МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри, канд. техн. наук,

доцент _____ Є. О. Давиденко

«___»____2022 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

«Вебзастосунок обліку медичних карток пацієнтів»

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 408.21810815

Студент _____ А. В. Кім
підпис

«____»_____2022 р.

Керівник канд. техн. наук, доцент _____ Г. В. Горбань
підпис

«____»_____2022 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва
підпис

«____»_____2022 р.

Миколаїв – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інформаційних технологій і програмних систем

ЗАТВЕРДЖУЮ

Завідувач кафедри _____ Є. О. Давиденко
підпис

«____»_____2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи бакалавра

Видано студенту групи 408 факультету комп'ютерних наук

Кім Артем Віталійович

(прізвище, ім'я, по-батькові)

1. Тема кваліфікаційної роботи бакалавра «Вебзастосунок обліку медичних карток пацієнтів»

Затверджено наказом по ЧНУ від «01»___грудня 2021 р.___ № 314

2. Строк представлення кваліфікаційної роботи «___» _____ 2022 р.

3. Очікуваний результат роботи та вихідні дані, якщо такі потрібні

Очікуваним результатом кваліфікаційної роботи бакалавра є розроблений вебзастосунок для медичних карток, зі застосуванням сучасних підходів, методів та використанням WEB-технологій. Вихідними даними є: медичні картки з лікарями , для наповнення сайту та технічне завдання замовника.

4. Перелік питань, що підлягають розробці.

– аналіз вимог до вебзастосунків обліку медичних карток;

– огляд існуючих вебзастосунків обліку медичних карток;

– розробка вимог до вебзастосунку на основі інформації про предметну

– область та існуючі аналоги;

– проектування вебзастосунку підбору гри за параметрами на основі пошуку асоціативних правил;

– програмна реалізація, тестування та від лагодження відповідного вебзастосунку.

5. Перелік графічних матеріалів.
   Презентація кваліфікаційної роботи.
6. Завдання до спеціальної частини.
   Дослідження питань охорони праці, які безпосередньо пов'язані з діяльністю розробника програмного забезпечення.
7. Консультанти:

Консультант	Кафедра(організація)	Частина роботи
А. О. Алексєєва	**Кафедра екології**	Охорона праці

Керівник роботи канд. техн. наук, доцент Горбань Г. В
(*посада, прізвище, ім'я, по-батькові)*

_____
(*підпис)*

Завдання прийнято до виконання Кім А. В.
(*прізвище, ім'я, по-батькові студента)*

_____
(*підпис)*

Дата видачі завдання «15» грудня 2022 р

КАЛЕНДАРНИЙ ПЛАН

Виконання кваліфікаційної роботи

Тема: «Вебзастосунок обліку медичних карток пацієнтів»

№	Найменування роботи	Початок	Закінчення	Примітки
1	Видача теми КРБ	20.10.22	01.11.22	Виконано
2	Отримання завдання на виконання КРБ	15.01.22	15.01.22	Виконано
3	Збір інформації для виконання КРБ та підбір літератури	16.01.22	30.01.22	Виконано
4	Конструювання БД (таблиці, зв'язки між ними, тощо)	04.02.22	16.02.22	Виконано
5	Створення БД	17.02.22	20.03.22	Виконано
6	Конструювання вебзастосунку (створення макету вебзастосунку)	21.03.22	25.03.22	Виконано
7	Створення вебзастосунку та наповнення його даними	26.03.22	25.04.22	Виконано
8	Тестування вебзастосунку	26.04.22	05.05.22	Виконано
9	Оформлення звіту з КРБ	06.05.22	27.05.22	Виконано
10	Підведення підсумків виконання КРБ	30.05.22	31.05.22	Виконано
11	Попередній захист КРБ	03.06.22	04.06.22	Виконано
12	Подання КРБ рецензенту	05.06.22	12.06.22	Виконано
13	Подання КРБ, її електронної копії та інших документів (відгуку, рецензії) до захисту	13.06.22	19.06.22	Виконано
14	Захист КРБ перед Державною екзаменаційною комісією (ДЕК)	17.06.22	17.06.22	Виконано

Розробив студент Кім Артем Віталійович
(прізвище, ім'я, по батькові студента)(*підпис*)

« » 2022 р.

Керівник роботи канд. техн. наук, доцент Горбань Гліб Валентинович
(посада, прізвище, ім'я, по батькові)(*підпис*)

« » 2022 р.

АНОТАЦІЯ

до кваліфікаційної роботи на тему:

«Вебзастосунок обліку медичних карток пацієнтів»

Студент 408 гр.: Кім Артем Віталійович

Керівник: канд. техн. наук, доцент Горбань Г. В.

В даній кваліфікаційній роботі бакалавра завдання є створення вебзастосунка для обліку медичних карток пацієнтів, яка вирішує проблему з зберіганням паперових медичних карток у сховищах медичних установ.

Актуальність цієї роботи корисна для українських мед установ, бо спростить процес створення та зберігання історій захворювань пацієнтів.

Об'єкт кваліфікаційної роботи є процес створення медичних карток для пацієнтів.

Предмет кваліфікаційної роботи є технології та підходи щодо розробки вебзастосунків, орієнтованих на лікарів та пацієнтів, які працюють з медичними картками.

Метою роботи є підвищення ефективності роботи медичних закладів шляхом створення вебзастосунку для керування медичними картками пацієнтів.

Кваліфікаційна робота бакалавра містить наступні розділи:

– Аналіз предметної області;

– Моделювання та проєктування вебзастосунку для медичних установ;

– Реалізація вебзастосунку.

Кваліфікаційна робота складається фахового розділу та спеціальної частини з охорони праці та безпеки в надзвичайних ситуаціях.

Робота складається зі вступу, трьох розділів, спеціальної частини з охорони праці, висновків та додатків.

У першому розділі кваліфікаційної роботи досліджено аналіз предметної області, в якій проаналізовано розвиток оцифрування медичної сфери. Досліджено сучасні технології розробки, сучасні клієнтські та серверні мови програмування, бази даних і види сучасних вебзастосунків. Проаналізовано аналоги електронних медичних карток і сформовано специфікації вимог до вебзастосунку.

У другому розділі кваліфікаційної роботи змодельовано діаграму варіантів використання. Створено найважливіші сценарії використання. Розроблено макет основних сторінок вебзастосунку. Створено діаграми, такі як: діаграма діяльності, фізична модель бази даних, діаграми компонентів для серверної та клієнтської частини і діаграму розгортання. В кінці другого розділу описано технології, які будуть використані для створення вебзастосунку.

В третьому розділі кваліфікаційної роботи розглянуто і описано архітектуру вебзастосунку, та його компонентів, такі як: модель, вид та контролер. Реалізація компонентів була додана до додатків. Також описано функції системи, та показано інтерфейс вебзастосунку.

КРБ викладена на _59__сторінки, вона містить _3__ розділи, _39__ілюстрацій, _25__ таблиці, _25_ джерел в переліку посилань.

Ключові слова: *база даних, вебзастосунок, дослідження, апробація, моделювання , аналогічних вебзастосунків.*

ASTRACT

of the Bachelor's Thesis

"Web Application of Patient Medical Records"

Student of group 408: Kim Artem Vitalievich

Supervisor: Ph.D. tech. Sc., Associate Professor Gorban G. V.

In this qualifying work of a bachelor, the task is to create a web application for accounting of medical records of patients, which solves the problem of storing paper medical records in the repositories of medical institutions.

The relevance of this work is useful for Ukrainian medical institutions, because it will simplify the process of creating and storing medical histories of patients.

The object of qualification work is the process of creating medical cards for patients.

The subject of the qualification work is technologies and approaches for the development of web applications aimed at doctors and patients who work with medical cards.

The aim of the work is to increase the efficiency of medical institutions by creating a web application for managing medical records of patients.

Thesis contains the following sections:

– Analysis of the subject area;

– Modeling and designing a web application for medical institutions;

– Implementation of a web application.

Qualification work consists of a professional section and a special part on labor protection and safety in emergencies.

The work consists of an introduction, three sections, a special part on labor protection, conclusions and appendices.

In the first section of the qualification work the analysis of the subject area is investigated, in which the development of digitization of the medical sphere is analyzed. Modern development technologies, modern client and server programming languages, databases and types of modern web applications are studied. Analogues of electronic

medical cards are analyzed and specifications of requirements for web application are formed.

In the second section of the qualification work the use diagram is modeled. The most important usage scenarios have been created. The layout of the main pages of the web application has been developed. Diagrams have been created, such as: activity diagram, physical database model, component and client component diagrams, and deployment diagram. The end of the second section describes the technologies that will be used to create the web application.

The third section of the qualification work considers and describes the architecture of the web application and its components, such as: model, type and controller. Implementation of components has been added to the applications. The system functions are also described, and the web application interface is shown.

The bachelor's thesis is presented on _59_ pages, it contains _3_ sections, _39__ illustrations, _25_ tables, _25__ sources in the list of references.

Keywords: database, web application, research, testing, modeling, similar web applications.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ГБ	–	Гігабайт
ГГц	–	Гігагерц
ЕМК	–	Електронна медична картка
ЕСОЗ	–	Електронна система охорони здоров’я
НСЗУ	–	Національна служба здоров'я України
ТБ	–	Терабайт

AJAX	–	Asynchronous JavaScript And XML
CSS	–	Cascading Style Sheets
ERD	–	Entity-Relationship diagram
MVC	–	Model-View-Controller
HTML	–	HyperText Markup Language
SPA	–	Single Page Application

ВСТУП

Кожен громадянин України має у медичному закладі свою медкартку. Медкартка заповнюється кожного разу, коли пацієнт відвідує медустанову, при цьому медкартки збільшуються в розмірах. В одних медзакладах виділяються під зберігання медичних карток стелажі, в інших – цілий кабінет або поверх. Медичні карти не захищені від загоряння або руйнування всередині приміщення, де зберігається архів, який може бути знищений.

