адміністратор	Формування звітів та статистики	Адміністратори мають можливість формувати різноманітні звіти щодо роботи амбулаторій, кількості прийомів, звернень та іншої статистичної інформації.

Рисунок 2.1 – Діаграма варіантів використання ПЗ

Наступним кроком деталізуємо виявлені прецеденти шляхом розробки специфікації варіантів використання.

Специфікації варіантів використання

У межах моделювання варіантів використання програмного комплексу для автоматизації облікової діяльності мережі сімейних амбулаторій були визначені основні бізнес-процеси та відповідні сценарії взаємодії користувачів із системою. Специфікації варіантів використання згруповані відповідно до ключових потоків обробки інформації. Вони подані у вигляді таблиць, що деталізують цілі, учасників, передумови, основні та альтернативні сценарії кожного випадку використання. У таблицях 2.3 - 2.9 наведено специфікації визначених (рис. 2.1) варіантів використання.

Таблиця 2.3 – Специфікація варіанту використання «Виписка електронних рецептів»

Складова	Опис
Опис прецеденту	Лікар формує та надсилає електронний рецепт пацієнту через систему.
Дійові особи прецеденту	Лікар, пацієнт
Передумови	Пацієнт звернувся до лікаря для отримання рецепту на лікарські засоби.
Потік подій	1) Лікар авторизується у системі. 2) Відкриває розділ електронних рецептів. 3) Обирає пацієнта зі списку. 4) Заповнює форму рецепту (назва препарату, дозування, термін дії). 5) Натискає «Виписати рецепт». 6) Система надсилає рецепт пацієнту у електронному вигляді.
Альтернативні потоки	Можливість виписати рецепт для кількох пацієнтів одночасно (наприклад, для сімейного обслуговування).
Постумови	Рецепт зберігається у системі та доступний для подальшого використання (наприклад, для перевірки аптекою чи повторного призначення).

Таблиця 2.4 – Специфікація варіанту використання «Запис на прийом»

Опис прецеденту	Пацієнт записується на прийом до лікаря через електронну систему.
Дійові особи прецеденту	Пацієнт, лікар
Передумови	Пацієнт має доступ до системи та зареєстрований у ній.
Потік подій	1) Пацієнт авторизується у системі. 2) Обирає лікаря та дату прийому. 3) Заповнює необхідні дані. 4) Підтверджує запис. 5) Система надсилає підтвердження запису.
Альтернативні потоки	Можливість запису для кількох пацієнтів (наприклад, для дітей чи інших членів родини).
Постумови	Запис зберігається у системі, інформація доступна пацієнту та лікарю.

Таблиця 2.5 – Специфікація варіанту використання «Скасування/редагування запису»

Опис прецеденту	Пацієнт або адміністратор скасовує чи редагує запис на прийом через електронну систему.
Дійові особи прецеденту	Пацієнт, адміністратор
Передумови	Існує запис на прийом, який потрібно змінити або скасувати.
Потік подій	1) Користувач авторизується у системі. 2) Обирає запис для редагування чи скасування. 3) Вносить зміни або підтверджує скасування. 4) Система оновлює інформацію та надсилає повідомлення.
Альтернативні потоки	Можливість адміністратору змінювати запис за іншого користувача.
Постумови	Запис оновлено або видалено, інформація актуальна у системі.

Таблиця 2.6 – Специфікація варіанту використання «Ведення електронної медичної картки»

Опис прецеденту	Лікар вносить та оновлює медичну інформацію пацієнта в електронній картці.
Дійові особи прецеденту	Лікар, пацієнт
Передумови	Пацієнт має електронну медичну картку у системі.
Потік подій	1) Лікар авторизується у системі. 2) Відкриває медичну картку пацієнта. 3) Додає або оновлює інформацію (діагнози, лікування, аналізи). 4) Система зберігає зміни та надає доступ пацієнту.
Альтернативні потоки	Пацієнт може переглядати свою медичну картку онлайн.
Постумови	Медична інформація актуальна та доступна для подальшого використання.

Таблиця 2.7 – Специфікація варіанту використання «Редагування/видалення особистих даних»

Опис прецеденту	Користувач редагує або видаляє власні особисті дані у системі.
Дійові особи прецеденту	Пацієнт, адміністратор
Передумови	Користувач має профіль у системі та права на редагування/видалення даних.
Потік подій	1) Користувач авторизується у системі. 2) Відкриває розділ особистих даних. 3) Вносить зміни або видаляє інформацію. 4) Система зберігає оновлені дані або видаляє їх із профілю.
Альтернативні потоки	Адміністратор може редагувати/видаляти дані за запитом користувача або у випадку порушення правил.
Постумови	Особисті дані актуальні або видалені, система містить лише достовірну інформацію.

Таблиця 2.8 – Специфікація варіанту використання «Формування звітів та статистики»

Складова	Опис
Опис прецеденту	Адміністратор формує різноманітні звіти та статистику щодо діяльності амбулаторії.
Дійові особи прецеденту	Адміністратор
Передумови	Існує накопичена база даних щодо прийомів, звернень, електронних рецептів та іншої інформації.
Потік подій	1) Адміністратор авторизується у системі. 2) Відкриває розділ «Звіти та статистика». 3) Обирає тип звіту (наприклад, кількість прийомів, звернень, виписаних рецептів). 4) Встановлює період та додаткові параметри. 5) Натискає «Сформувати звіт». 6) Система генерує та відображає звіт, надає можливість експорту у PDF, Excel чи інший формат.
Альтернативні потоки	Можливість автоматичного формування звітів за розкладом, або інтеграції з зовнішніми системами для аналізу даних.
Постумови	Звіт зберігається у системі та доступний для перегляду, аналізу та подальшого використання (наприклад, для подання у державні органи).

2.2 Архітектура програмного забезпечення

Одним із ключових етапів проєктування програмного забезпечення є вибір архітектури, яка забезпечить необхідні атрибути якості – масштабованість, безпечність, зручність використання, супроводжуваність тощо. Було проаналізовано кілька архітектурних стилів і обґрунтовано вибір найбільш придатного варіанту.

Для прийняття обґрунтованого рішення щодо архітектурного стилю було проведено порівняльний аналіз чотирьох популярних підходів. В таблиці 2.9 подано оцінку їх відповідності ключовим атрибутам якості системи.

Таблиця 2.9 – Порівняльний аналіз архітектурних стилів за атрибутами якості

Атрибут якості	Багатошарова архітектура абстракцій	Архітектура потоків даних	Архітектура MVC	Мікросервісна архітектура
Зручність використання	+2	-1	+2	+2
Безпечність	+2	-1	+1	+2
Супроводжуваність	+1	+1	+1	+1
Ефективність	0	0	+2	+1
Надійність	+1	-2	+2	+1
Масштабованість	0	-1	+1	+1
Переносимість	+1	+2	+1	+1

Архітектура MVC (Model-View-Controller) була обрана як оптимальна з огляду на її структурованість та численні переваги, що забезпечують ефективність розробки, підтримки та масштабування програмного забезпечення. Основною причиною вибору цієї архітектури є її здатність чітко розділяти функціональні аспекти системи на три незалежні компоненти: модель, яка відповідає за дані та логіку їх обробки; вид, який забезпечує відображення інформації користувачу; та контролер, що керує взаємодією між моделлю та видом. Такий підхід дозволяє підвищити гнучкість і зручність внесення змін, а також спрощує процес тестування та супроводу системи.

Багатошарова архітектура абстракцій передбачає розділення системи на декілька шарів, кожен з яких виконує свою окрему функцію. Такий підхід створює чітку структуру взаємодії між компонентами, завдяки чому підвищується безпечність та супроводжуваність програмного продукту. Зокрема, багатошарова архітектура дозволяє уникати прямого доступу до даних з боку користувача, забезпечуючи логічний контроль над процесами та підвищуючи захищеність системи від несанкціонованих втручань. Це сприяє

підвищенню надійності та довготривалої підтримки програмного забезпечення.

Висока підтримка безпечності (оцінка +2) та супроводжуваності (+1) обумовлена тим, що багатошарова архітектура дозволяє чітко розмежувати функціональні обов’язки між різними рівнями системи. Це мінімізує ризик помилок при внесенні змін та забезпечує ефективний контроль над логікою роботи програмного продукту. У разі оновлення або модифікації одного з шарів зміни не впливають на інші компоненти, що значно спрощує процес розробки та підтримки.

Масштабованість у багатошаровій архітектурі має певні обмеження (оцінка 0), оскільки горизонтальне масштабування шарів часто пов’язане зі складністю синхронізації та розподілу навантаження між ними. Це може ускладнювати розширення системи, особливо за умов зростання кількості користувачів або функціональності. Проте вертикальне масштабування та модифікація окремих шарів залишаються достатньо простими завдяки чіткому розділенню логіки.

Архітектура потоків даних орієнтована на організацію обробки та передачі даних між модулями у вигляді потоків, що дозволяє досягти високої модульності та переносимості рішень.

Недостатній рівень безпечності (оцінка -1) пов’язаний зі складністю контролю за передачею даних між потоками. Це може призводити до виникнення помилок у випадках некоректної синхронізації або обробки інформації, а також створює додаткові ризики для захищеності системи.

Високий рівень переносимості (+2) пояснюється модульністю даних, яка дозволяє легко адаптувати систему до нових умов чи змін середовища. Архітектура потоків даних забезпечує простоту інтеграції та розширення функціоналу, що особливо актуально для сучасних інформаційних систем.

Модель-вид-контролер (MVC) є класичним підходом до побудови програмних систем, що дозволяє чітко розділити ролі та відповідальність між компонентами.

Чітке розділення ролей (+2 для зручності) забезпечує прозорість взаємодії між користувачем, бізнес-логікою та базою даних. Це сприяє легкому внесенню змін до окремих частин системи без ризику порушення її роботи в цілому, а також оптимізує процес розробки та тестування.

Вищий рівень супроводжуваності (+1) досягається завдяки ізоляції бізнес-логіки, що дозволяє розробникам оперативно реагувати на зміни вимог або виправлення помилок, не зачіпаючи інші компоненти системи.

Мікросервісна архітектура базується на розподілі системи на незалежні сервісні компоненти, кожен з яких виконує певну функцію та може масштабуватися окремо.

Висока підтримка супроводжуваності (+1) та масштабованості (+1) забезпечується незалежністю компонентів: кожен сервіс можна оновлювати, масштабувати чи замінювати без впливу на інші частини системи. Це особливо важливо для великих проєктів із складною логікою та великою кількістю користувачів.

Високий рівень безпечності (+2) досягається завдяки можливості ізоляції кожного сервісу, що дозволяє обмежити доступ до даних та функціоналу, запобігати поширенню помилок і забезпечувати ефективний захист від зовнішніх загроз.

Таким чином для проєкту найкраще підходить архітектура MVC через її сильні сторони:

- чітке розділення відповідальностей. Архітектура MVC забезпечує структурний поділ системи на три незалежні компоненти – Model, View та Controller. Це мінімізує зв’язність між частинами застосунку та спрощує підтримку, тестування й подальший розвиток;

- покращена підтримуваність. Завдяки ізоляції бізнес-логіки (Model) від інтерфейсу користувача (View) зміни у відображенні даних не потребують модифікації логіки обробки, і навпаки. Це знижує ризик помилок та прискорює внесення змін;

- масштабованість інтерфейсів. MVC дозволяє легко додавати нові типи клієнтів (вебінтерфейс, мобільний застосунок, панель адміністратора), використовуючи спільну модель даних та контролери. Це забезпечує єдину точку доступу до бізнес-логіки;

- паралельна розробка. Команди можуть працювати над Model, View і Controller незалежно одна від одної. Це підвищує продуктивність розробки та зменшує конфлікти при інтеграції;

- гнучкість у виборі технологій. View може бути реалізований за допомогою різних шаблонізаторів або UI-фреймворків, тоді як Model і Controller можуть використовувати інші технологічні стеки. Це дозволяє адаптувати систему до вимог конкретного середовища;

- покращене тестування. Окреме тестування моделей і контролерів значно спрощується, оскільки вони не залежать від інтерфейсу користувача. Це підвищує надійність системи та якість програмного забезпечення.

На рисунку 2.2 наведено діаграму компонентів системи, яка ілюструє основні складові архітектури, а також взаємозв’язки між ними.

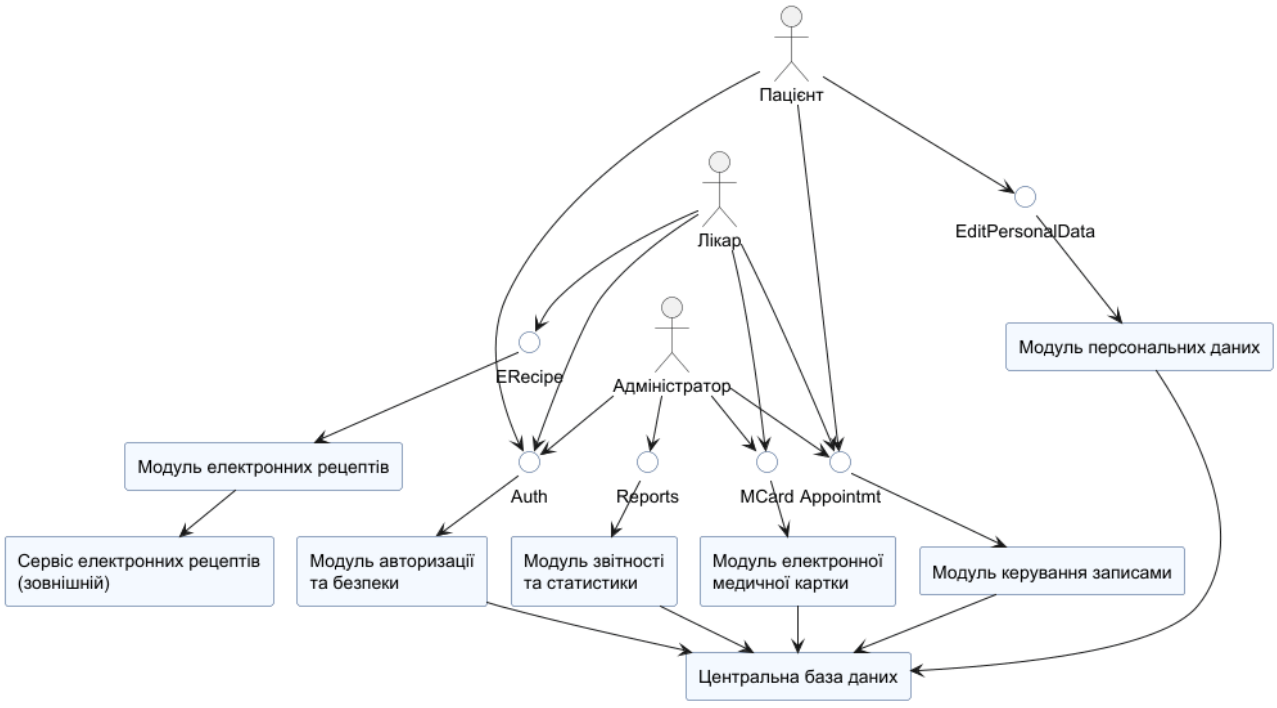

Рисунок 2.2 – Діаграма компонентів програмної системи

Ґрунтуючись на діаграмі, система представлена вебзастосунком та складається з окремих модулів, зокрема: модуль авторизації та безпеки, модуль персональних даних, модуль керування записами, модуль електронних рецептів, модуль звітності та статистики, модуль електронної медичної картки, та централізованої бази даних.

Кожен компонент виконує чітко визначену роль, що відповідає принципам архітектури MVC. Це дозволяє досягти високої гнучкості, розділення відповідальності, а також забезпечує ізоляцію збоїв між окремими частинами системи.

На рисунку 2.3 зображено діаграму розгортання, яка демонструє, як саме розміщуються компоненти системи на фізичних або віртуальних вузлах інфраструктури.

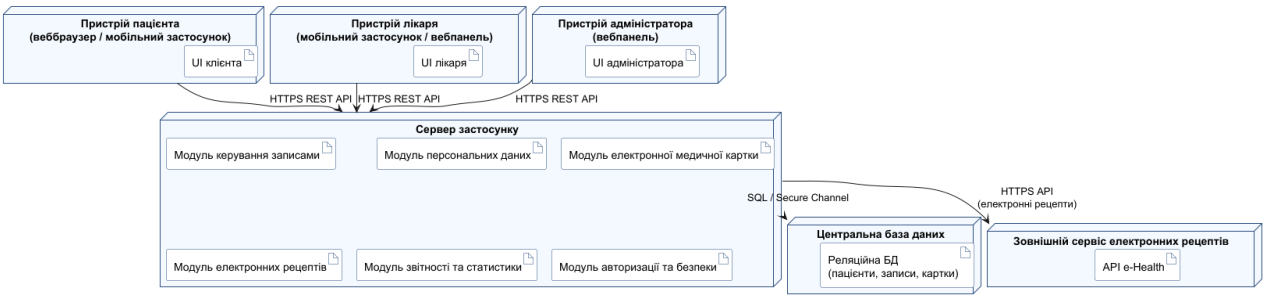

Рисунок 2.3 – Діаграма розгортання ПЗ

На рисунку можна побачити фізичну та логічну структуру ПЗ облікової діяльності сімейної амбулаторії, включно з основними вузлами (клієнтські пристрої, сервер застосунку, центральна база даних, зовнішні сервіси) та каналами взаємодії між ними. Архітектура побудована за багатошаровим принципом, що забезпечує чітке розділення відповідальностей, масштабованість та підвищену надійність.

Клієнтський шар

До клієнтського шару належать: *Пристрій пацієнта* (веббраузер або мобільний застосунок), *Пристрій лікаря* (мобільний застосунок або вебпанель), *Пристрій адміністратора* (вебпанель).

На кожному з цих пристроїв розгорнуто відповідний **користувацький**

інтерфейс (UI), який реалізує функції взаємодії користувача із ПЗ: запис на прийом, перегляд та редагування даних, ведення медичної документації, формування звітів тощо. Клієнтський шар не містить бізнес-логіки системи, а виступає переважно як тонкий клієнт, що надсилає запити до серверного шару та відображає отримані результати.

Серверний шар (сервер застосунку)

Сервер застосунку є центральним елементом архітектури, на якому розгорнуто основні програмні модулі: *Модуль керування записами* – обробка операцій запису, скасування та редагування прийомів; *Модуль персональних даних* – управління обліковими записами пацієнтів, їх персональною інформацією та налаштуваннями; *Модуль електронної медичної картки* – створення, оновлення та зберігання медичних записів, історії хвороби, результатів обстежень; *Модуль електронних рецептів* – формування та передавання електронних рецептів до зовнішнього сервісу; *Модуль звітності та статистики* – агрегування даних, формування аналітичних звітів щодо роботи амбулаторій, прийомів, звернень тощо; *Модуль авторизації та безпеки* – автентифікація користувачів, керування ролями (пацієнт, лікар, адміністратор), контроль доступу до ресурсів, аудит дій.

Серверний шар реалізує бізнес-логіку системи, забезпечує цілісність даних, узгодженість транзакцій та централізоване застосування політик безпеки.

Шар даних (центральна база даних)

Центральна база даних представлена окремим вузлом, на якому розгорнуто реляційну СУБД. У ній зберігаються: *дані про пацієнтів* (персональні дані, контактна інформація); *записи про прийоми* (дата, час, лікар, статус); *електронні медичні картки* (діагнози, призначення, результати обстежень); *службові дані для звітності та аудиту*.

Шар даних ізольований від клієнтських пристроїв: доступ до нього здійснюється виключно через сервер застосунку. Це забезпечує централізований контроль доступу, підвищує безпеку та спрощує

адміністрування.

Зовнішні сервіси

Окремим вузлом представлено **зовнішній сервіс електронних рецептів** (наприклад, державний або комерційний e-Health-сервіс). Взаємодія з ним здійснюється через стандартизований **API**, що дозволяє: надсилати сформовані електронні рецепти; отримувати підтвердження про успішну реєстрацію рецепта; за потреби – виконувати додаткові перевірки (валідація, статус рецепта тощо).

Цей підхід спрощує інтеграцію з національною або галузевою інформаційними системами та забезпечує відповідність нормативним вимогам.

Протоколи та канали взаємодії

Комунікація між клієнтськими пристроями (пацієнта, лікаря, адміністратора) та сервером застосунку здійснюється за допомогою протоколу HTTPS, який забезпечує шифрування трафіку, конфіденційність та цілісність даних, що передаються мережею та REST-орієнтованого API – запити формуються у вигляді HTTP-методів (GET, POST, PUT, DELETE), що дозволяє стандартизувати доступ до ресурсів (записи, пацієнти, картки, звіти).

Такий підхід забезпечує технологічну незалежність клієнтів (веб, мобільні застосунки) та спрощує розширення системи новими інтерфейсами.

Зв’язок між сервером застосунку та центральною базою даних реалізовано через SQL-протокол поверх захищеного каналу (наприклад, TLS-шифрування на рівні з’єднання).

Сервер застосунку виконує роль єдиного посередника між бізнес-логікою та даними, що дозволяє: централізовано керувати транзакціями; застосовувати політики цілісності та узгодженості; реалізувати механізми резервного копіювання та відновлення.

Комунікація з зовнішнім сервісом електронних рецептів здійснюється через HTTPS-протокол для захищеного обміну даними та REST API – для надсилання структурованих запитів (наприклад, у форматі JSON або XML) та отримання відповідей.

Це дозволяє інтегрувати систему з національними e-Health-платформами; забезпечити юридично значимий обіг електронних рецептів; розширювати функціональність без модифікації внутрішньої структури системи.

На рівні архітектури розгортання передбачено: автентифікацію та авторизацію на сервері застосунку (модуль авторизації та безпеки); шифрування трафіку між клієнтами, сервером застосунку та зовнішніми сервісами (HTTPS); ізоляцію шару даних від прямого доступу ззовні (доступ лише через сервер застосунку); можливість розміщення різних вузлів у різних мережевих сегментах (наприклад, сервер застосунку в DMZ, база даних – у внутрішній захищеній мережі).

2.3 Ескізний проєкт програмного забезпечення

У рамках ескізного проєкту з метою показати основні етапи користувацького сценарію, а також виявити можливі альтернативні шляхи виконання дій були обрані три ключові варіанти використання:

«Запис на прийом», «Ведення електронної медичної картки» та «Виписка електронного рецепта».

Діаграма діяльності для варіанту «Запис на прийом» зображена на рисунку 2.4.

Рисунок 2.4 – Діаграма діяльності для варіанту «Запис на прийом»

На рисунку 2.5 наведені діаграми діяльності для варіантів «Ведення електронної медичної картки» та «Виписка електронного рецепта».

Рисунок 2.5 – Діаграми діяльності для варіантів «Ведення електронної медичної картки» та «Виписка електронного рецепта»

Розробка моделей даних програмного забезпечення для автоматизації облікової діяльності мережі сімейних амбулаторій.

У межах ескізного проєктування було сформульовано ключові сутності та зв’язки моделі даних програмного забезпечення для автоматизації облікової діяльності мережі сімейних амбулаторій. Для побудови логічної моделі даних

було використано програмний засіб DBeaver[7]. Логічна модель даних забезпечує формалізований опис інформаційних об’єктів предметної області, їх атрибутів та взаємозв’язків, що дозволяє узгодити вимоги користувачів із майбутньою структурою бази даних, зменшити дублювання інформації, забезпечити цілісність записів і спростити подальшу реалізацію фізичної схеми у СУБД.

Логічна модель охоплює такі основні групи сутностей: користувачі системи (пацієнти та працівники), організаційна структура амбулаторій, довідникові дані (послуги, спеціальності, статуси), облікові події (записи на прийом, візити), медична документація (записи електронної картки, рецепти) та службові дані безпеки (ролі, права, журнал дій).

Логічна модель даних, збудована у засобі Dbeaver, показана на рисунку 2.6.

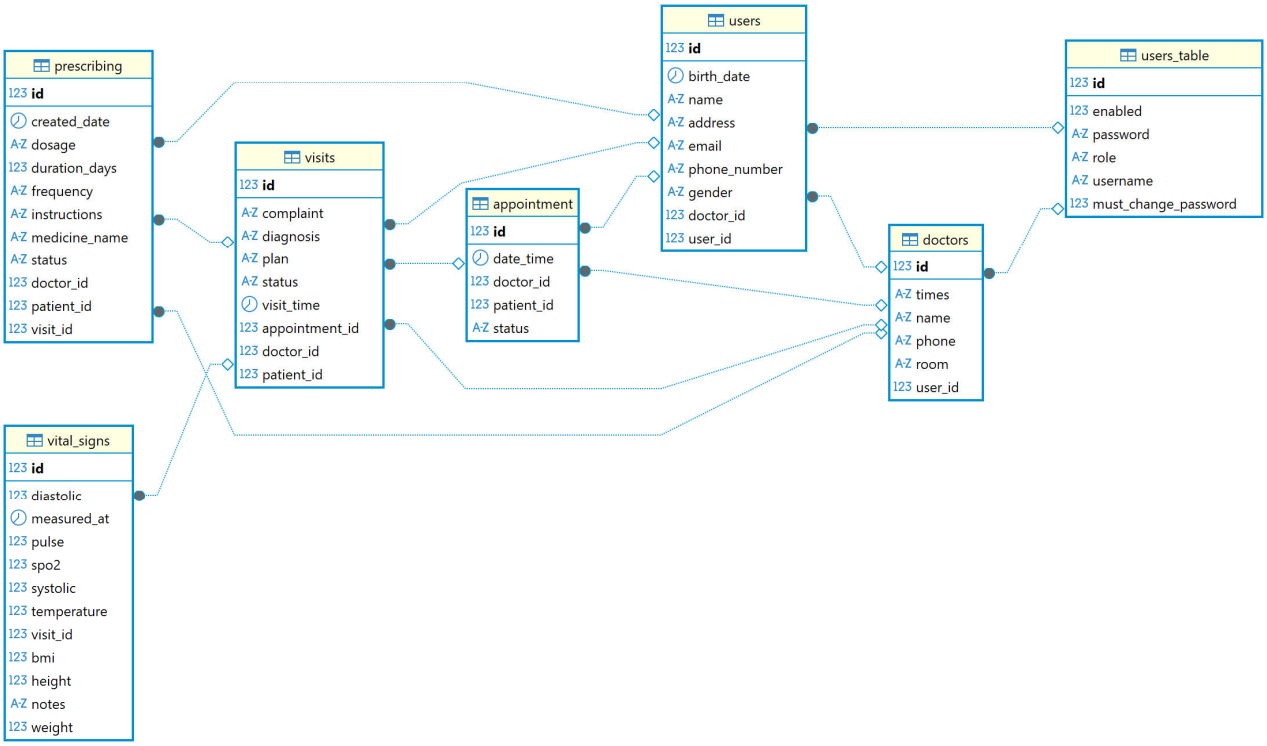

Рисунок 2.6 – Логічна модель даних

Нижче наведено опис ключових сутностей та їхнє призначення.

Сутність *users* (Пацієнти)

Містить ідентифікаційну, контактну та демографічну інформацію, яка необхідна для організації прийомів та ведення медичної документації.

Атрибути:

- id – унікальний ідентифікатор пацієнта.
- name – ПІБ пацієнта.
- birth_date – дата народження.
- gender – стать.
- address – адреса проживання.
- email – електронна пошта для комунікації.
- phone_number – контактний номер телефону.
- doctor_id – закріплений лікар (зовнішній ключ на *doctors*).
- user_id – зв’язок із обліковим записом у таблиці *users_table* (для автентифікації).

Забезпечує зберігання основних даних пацієнтів, використовується у всіх ключових процесах: запис на прийом, ведення медичної картки, формування рецептів.

Сутність *users_table* (Облікові записи системи)

Сутність використовується модулем авторизації та автентифікації. Містить технічні дані, необхідні для входу в систему та керування ролями.

Атрибути:

- id – унікальний ідентифікатор облікового запису.
- username – логін користувача.
- password – хешований пароль.
- role – роль у системі (пацієнт, лікар, адміністратор).
- enabled – статус активності облікового запису.
- must_change_password – вимога зміни пароля при наступному вході.

Забезпечує безпечний доступ до системи, реалізує контроль доступу та розмежування прав користувачів.

Сутність *doctors* (Лікарі)

Містить інформацію про медичних працівників, які здійснюють прийоми, ведуть медичні картки та виписують рецепти.

Атрибути:

- id – унікальний ідентифікатор лікаря.
- name – ПІБ лікаря.

- phone – контактний номер.
- room – номер кабінету.
- times – робочий графік.
- user_id – зв’язок із обліковим записом у *users_table*.

Використовується для планування прийомів, ведення медичної документації та формування електронних рецептів.

Сутність appointment (Записи на прийом)

Відображає процес планування прийомів між пацієнтом і лікарем.

Атрибути:

- id – унікальний ідентифікатор запису.
- date_time – дата та час прийому.
- doctor_id – лікар, до якого здійснено запис.
- patient_id – пацієнт, який записався.
- status – статус запису (створено, підтверджено, скасовано тощо).

Є ключовою сутністю для організації роботи амбулаторії та формує основу для подальших медичних подій (візитів, вимірювань, призначень).

Сутність visits (Візити)

Описує фактичний прийом пацієнта у лікаря, включно з медичними даними та результатами огляду.

Атрибути:

- id – унікальний ідентифікатор візиту.
- complaint – скарги пацієнта.
- diagnosis – встановлений діагноз.
- plan – план лікування.
- status – статус візиту (завершено, триває тощо).
- visit_time – час проведення візиту.
- appointment_id – пов’язаний запис на прийом.
- doctor_id, patient_id – учасники візиту.

Формує основний медичний запис, який може містити вимірювання, призначення та рецепти.

Сутність *vital_signs* (Показники життєдіяльності)

Містить результати вимірювань, отриманих під час візиту.

Атрибути:

- id – унікальний ідентифікатор вимірювання.
- systolic, diastolic – артеріальний тиск.
- pulse – пульс.
- spo2 – рівень насичення крові киснем.
- temperature – температура тіла.
- height, weight, bmi – антропометричні дані.
- measured_at – час вимірювання.
- notes – додаткові примітки.
- visit_id – зв’язок із візитом.

Забезпечує фіксацію об’єктивних показників стану пацієнта, які використовуються для діагностики та моніторингу.

Сутність *prescribing* (Призначення та рецепти)

Містить інформацію про лікарські призначення, сформовані під час візиту.

- Атрибути:
- id – унікальний ідентифікатор призначення.
- medicine_name – назва лікарського засобу.
- dosage – дозування.
- frequency – частота прийому.
- duration_days – тривалість курсу.
- instructions – додаткові рекомендації.
- status – статус рецепта (чернетка, підтверджено, відправлено).
- created_date – дата створення.
- doctor_id, patient_id – учасники процесу.
- visit_id – візит, у межах якого сформовано призначення.

Забезпечує формування та зберігання електронних рецептів, а також інтеграцію з зовнішніми e-Health сервісами.

Сформулюємо схему зв’язків

2) *users_table* ↔ *users*

Тип зв’язку: 1:1 або 1:N. Кожен обліковий запис (*users_table*) може відповідати одному пацієнту (*users*). Таблиця *users_table* використовується виключно для автентифікації та авторизації, тоді як *users* містить медичні та персональні дані. Зв’язок призначений для забезпечення розмежування технічних облікових даних і персональної інформації пацієнтів.

2) *users_table* ↔ *doctors*

Тип зв’язку: 1:1. Кожен лікар має власний обліковий запис у системі. Обліковий запис визначає роль «лікар» та дозволяє доступ до функцій ведення медичної документації. Зв’язок призначений для забезпечення контролю доступу лікарів до професійних функцій системи.

3) Зв’язок *doctors* ↔ *users* (пацієнти)

Тип зв’язку: 1:N. Один лікар може бути закріплений за багатьма пацієнтами. Пацієнт може мати лише одного основного лікаря. Зв’язок призначений для організації первинної медичної допомоги та маршрутизації пацієнтів.

4) Зв’язок *appointment* ↔ *doctors*

Тип зв’язку: 1:N. Один лікар може мати багато записів на прийом. Кожен запис на прийом прив’язаний до конкретного лікаря. Зв’язок призначений для планування графіка роботи лікаря.

5) Зв’язок *appointment* ↔ *users* (пацієнти)

Тип зв’язку: 1:N. Пацієнт може створити багато записів на прийом. Кожен запис належить одному пацієнту. Зв’язок призначений для формування історії звернень пацієнта.

6) Зв’язок *visits* ↔ *appointment*

Тип зв’язку: 1:N. Один запис на прийом може мати один або кілька фактичних візитів. Кожен візит прив’язаний до конкретного запису. Зв’язок призначений для фіксації фактичного прийому, що відбувся на основі запису.

7) Зв’язок *visits* ↔ *doctors*

Тип зв’язку: 1:N. Лікар може провести багато візитів. Кожен візит має одного відповідального лікаря. Зв’язок призначений для відображення

медичної діяльності лікаря.

8) Зв’язок *visits* ↔ *users* (пацієнти)

Тип зв’язку: 1:N. Пацієнт може мати багато візитів. Кожен візит належить одному пацієнту. Зв’язок призначений для формування медичної історії пацієнта.