Електронна медична картка – це історія звернень та захворювань пацієнта, яка зберігається не на папері, а в електронній системі.

Головними перевагами електронних медкарток над паперовими є надійність зберігання даних пацієнтів та зручність використання історією пацієнта. Електронні медкартки можна переглядати скрізь де є комп'ютер та інтернет.

Кількість мед установ які бажають мати вебзастосунок, який зберігатиме медкартки в електронному вигляді, стрімко збільшується, що підвищує рівень зацікавленості в цій сфері. Таким чином, все більше лікарів вимагають інструменту для введення історії пацієнта.

Вебзастосунок позбавить мед установу від необхідності зберігати медкартки в мед закладах, а пацієнти зможуть за кілька кліків дізнатися всю інформацію про їх здоров'я. Отже, **актуальність** цієї роботи корисна для українських медустанов, бо спростить процес створення та зберігання історій захворювань пацієнтів.

Сучасний вебзастосунок розробляється за допомогою таких основних технологій: AJAX, CSS, HTML, ASP.NET Core MVC, T-SQL.

Об’єкт кваліфікаційної роботи є процес створення медичних карток для медустанов.

Предмет кваліфікаційної роботи є технології та підходи щодо розробки вебзастосунків орієнтованих на лікарів, які працюють з медичними картками.

Тому, було досліджено можливість створення вебзастосунку, який буде створювати та зберігати історію звернень та захворювань пацієнта.

Метою роботи є підвищення ефективності роботи медичних закладів шляхом створення вебзастосунку для керування медичними картками пацієнтів.

Відповідно до мети визначено такі **завдання**:

- аналіз предметної області та існуючих аналогів;
- формування специфікацію вимог до вебзастосунку;
- створення макету вебзастосунку;
- моделювання вебзастосунку;
- підбір технологій для ефективної розробки вебзастосунку;
- розробка вебзастосунку.

Сфера застосування:

Вебзастосунок буде використовуватися українськими поліклініками для обліку пацієнтів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Диджиталізація медичної сфери в Україні

В Україні проходить цифрова трансформація в медичній системі, яка розпочалась у 2016 році Міністерством охорони здоров'я та Національною службою здоров'я України [2].

Цілі трансформації медичної системи: забезпечення прозорості фінансування; надання можливості працювати без паперу; сформування бізнес-середовища; створення простору для інновацій в медицині; сприяти розвитку медичного IT-ринку. Для вирішення цілій було створено електрону систему «eHealth».

Електронна система охорони здоров'я «eHealth» – це інформаційно-телекомунікаційна система, в якій користувач через МІС взаємодіє з центральною базою даних.

Система eHealth складається з:

1) Центральної бази даних (ЦБД) – інформаційно-телекомунікаційна система, яка містить передбачені законодавством реєстри, програмні модулі, інформаційну систему НСЗУ в частині, необхідній для реалізації державних фінансових гарантій та інших [2].

Також, забезпечує виконувати такі дії, як обмін інформації та документами між реєстрами, перегляд, створення, електронними медичними інформаційними системами, державними електронними інформаційними ресурсами.

2) МІС (електронна медична інформаційна система) — інформаційно-телекомунікаційна система, головне призначення якої є автоматизація первинних процесів, корельована із роботою медичних установ загальної та вузької спеціалізації, яка дає змогу створювати, переглядати, обмінюватися інформацією в електронній формі [2].

На початок 2018 року в Україні відмовились від ведення паперової документації у сфері охорони здоров'я. Терапевти, сімейні лікарі та педіатри

медичних установ, що під'єднані до Електронної системи охорони здоров’я (ЕСОЗ), отримали змогу вносити інформацію про пацієнтів, створюючи для них ЕМК.

1.2 Електронні медичні картки

Раніше документування історій хвороби було трудомістким і дорогим, але впровадження програмного забезпечення для електронних медичних карток змінило цю ситуацію. Медичні організації всіх розмірів та спеціальностей забезпечують більш точне лікування, що призводить до підвищення ефективності за допомогою систем ЕМК.

Електронна медична картка (ЕМК) — це цифрова версія всієї інформації, яку ви зазвичай знаходите в паперовій книжці мед установи: історія хвороби, діагнози, ліки, історія імунізації, алергія, результати лабораторних досліджень та записки лікаря.

1) Хмарні системи ЕМК. Які розміщені на сторонньому сервері, доступ до якого здійснюється через Інтернет. Сторонній постачальник обслуговує все серверне обладнання, забезпечує безпеку та будь-які оновлення програмного забезпечення. Вони пропонуються на основі недорогої місячної підписки. Це одна з найбільших переваг цих послуг; медична практика не повинна нести витрати на обладнання та обслуговування, які пов’язані з локальним розгортанням медичних систем.

2) Система ЕМК на базі сервера. Яка розміщена в медустанові. Хоча перевага локального розгортання полягає в тому, що ви маєте контроль над обладнанням, це також є недоліком. Медичні установи, які впроваджують системи ЕМК, повинні мати IT-фахівця для підтримки апаратного забезпечення, програмного забезпечення та даних, які вони зберігають. Якщо обладнання вийшло з ладу, це може призвести до несподіваних витрат.

1.3 Дослідження сучасних технологій розробки для рішення проблем у даній предметній області

1.3.1 Мови програмування

Рік за роком і навіть місяць за місяцем тенденції в мовах програмування постійно міняються. Веброзробники використовують різні мови програмування в залежності від того, що вони створюють:

– Деякі мови програмування застосовуються до зовнішнього інтерфейсу або сторони клієнта. Інтерфейсна веброзробка розробляє всі інтерфейси та візуальні елементи, які бачать користувачі, від кнопок до графіки. Такі мови, як HTML і CSS, є інтерфейсними мовами.

– Інші мови програмування, такі як Python і Java, в першу чергу призначені для розробки вебзастосунків. Ці так звані «мови на стороні сервера» забезпечують комунікацію між інтерфейсами та базами даних і серверами, які його підтримують.

1.3.2 Бази даних

Керування даними у вебзастосунку відіграє вирішальну роль у забезпеченні чудової взаємодії з користувачем. Незалежно від того, наскільки добре розроблений інтерфейс програми або наскільки чистий код, все це не має значення, якщо вебзастосунок не може швидко отримувати, обробляти та передавати інформацію. Дані також потрібно захищати від потенційних зловмисників. На щастя, усіх цих моментів можна досягти, вибравши правильну систему керування базами даних.

При виборі сучасної бази даних одним з найважливіших рішень є вибір реляційної (SQL) або нереляційної (NoSQL) структури даних. Хоча обидва варіанти є життєздатними, між ними є ключові відмінності, про які користувачі повинні пам’ятати, приймаючи рішення.

П'ять критичних відмінностей між SQL і NoSQL [6]:

1) Бази даних SQL є реляційними, бази даних NoSQL нереляційними;

2) Бази даних SQL використовують структуровану мову запитів і мають попередньо визначену схему. Бази даних NoSQL мають динамічні схеми для неструктурованих даних;

3) Бази даних SQL масштабуються вертикально, тоді як бази даних NoSQL масштабуються горизонтально;

4) Бази даних SQL засновані на таблицях, у той час як бази даних NoSQL є сховищами документів, ключів-значень, графіків або широких стовпців;

5) Бази даних SQL найкраще підходять для багаторядкових транзакцій, а NoSQL найкраще підходить для неструктурованих даних, таких як документи або JSON.

1.3.3 Архітектура сучасних вебзастосунків

Змінюється також архітектура вебзастосунків, від застарілої архітектури HTML-вебзастосунка до мікросервісної. Розглянемо трошки детальніше види архітектур.

1) Мікросервіси (або архітектура мікросервісів) — це архітектурний підхід на основі хмарної архітектури, в якому одна програма складається з багатьох слабко пов’язаних і незалежно розгорнутих менших компонентів або служб.

Переваги для бізнесу та організацій:

– код можна оновлювати простіше – нові функції можна додавати, не торкаючись усієї програми;

– команди можуть використовувати різні стеки та різні мови програмування для різних компонентів;

– компоненти можна масштабувати незалежно один від одного, зменшуючи втрати та витрати, пов’язані з масштабуванням цілих додатків, оскільки одна функція може мати надто велике навантаження.

Недоліки мікросервісної архітектури:

– складності розрахунку та розміру мікросервісу в кастомних рішеннях: весь функціонал розбитий на кілька блоків, що може ускладнити правильний розрахунок їх кількості та призначення;

– складна процедура тестування між розподіленими середовищами: тестування всієї програми з огляду на її децентралізацію може бути складним;

– складне управління зі збільшенням кількості послуг: такі системи децентралізовані, що ускладнює управління зі зростанням їх кількості;

– відносно висока вартість реалізації: перехід від моноліту до мікросервісу зазвичай займає багато грошей та часу.

2) Програма SPA — це окрема сторінка, яка постійно взаємодіє з користувачем, динамічно перезаписуючи поточну сторінку, а не завантажуючи з сервера нові сторінки.

Вони бувають двох типів:

– вміст кількох сторінок на одній сторінці «оболонки» з іншої;

– використання AJAX для отримання контенту кожного разу, коли користувач натискає посилання, тому на сторінці буде оновлюватися лише змінений контент, а решта сторінки залишається незмінною.

Переваги односторінкового вебзастосунку:

– Швидкий. Вебзастосунок на одній сторінці оновлює лише потрібний вміст, це значно покращує час завантаження. Більшість ресурсів, таких як HTML, CSS і скрипти, завантажуються лише один раз, на самому початку. Крім того, завантажуючи лише невеликі фрагменти контенту, SPA-мітки прискорюють взаємодію.

– Адаптований макет. SPA легко адаптуються до браузерів мобільних пристроїв, а також до дизайну мобільних програм, якщо ви хочете запропонувати їх своїм користувачам разом із вебверсією.