9) Зв’язок *vital_signs* ↔ *visits*

Тип зв’язку: 1:N. Під час одного візиту може бути зафіксовано багато вимірювань. Кожен запис показників життєдіяльності належить одному візиту. Зв’язок призначений для зберігання об’єктивних медичних даних, отриманих під час огляду.

10) Зв’язок *prescribing* ↔ *visits*

Тип зв’язку: 1:N. Під час одного візиту лікар може створити кілька призначень або рецептів. Кожне призначення належить одному візиту. Зв’язок призначений для формування лікувальних призначень у межах конкретного медичного огляду.

11) Зв’язок *prescribing* ↔ *doctors*

Тип зв’язку: 1:N. Лікар може виписати багато рецептів. Кожен рецепт має одного автора. Зв’язок призначений для фіксації юридичної відповідальності за призначення.

12) Зв’язок *prescribing* ↔ *users* (пацієнти)

Тип зв’язку: 1:N. Пацієнт може отримати багато призначень. Кожне призначення належить одному пацієнту. Зв’язок призначений для формування історії лікування пацієнта.

Інтерфейс користувача

У рамках підготовки ескізного проєкту було здійснено декілька варіантів візуального оформлення сторінок адміністрування системи. Мета – визначити оптимальне розташування елементів керування, читабельність інформації та ергономіку інтерфейсу. Нижче наведено два початкових прототипи, які демонструють розміщення елементів керування для консолі адміністратора системи.

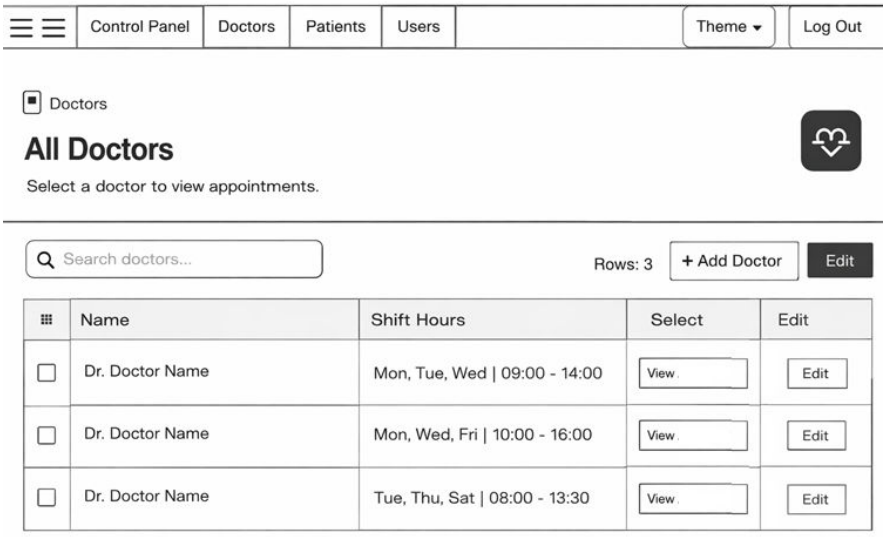

Рисунок 2.7 – Прототип сторінки адміністрування лікарів

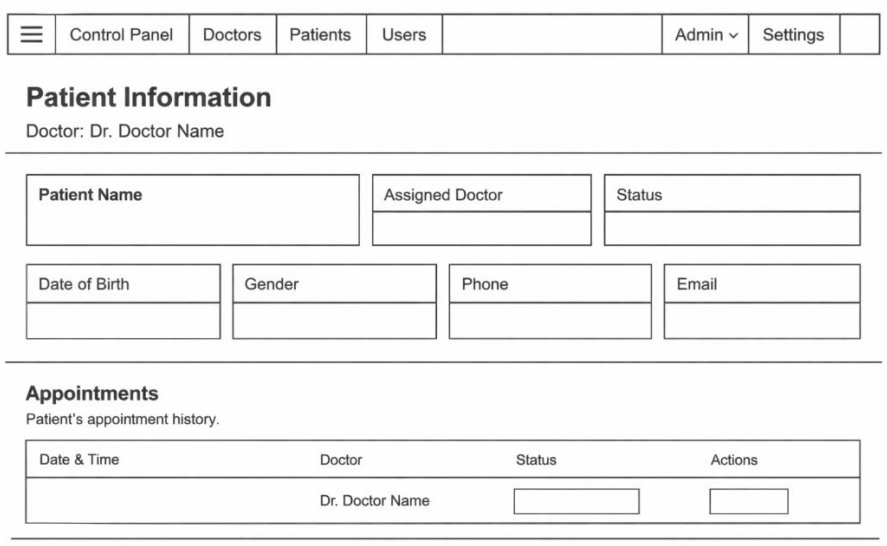

Рисунок 2.8 – Прототип сторінки редагування інформації пацієнта

Таким чином, у ескізному проєкті було розроблено логічну структуру бази даних для зберігання інформації та прототипи вебсторінок клієнтського застосунку. Наступним етапом є розроблення технічного проєкту програмного забезпечення

2.4 Технічний проєкт програмного забезпечення

Діаграма класів

На цьому етапі проєктування було побудовано діаграму класів, що відображає основні сутності системи, їх атрибути та взаємозв’язки. Вона служить логічною

моделлю структури об’єктів програмного забезпечення та є важливим інструментом для реалізації об’єктно-орієнтованого підходу. Оскільки програмне забезпечення розробляється мовою програмування Java із використанням фреймворків Spring та Hibernate, на діаграмі будуть показані як класи сутностей, призначених для відображення сутностей БД, сервісів, що реалізують бізнес-логіку, контролерів, які призначені для взаємодії з користувачем, а також інтерфейси репозиторіїв, що реалізують запити до БД. Також на діаграмі класів присутні класи конфігурації: налаштувань перемикання мовних інтерфейсів Spring Web MVC та налаштувань безпеки для ефективного використання можливостей Spring Security. Діаграма класів програмного забезпечення автоматизації облікової діяльності мережі сімейних амбулаторій представлена на рисунку 2.9.

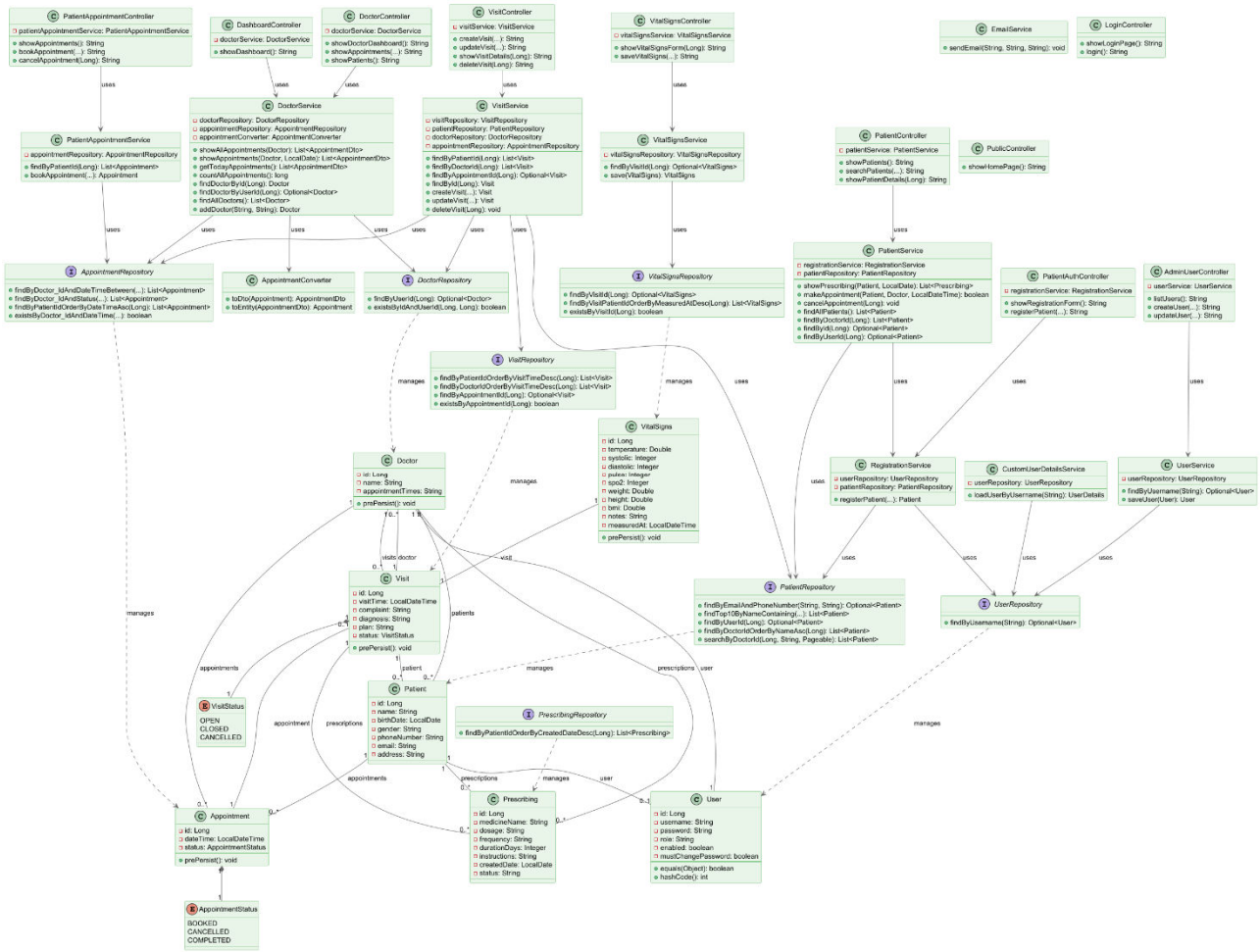

Рисунок 2.9 – Діаграма класів програмного забезпечення

На рисунку 2.10 показано діаграму класів-сутностей, що відображають таблиці бази даних при використанні JPA (Hibernate).

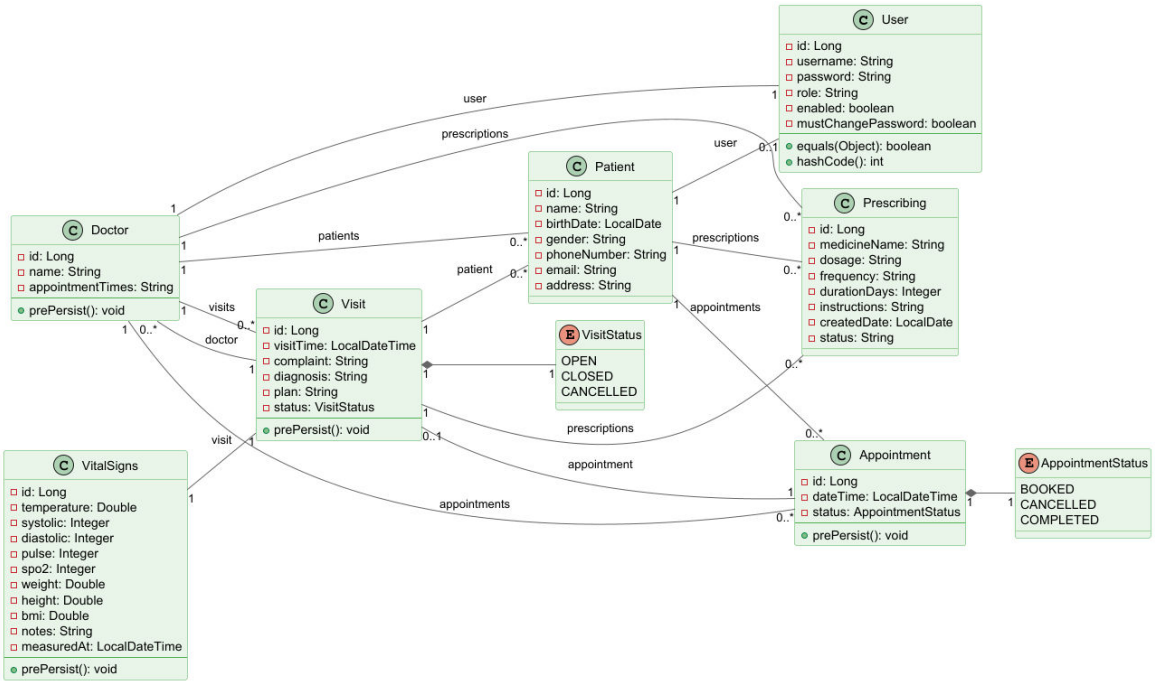

Рисунок 2.10 – Діаграма класів-сутностей

Динамічна модель описує поведінку системи в часі та ілюструє порядок взаємодії між об’єктами під час виконання ключових сценаріїв. Для цього використовуються діаграми послідовностей, що детально відображають обмін повідомленнями між компонентами програмного комплексу.

На рисунках 2.11 та 2.12 зображено діаграми послідовності для варіантів використання «Запис на прийом», «Редагування гостьового рахунку» відповідно.

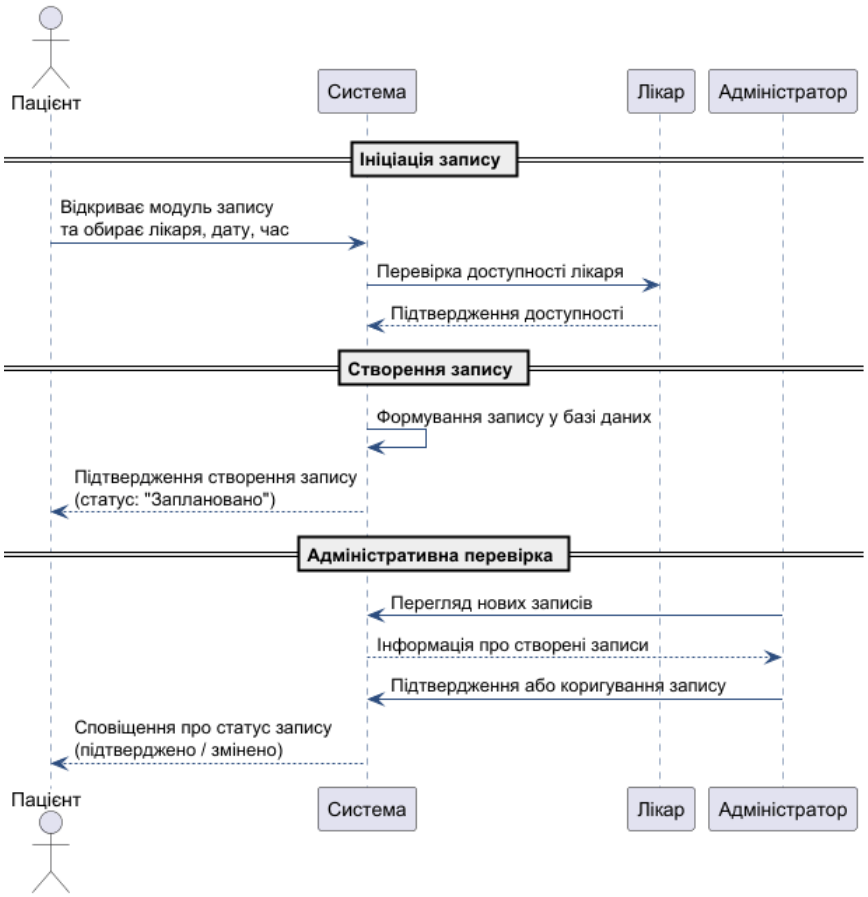

Рисунок 2.11 – Діаграма послідовності «Запис на прийом»

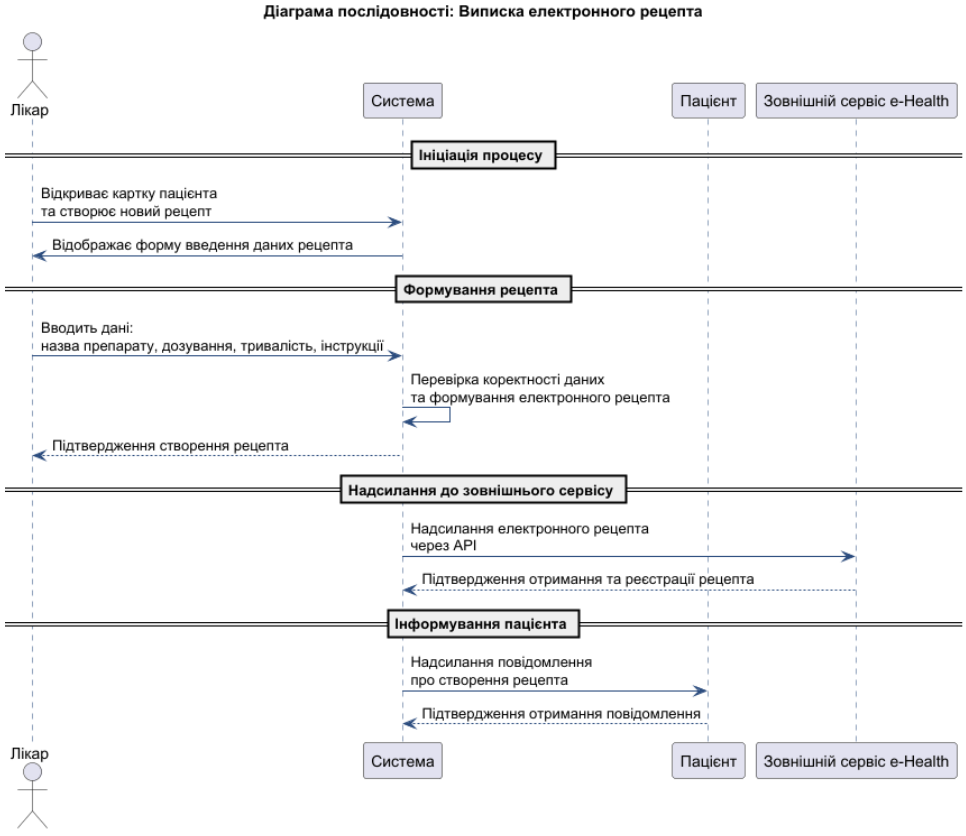

Рисунок 2.12 – Діаграма послідовності «Виписка електронних рецептів»

Таким чином, у ході розроблення технічного проєкту було створено моделі статичного представлення програмного забезпечення у вигляді діаграми класів програми та діаграми класів-сутностей для взаємодії з БД. Також були розроблені моделі динамічної поведінки програмного забезпечення у процесі його роботи.

2.5 Робочий проєкт програмного забезпечення

Вибір інструментарію розробки

У представленій кваліфікаційній роботі розробляється комплексний програмний застосунок, що складається з трьох основних компонентів: бази даних, серверного застосунка (backend) та клієнтського інтерфейсу (frontend). Для кожного з них обрано сучасні технології, що дозволяють забезпечити високий рівень функціональності, безпеки та зручності у використанні, а також сприяють ефективній розробці та підтримці продукту.

База даних

- Для збереження даних в системі використовується СУБД MySQL. MySQL – це реляційна система керування базами даних, яка вирізняється високою продуктивністю та стабільністю при роботі з великими обсягами даних [5 – 8]. Вона підтримує стандарт SQL і забезпечує ефективне виконання складних запитів завдяки оптимізованому механізму індексування. Однією з ключових особливостей MySQL є підтримка різних типів таблиць (наприклад, InnoDB та MyISAM), що дозволяє обирати оптимальний механізм зберігання залежно від задачі. Система має вбудовані засоби реплікації, що забезпечують масштабованість і відмовостійкість. MySQL добре інтегрується з більшістю мов програмування та вебфреймворків, що робить її популярною у веброзробці [6]. Вона є кросплатформною та може працювати на Windows, Linux і macOS. Завдяки відкритому вихідному коду MySQL має широку спільноту, регулярні оновлення та великий набір інструментів для адміністрування.
- Для проєктування структури БД використовується засіб DBeaver

Community – це безкоштовна, відкрита версія універсального інструмента для роботи з базами даних, який підтримує широкий спектр СУБД, включно з MySQL, PostgreSQL, Oracle, SQL Server, SQLite та багатьма іншими. Він має зручний графічний інтерфейс, що дозволяє виконувати запити, переглядати структуру бази, редагувати дані та керувати підключеннями без необхідності використовувати консоль. Однією з ключових переваг DBeaver Community є кросплатформність – застосунок працює на Windows, Linux і macOS [4]. Інструмент підтримує ER-діаграми, автодоповнення SQL, імпорт та експорт даних, а також роботу з кількома підключеннями одночасно. Завдяки відкритому коду та активній спільноті DBeaver регулярно оновлюється та має великий набір плагінів. Він підходить як для розробників, так і для аналітиків та адміністраторів баз даних, забезпечуючи стабільну та інтуїтивну роботу з даними.

Серверний застосунок (backend):

- Для розробки коду програмних модулів бекенду: робота з БД, бізнес-логіка, опрацювання запитів користувача – використовується мова програмування Java. Java – універсальна мова програмування, об’єктно-орієнтована, кросплатформна та високопродуктивна мова, створена для розробки надійних і масштабованих застосунків [9]. Вона працює за принципом «write once, run anywhere», оскільки код виконується у віртуальній машині Java (JVM), що забезпечує незалежність від операційної системи. Java має статичну типізацію, що підвищує надійність програм і зменшує кількість помилок на етапі компіляції. Мова підтримує багатопотоковість, що дозволяє створювати високонавантажені та паралельні системи [10]. Завдяки великій стандартній бібліотеці та екосистемі фреймворків (Spring, Hibernate, JavaFX) Java широко використовується у веброзробці, корпоративних системах, мобільних застосунках (Android) та серверних рішеннях. Вона має високу безпеку, що робить її популярною у фінансових та банківських системах. Активна спільнота та регулярні оновлення забезпечують стабільний розвиток мови та її інструментів [11].

- Для створення комплексного програмного застосунка мовою Java використовується сучасна версія фреймворку Spring. Spring-фреймворк – це потужний, модульний та гнучкий фреймворк для Java, який спрощує розробку корпоративних застосунків [12]. Він базується на принципах інверсії керування (IoC) та аспектно-орієнтованого програмування (AOP), що дозволяє зменшити зв'язність компонентів і підвищити масштабованість системи. Spring надає розробнику великий набір модулів: для веброзробки (Spring MVC), роботи з базами даних (Spring Data), безпеки (Spring Security) [13], інтеграції (Spring Integration), мікросервісів (Spring Boot, Spring Cloud) та багато інших. Завдяки Spring Boot створення застосунків стає значно швидшим, оскільки він автоматизує конфігурацію та пропонує вбудований сервер [14]. Фреймворк підтримує сучасні архітектурні підходи, включно з REST, мікросервісами та реактивним програмуванням. Він широко використовується у великих комерційних проєктах завдяки своїй надійності, продуктивності та активній екосистемі [15]. Для взаємодії з БД засобами роботи Spring Data, використовується ORM фреймворк Hibernate [16] – основна реалізація стандарту JPA на платформі Spring.

Клієнтський застосунок (frontend):

- Оскільки застосунок розробляється для роботи у вебсередовищі, то для представлення клієнтського інтерфейсу використовується шаблонізатор Thymeleaf, який працює на основі стандартного HTML5 в комплексі з CSS3 [17]. На теперішній час, Thymeleaf є стандартним шаблонізатором Web-сторінок для Spring MVC[18]. Для стильового оформлення сторінок використовується фреймворк Bootstrap 5, який завдяки великій бібліотеці компонентів та гнучкій системі налаштувань дозволяє створювати зручні динамічні інтерфейси із використанням мови JavaScript для реалізації операцій на клієнтському боці [19].

Звісно, для роботи із таким широким спектром технологій потрібні зручні засоби розробки. Для створення усіх програмних модулів, налагодження програми при використанні такого стеку технологій було використано

середовище IntelliJ IDEA від компанії JetBrains. IntelliJ IDEA – це одне з найпотужніших і найпопулярніших інтегрованих середовищ розробки (IDE) для Java та інших JVM-мов [20]. Вона вирізняється глибоким аналізом коду, розумним автодоповненням, рефакторингом та інструментами для підвищення продуктивності розробника. IntelliJ IDEA підтримує широкий спектр фреймворків і технологій: Spring, Hibernate, Maven, Gradle, JavaFX, Kotlin, Scala та багато інших. Середовище має вбудовані засоби для роботи з базами даних, системами контролю версій (Git, GitHub, GitLab), тестуванням і профілюванням. Завдяки інтелектуальним підказкам і автоматизації рутинних задач IDE значно прискорює розробку складних застосунків. Версія Community є безкоштовною та підходить для більшості Java-проєктів, тоді як Ultimate пропонує розширені можливості для веб- і корпоративної розробки. Починаючи з 2026 року IntelliJ IDEA постачається як єдиний дистрибутив, який при наявності ліцензії надає більш потужні та просунуті можливості роботи з БД, Web та іншими технологіями, в той же час, при відсутності ліцензії середовище IntelliJ IDEA надає базові функції роботи мовами Java та Kotlin.

Для *тестування* застосунка розглянемо сценарії тестування за методом *чорної скриньки*, розбивши усі можливі шляхи виконання на два класи еквівалентності: безпомилковий та із помилками.

Розглянемо успішний сценарій.

1 Вхід в систему

Будь який користувач відкриває початкову сторінку застосунк, де можна ознайомитись з короткою інформацією про КНП ЦПМСД№1. Натиснувши кнопку «Увійти» він потрапляє на сторінку входу, де клієнт взаємодіє з frontend-кодом, що створений із використанням JavaScript [21] (рисунок 2.13):

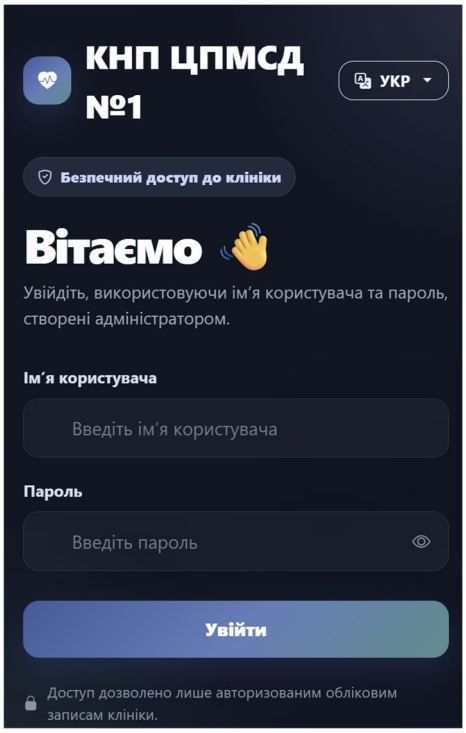

Рисунок 2.13 – Сторінка входу до застосунку

- Вводить **ім'я користувача** та **пароль**, створені адміністратором
- Натискає кнопку **"Увійти"**
- Якщо реєстраційні дані не введено, засоби JavaScript заблокують відправлення запиту на автентифікацію [21] і користувач буде змушений ввести свої дані
- Система автентифікує користувача і, якщо він є лікарем (відповідна роль вказана у таблиці users_table) **автоматично перенаправляє на сторінку "Мої пацієнти"** (рисунок 2.15), якщо реєстраційні дані введені неправильно, відображається повідомлення про це (рисунок 2.14).

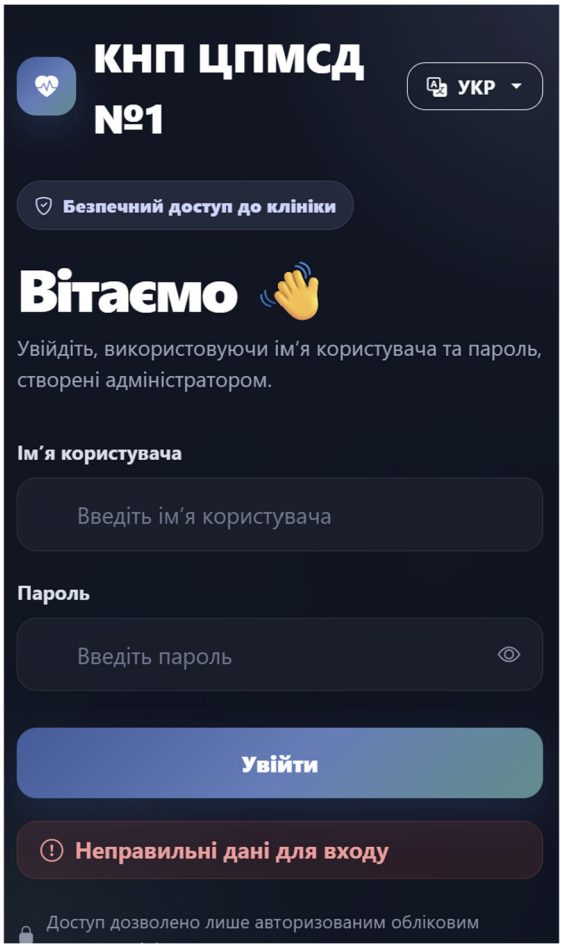

Рисунок 2.14 – Сторінка входу із повідомленням про помилку

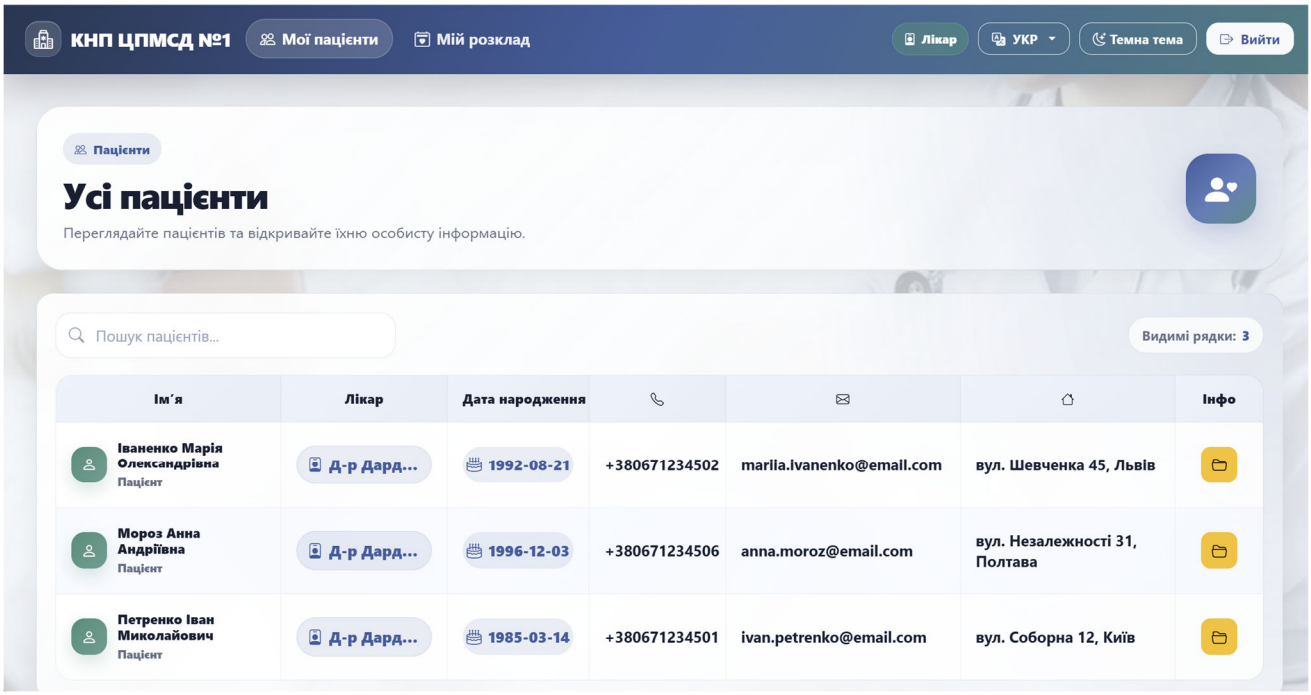

Рисунок 2.15 – Сторінка лікаря за замовчуванням – «Мої пацієнти»

2 Головна сторінка - "Мої пацієнти"

Після входу лікар одразу бачить список **своїх прикріплених пацієнтів**:

Таблиця пацієнтів містить:

- **Ім'я** пацієнта
- **Дата народження**
- **Телефон**
- Кнопка **"Переглянути інформацію"**

Функції на сторінці:

- **Пошук** пацієнтів за ім'ям, телефоном (реалізовано в асинхронному режимі за рахунок запитів JavaScript) [21]
- **Фільтрація** видимих рядків таблиці
- Можливість **додати нового пацієнта** (кнопка "Додати пацієнта")

3 Перегляд медичної картки пацієнта

Лікар натискає **"Переглянути інформацію"** біля потрібного пацієнта і потрапляє на детальну сторінку:

Особиста інформація:

- ПІБ, стать, дата народження
- Телефон, email, адреса
- Прикріплений лікар
- Кнопка **"Редагувати пацієнта"**

Вкладка "Записи":

- **Історія всіх записів** цього пацієнта
- Для кожного запису: дата/час, статус (Заплановано/Завершено/Скасовано)
- Кнопка **"Почати візит"** - для переходу від запису до медичного візиту

Вкладка "Медичні візити":

- **Історія всіх візитів** пацієнта до лікаря
- Для кожного візиту відображається:
  - Дата і час візиту
  - Статус (Відкрито/Закрито/Скасовано)
  - **Скарга** пацієнта