1.4 Огляд та аналіз наявних аналогів

1.4.1 Система управління даними «Doctor ELEKS»

Розумна система управління даними «Doctor ELEKS» забезпечує клініки єдиною уніфікованою медичною картою. Він допомагає особам, які здійснюють догляд, безпечно зберігати та отримувати доступ до відповідної інформації про пацієнтів відповідно до стандартів медичних даних HIPAA, HL7, CDA, ICD. Ви можете отримати доступ до відповідних клінічних записів у будь-який час, у будь-якому місці та на будь-якому пристрої, забезпечуючи безперебійне обслуговування пацієнтів із повною впевненістю в безпеці.

Doctor ELEKS — це програмне рішення, яке можна налаштувати, масштабоване; вже успішно розгорнуто у понад 200 медичних закладах. Він пропонує інноваційний «деревоподібний перегляд» шаблонів документів і повністю оптимізований інтерфейс для легкого залучення користувачів [16].

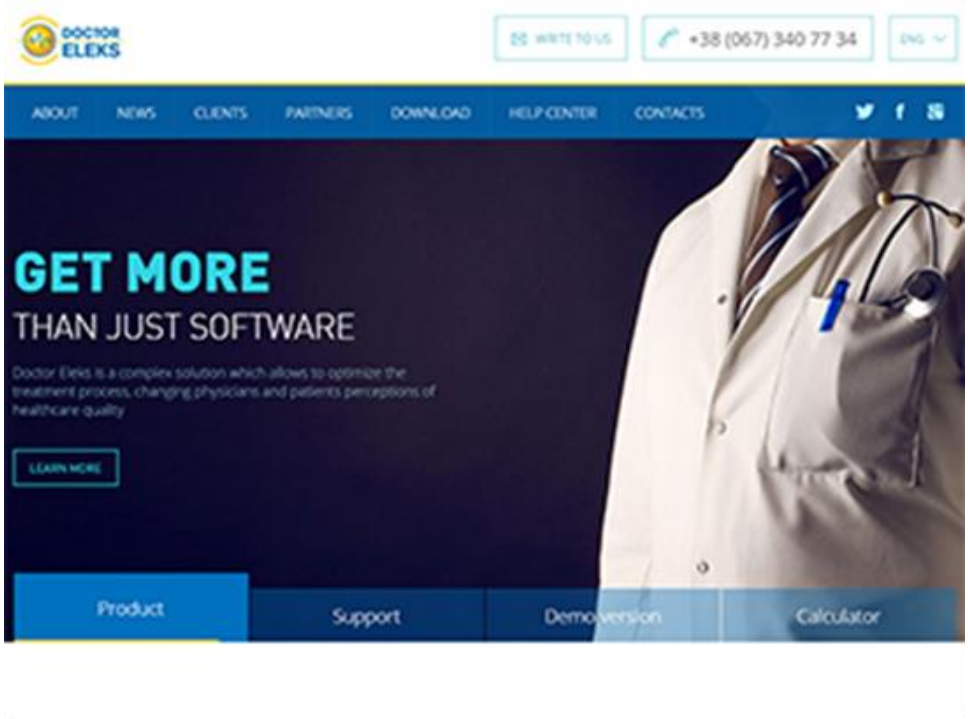

Рисунок 1.1 – Головна сторінка

Основні функції системи «Doctor ELEKS»:

– фіксація комунікацій з пацієнтом;

– створення медичної картки пацієнта та редагування її даних;
– перегляд історії візитів пацієнта;
– керування курсами пацієнта;
– пошук медичної картки пацієнта;
– шаблони медичних пацієнтів;
– створення завдань для медичного персоналу;
– форматування звітів та облікової документації відповідно до форм МОЗ України;
– створення електронного рецепта;
– експорт звітів у файли формату pdf та xlsx.

Плюси. Потужний функціонал, наявність комунікаційного сервера для обміну даними у форматі HL7 із суміжними ІС, зовнішніми лабораторіями, страховими компаніями. Присутній інтеграція з Toshiba УЗД, підтримується імпорт DICOM-зображень, підключення DICOM-сумісного обладнання та багато іншого. Doctor ELEKS підключений до системи eHealth, система пройшла перевірку і рекомендована до використання МОЗ України.

Мінуси:

– немає доступу для пацієнта;
– неможливість онлайн-запису в чергу;
– відсутність кросплатформеної підтримки;
– застарілий інтерфейс системи.

1.4.2 Вебсистема «InSync Healthcare Solutions»

InSync Healthcare Solutions пропонує хмарне програмне забезпечення ЕМК з керуванням пацієнтом, електронним виписуванням рецептів і телемедициною. Медичні працівники можуть вибирати з більш ніж 80 типів оцінки та створювати власні форми для підвищення якості обслуговування[21].

InSync пропонує комплексну реалізацію, налаштовану відповідно до потреб кожної практики, а також послуги з навчання, щоб усі користувачі почувалися комфортна в системі[21].

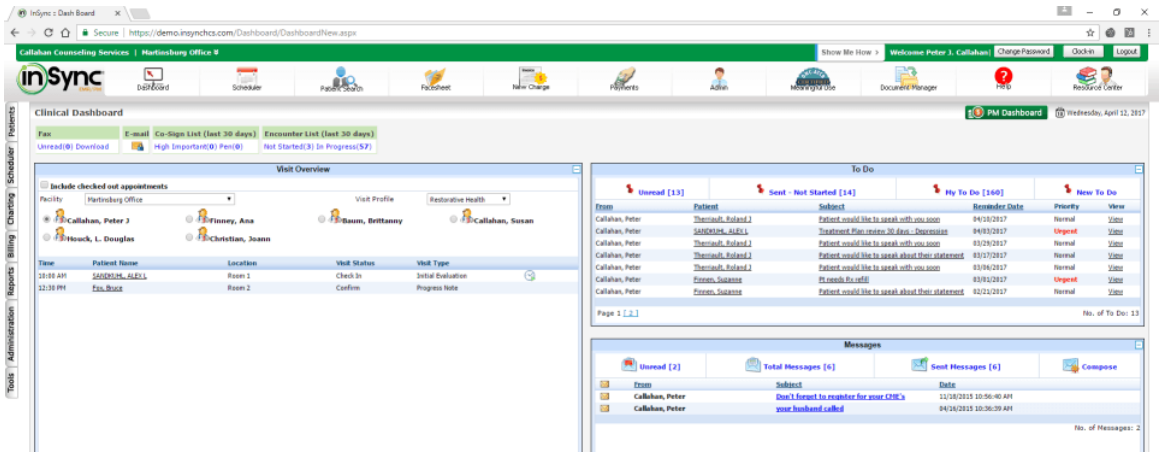

Рисунок 1.2 – Головний інтерфейс вебзастосунку

Основні функції системи «InSync Healthcare Solutions»:

– прийом та реєстрація пацієнтів;
– створення медичної картки пацієнта та редагування її даних;
– запис пацієнта на прийом до лікаря;
– форматування та друк статистичних звітів;
– редагування деталей візиту(видалення, перенесення, час, лікар, кабінет);
– можливість прикріплювати медичні документи та зображення;
– створення електронного рецепта;
– форматування журналів лабораторних обстежень;
– створення та друк результатів лабораторних обстежень.

Плюси:

– налаштовується під потреби організації;
– регулярні оновлення для покращення функцій;
– доброзичлива та корисна команда підтримки.

Мінуси:

– звіти не повністю налаштовуються;
– зависання вебзастосунку.

1.4.3 Вебсистема «InSync Healthcare Solutions»

CureMD — це хмарне рішення, яке виконує низку адміністративних та клінічних завдань медичного закладу, від планування візитів та документації пацієнтів до процесу виставлення рахунків та оплати [13]. Він підходить для різних організацій охорони здоров'я та спеціальностей усіх розмірів. Користувачі можуть легко отримувати доступ до медичної інформації пацієнта та оновлювати її, а також автоматизувати трудомісткі завдання, що повторюються.

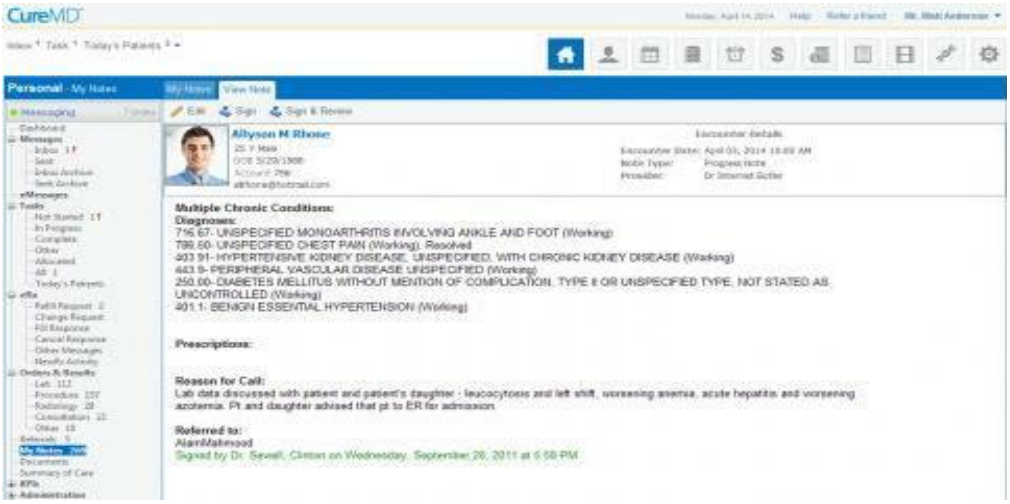

Рисунок 1.3 – Сторінка пацієнта

Основні функції системи «CureMD»:

– запис пацієнта на прийом до лікаря;
– редагування деталей візиту(видалення, перенесення, час, лікар, кабінет);
– створення медичної картки пацієнта та редагування її даних;
– створення звітів на основі внесених в систему даних;
– форматування спеціалізованих звітів щодо діяльності відділень;
– перегляд історії змін документів.