  - **Діагноз** лікаря
  - **План лікування**
- Кнопка **"Додати візит"** - для створення нового візиту
- Можливість **редагувати** або **видалити** існуючі візити

Вкладка "Життєві показники":

- **Історія вимірювань** пацієнта:
  - Температура (°C)
  - Артеріальний тиск (систолічний/діастолічний)
  - Пульс (уд/хв)
  - SpO2 (%)
  - Вага (кг)
  - Зріст (см)
  - ІМТ (автоматично розраховується)
  - Додаткові нотатки
- Показники додаються через форму візиту

Вкладка "Рецепти":

- **Список призначених ліків**:
  - Назва ліків
  - Дозування (наприклад, 500mg)
  - Частота прийому (1 раз/2 рази/3 рази на день, кожні 8/12 годин, за потреби)
  - Тривалість лікування (у днях)
- Кнопка **"Додати рецепт"**
- Можливість **редагувати** існуючі рецепти

4 Робота з записами

Переглянути всі записи на певну дату:

Через верхнє меню:

- Натискає **"Мій розклад"**
- **Вибирає дату** через календар
- Бачить список записів на обрану дату:

- Дата і час
- Ім'я пацієнта
- Телефон пацієнта
- Статус запису
- Кнопка **"Почати візит"**

Додати новий запис:

- Натискає **"Додати запис"**
- Шукає пацієнта за ім'ям/телефоном
- Обирає дату
- Обирає доступний час
- Зберігає запис

5 Проведення прийому (створення візиту)

Варіант А - З існуючого запису:

1. На сторінці запису/розкладу натискає **"Почати візит"**
2. Система автоматично створює візит з прив'язкою до запису

Варіант Б - Без попереднього запису (walk-in):

1. На сторінці пацієнта натискає **"Додати візит"**
2. Створює візит без прив'язки до запису

Заповнення форми візиту:

- **Час візиту** (автоматично або вручну)
- **Статус** - Відкрито/Закрито/Скасовано
- **Скарга** - на що скаржиться пацієнт
- **Діагноз** - висновок лікаря
- **План лікування** - рекомендації, повторний огляд, нотатки

Додавання життєвих показників до візиту:

- Натискає кнопку додавання показників у формі візиту
- Вводить виміряні дані (тиск, температура, пульс, вага тощо)
- Зберігає

Призначення ліків:

- Натискає **"Додати рецепт"** у розділі рецептів
- Заповнює: назву, дозування, частоту, тривалість
- Зберігає рецепт

6 Навігаційне меню лікаря

У верхньому меню лікар має доступ до:

- **Мої пацієнти** - головна сторінка зі списком прикріплених пацієнтів
- **Мій розклад** - перегляд записів на певну дату
- **Моя інформація** - особиста інформація лікаря, графік роботи
- **Зміна мови** (Українська/English)
- **Перемикач теми** (світла/темна)
- **Вийти** - вихід з системи

Висновки за розділом 2

У розділі 2 описано повний цикл проєктування та підготовки до реалізації програмного комплексу для автоматизації облікової діяльності мережі сімейних амбулаторій.

1 Проведено деталізацію вимог і побудовано модель варіантів використання, визначено акторів та сформовано специфікації ключових сценаріїв (запис на прийом, ведення електронної медичної картки, виписка е-рецептів, звітність).

2 Обґрунтовано вибір архітектури MVC та описано компоненти й розгортання системи, що забезпечують масштабованість і керованість доступу.

3 На етапах ескізного, технічного й робочого проєктів розроблено логічну модель даних, структуру сутностей і зв’язків, а також визначено технологічний стек (Java, Spring, Hibernate, Thymeleaf, Bootstrap, MySQL) і підхід до тестування за методом «чорної скриньки».

4 Отримані результати створюють основу для подальшої реалізації, інтеграції та перевірки працездатності системи у розділі 3.

3 РЕЗУЛЬТАТИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ОБЛІКОВОЇ ДІЯЛЬНОСТІ МЕРЕЖІ СІМЕЙНИХ АМБУЛАТОРІЙ

Під час виконання роботи було виконано:

1) Проаналізовано предметну область: визначено особливості облікових процесів у мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська, описано основні об’єкти обліку (пацієнти, персонал, прийоми, візити, призначення) та сформульовано проблеми, що виникають при використанні розрізнених і ручних інструментів.

2) Сформовано вимоги та постановку задачі: проаналізовано наявні програмні рішення для медичних закладів, визначено їх переваги та обмеження, після чого сформульовано функціональні й нефункціональні вимоги до розроблюваного ПЗ та зафіксовано їх у постановці задачі й технічному завданні.

3) Виконано проєктування програмного комплексу: визначено акторів системи та побудовано модель варіантів використання, підготовлено специфікації ключових прецедентів (запис на прийом, ведення електронної медичної картки, виписка електронних рецептів, формування звітів), обґрунтовано архітектурний підхід та розроблено структуру модулів і взаємодію компонентів.

4) Реалізовано програмне забезпечення: розроблено вебзастосунок із модулями авторизації та розмежування доступу, ведення довідників (лікарі, послуги, амбулаторії/кабінети), обліку пацієнтів, керування записами на прийом і візитами, ведення електронної медичної картки, формування електронних призначень/рецептів та генерації звітності за обліковими даними.

5) Спроєктовано та налаштовано сховище даних: побудовано логічну модель і реалізовано реляційну базу даних MySQL, налаштовано доступ через ORM-рівень, забезпечено цілісність даних і підготовлено довідники, необхідні для коректного ведення обліку та формування звітів.

6) Забезпечено сумісність та зручність експлуатації: вебінтерфейс

коректно відображається у сучасних браузерах (Chrome, Edge, Firefox), підтримується робота на типових робочих місцях під управлінням Windows 10/11, а також передбачено експорт звітів і форм у поширені формати.

7) Протестовано систему: проведено модульне тестування серверної логіки та контролерів, інтеграційне тестування взаємодії з базою даних, перевірено коректність обробки типових сценаріїв (запис/скасування прийому, створення візиту, оформлення призначень, формування звітів), а також виконано приймальні перевірки з урахуванням ролей користувачів.

8) Підготовлено документацію: оформлено технічне завдання (Додаток А), інструкцію користувача (Додаток Б), опис програми (Додаток В) та програму і методику випробувань (Додаток Г), що забезпечує можливість впровадження, супроводу та подальшого розвитку програмного комплексу.

У результаті виконаної роботи створено цілісний програмний комплекс, який забезпечує централізований облік ключових даних і підтримку типових операцій у мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська, підвищуючи оперативність ведення записів і якість підготовки звітності.

Розроблене ПЗ забезпечує:

- централізацію облікових даних (єдині довідники, пацієнти, лікарі, записи, візити) та зменшення дублювання інформації між підрозділами;
- підвищення оперативності роботи персоналу (швидкий пошук, планування прийому, ведення медичної картки, формування звітів) завдяки уніфікованим формам і автоматичному заповненню типових полів;
- дотримання вимог інформаційної безпеки (автентифікація, розмежування прав доступу за ролями, журналювання дій користувачів) та зниження ризику помилок завдяки валідації даних;
- можливість формування регламентної та управлінської звітності (за періодами, лікарями, амбулаторіями, видами послуг), що підтримує контроль виконання планових показників і прийняття управлінських рішень.

Доопрацювання та впровадження такого рішення у КНП «ЦПМСД №1» м. Краматорська дозволить скоротити час на рутинні облікові операції,

підвищити точність і повноту даних, а також забезпечити більш прозорий контроль процесів і показників діяльності мережі амбулаторій.

Висновки за розділом 3

У розділі 3 наведено результати реалізації розробленого програмного комплексу та підтверджено його відповідність поставленим вимогам.

1. Реалізовано вебзастосунок із модулями обліку пацієнтів і лікарів, керування записами на прийом та візитами, ведення електронної медичної картки, формування призначень/рецептів і звітності, а також механізмами автентифікації та розмежування доступу.

2. Створено та налаштовано реляційну базу даних MySQL, забезпечено цілісність і узгодженість даних.

3. Проведено модульне та інтеграційне тестування ключових сценаріїв і підготовлено комплект документації, включно з програмою та методикою випробувань.

4. Отриманий результат може бути використаний для подальшого впровадження в мережі амбулаторій та розвитку функціоналу.

4 ОХОРОНА ПРАЦІ У ВІДІЛЛІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Охорона праці – це комплексна система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних та лікувально- профілактичних заходів, спрямованих на збереження життя, здоров'я, працездатності й продуктивності працівників у процесі виконання ними трудових обов'язків. У сфері інформаційних технологій охорона праці набуває особливої специфіки, оскільки основні ризики пов'язані не зі шкідливими виробничими умовами в традиційному розумінні, а з тривалим статичним навантаженням, психоемоційним напруженням, а також ергономічними особливостями робочого середовища.

У контексті розробки програмного забезпечення для ЦПМСД №1 м.Краматорськ, охорона праці відіграє ключову роль. Працівники, що займаються програмуванням, проєктуванням, тестуванням та супроводом програмного продукту, виконують значну частину своїх обов'язків у сидячому положенні, за монітором комп'ютера. Такий характер роботи потребує ретельного дотримання санітарно-гігієнічних та ергономічних норм для запобігання хронічному перенапруженню, зору, опорно-рухового апарату та нервової системи.

Характеристика робочого місця розробника ПЗ

Робоче місце програміста, незалежно від того, чи працює він в офісі, чи віддалено, має бути обладнане відповідно до встановлених норм і стандартів. До основних елементів робочого місця належать:

- комп'ютерне обладнання: монітор з достатньою діагоналлю та регулюванням яскравості, клавіатура, миша;

- ергономічні меблі: регульований стілець з підтримкою попереку, зручний стіл на відповідній висоті;
- джерела освітлення: природне (вікна) та штучне (настільні або стельові лампи з нейтральним світлом);
- засоби комунікації та підключення до мережі Інтернет;
- засоби для підтримання температурного та вологісного режимів (кондиціонери, зволожувачі повітря).

Облаштування робочого місця має відповідати вимогам нормативних документів:

- ДСТУ EN ISO 9241 [22] (Ергономічні вимоги до офісної роботи з відеотерміналами);
- ДБН В.2.2-10-2001 [23] (Будинки і споруди. Будинки та приміщення для постійного перебування людей);
- Санітарні правила 3.3.2.007-98 (Гігієнічні вимоги до ПЕОМ та організації праці користувачів).

4.1 Шкідливі та небезпечні виробничі фактори при розробці ПЗ

Розробка програмного забезпечення передбачає вплив низки потенційно шкідливих та небезпечних чинників, які можуть негативно вплинути на здоров’я працівника:

Фізичні фактори:

- тривале перебування у статичному положенні (сидячи);
- недостатнє або занадто яскраве освітлення, яке створює напруження для очей;
- шумове навантаження, особливо в офісах з відкритим плануванням (open space).

- Психофізіологічні фактори:
- високий рівень концентрації уваги протягом тривалого часу;
- робота з великими обсягами інформації та кодом;
- стресові ситуації, пов’язані з дедлайнами, помилками або технічними збоями.

Хімічні та електромагнітні фактори:

- можливий вплив електромагнітного випромінювання від моніторів і мережевого обладнання (хоча в межах допустимих норм);
- недостатня вентиляція в приміщенні може призводити до накопичення шкідливих речовин.

4.2 Заходи щодо забезпечення охорони праці при роботі з комп’ютером

Для мінімізації впливу шкідливих чинників і забезпечення безпечних умов праці необхідно впровадити низку заходів:

Організаційні заходи:

- дотримання режиму праці та відпочинку: рекомендовані перерви – 10-15 хвилин після кожної години роботи за ПК;
- чергування розумових та рутинних завдань (програмування, перегляд документації, тестування);
- регулярне проходження інструктажів та навчання з охорони праці, пожежної безпеки та надання першої допомоги.
- Санітарно-гігієнічні заходи [24]:
- підтримання температури в межах 20–24°C;
- освітлення з яскравістю не менше 300–500 лк, бажано з можливістю регулювання;

- регулярне провітрювання приміщення, підтримання вологості на рівні 40–60%;
- облаштування робочої зони в місцях, захищених від прямих сонячних променів та відблисків.

Ергономічні заходи:

- встановлення монітора на відстані 60-70 см від очей, верхній край екрана має бути на рівні очей;
- використання офісних крісел з можливістю регулювання висоти та кута нахилу спинки;
- правильне розташування клавіатури і миші – на одному рівні з руками, з можливістю підтримки зап’ясть.

Психологічна підтримка та профілактика емоційного вигорання:

- формування позитивної атмосфери в команді;
- забезпечення розумного навантаження та уникнення понаднормової роботи;
- надання працівникам можливості працювати за гнучким графіком або дистанційно;
- мотиваційні заходи, тренінги з тайм-менеджменту, ментального здоров’я.

Висновки за розділом 4

Урахування вимог охорони праці є необхідною умовою стабільного функціонування ІТ-підрозділів. Якісно організоване робоче середовище не лише знижує ризики професійних захворювань, а й сприяє підвищенню продуктивності праці, зниженню кількості помилок у програмному коді та формує відповідальне ставлення працівників до власного здоров’я та результатів праці.

ВИСНОВКИ

В ході кваліфікаційної роботи було досягнуто поставленої мети, а саме розроблено програмне забезпечення для автоматизації облікової діяльності мережі сімейних амбулаторій. Розроблення проходило у декілька етапів: проведено аналіз та формулювання практичної задачі інженерії програмного забезпечення з розробки програмного продукту для автоматизації облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська, що характеризується комплексністю та невизначеністю умов; сформовано вимоги до програмного забезпечення; на основі поставленої задачі було розроблено модулі програмного забезпечення.

У межах кваліфікаційної роботи було виконано такі основні завдання:

- проаналізовано роботу мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська», зокрема сучасні тенденції автоматизації діяльності;
- зроблено висновки щодо доцільності розробки застосунку для автоматизації обліку роботи амбулаторій;
- сформулювано постанову задачі на розробку програмного забезпечення для автоматизації обліку роботи сімейних амбулаторій;
- проведено аналіз вимог програмного забезпечення для автоматизації обліку роботи сімейних амбулаторій;
- проведено аналіз існуючих аналогів застосунків для автоматизації обліку роботи медичних закладів;
- визначено функціональні, інформаційні та технічні вимоги до розроблюваного комплексу, включаючи вимоги до підтримки обробки візитів пацієнтів, управління особистими даними пацієнтів, створення звітної документації та дотримання вимог інформаційної безпеки;
- проведено декомпозицію проєкту на окремі функціональні підсистеми: вебінтерфейс для пацієнтів, лікарів та адміністраторів;
- розроблено архітектуру програмного забезпечення, включно з

визначенням внутрішніх взаємозв’язків між компонентами, каналів взаємодії з центральною базою даних, а також механізмів авторизації та захисту інформації;

- реалізовано програмні компоненти системи із використанням сучасних інструментів та технологій розробки;
- проведено комплексне тестування функціональності, зокрема модульні та інтеграційні випробування, а також приймальні тести за участю представників кінцевого користувача;
- підготовлено повний комплект технічної та експлуатаційної документації, що включає технічне завдання, опис програми, інструкцію користувача та методику випробувань;
- надано практичні рекомендації щодо впровадження системи в умовах реальної експлуатації з урахуванням вимог охорони праці та інформаційної безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ISO/IEC/IEEE 29148:2018. Systems and software engineering. Life cycle processes. Requirements engineering. [Чинний від 2018-11-28]. Вид. офіц. Швейцарія, 2018. 9с. (Інформація та документація).

2. Офіційний сайт Центру сімейної медицини №1 м.Краматорська . URL: https://kramfamilycenter1.dn.ua (дата звернення: 01.04.2025).

3. Офіційний сайт SAP ”Обслуговування клієнтів”. URL: https://www.sap.com/ukraine/products/crm/customer-service.html (дата звернення: 01.04.2026).

4. DBeaver. Офіційний сайт : вебсайт. – URL: https://dbeaver.io (дата звернення: 09.05.2026).

5. MySQL 8.0 Reference Manual : official documentation : website. – URL: https://dev.mysql.com/doc/refman/8.0/en/ (accessed: 09.05.2026).

6. Bell C., Kindahl J., Thalmann L. MySQL High Availability: Tools for Building Robust Data Centers. Sebastopol: O’Reilly Media, 2010. 370 p.

7. DuBois P. MySQL Cookbook: Solutions for Database Developers and Administrators. 3rd ed. Sebastopol: O’Reilly Media, 2014. 884 p.