Плюси:

– має мобільний застосунок для iOS;
– дозволяє обмінюватися даними між зацікавленими сторонами та підключатися до платників, ракових регістрів, обміну медичною інформацією,

служб радіології та візуалізації, пацієнтів, аптек та обладнання, сумісного з DICOM;

Таблиця 1.1 – Порівняльна таблиця аналогів ЕМК;

	CureMD	InSync Healthcare Solutions	Doctor ELEKS
Генерування історії пацієнта на друк	+	-	+
Завантаження медичних карток в форматі pdf	-	+	+
Додання відгука до лабораторних досліджень	+	+	+
Випис ліків	-	+	-
Запис до лікаря	-	+	+

У табл.1.1 наведено наявність необхідного функціоналу. Усі недоліки будуть виправлені в системі, що розробляється.

1.5 Специфікація вимог

1.5.1 Призначення проєкту

Вебзастосунок має:

– зберігати інформацію: персональні дані всіх користувачів, історію обстежень пацієнта, ліків, результатів лабораторних досліджень та записів до лікаря;

– виводити інформацію, яку запитує лікар: свої персональні дані, список доданих до системи пацієнтів, історію звернень, історію записів, історію відгуків до лабораторних досліджень;

– виводити інформацію, яку запитує адміністратор: облікові дані лікарів, історію записів, інформацію про пацієнта.

1.5.2 Опис основних функцій системи

Вебзастосунок обліку медичних карток пацієнтів має наступний набір функціональних можливостей:

– авторизація користувачів в системі з різним доступом до інформаційної системи;

– здійснення CRUD-операцій адміністратором над даними про обліковий запис лікаря;

– здійснення CRUD-операцій адміністратором над даними про запис пацієнта до лікаря;

– здійснення CRUD-операцій лікарем над даними про запис пацієнта до лікаря;

– здійснення CRUD-операцій лікарем над даними про обстеження пацієнта;

– здійснення CRUD-операцій лікарем над даними про відгук до лабораторних досліджень;

– друк медичної картки пацієнта;

– завантаження інформації про обстеження пацієнта в форматі pdf;

1.6 Опис ролей в системі

– гість (неавторизований користувач);

– лікар (користувач, який здійснює управління своїми даними облікового запису, створює відгук на обстеження, виписує ліки пацієнта, розшифровує лабораторні дослідження та створює відгук, змінює дату прийому пацієнта);

– адміністратор(користувач, який створює обліковий запис для лікаря, додає пацієнта до системи, створює запис пацієнта до лікаря).

1.7 Вимоги до технічного забезпечення

Таблиця 1.2 – Мінімальні системні вимоги до сервера

Комплектуючі	Характеристики
Процесор	6-ядерний процесор з тактовою частотою не менш ніж 3.60ГГц.
Дисковий простір	Не менш ніж 1 терабайт
Оперативна пам’ять	Не менше 16 гігабайт
Операційна система	64-розрядна, Windows 7 Server та вище, Ubuntu Server 22.04 LTS
Трафік/с	1 ГБ/с 40ТБ

Таблиця 1.3 – Оптимальні системні вимоги до сервера

Комплектуючі	Характеристики
Процесор	16-ядерний процесор з тактовою частотою не менш ніж 3.6 ГГц
Дисковий простір	Не менш ніж 4 терабайти
Оперативна пам’ять	Не менше 32 гігабайт
Операційна система	64-розрядна, Windows 10 Server, Ubuntu Server 22.04 LTS
Комплектуючі	Характеристики
Трафік/с	1 ГБ/с 40ТБ

Таблиця 1.4 – Вимоги до браузера для коректної роботи клієнтського застосунку

Назва продукту	Мінімальна версія
Google Chrome	Частково з версії 2, повна підтримка з версії 21 та вище
Safari	Частково з версії 3.1, повна підтримка з версії 5 та вище
Internet Explorer	Частково з версії 9, повна підтримка з версії 10 та вище

Специфікації вимог допоможуть розробнику створити відповідний прийнятим правилам вебзастосунок, що є основою успішної розробки.

Висновки до розділу 1

В першому розділі кваліфікаційної роботи досліджено аналіз предметної області, в якій проаналізовано розвиток оцифрування медичної сфери. Також проаналізовано поняття електронної медичної картки та його типів.

Крім того, досліджено сучасні технології розробки для розв'язання проблеми у даній предметній області. Проаналізовано сучасні клієнтські та серверні мови програмування, бази даних і архітектури сучасних вебзастосунків.

Також проаналізовано аналоги ЕМК в Україні, та у всьому світі. Виявлені функціональні спроможності, плюси та мінуси ЕМК.

Дослідивши аналоги ЕМК сформульовано специфікацію вимог: призначення проекту; загальний опис; сфери застосування; опис основних функцій системи; опис ролей користувачів в системі; вимоги до технічного забезпечення.

Таким чином, мені вдалося виконати всі завдання для першої частини кваліфікаційної роботи, що в подальшому дає можливість виконувати інші частини.

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ МЕДИЧНИХ УСТАНОВ

2.1 Моделювання прецедентів

Для моделювання прецедентів було створено діаграму прецедентів та найважливіші сценарії використання. Діаграма прецедентів є цінними для візуалізації функціональних вимог системи, що вплине на вибір дизайну та пріоритети розробки.

Створена модель включає в себе список акторів, список прецедентів, схему використання та їх опис. Згідно з вимогами до системи, що моделюється, можна виділити наступних акторів: гість, лікар, адміністратор.

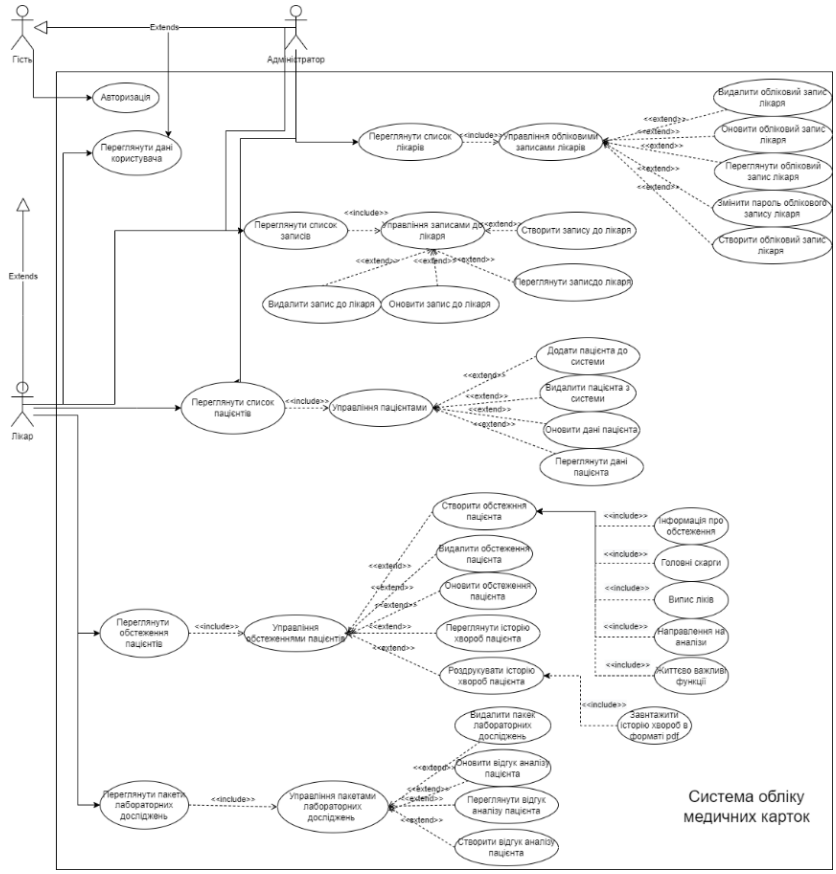

Рисунок 2.1 – Діаграма використання для системи обліку медичних карток

До діаграми використання створено найважливіші сценарії для випадків використання, що представлені на табл. 2.1–2.5.

Таблиця 2.1 – Створення запису до лікаря

Діючі особи	Система, Адміністратор
Мета	Створення запису до лікаря.
Передумова	Адміністратор авторизований до системи .
Успішний сценарій: 1. Адміністратор переходить до сторінки з записами пацієнтів; 2. Адміністратор натискає кнопку «Створити»; 3. Відкривається модальне вікно; 4. В модальному вікні адміністратор заповнює форму; 5. Адміністратор натискає кнопку «Зберегти»; 6. Система відправляє дані форми до таблиці бази даних; 7. База даних відправляє до системи успішне підтвердження отримання даних; 8. Система відправляє повідомлення адміністратору про успішне додавання запису до лікаря; 9. Система додає звернення до таблиці звернень.	
Результат	Адміністратор додав запис пацієнта до лікаря в таблицю звернень.
Розширення	
1а	Адміністратор не знаходить необхідного пацієнта в комбінованому списку, та виходить з модального вікна.

Таблиця 2.2 – Створення обстеження

Діючі особи	Система, Лікар
Мета	Додання обстеження пацієнта
Передумова	Лікар авторизований до системи

Кінець таблиці 2.2

Успішний сценарій: 1. Лікар переходить до сторінки з обстеженнями пацієнтів; 2. Лікар натискає кнопку «Створити»; 3. Відкривається модальне вікно; 4. В модальному вікні лікар заповнює форму про прийом до лікаря ; 5. В модальному вікні лікар заповнює форму про основні скарги пацієнта; 6. В модальному вікні лікар заповнює таблицю направленнями на аналізи; 7. В модальному вікні лікар заповнює таблицю ліками для пацієнта ; 8. В модальному вікні лікар заповнює форму про життєво-важливих функцій; 9. Адміністратор натискає кнопку «Зберегти»; 10. Система відправляє дані модального вікна до таблиці бази даних; 11. База даних відправляє до системи успішне підтвердження отримання даних; 12. Система відправляє повідомлення лікарю про успішне додавання обстеження; 13. Система додає звернення до таблиці звернень.	
Результат	Лікар додав обстеження пацієнта до таблиці обстежень.
Розширення	
1а	Лікар не знаходить необхідного пацієнта, та виходить з модального вікна;

Таблиця 2.3 – Роздрукувати історію пацієнта

Діючі особи	Система, Браузер, Лікар
Мета	Друк історії хвороб пацієнта.
Передумова	Лікар авторизований до системи.

Кінець таблиці 2.3

Успішний сценарій: 1. Лікар переходить до сторінки з обстеженнями пацієнтів; 2. Лікар знаходить обстеження необхідного пацієнта; 3. Лікар натискає кнопку «Роздрукувати»; 4. Лікар переходить на сторінку, в якій бачить всю історію хвороб пацієнта; 5. Лікар натискає кнопку «Друк»; 6. Система генерує документ для друку; 7. Лікар переходить у браузерне вікно для вибору та налаштування принтера.	
Результат	Лікар роздрукував згенерований документ з історіями хвороб пацієнта.

Таблиця 2.4 – Створити відгук для лабораторних досліджень пацієнта

Діючі особи	Система, Лікар
Мета	Створення відгуку для лабораторних досліджень пацієнта.
Передумова	Лікар авторизований до системи.
Успішний сценарій: 1. Лікар переходить до сторінки з відгуками на пакет лабораторних досліджень; 2. Лікар натискає кнопку «Створити»; 3. Відкривається модальне вікно з формою та таблицею; 4. В модальному вікні лікар заповнює форму про деталі пакета лабораторних досліджень; 5. В модальному вікні лікар заповнює таблицю відгуками про лабораторні дослідження пакета; 6. Лікар натискає кнопку «Зберегти». 7. База даних відправляє до системи успішне підтвердження отримання даних;	

Кінець таблиці 2.4

Продовження успішного сценарію: 8. Система відправляє повідомлення лікарю про успішне додавання відгуків до лабораторних досліджень пакета; 9. Система додає деталі пакета лабораторних досліджень в таблицю.	
Результат	Відгуки для лабораторних досліджень пакету успішно додались до бази даних, та к історії хвороби пацієнта.
Розширення	
1а	Лікар не знаходить необхідного пацієнта в комбінованому списку, та виходить з модального вікна.

Таблиця 2.5 – Авторизація користувача

Діючі особи	Система, Гість
Мета	Авторизація користувача.
Передумова	Користувач не авторизований.
Успішний сценарій: 1. Гість вводить свою пошту та пароль; 2. Система перевіряє дані; 3. Користувач авторизувався в системі.	
Результат	Користувач авторизувався в системі.
Розширення	
1а	Система не знаходить користувача в системі.
2а	Гість припустився помилки при введенні облікових даних.

У деяких методологіях вони можуть початися з коротких бізнес-сценаріїв, потім перерости в детальні сценарії використання системи, а потім у надзвичайно детальні та всеосяжні тести.

2.2 Розробка макету вебзастосунку

При розробці вебзастосунку корисно мати уявлення про те, як він має виглядати. Після визначення вимог і створення діаграми використання та сценаріїв до прецедентів будо створено макет вебзастосунку.

В ідеалі макет стає точкою сходження думок клієнта, дизайнера, маркетолога, копірайтера та програміста. Всі вони повинні бути задоволені готовим макетом. У цьому випадку тривалість проекту скорочується до мінімуму і нічого переробляти не потрібно.

Moqups — це візуальний інструмент для спільної роботи, який поєднує в собі дошку, діаграми та функції дизайну в одному онлайн-застосунку [12]. Вебзастосунок є безкоштовним програмним забезпеченням, але тільки для некомерційних організацій. Для того щоб отримати повнофункціональний обліковий запис, потрібно відправити запит на отримання ліцензії. Для створення макету було використано найсучасніший вебзастосунок Moqups.

Відрізняється Moqups від інших каркасних платформ, тим, що вони дозволяють користувачам перемикатися між каркасами, діаграмами та прототипами без необхідності перемикання платформ.

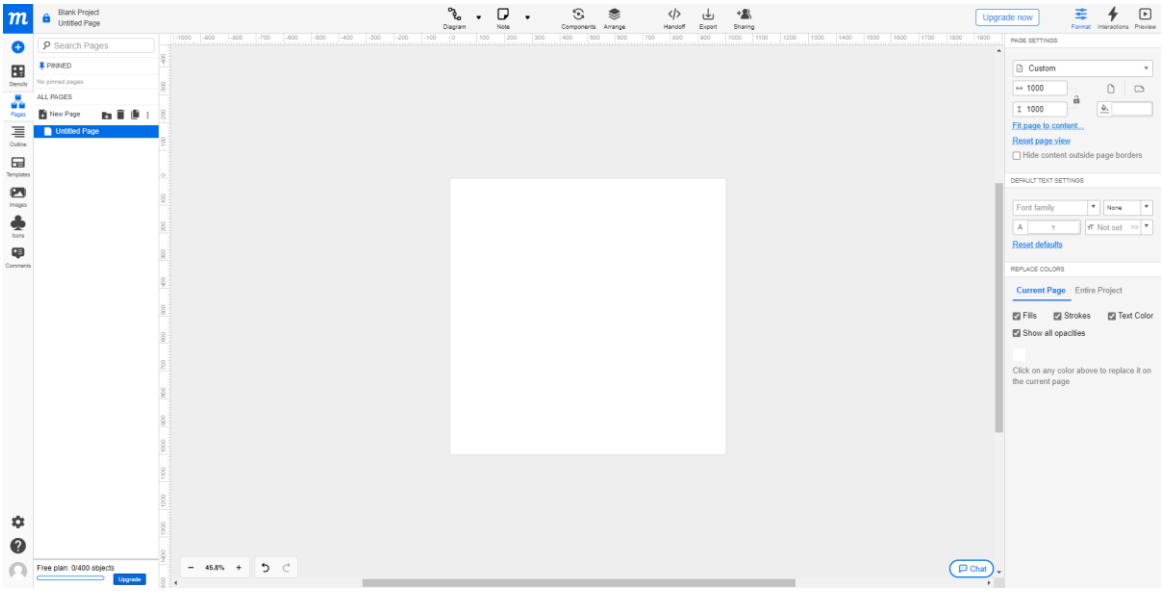

Рисунок 2.2 – Інтерфейс Moqups

2.2.1 Макет сторінки входу

Початок роботи користувача починається з сторінки авторизації, яка дозволяє неавторизованому користувачеві увійти до вебзастосунку. Щоб увійти до системи, користувач повинен вести свої ідентифікаційні дані. На рис. 2.3 зображена форма, яка має декілька обов’язкових полів для заповнення, а саме: електронну пошту і пароль користувача. Якщо користувач захоче щоб система запам’ятала його введені дані, він може увімкнути прапорець з надписом «Запам’ятати мене».

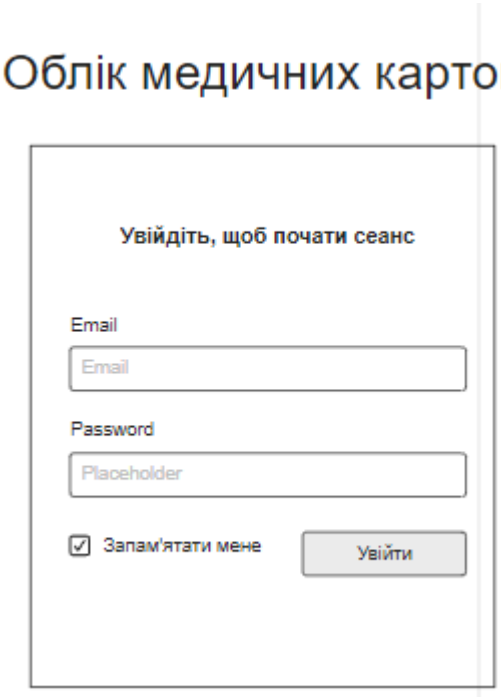

Рисунок 2.3 – Макет форми авторизації

2.2.2 Макети сторінок користувача з ролем «Лікар»

Після аутентифікації та авторизації відбувається входження на сторінку з певними ресурсами, що належать ролі авторизованого користувача. На рис. 2.4 зображена структура інтерфейсу сторінки, яка містить: навігаційну панель з лівої сторони, заголовок та підвал. Навігаційна панель містить в собі: назву вебзастосунку, ім’я і прізвище користувача, та посилання на контент. Заголовок містить кнопку «Вийти з акаунта». Підвал містить інформацію про проєкт.

Якщо на сторінці є таблиця та користувачу потрібно збільшити кількість рядків в таблиці, він може за допомогою комбінованого списку вибрати кількість рядків на одній таблиці. Таблиця має лічильник рядків та пагінацію сторінок.

Навігаційна панель користувача з ролем «Лікар» містить такі посилання на контент: історія записів, історію обстежень, історія відгуків на аналізи, профіль користувача.

Контент посилання «Історія записів»

Контент посилання «Історія записів» на рис. 2.4 містить таблицю, в якій містять записи пацієнтів до лікаря. У колонці з назвою «Дії» знаходиться дві кнопки: змінити та видалити.