8. Karwin B. SQL Antipatterns: Avoiding the Pitfalls of Database Programming. Dallas: Pragmatic Bookshelf, 2010. 336 p.

9. Bloch J. Effective Java. 3rd ed. Boston: Addison-Wesley Professional, 2018. 416 p.

10. Goetz B., Peierls T., Bloch J. et al. Java Concurrency in Practice. Boston: Addison-Wesley Professional, 2006. 432 p.

11. Horstmann C. S. Core Java, Volume I: Fundamentals. 14th ed. Oracle Press, 2026. 816 p.

12. Walls C. Spring in Action. 6th ed. Shelter Island: Manning Publications, 2022. 520 p.

13. Spilca L. Spring Security in Action. 2nd ed. Shelter Island: Manning

Publications, 2024. 440 p.

14. Bauer C., King G., Gregory G. Java Persistence with Hibernate. 2nd ed. Shelter Island: Manning Publications, 2015. 608 p.

15. Hibernate ORM documentation : official website. – URL: https://hibernate.org/orm/documentation/ (accessed: 09.05.2026).

16. Walls C. Spring Boot in Action. Shelter Island: Manning Publications, 2016. 264 p.

17. Thymeleaf Documentation: official website. – URL: https://www.thymeleaf.org/documentation.html (accessed: 09.05.2026).

18. Deblauwe W. Taming Thymeleaf: Practical Guide to Building a Web Application with Spring Boot and Thymeleaf. Leanpub, 2020. URL: https://www.wimdeblauwe.com/books/taming-thymeleaf/ (accessed: 09.05.2026).

19. Bootstrap Documentation: official website. – URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/ (accessed: 09.05.2026).

20. IntelliJ IDEA Documentation: official website. – URL: https://www.jetbrains.com/help/idea/ (accessed: 09.05.2026).

21. MDN Web Docs. JavaScript Guide: website. – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (accessed: 09.05.2026).

22. ДСТУ EN ISO 9241-5:2004. Вимоги до ергономіки офісного обладнання для роботи з дисплеями.[Чинний від 2006-01-01]. Вид. офіц. Київ: Держспоживстандарт України, 2006. 34с. (Інформація та документація).

23. ДБН В.2.2-10-2001. Приміщення житлові та громадські. Основи проектування. [Чинний від 2001-06-04]. Вид. офіц. Київ, 2001. 171с. (Інформація та документація).

24. ДСанПіН 3.3.2.007-98. Санітарні правила і норми роботи з ПК.[Чинний від 1998-12-10]. Вид. офіц. Київ: Міністерство охорони здоров’я України, 1998. 22с. (Інформація та документація).

ДОДАТОК А

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку програмного забезпечення автоматизації облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська»

А.1 Вступ

Програмне забезпечення призначене для підвищення ефективності роботи сімейних амбулаторій КНП «ЦПМСД №1» м.Краматорська за рахунок автоматизації обліку візитів пацієнтів, призначень та планування годин прийомів лікарів.

А.2 Підстава для розробки

Підставою для розробки програмного забезпечення є Наказ ректора НУК на затвердження тем кваліфікаційних робіт бакалаврів зі спеціальності 121 «Інженерія програмного забезпечення» №275-уч від 07.04.2026 р.

А.3 Призначення розробки

Розробка призначена для автоматизації процесів сімейних амбулаторій: обліку лікарів та пацієнтів, планування прийомів, збереження історій звернень пацієнтів та призначених лікувань, формування звітів і повідомлень для пацієнтів.

А.4 Вимоги до програми або програмного виробу

А.4.1 Вимоги до функціональних характеристик:

- реєстрація пацієнтів та лікарів;
- оформлення та облік відвідувань;
- формування призначень та рецептів за результатом візиту;
- введення та обробка лабораторних досліджень та вимірювань життєвих показників пацієнтів;

- захист доступу до даних через систему ролей.

А.4.2Вимоги до надійності

- стійкість роботи при обробці великих обсягів даних;
- перевірка коректності вхідних даних;
- можливість резервного копіювання бази даних;
- обробка помилок введення/збереження даних.

А.4.3 Умови експлуатації:

- експлуатація в офісних умовах при температурі 18–26°C та вологості 40–60%;
- програмна система розрахована на користувача із середнім рівнем комп'ютерної грамотності.

А.4.4 Вимоги до складу і параметрів технічних засобів:

- ОС: Windows 11 або вище;
- мінімальні ресурси клієнтської машини: 4 ГБ ОЗП, 2 ГГц процесор;
- СУБД: MySQL;
- підтримка експорту у PDF, CSV.

А.4.5 Вимоги до інформаційної та програмної сумісності:

- програма повинна функціонувати під управлінням ОС Windows 11;
- підтримка браузерного доступу через вебінтерфейс.
-

А.4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

А.4.7 Вимоги до транспортування і зберігання

Зберігання файлів розробки на корпоративному сервері або в хмарних

сховищах із резервним копіюванням.

А.4.8 Спеціальні вимоги

Спеціальних вимог не передбачається.

А.5 Вимоги до програмної документації:

- Технічне завдання;
- Опис програми;
- Інструкція користувача;
- Програма та методика випробувань;
- -Тексти програмних модулів.

А.6 Техніко-економічні показники

Техніко-економічні показники не розраховуються.

А.4.7 Стадії і етапи розробки

Стадії та етапи розробки програмного застосунку приведено у табл А.1

Таблиця А.1-Стадії та етапи розробки ПЗ

Стадія	**Етапи робіт**	**Термін виконання**
Технічне завдання	Збирання та аналіз вимог, документування вимог.	28.03.2026
Ескізний проєкт	Розроблення діаграми варіантів використання, розроблення специфікацій, розроблення діаграм діяльності.	15.03.2026
Технічний проєкт	Розроблення статичної моделі ПЗ, розроблення динамічної моделі ПЗ, деталізація алгоритмів, модулів	25.04.2026
Робочий проєкт	Вибір засобів розроблення та мов програмування, розроблення програмного коду, програмування, тестування, документування.	01.06.2026

А.8 Порядок контролю і приймання

На кожному етапі створення програмного забезпечення контролюють процеси аналізу та проєктування його складових з урахуванням вимог, викладених у технічному завданні. Кожна фаза розроблення має бути представлена у визначені терміни та погоджена із замовником.

Контроль здійснюється шляхом поетапного перегляду та погодження результатів робіт (аналіз, проєктування, реалізація, тестування) з оформленням звітів/актів перевірки, фіксацією зауважень і термінів їх усунення.

Приймання виконується за затвердженою методикою і програмою випробувань, а результати фіксують у протоколі проведення перевірок. Якщо під час приймання програмного продукту виявлено помилки, складають відповідний акт, у якому їх описують. Цей документ підписують представники замовника й розробника, а затверджують керівники обох сторін. Розробник зобов’язаний упродовж не більше двох тижнів виправити всі виявлені недоліки та повідомити замовника про готовність до повторної перевірки.

ДОДАТОК Б

ІНСТРУКЦІЯ КОРИСТУВАЧА ДЛЯ АДМІНІСТРАТОРА

Система управління КНП ЦПМСД №1

1. Вхід у систему

1.1 Початок роботи

1. Відкрийте браузер і перейдіть на адресу системи клініки
2. На сторінці входу введіть:
   - **Ім’я користувача**: ваш логін адміністратора
   - **Пароль**: ваш пароль
3. Натисніть кнопку **"Увійти"**

Примітка: Облікові дані адміністратора створюються при початковому встановленні системи.

1.2 Можливі помилки

- **"Неправильні дані для входу"** - перевірте правильність введення логіну та пароля
- **"Цей обліковий запис ще не пов’язаний із записом лікаря або пацієнта"** - для адміністратора це нормально, обліковий запис не потребує прив’язки

2. Головна панель керування

Після успішного входу ви потрапляєте на **Панель керування** - головну сторінку адміністратора.

2.1 Статистичні картки

На панелі відображаються три основні показники:

Картка	Опис
Лікарі	Загальна кількість зареєстрованих лікарів
Пацієнти	Загальна кількість зареєстрованих пацієнтів
Прийоми	Загальна кількість записів на прийом

2.2 Картка адміністрування

У верхній частині панелі знаходиться спеціальна картка **"Адміністрування"**:

- **Призначення**: швидкий доступ до управління користувачами
- **Кнопка**: "Керування користувачами"
- **Функція**: створення облікових записів та призначення ролей

2.3 Сьогоднішні записи

Таблиця **"Сьогоднішні найближчі записи"** показує:

- Дату і час прийому
- Ім’я пацієнта
- Телефон пацієнта
- Загальна кількість записів на сьогодні

2.4 Навігаційне меню адміністратора

У верхньому меню доступні розділи:

- **Панель керування** - головна сторінка
- **Лікарі** - управління лікарями
- **Пацієнти** - управління пацієнтами
- **Користувачі** - управління обліковими записами
- **Мова** - перемикач мови (УКР/EN)
- **Тема** - світлий/темний режим
- **Вийти** - вихід з системи

3. Управління користувачами

Це **розділ** для адміністратора, де створюються облікові записи для всіх працівників та пацієнтів клініки.

3.1 Перехід до управління користувачами

Спосіб 1: Натисніть кнопку **"Керування користувачами"** на панелі керування

Спосіб 2: Виберіть **"Користувачі"** в верхньому меню

3.2 Перегляд користувачів

На сторінці користувачів відображається таблиця з колонками:

Колонка	Опис
Ім’я користувача	Логін для входу в систему
Роль	Адміністратор / Лікар / Пацієнт
Статус	Увімкнено / Вимкнено
Пов’язаний запис	Прізвище лікаря або пацієнта (якщо прив’язано)
Редагування	Кнопка "Редагувати"

Функції:

- **Пошук** - знайдіть користувача за ім’ям
- **Фільтр** - відображення певної кількості рядків

3.3 Створення нового користувача

Крок 1: Відкриття форми

Натисніть кнопку **"Додати користувача"** (праворуч вгорі)

Крок 2: Заповнення основної інформації

Обов’язкові поля:

- **Ім’я користувача** - унікальний логін (наприклад: ivanov_doctor, petrov_patient)
- **Пароль** - надійний пароль для першого входу
- **Роль** - виберіть одну з трьох:
- **Адміністратор** - повний доступ до системи
- **Лікар** - доступ до пацієнтів та записів
- **Пацієнт** - доступ до власної медичної картки

Додаткові налаштування:

- **Увімкнено** - чекбокс (за замовчуванням активний)
- Увімкнено = користувач може входити в систему

- Вимкнено = користувач заблокований

Крок 3: Прив’язка до запису

Для ролі "Лікар":

1. З’явиться поле **"Прив’язати до лікаря"**
2. Відкрийте випадаючий список **"Оберіть лікаря"**
3. Виберіть потрібного лікаря зі списку
4. Це дозволить лікарю бачити своїх пацієнтів після входу

Для ролі "Пацієнт":

1. З’явиться поле **"Прив’язати до пацієнта"**
2. Відкрийте випадаючий список **"Оберіть пацієнта"**
3. Виберіть потрібного пацієнта зі списку
4. Це дозволить пацієнту бачити свою медичну картку

Для ролі "Адміністратор":

- Прив’язка не потрібна
- Адміністратор має доступ до всієї системи

Крок 4: Збереження

Натисніть кнопку **"Зберегти"**

Успішно: З’явиться повідомлення "Користувача успішно збережено"

Помилка: "Не вдалося зберегти користувача" - перевірте заповнення всіх полів

3.4 Редагування користувача

1. Знайдіть потрібного користувача в таблиці
2. Натисніть кнопку **"Редагувати"**
3. У формі можна змінити:
   - Пароль (залиште порожнім, якщо не хочете змінювати)
   - Роль
   - Статус (увімкнено/вимкнено)
   - Прив’язку до лікаря/пацієнта

4. Натисніть **"Зберегти"**

3.5 Типові сценарії

Сценарій 1: Новий лікар в клініці

1. **Спочатку створіть лікаря** (розділ 4)
2. **Потім створіть користувача:**
   - Ім’я користувача: `ivanov_doctor`
   - Пароль: `TempPass123`
   - Роль: DOCTOR
   - Прив’язка: виберіть "Іванов Іван Іванович"
   - Увімкнено: X
3. Передайте лікарю логін і пароль для першого входу

Сценарій 2: Новий пацієнт

1. **Спочатку створіть пацієнта** (розділ 5)
2. **Потім створіть користувача:**
   - Ім’я користувача: petrov_patient
   - Пароль: Patient123
   - Роль: **Пацієнт**
   - Прив’язка: виберіть "Петров Петро Петрович"
   - Увімкнено: X
3. Передайте пацієнту логін і пароль

Сценарій 3: Тимчасове блокування користувача

1. Знайдіть користувача
2. Натисніть **"Редагувати"**
3. Зніміть галочку **"Увімкнено"**
4. Збережіть
5. Користувач не зможе увійти в систему до розблокування

4. Управління лікарями

4.1 Перегляд лікарів

1. Виберіть **"Лікарі"** у верхньому меню
2. Відображається таблиця всіх лікарів:
   - Ім’я лікаря
   - Час прийому (графік роботи)
   - Кнопка "Переглянути записи"
   - Кнопка "Редагувати"

Функції:

- **Пошук** - введіть прізвище лікаря
- **Фільтр рядків** - обмежте кількість видимих записів

4.2 Додавання нового лікаря

Крок 1: Відкрийте форму

Натисніть **"Додати лікаря"**

Крок 2: Введіть ім’я

Ім’я лікаря: введіть повне ПІБ (наприклад: "Іванов Іван Іванович")

Крок 3: Створіть графік роботи

У системі є **Конструктор графіка** з двома режимами:

Режим А: Конструктор часових блоків

1. Натисніть **"Додати часовий блок"**
2. Для кожного блоку вкажіть:
   - **Дні тижня** - оберіть один або декілька (Пн, Вт, Ср, Чт, Пт, Сб, Нд)
   - **Час від** - початок робочого дня (формат 24-год: ГГ:ХХ)
   - **Час до** - кінець робочого дня (формат 24-год: ГГ:ХХ)
3. Натисніть **"Згенерувати час прийому"**
4. Система автоматично створить слоти через кожні 30 хвилин

Приклад:

```
Блок 1:
Дні: Пн, Вт, Ср
Від: 09:00
До: 13:00

Блок 2:
Дні: Чт, Пт
Від: 14:00
До: 18:00

Результат: Пн, Вт, Ср | 09:00 - 13:00 ; Чт, Пт | 14:00 - 18:00
```

Режим Б: Ручне введення

1. Натисніть **"Показати"** біля поля "Час прийому"
2. Введіть текст у форматі:

```
День, День | ГГ:ХХ - ГГ:ХХ ; День | ГГ:ХХ - ГГ:ХХ
```

Приклад:

```
Пн, Ср, Пт | 08:00 - 12:00 ; Вт, Чт | 13:00 - 17:00
```

Важливо: Використовуйте 24-годинний формат часу (00:00 до 23:59)

Крок 4: Збережіть

Натисніть **"Зберегти"**

4.3 Редагування лікаря

1. Знайдіть лікаря в таблиці
2. Натисніть **"Редагувати"**
3. Змініть ім’я або графік роботи
4. Збережіть зміни

4.4 Перегляд записів лікаря

1. Знайдіть лікаря в таблиці
2. Натисніть **"Переглянути записи"**
3. Виберіть дату через календар
4. Перегляньте список записів на цю дату

5. Управління пацієнтами

5.1 Перегляд пацієнтів

1. Виберіть **"Пацієнти"** у верхньому меню
2. Таблиця показує:

- Ім’я пацієнта
- Прикріплений лікар
- Дата народження
- Кнопка "Переглянути інформацію"
- Кнопка "Редагувати"

5.2 Додавання нового пацієнта

Крок 1: Відкрийте форму

Натисніть **"Додати пацієнта"**

Крок 2: Заповніть основну інформацію

Обов’язкові поля:

- **Ім’я** - повне ПІБ пацієнта
- **Лікар** - виберіть лікаря зі списку (до якого прикріплюється пацієнт)
- **Дата народження** - формат ДД.ММ.РРРР
- **Стать** - Чоловіча / Жіноча

Контактні дані:

- **Номер телефону** - у форматі +380…
- **Електронна пошта** - email пацієнта
- **Адреса** - місце проживання

Крок 3: Збережіть

Натисніть **"Зберегти"**

5.3 Редагування пацієнта

1. Знайдіть пацієнта в таблиці
2. Натисніть **"Редагувати"**
3. Змініть необхідні дані
4. Можна змінити прикріпленого лікаря
5. Збережіть зміни

5.4 Перегляд медичної картки

1. Знайдіть пацієнта

2. Натисніть **"Переглянути інформацію"**
3. Відкриється детальна сторінка з вкладками:

Вкладки медичної картки:

- **Записи** - історія записів на прийом
- **Медичні візити** - нотатки лікаря, діагнози
- **Життєві показники** - історія вимірювань
- **Рецепти** - призначені ліки

Примітка: Адміністратор може переглядати медичні картки, але основну роботу з ними виконує лікар.

6. Управління записами

6.1 Перегляд записів

Є два способи перегляду записів:

Спосіб 1: Через конкретного лікаря

1. Розділ **"Лікарі"** → Виберіть лікаря → **"Переглянути записи"**
2. Виберіть дату
3. Перегляньте записи на цю дату

Спосіб 2: Через панель керування

- На головній сторінці відображаються сьогоднішні записи

6.2 Створення запису

Крок 1: Відкрийте форму

1. Перейдіть до розділу записів лікаря
2. Натисніть **"Додати запис"**

Крок 2: Виберіть пацієнта

1. У полі **"Пацієнт"** почніть вводити ім’я або телефон
2. З’явиться список пацієнтів
3. Виберіть потрібного зі списку

Крок 3: Виберіть дату і час

1. **Дата** - оберіть дату прийому через календар
2. **Час** - виберіть доступний час зі списку

– Відображаються тільки вільні слоти відповідно до графіка лікаря
– Заброньовані слоти не відображаються

Крок 4: Збережіть

Натисніть **"Зберегти"**

6.3 Редагування запису

1. У таблиці записів знайдіть потрібний запис
2. Натисніть кнопку **"Редагувати"**
3. Можна змінити:
   – Пацієнта
   – Дату
   – Час
   – Статус (Заплановано/Завершено/Скасовано)
4. Збережіть зміни

6.4 Статуси записів

Статус	Значення	Коли встановлюється
Заплановано	Запис активний, очікується візит	При створенні запису
Завершено	Пацієнт відвідав лікаря	Після проведення прийому
Скасовано	Запис скасовано	Вручну адміністратором або пацієнтом

7. Налаштування системи

7.1 Зміна мови інтерфейсу

1. У правому верхньому куті знайдіть перемикач мови
2. Натисніть **"УКР"** або **"EN"**
3. Інтерфейс миттєво перемкнеться на обрану мову

Доступні мови: - UA Українська (УКР) - GB English (EN)

7.2 Зміна теми оформлення

1. У правому верхньому куті знайдіть перемикач теми
2. Натисніть на іконку місяця або сонця
3. Тема миттєво зміниться

Доступні теми:

- **Світлий режим** - білий фон, темний текст
- **Темна тема** - темний фон, світлий текст

Корисно: Темна тема зменшує навантаження на очі при роботі вночі або в темному приміщенні.

7.3 Вихід з системи

1. Натисніть кнопку **"Вийти"** у верхньому меню
2. Підтвердіть вихід у діалоговому вікні
3. Система перенаправить вас на сторінку входу

Контрольний список для адміністратора

Щоденні завдання

- ❑ Перевірити сьогоднішні записи на панелі керування
- ❑ Переконатися, що всі лікарі мають доступ до системи
- ❑ Відповісти на запити щодо відновлення паролів

При додаванні нового лікаря

- ❑ Створити запис лікаря в розділі "Лікарі"
- ❑ Налаштувати графік роботи лікаря
- ❑ Створити обліковий запис користувача з роллю "Лікар"
- ❑ Прив’язати користувача до лікаря
- ❑ Передати логін і пароль лікарю
- ❑ Перевірити, що лікар може увійти в систему

При додаванні нового пацієнта

- ❑ Створити запис пацієнта в розділі "Пацієнти"
- ❑ Прикріпити пацієнта до лікаря

- ❑ Заповнити контактні дані
- ❑ За потреби створити обліковий запис з роллю "Пацієнт"
- ❑ Прив’язати користувача до пацієнта
- ❑ Передати логін і пароль пацієнту

При зміні розкладу лікаря

- ❑ Перевірити існуючі записи на дати, що змінюються
- ❑ Попередити пацієнтів про зміни (за потреби)
- ❑ Оновити графік роботи лікаря
- ❑ Перенести записи, що конфліктують

Часті помилки та їх вирішення

Помилка: "Не вдалося зберегти користувача"

Причини: - Ім’я користувача вже існує - Не заповнено обов’язкові поля - Не вибрано роль

Рішення: - Придумайте унікальне ім’я користувача - Перевірте заповнення всіх полів - Обов’язково виберіть роль зі списку

Помилка: "Не вдалося зберегти лікаря"

Причини:

- Некоректний формат графіка роботи
- Не введено ім’я лікаря

Рішення:

- Використовуйте конструктор графіка замість ручного введення
- Перевірте формат часу (24-годинний)
- Введіть повне ПІБ лікаря

Помилка: "Не вдалося зберегти запис"

Причини:

- Не вибрано пацієнта зі списку
- Обраний час вже зайнятий

- Дата у минулому

Рішення:

- Обов’язково виберіть пацієнта зі списку пошуку (не просто введіть ім’я)
- Оберіть інший вільний час
- Перевірте правильність введеної дати

Користувач не може увійти

Перевірте:

1. Чи увімкнено обліковий запис (галочка "Увімкнено")
2. Чи правильний пароль
3. Чи прив’язаний користувач до лікаря/пацієнта (для ролей крім адміністратора)

Безпека та конфіденційність

Рекомендації щодо паролів

- Створюйте надійні паролі (мінімум 8 символів)
- Використовуйте комбінацію літер, цифр та символів
- Рекомендуйте користувачам змінити пароль після першого входу
- Не зберігайте паролі у відкритому вигляді (система не зберігає паролі, лише їхні хеш-коди)

Управління доступом

- Вчасно відключайте облікові записи звільнених працівників
- Регулярно перевіряйте список активних користувачів
- Використовуйте статус "Вимкнено" замість видалення (для збереження історії)

Резервне копіювання

Важливо: Регулярно створюйте резервні копії бази даних системи. Зверніться до системного адміністратора щодо налаштування автоматичного резервного копіювання.

Підтримка

При виникненні технічних проблем, які не вирішуються за допомогою цієї інструкції, зверніться до технічної підтримки системи.

ДОДАТОК В

ПРОГРАМА ТА МЕТОДИКА ВИПРОБУВАНЬ

1 Загальні відомості

Даний документ встановлює програму та методику випробувань програмного забезпечення «Автоматизація облікової діяльності мережі сімейних амбулаторій КНП «ЦПМСД №1» м. Краматорська» (далі – Система). Документ визначає обсяг, порядок виконання та критерії оцінювання результатів випробувань з метою підтвердження відповідності Системи вимогам технічного завдання (Додаток А) і готовності до дослідної/промислової експлуатації.

Вихідні дані та посилання на документацію, що використовується під час випробувань: технічне завдання (Додаток А), інструкція користувача (Додаток Б)

2 Об’єкт і мета випробувань

Об’єктом випробувань є вебзастосунок Системи, розгорнутий на сервері застосунку та підключений до реляційної бази даних MySQL, що забезпечує роботу ролей користувачів: адміністратор, лікар, пацієнт. Метою випробувань є перевірка коректності реалізації функціональних можливостей (облік пацієнтів і лікарів, керування записами на прийом, ведення візитів і електронної медичної картки, фіксація показників життєдіяльності, формування призначень/рецептів, звітність), а також підтвердження нефункціональних вимог щодо безпеки, надійності, продуктивності та сумісності.

3 Умови та засоби випробувань

Випробування проводяться у тестовому середовищі, що імітує робоче розгортання Системи. Мінімальна конфігурація: сервер застосунку (ОС Windows/Linux; JDK 21+; Spring Boot; сервер додатка або вбудований Tomcat), СУБД MySQL 8.x; клієнтські робочі місця з ОС Windows 10/11 та браузерами Google Chrome/ Microsoft Edge/ Mozilla Firefox актуальних версій. Доступ до Системи здійснюється через HTTPS (за наявності налаштування) або HTTP у

локальному стенді.

Засоби випробувань: веббраузер; інструмент адміністрування БД (DBeaver/Workbench) для контролю стану таблиць і перевірки цілісності даних; журнал серверу застосунку (лог-файли); засоби створення резервної копії БД (mysqldump). Для тестування використовуються попередньо створені облікові записи: ADMIN_01 (роль «адміністратор»), DOC_01 (роль «лікар»), PAT_01 (роль «пацієнт»), а також набір тестових пацієнтів/лікарів/послуг/розкладів, підготовлений відповідно до предметної області.

Перед початком випробувань необхідно:

(1) розгорнути Систему на тестовому сервері;

(2) виконати ініціалізацію бази даних та застосувати міграції (за наявності);

(3) імпортувати довідники (послуги, статуси, спеціальності) і створити тестові облікові записи;

(4) перевірити доступність вебінтерфейсу з клієнтського ПК;

(5) зафіксувати версію збірки Системи та дату/час початку випробувань.

4 Склад випробувань і порядок проведення

Випробування виконуються у такій послідовності:

4.1 перевірка розгортання та доступності;

4.2 функціональні випробування модулів;

4.3 інтеграційні випробування взаємодії з БД та зовнішніми сервісами (за наявності);

4.4 випробування засобів захисту (автентифікація, авторизація, аудит);

4.5 випробування сумісності (браузери/ОС);

4.6 випробування надійності та відновлення (резервне копіювання/відновлення, обробка помилок);

4.7 оцінка продуктивності для типових операцій;

4.8 приймальні випробування за ролями.

4.1 Перевірка розгортання та доступності: запуск серверу застосунку, підключення до MySQL, відкриття стартової сторінки та сторінки входу. Очікуваний результат – відсутність критичних помилок у логах, коректне відображення сторінок.

4.2 Функціональні випробування: перевірка реалізації сценаріїв для ролей «адміністратор», «лікар», «пацієнт» згідно з контрольними прикладами (розділ 5). Очікуваний результат – коректне виконання операцій CRUD, зміна статусів, валідація даних, формування документів/звітів.

4.3 Інтеграційні випробування: перевірка цілісності даних у БД (зовнішні ключі, каскадні дії), коректності транзакцій при створенні запису/візиту/рецепта, а також взаємодії з зовнішнім сервісом е-рецептів (якщо підключений у стенді). Очікуваний результат – дані зберігаються узгоджено, помилки інтеграції обробляються з повідомленням користувачу.

4.4 Випробування безпеки: перевірка автентифікації, зміни пароля (за наявності), розмежування прав доступу за ролями, заборони доступу до сторінок/операцій без прав, фіксації подій у журналі. Очікуваний результат – користувачі бачать лише дозволені функції, несанкціонований доступ блокується.

4.5 Випробування сумісності: виконання базових сценаріїв у Chrome/Edge/Firefox, перевірка коректності верстки та відсутності критичних відмінностей. Очікуваний результат – функції доступні та працюють однаково в підтримуваних браузерах.

4.6 Випробування надійності та відновлення: створення резервної копії БД, відновлення на чистому екземплярі, перевірка збереження користувацьких даних; перевірка обробки помилкових введень і відмовостійкої поведінки (повідомлення, відсутність втрати даних). Очікуваний результат – успішне відновлення, відсутність пошкодження даних.

4.7 Оцінка продуктивності: вимірювання часу відповіді для типових операцій (пошук пацієнта, відкриття картки, збереження візиту, формування звіту)

при стандартному навантаженні. Очікуваний результат – час відповіді не перевищує 1 с для типових операцій пошуку/збереження (відповідно до вимог).

5 Контрольні приклади (тест-кейси)

ID	Роль	Передумови	Кроки	Очікуваний результат	Примітки
TC-01	Усі	Система розгорнута, є обліковий запис	1) Відкрити сторінку входу. 2) Ввести логін/пароль. 3) Натиснути «Увійти».	Успішна автентифікація та перенаправлення на стартову сторінку ролі.	ADMIN_01, DOC_01, PAT_01
TC-02	Усі	Користувач неавторизований	1) Перейти за URL сторінки, доступної лише після входу.	Відмова у доступі та перенаправлення на вхід/повідомлення про заборону.	Перевірити різні сторінки
TC-03	Адмін	ADMIN_01 авторизований	1) Відкрити довідник «Лікарі». 2) Додати нового лікаря. 3) Зберегти.	Запис з’являється у списку, зберігається у БД, валідація обов’язкових полів.	ПІБ, телефон, кабінет, графік
TC-04	Адмін	Є пацієнт і лікар	1) Створити запис на прийом (дата/час). 2) Перевірити відображення у розкладі лікаря.	Запис створено, статус «Заплановано», конфлікти часу блокуються.	Перевірити перетин інтервалів
TC-05	Лікар	DOC_01 має запланований запис	1) Відкрити «Мій розклад». 2) Натиснути «Почати візит». 3) Заповнити скарги, діагноз, план. 4) Зберегти.	Створено візит, дані збережено у картці, статус візиту коректний.	Перевірити прив’язку до appointment
TC-06	Лікар	Є відкритий візит	1) Додати показники (АТ, пульс, t°, зріст, вага). 2) Зберегти.	Показники збережено, ІМТ розраховано автоматично (за наявності), відображення в історії.	Перевірити діапазони валідації

ID	Роль	Передумови	Кроки	Очікуваний результат	Примітки
ТС-07	Лікар	Є візит пацієнта	1) Створити призначення/рецепт (препарат, дозування, частота, тривалість). 2) Зберегти/ відправити.	Рецепт збережено, доступний у картці; при відправленні – фіксується статус та результат.	За наявності інтеграції – перевірити API
ТС-08	Адмін	Є дані за період	1) Відкрити «Звіти». 2) Обрати тип і період. 3) Сформувати. 4) Експортувати у PDF/CSV.	Звіт сформовано коректно, експорт створює файл, дані відповідають БД.	Перевірити фільтри (лікар/амбулаторія/послуга)
ТС-09	Пацієнт	PAT_01 авторизований	1) Переглянути доступні записи/історію. 2) Спробувати переглянути дані іншого пацієнта.	Доступ лише до власних даних; спроба доступу до чужих – заборонена.	Перевірка авторизації на рівні серверу
ТС-10	Адмін	Є резервна копія БД	1) Виконати резервне копіювання. 2) Розгорнути на чистій БД. 3) Перевірити дані та входи.	Відновлення успішне, дані доступні, система запускається без помилок.	mysqldump

6 Критерії приймання результатів

Система вважається такою, що пройшла випробування, якщо виконуються умови: (1) тест-кейси ТС-01…ТС-10 виконані з результатом «успішно» або з незначними зауваженнями, що не впливають на основні функції; (2) відсутні дефекти категорії «критичний/високий пріоритет» (збої, втрата даних, порушення прав доступу); (3) підтверджено відповідність ключовим нефункціональним вимогам: час відповіді для типових операцій у нормальних умовах не перевищує 1 с, система працює у підтримуваних браузерах, наявне резервне копіювання/відновлення; (4) сформовано протокол випробувань із підписами відповідальних осіб.

7 Оформлення результатів випробувань

За результатами випробувань оформлюється протокол (звіт), що містить: найменування Системи та версію; дату і місце проведення; склад комісії/відповідальних осіб; опис стенду (ПЗ/ОС/браузери/версія MySQL); перелік виконаних тест-кейсів із фактичними результатами; перелік виявлених дефектів із пріоритетами та рекомендаціями щодо усунення; висновок про відповідність/невідповідність Системи вимогам та рішення щодо допуску до експлуатації.

ДОДАТОК Г.