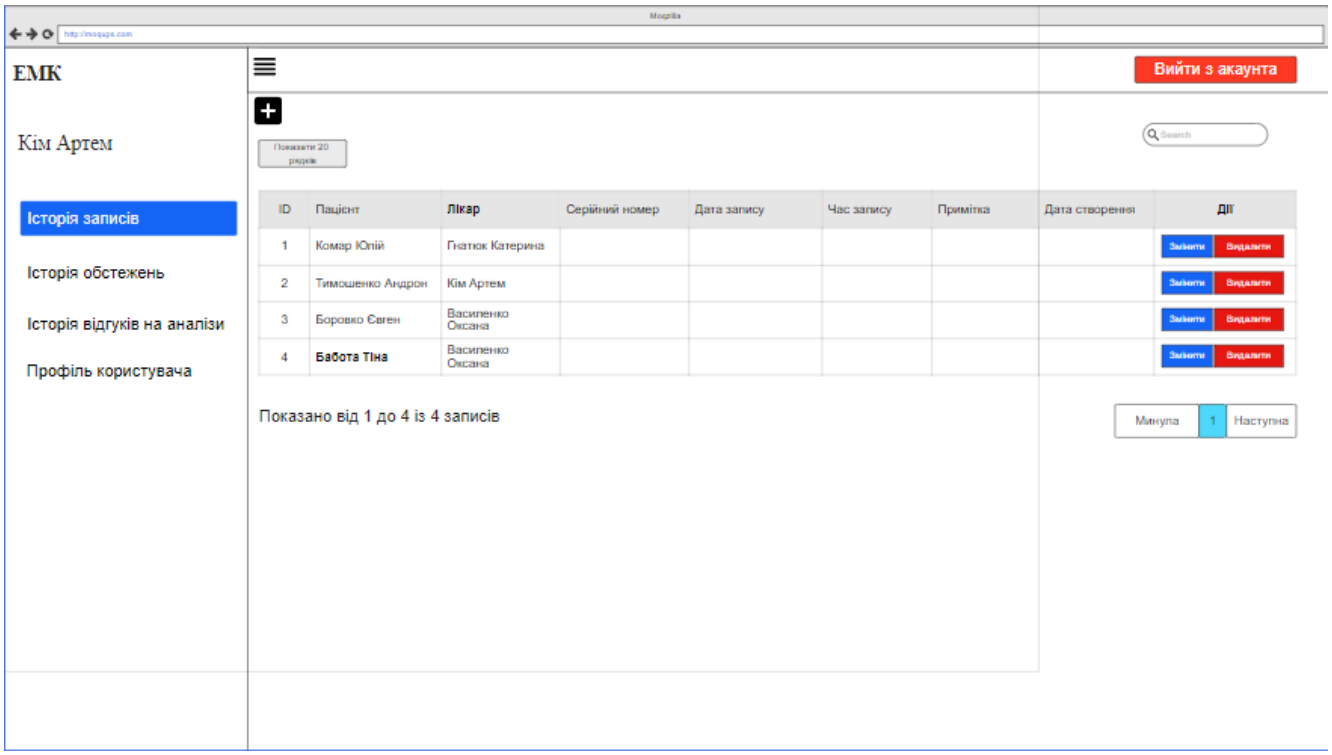

Рисунок 2.4 – Макет посилання «Історія записів»

Це посилання є як у лікаря, так і у адміністратора. Адміністратору потрібне це посилання, щоб записати пацієнта до лікаря і для управління записами пацієнтів. Лікарю потрібне це посилання, щоб планувати свій робочий час. Якщо адміністратора немає на його робочому місці, лікар може самостійно записати пацієнта.

Якщо лікарю потрібно додати запис пацієнта до системи, йому потрібно натиснути на кнопку з іконкою «Додати». Після цього спливає модальне вікно. Модальне вікно на рис. 2.5 містить форму для заповнення інформацією про запис до лікаря. Після заповнення запису, лікар може зберегти або закрити модальне вікно.

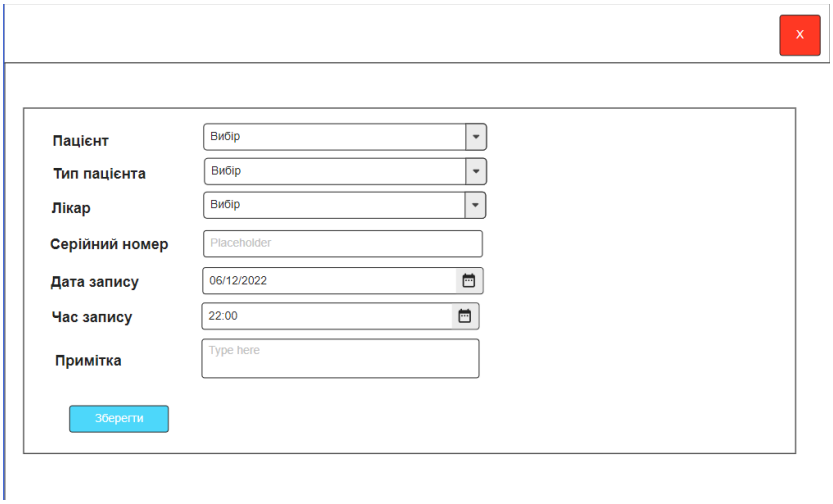

Рисунок 2.5 – Макет модального вікна для створення запису до лікаря

Контент посилання «Історія обстежень»

Контент посилання «Історія обстежень» на рис. 2.6 містить основну інформацію про звернення пацієнта до лікаря у таблиці. У колонці з назвою «Дії» знаходиться чотири кнопки: історія, друк, змінити і видалити.

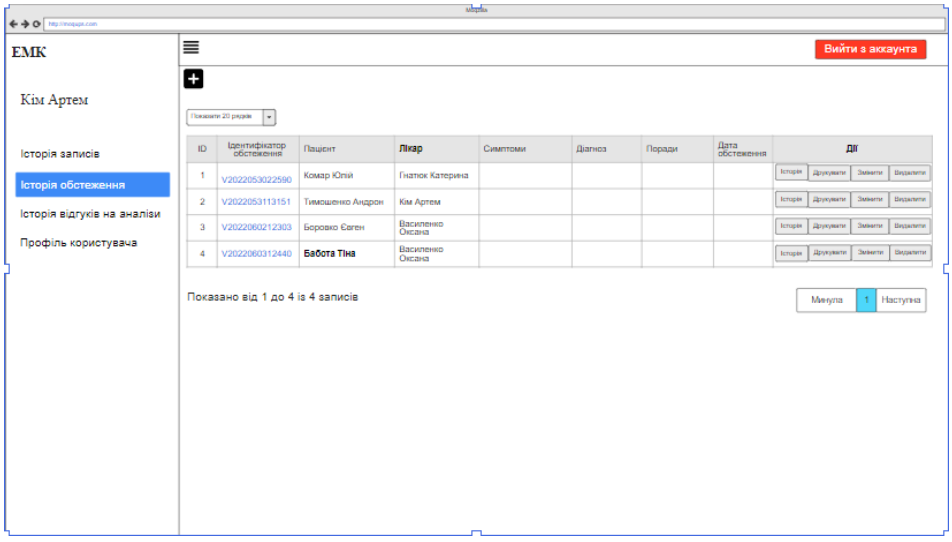

Рисунок 2.6 – Макет посилання «Історія обстежень»

Якщо лікарю потрібно додати обстеження пацієнта до системи, йому потрібно натиснути на кнопку з іконкою «Додати». Після цього спливає модальне вікно. Модальне вікно на рис. 2.7 містить головну інформацію звернення, і має форму з вкладками для документування обстеження пацієнта, що звертається до лікаря. Форма з вкладками складається з: головні скарги, лікування, розлідування і життєво-важливих функцій.

У вкладці «Головні скарги» лікар описує симптоми хворого пацієнта, надає повний висновок медогляду, робить запис про розвиток хвороби, і надає повний висновок хворого.

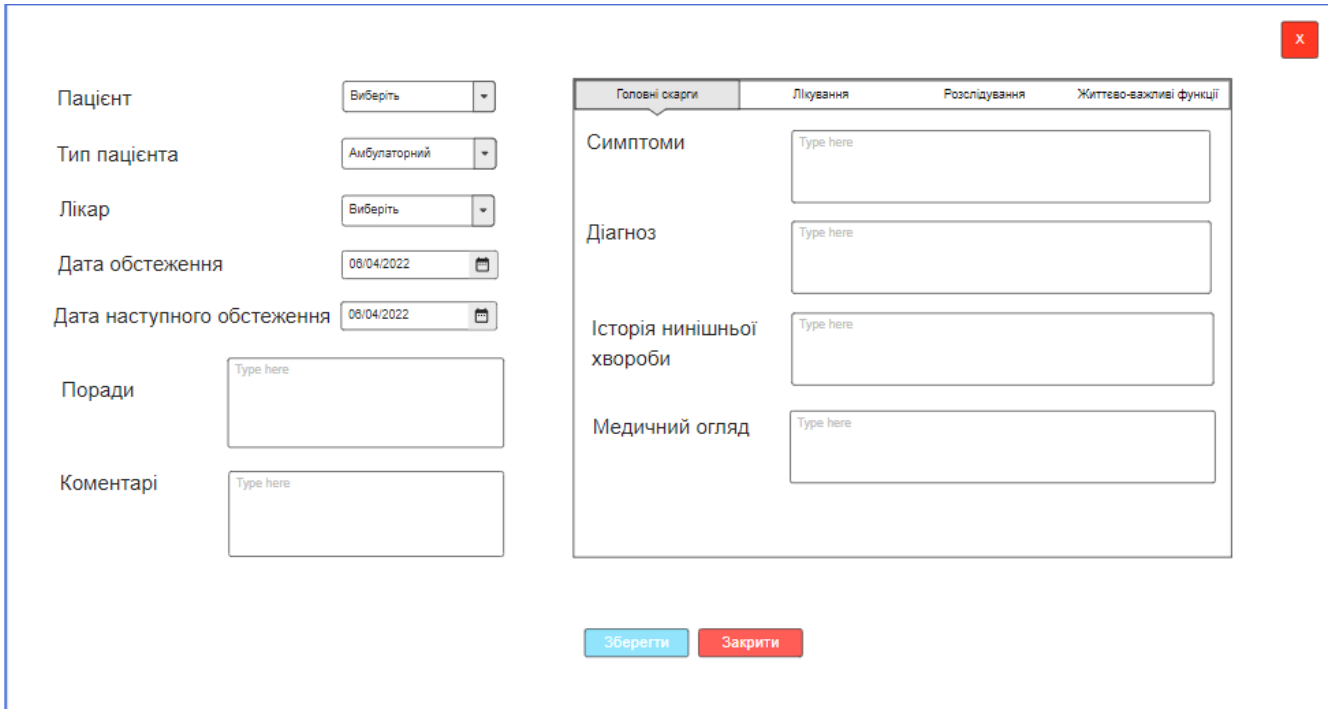

Рисунок 2.7 – Макет модального вікна для створення історії обстеження пацієнта» та вкладка «Головні скарги»

У вкладці «Лікування» на рис. 2.8 лікар вибирає ліки, які допоможуть пацієнтові одужати. Також надає інструкцію на використання ліків. Таблиця має: два комбіновані списки для вибору необхідного значення із довідників, одне поле для ведення необхідної кількості ліків, один перемикач для вибору коли вживати ліки, і одну кнопку для додання лікарського засобу з інструкцією в таблицю.