ВИХІДНІ ТЕКСТИ ПРОГРАМНИХ МОДУЛІВ

Appointment.java

```java
package ua.edu.nuos.doctors.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.time.LocalDateTime;

@NoArgsConstructor
@Entity
@Getter
@Setter
@Table(name = "appointment")
public class Appointment {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;

    @Column(name = "date_time", nullable = false)
    private LocalDateTime dateTime;

    @Enumerated(EnumType.STRING)
    @Column(name = "status", nullable = false, length = 20)
    private AppointmentStatus status;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "patient_id")
    private Patient patient;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "doctor_id")
    private Doctor doctor;

    @PrePersist
    public void prePersist() {
        if (status == null) {
            status = AppointmentStatus.BOOKED;
        }
    }
}
```

AppointmetStatus.java

```java
package ua.edu.nuos.doctors.entity;

public enum AppointmentStatus {
    BOOKED,
    CANCELLED,
```

```
    COMPLETED
}
```

Doctor.java

```
package ua.edu.nuos.doctors.entity;

import jakarta.annotation.PostConstruct;
import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.util.Collection;

@Entity
@NoArgsConstructor
@Table(name = "doctors")
@Getter
@Setter
public class Doctor {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;

    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "times", nullable = false)
    private String appointmentTimes;

    @OneToMany(mappedBy = "doctor")
    private Collection<Appointment> appointments;

    @OneToMany(mappedBy = "doctor")
    private Collection<Patient> patients;

    @OneToMany(mappedBy = "doctor")
    private Collection<Visit> visits;

    @OneToOne
    @JoinColumn(name = "user_id", unique = true)
    private User user;

    public Doctor(String name, String appointmentTimes) {
        this.name = name;
        this.appointmentTimes = appointmentTimes;
    }

    @PrePersist
    public void prePersist() {
        if (appointmentTimes == null) {
```

```
            appointmentTimes = "";
        }
    }
}
```

Patient.java

```
package ua.edu.nuos.doctors.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.time.LocalDate;
import java.util.Collection;

@NoArgsConstructor
@Entity
@Getter
@Setter
@Table(name = "users")
public class Patient {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;

    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "birth_date", nullable = false)
    private LocalDate birthDate;

    @Column(name = "gender")
    private String gender;

    @Column(name = "phone_number", nullable = false)
    private String phoneNumber;

    @Column(name = "email", nullable = false)
    private String email;

    @Column(name = "address", nullable = true)
    private String address;

    @ManyToOne
    @JoinColumn(name = "doctor_id")
    private Doctor doctor;

    @OneToOne
    @JoinColumn(name = "user_id", unique = true)
    private User user;

    @OneToMany(mappedBy = "patient")
    private Collection<Appointment> appointments;
```

```
    @OneToMany(mappedBy = "patient")
    private Collection<Prescribing> prescriptions;
}
```

Prescribing.java

```
package ua.edu.nuos.doctors.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.time.LocalDate;

@NoArgsConstructor
@Getter
@Setter
@Table(name = "prescribing")
@Entity
public class Prescribing {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "medicine_name", nullable = false)
    private String medicineName;


    @Column(name = "dosage")
    private String dosage;

    @Column(name = "frequency")
    private String frequency;

    @Column(name = "duration_days")
    private Integer durationDays;

    @Column(name = "instructions")
    private String instructions;

    @Column(name = "created_date", nullable = false)
    private LocalDate createdDate;

    // ACTIVE / COMPLETED / CANCELLED
    @Column(name = "status", nullable = false)
    private String status;

    @ManyToOne
    private Visit visit;

    @ManyToOne
    private Patient patient;

    @ManyToOne
```

```
    private Doctor doctor;
}
```

User.java

```
package ua.edu.nuos.doctors.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import org.hibernate.proxy.HibernateProxy;

import java.util.Objects;

@Getter
@Setter
@Entity
@Table(name = "users_table")
public class User {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    @Column(nullable = false)
    private String role;     // ROLE_ADMIN, ROLE_DOCTOR, ROLE_PATIENT

    @Column(nullable = false)
    private boolean enabled;

    @Column(name = "must_change_password", nullable = false)
    private boolean mustChangePassword = false;

    @Override
    public final boolean equals(Object o) {
        if (this == o) return true;
        if (o == null) return false;
        Class<?> oEffectiveClass = o instanceof HibernateProxy ?
((HibernateProxy) o).getHibernateLazyInitializer().getPersistentClass() :
o.getClass();
        Class<?> thisEffectiveClass = this instanceof HibernateProxy ?
((HibernateProxy) this).getHibernateLazyInitializer().getPersistentClass() :
this.getClass();
        if (thisEffectiveClass != oEffectiveClass) return false;
        User user = (User) o;
        return getId() != null && Objects.equals(getId(), user.getId());
    }

    @Override
```

```
    public final int hashCode() {
        return this instanceof HibernateProxy ? ((HibernateProxy)
this).getHibernateLazyInitializer().getPersistentClass().hashCode() :
getClass().hashCode();
    }
}
```

Visit.java

```
package ua.edu.nuos.doctors.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.time.LocalDateTime;

@NoArgsConstructor
@Entity
@Getter
@Setter
@Table(name = "visits")
public class Visit {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "visit_time", nullable = false)
    private LocalDateTime visitTime;

    @Column(name = "complaint", columnDefinition = "TEXT")
    private String complaint;

    @Column(name = "diagnosis", columnDefinition = "TEXT")
    private String diagnosis;

    @Column(name = "plan", columnDefinition = "TEXT")
    private String plan;

    @Enumerated(EnumType.STRING)
    @Column(name = "status", nullable = false, length = 20)
    private VisitStatus status;

    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "appointment_id")
    private Appointment appointment;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "patient_id", nullable = false)
    private Patient patient;
```

```
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "doctor_id", nullable = false)
    private Doctor doctor;

    @PrePersist
    public void prePersist() {
        if (visitTime == null) {
            visitTime = LocalDateTime.now();
        }

        if (status == null) {
            status = VisitStatus.OPEN;
        }
    }
}
```

VisitStatus.java

```
package ua.edu.nuos.doctors.entity;

public enum VisitStatus {
    OPEN,
    CLOSED,
    CANCELLED
}
```

VitalSigns.java

```
package ua.edu.nuos.doctors.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.time.LocalDateTime;

@NoArgsConstructor
@Getter
@Setter
@Entity
@Table(name = "vital_signs")
public class VitalSigns {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "temperature")
    private Double temperature;

    @Column(name = "systolic")
    private Integer systolic;
```

```
    @Column(name = "diastolic")
    private Integer diastolic;

    @Column(name = "pulse")
    private Integer pulse;

    @Column(name = "spo2")
    private Integer spo2;

    @Column(name = "weight")
    private Double weight;

    @Column(name = "height")
    private Double height;

    @Column(name = "bmi")
    private Double bmi;

    @Column(name = "notes", columnDefinition = "TEXT")
    private String notes;

    @Column(name = "measured_at", nullable = false)
    private LocalDateTime measuredAt;

    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "visit_id", nullable = false, unique = true)
    private Visit visit;

    @PrePersist
    public void prePersist() {
        if (measuredAt == null) {
            measuredAt = LocalDateTime.now();
        }

        if (weight != null && height != null && height > 0) {
            double heightInMeters = height / 100.0;
            bmi = weight / (heightInMeters * heightInMeters);
            bmi = Math.round(bmi * 10.0) / 10.0;
        }
    }
}
```

PatientRepository.java

```
package ua.edu.nuos.doctors.repository;

import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import ua.edu.nuos.doctors.entity.Patient;

import java.util.List;
```

```
import java.util.Optional;

public interface PatientRepository extends JpaRepository<Patient, Long> {
    Optional<Patient> findByEmailAndPhoneNumber(String email, String
phoneNumber);

    List<Patient>
findTop10ByNameContainingIgnoreCaseOrPhoneNumberContainingIgnoreCaseOrderByNameA
sc(String name, String phone);

    Optional<Patient> findByUserId(Long userId);

    List<Patient> findByDoctorIdOrderByNameAsc(Long doctorId);

    boolean existsByIdAndDoctorId(Long id, Long doctorId);

    @Query("""
            select p from Patient p
            where p.doctor.id = :doctorId
              and (
                  lower(p.name) like lower(concat('%', :query, '%'))
                  or lower(p.phoneNumber) like lower(concat('%', :query, '%'))
              )
            order by p.name asc
            """)
    List<Patient> searchByDoctorId(@Param("doctorId") Long doctorId,
                                   @Param("query") String query,
                                   Pageable pageable);
}
```

SecurityConfiguration.java

```
package ua.edu.nuos.doctors.config;

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.AllArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecuri
ty;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import ua.edu.nuos.doctors.service.CustomUserDetailsService;

@Configuration
@EnableWebSecurity
```

```
@AllArgsConstructor
public class SecurityConfiguration {

    private final CustomUserDetailsService userDetailsService;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
                .authorizeHttpRequests(auth -> auth

                        // Public pages for visitors
                        .requestMatchers(
                                "/",
                                "/about",
                                "/contact",
                                "/login",
                                "/test/**",
                                "/css/**",
                                "/js/**",
                                "/images/**",
                                "/webjars/**"
                        ).permitAll()

                        // Admin pages
                        .requestMatchers("/dashboard").hasRole("ADMIN")
                        .requestMatchers("/admin/**").hasRole("ADMIN")

                        // Doctor pages
                        .requestMatchers("/doctors/**").hasAnyRole("ADMIN",
"DOCTOR")

                        // Patient pages
                        .requestMatchers("/patients/me").hasRole("PATIENT")
                        .requestMatchers("/patients/show").hasAnyRole("ADMIN",
"DOCTOR")
                        .requestMatchers("/patients/**").hasAnyRole("ADMIN",
"DOCTOR", "PATIENT")

                        // Medical actions
                        .requestMatchers("/visits/**").hasAnyRole("ADMIN",
"DOCTOR")
                        .requestMatchers("/vital-signs/**").hasAnyRole("ADMIN",
"DOCTOR")

                        // Everything else needs login
                        .anyRequest().authenticated()
                )
                .formLogin(form -> form
                        .loginPage("/login")
                        .successHandler(customSuccessHandler())
                        .permitAll()
                )
                .logout(logout -> logout
                        .logoutUrl("/logout")
```

```
                        .logoutSuccessUrl("/")
                        .invalidateHttpSession(true)
                        .clearAuthentication(true)
                        .deleteCookies("JSESSIONID")
                        .permitAll()
                )
                .userDetailsService(userDetailsService);

        return http.build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationSuccessHandler customSuccessHandler() {
        return (HttpServletRequest request, HttpServletResponse response,
Authentication authentication) -> {
            String redirectUrl = "/login";

            if (authentication.getAuthorities().contains(new
SimpleGrantedAuthority("ROLE_ADMIN"))) {
                redirectUrl = "/dashboard";
            } else if (authentication.getAuthorities().contains(new
SimpleGrantedAuthority("ROLE_DOCTOR"))) {
                redirectUrl = "/patients/show";
            } else if (authentication.getAuthorities().contains(new
SimpleGrantedAuthority("ROLE_PATIENT"))) {
                redirectUrl = "/patients/me";
            }

            response.sendRedirect(redirectUrl);
        };
    }
}
```

VisitService.java

```
package ua.edu.nuos.doctors.service;

import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import ua.edu.nuos.doctors.entity.*;
import ua.edu.nuos.doctors.repository.AppointmentRepository;
import ua.edu.nuos.doctors.repository.DoctorRepository;
import ua.edu.nuos.doctors.repository.PatientRepository;
import ua.edu.nuos.doctors.repository.VisitRepository;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;
```

```
@Service
@RequiredArgsConstructor
public class VisitService {

    private final VisitRepository visitRepository;
    private final PatientRepository patientRepository;
    private final DoctorRepository doctorRepository;
    private final AppointmentRepository appointmentRepository;

    public List<Visit> findByPatientId(Long patientId) {
        return visitRepository.findByPatientIdOrderByVisitTimeDesc(patientId);
    }

    public List<Visit> findByDoctorId(Long doctorId) {
        return visitRepository.findByDoctorIdOrderByVisitTimeDesc(doctorId);
    }

    public Optional<Visit> findByAppointmentId(Long appointmentId) {
        return visitRepository.findByAppointmentId(appointmentId);
    }

    public Visit findById(Long id) {
        return visitRepository.findById(id)
                .orElseThrow(() -> new IllegalArgumentException("Visit not found
with id: " + id));
    }

    @Transactional
    public Visit createVisit(Long patientId,
                             Long doctorId,
                             Long appointmentId,
                             LocalDateTime visitTime,
                             String complaint,
                             String diagnosis,
                             String plan,
                             VisitStatus status) {

        Visit visit = new Visit();

        visit.setComplaint(complaint);
        visit.setDiagnosis(diagnosis);
        visit.setPlan(plan);
        visit.setStatus(status != null ? status : VisitStatus.OPEN);

        if (appointmentId != null) {
            Appointment appointment =
appointmentRepository.findById(appointmentId)
                    .orElseThrow(() -> new IllegalArgumentException("Appointment
not found with id: " + appointmentId));

            if (visitRepository.existsByAppointmentId(appointmentId)) {
                throw new IllegalArgumentException("This appointment already has
a visit");
            }
```

```
            if (appointment.getPatient() == null ||
!appointment.getPatient().getId().equals(patientId)) {
                throw new IllegalArgumentException("This appointment does not
belong to this patient");
            }

            if (appointment.getDoctor() == null) {
                throw new IllegalArgumentException("This appointment does not
have a doctor");
            }

            visit.setAppointment(appointment);
            visit.setPatient(appointment.getPatient());
            visit.setDoctor(appointment.getDoctor());
            visit.setVisitTime(visitTime != null ? visitTime :
appointment.getDateTime());

        } else {
            Patient patient = patientRepository.findById(patientId)
                    .orElseThrow(() -> new IllegalArgumentException("Patient not
found with id: " + patientId));

            if (doctorId == null) {
                throw new IllegalArgumentException("Doctor is required for walk-
in visit");
            }

            Doctor doctor = doctorRepository.findById(doctorId)
                    .orElseThrow(() -> new IllegalArgumentException("Doctor not
found with id: " + doctorId));

            visit.setPatient(patient);
            visit.setDoctor(doctor);
            visit.setVisitTime(visitTime != null ? visitTime :
LocalDateTime.now());
        }

        return visitRepository.save(visit);
    }

    @Transactional
    public Visit updateVisit(Long visitId,
                             LocalDateTime visitTime,
                             String complaint,
                             String diagnosis,
                             String plan,
                             VisitStatus status) {

        Visit visit = findById(visitId);

        if (visitTime != null) {
            visit.setVisitTime(visitTime);
        }
```

```
        visit.setComplaint(complaint);
        visit.setDiagnosis(diagnosis);
        visit.setPlan(plan);

        if (status != null) {
            visit.setStatus(status);
        }

        return visitRepository.save(visit);
    }

    @Transactional
    public void deleteVisit(Long visitId) {
        visitRepository.deleteById(visitId);
    }
}
```

ДОДАТОК Д.

ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Д.1. Загальні відомості

Д.1.1 Позначення та найменування програми

Найменування: Застосунок для автоматизації облікової діяльності мережі сімейних амбулаторій

Позначення: ClinicApplication

Д.1.2 Програмне забезпечення, необхідне для функціонування програми

Застосунок є кросплатформеним веб-застосунком, що працює у середовищі браузера та серверної частини із використанням фреймворку Spring Boot мовою Java. Для його функціонування необхідна операційна система:

- Windows 10 або вище;
- Linux з ядром 4.x або вище;
- Mac OS не нижче версії 10.12.

Також потрібна наявність:

- Java Development Kit (версії не нижче 21), рекомендується BellSoft Liberica JDK;
- СУБД MySQL;
- веб-браузера (Google Chrome, Mozilla Firefox, Safari, тощо).

Д.1.3 Мови програмування

Основні мови програмування, використані при реалізації:

- Java (бекенд, серверна логіка);
- HTML/CSS (представлення);

- JavaScript (клієнтська взаємодія).

Д.2 Функціональне призначення

Д.2.1 Класи вирішуваних завдань і призначення програми

Програмне забезпечення призначене для автоматизації облікової діяльності мережі сімейних амбулаторій та реалізує такі функції:

- Додавання, редагування, видалення даних про лікарів, пацієнтів, прийоми, призначення;
- Генерація електронних рецептів;
- Формування звітів щодо прийомів;
- Управління графіками прийому лікарів;
- Ведення електронних медичних карток;
- Збереження історії прийомів та призначень;
- Можливість запису на прийом та отримання списку рекомендацій для пацієнтів.

Д.2.2 Функціональні обмеження

- Мінімальні апаратні ресурси клієнтського пристрою: 4 ГБ оперативної пам’яті, 1 ГГц процесор;
- Час відповіді на стандартний запит не повинен перевищувати 1 секунди;
- Застосунок функціонує лише за наявності доступу до мережі Інтернет (для клієнт-серверної взаємодії).

Д.3 Опис логічної структури

Програмне забезпечення побудовано за принципами багаторівневої клієнт-серверної архітектури із застосуванням шаблону MVC:

- Model (Модель) – реалізована за допомогою ORM Hibernate, як стандартної реалізації механізму Java Persistence у фреймворку Spring Data – відповідає за доступ до бази даних;
- View (Подання) – HTML/CSS-сторінки, що відображають дані для користувача, створені із використанням шаблонізатора Thymeleaf;
- Controller (Контролер) – реалізований у Java, відповідає за обробку HTTP-запитів, керування логікою взаємодії між Model і View.

Основні модулі:

- Модуль автентифікації;
- Модуль адміністратора;
- Модуль лікаря;
- Модуль пацієнта.

Д.4 Технічні засоби та програмні засоби

Апаратні засоби:

- ПК або ноутбук з процесором від 1 ГГц;
- Оперативна пам’ять від 4 ГБ;
- Жорсткий диск із вільним місцем не менше 500 МБ.

Програмні засоби:

- Операційна система: Windows 10+, Linux 4.x+, Mac OS 10.12+;
- СУБД MySQL
- Віртуальна машина Java (у складі Java Development Kit);
- Веб-браузер (для клієнта).