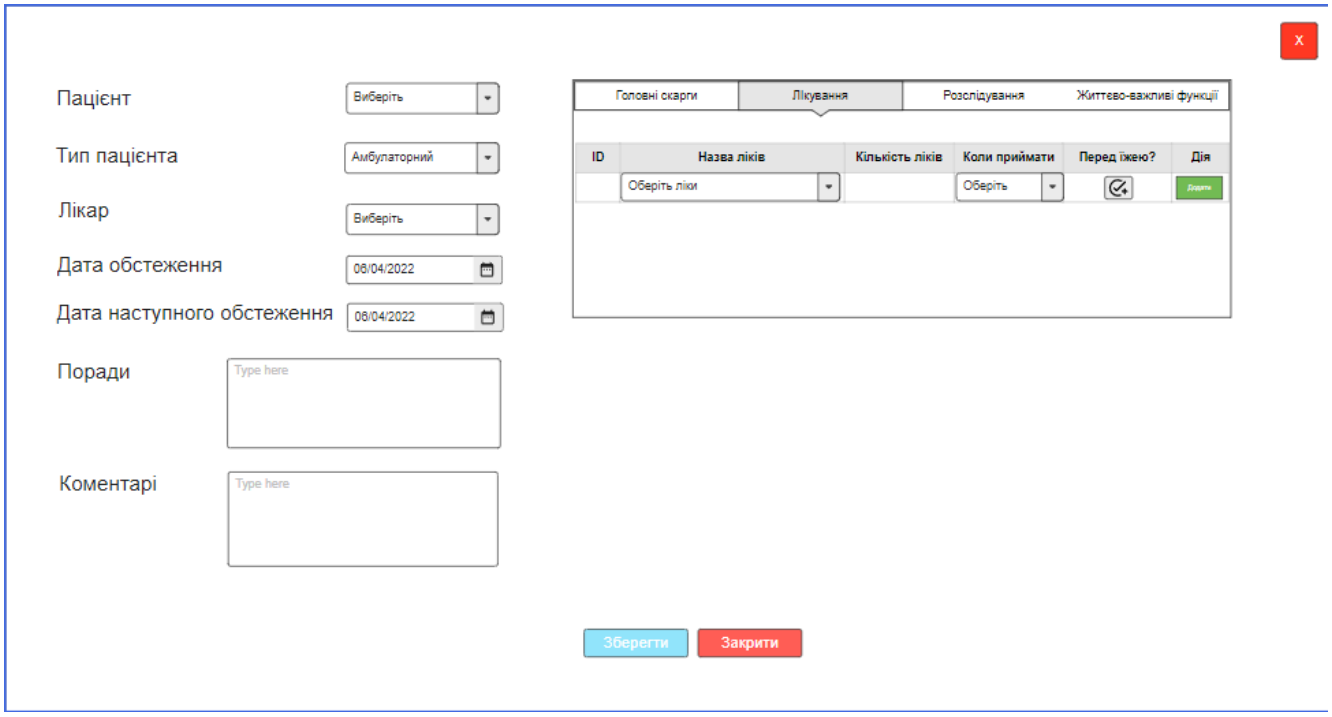

Рисунок 2.8 – Макет модального вікна для вибору ліків

У вкладці «Розслідування» на рис. 2.9 лікар дає пацієнту направлення на аналізи. Лікар обирає аналіз в комбінованому списку для вибору необхідного значення із довідника. Також автоматично надається ціна за тест, яка прикріплена к аналізу.

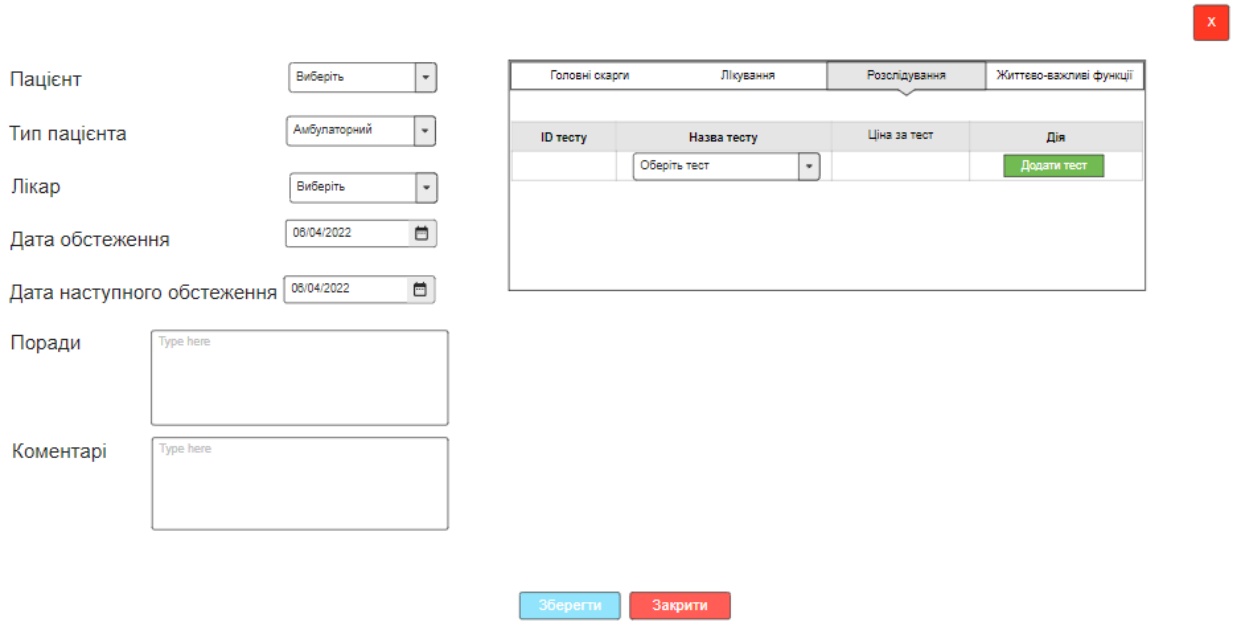

Рисунок 2.9 – Макет модального вікна для випису направлення на лабораторні дослідження

Контент посилання «Історія відгуків на аналізи»

Контент посилання «Історія відгуків на аналізи» на рис. 2.10 містить основну інформацію про аналізи пацієнтів у таблиці. У колонці з назвою «Дії» знаходиться дві кнопки: змінити та видалити.

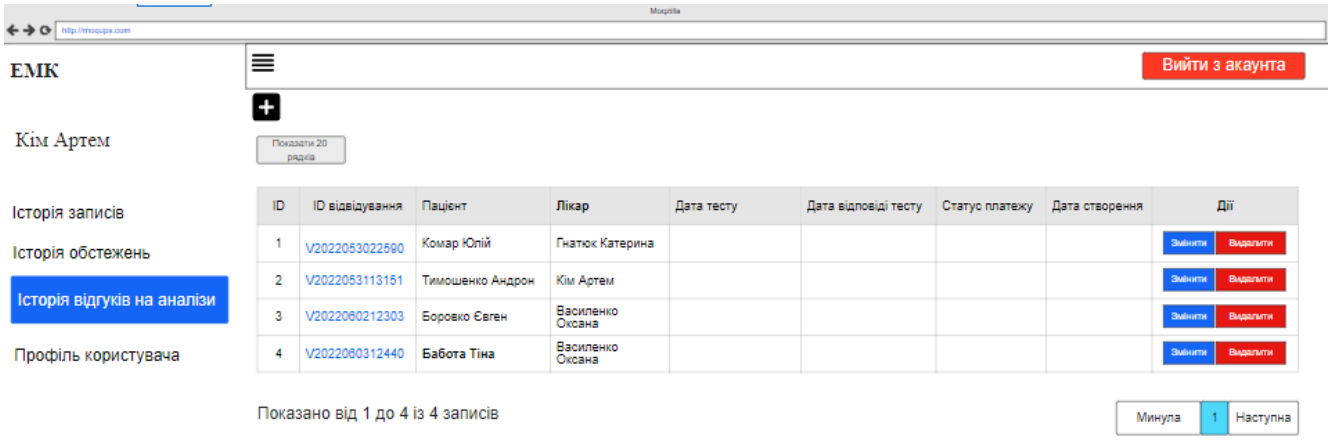

Рисунок 2.10 – Макет посилання «Історія відгуків на аналізи»

Для того, щоб лікар дав відгук на готовий лабораторний тест пацієнта, він повинен натиснути на кнопку з іконкою «Додати». Після цього лікар повинен вибрати пацієнта і статус платежу за тест. Лікар повинен вибрати достовірну дату лабораторного тесту, та вибрати дату коли відповів на тест. Знайти назву потрібного тесту. Написати відгук та зауваження до результату. Після заповнення форми, лікар повинен натиснути на кнопку «Додати тест». Після цього, тест прикріпиться до історії вибраного пацієнта.

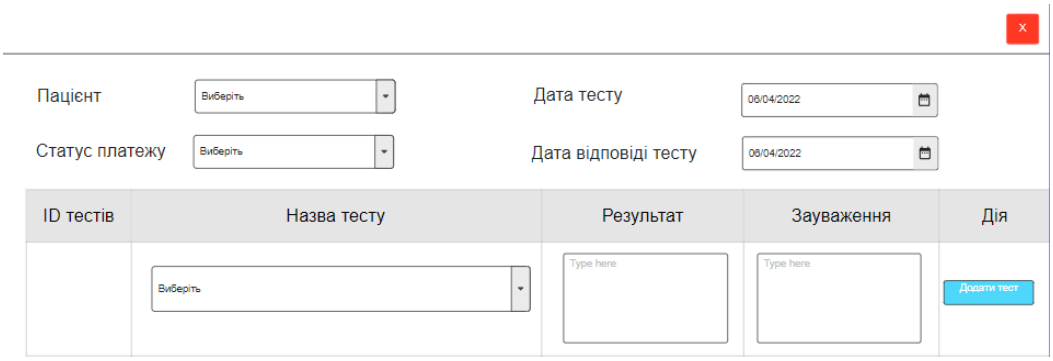

Рисунок 2.11 – Макет модального вікна для надання відгуку на готові лабораторні дослідження

2.2.3 Макети сторінок користувача з ролем «Адміністратор»

Навігаційна панель користувача з ролем «Адміністратор» містить такі посилання на контент: список лікарів, список пацієнтів, профіль користувача.

Контент посилання «Список лікарів»

Контент посилання «Список лікарів» на рис. 2.12 містить основну інформацію облікових записів лікарів. У колонці з назвою «Дії» знаходиться комбінований список дій: змінити, видалити, змінити пароль.

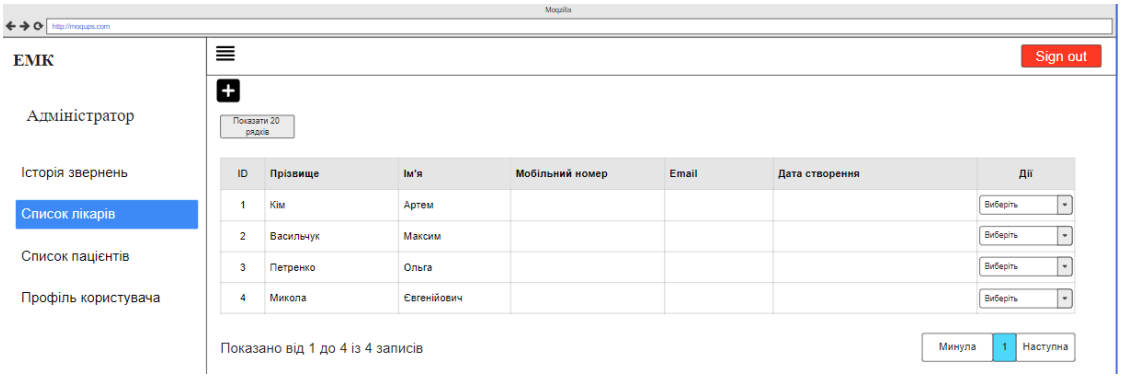

Рисунок 2.12 – Макет контенту посилання «Список лікарів»

Для того, щоб створити обліковий запис лікаря адміністратор повинен натиснути на кнопку з іконкою «Додати». Після цього адміністратор повинен заповнити форму даними про лікаря: прізвище, ім'я, мобільний номер, email, пароль, вибрати тип лікаря, адресу та країну. Після заповнення форми, адміністратор повинен натиснути на кнопку «Зберегти».

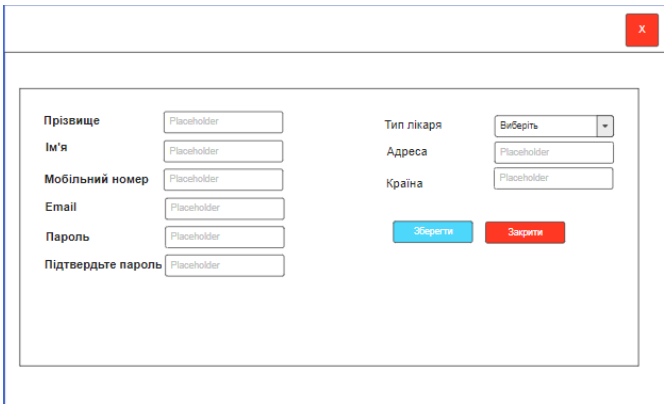

Рисунок 2.13 – Макет модального вікна для створення облікового запису лікаря

2.3 Діаграма діяльності

Діаграма діяльності використовується для опису бізнес-процесів і варіантів використання, а також документування реалізації системних процесів. Діаграма відображає бізнес-процеси, логіку процедур і потоки робіт-переходи, від однієї діяльності до іншої. Діяльність можна описати як операцію системи. Основна мета діаграм діяльності – охоплення динамічної поведінки системи.

Поведінка системи можна подати у вигляді діаграми діяльності, наведеної на рис 2.14.

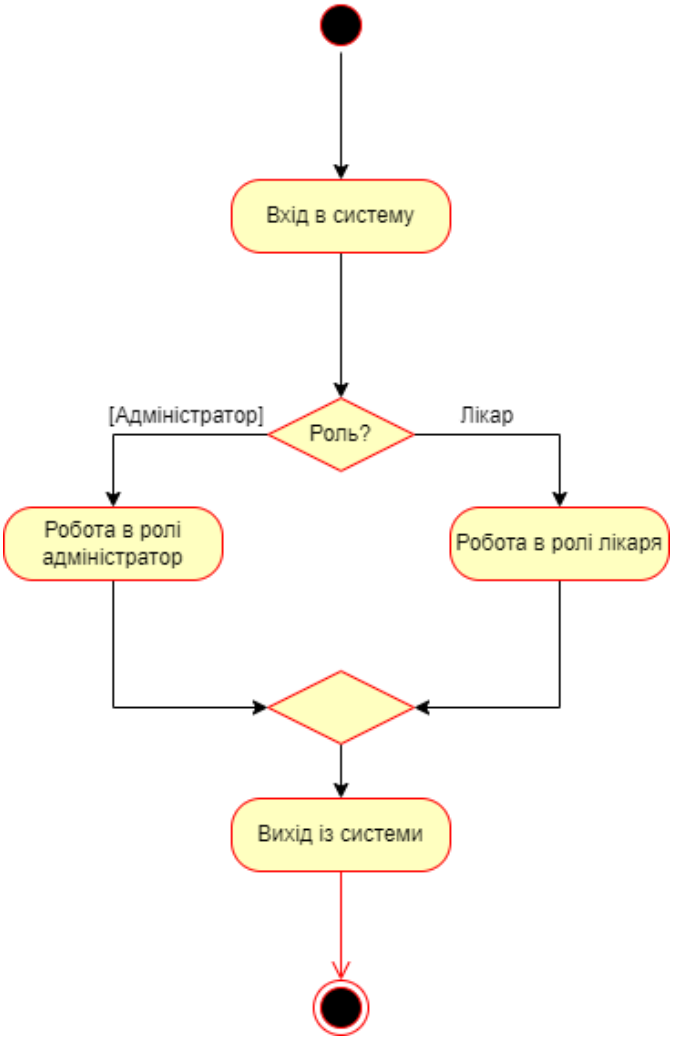

Рисунок 2.14 – Загальна діаграма діяльності

На діаграми діяльності показано розподіл авторизованих користувачів за ролями та їх діяльність у системі.

2.4 ER-діаграма та його описання

Діаграма взаємозв'язків сутностей (ERD) є невід’ємною частиною моделювання будь-якої системи. Діаграма використовуються в програмній інженерії на етапах планування програмного проекту, та допомагає пояснити логічну структуру баз даних. Діаграми ER створюються на основі трьох основних концепцій: сутності, атрибута та відносин.

У ER-моделях та моделях даних зазвичай виділяють до трьох рівнів деталізації:

– Концептуальна модель даних – схема найвищого рівня з мінімальною кількістю подробиць. Перевага цього підходу полягає у можливості відобразити загальну структуру моделі та всю архітектуру системи. Менш масштабні системи можуть обійтися без цієї моделі. І тут можна відразу переходити до логічної моделі [6].

– Логічна модель даних містить більш детальну інформацію, ніж концептуальна модель. На цьому рівні визначаються докладніші операційні та транзакційні сутності. Логічна модель не залежить від технології, в якій вона застосовуватиметься [6].

– Фізична модель даних: на основі кожної логічної моделі даних можна становити одну або дві фізичні моделі. В останніх має бути достатньо технічних подробиць для складання та впровадження самої бази даних [6].

В додатку Б спроектована логічна модель, щоб показати структуру бази даних. Модель містить 10 таблиць: User, RoleType, Doctor, InfoPatient, PatientAppoinment, PatientTest, PatientTestDetail, LabTest, LabTestCategories, CheckupSummaryPatient.

Таблиця 2.6 – Опис таблиць

Таблиця	Опис
User	Таблиця створена для створення користувача.
RoleType	Таблиця створена для надання ролі.

Кінець таблиці 2.6

Таблиця	Опис
PatientTest	Таблиця призначена для зберігання інформації про тест пацієнта.
PatientTestDetail	Таблиця призначена для зберігання інформації про результат лабораторного дослідження для пацієнта.
LabTest	Таблиця призначена для зберігання інформацію про лабораторне дослідження.
LabTestCategories	Таблиця призначена для зберігання інформації про категорію тесту.
CheckupSummaryPatient	Таблиця призначена для зберігання інформації про обстеження пацієнта
PatientAppoinment	Таблиця призначена для зберігання інформації про запис пацієнта до лікаря.
Doctor	Таблиця створена для ролі «Лікар», шо доповнює користувача.
InfoPatient	Таблиця містить всю інформацію про пацієнта в системі.

Таблиця 2.7 – Таблиця «User»

Ключ	**Поле**	**Значення**
PK	Id	Ідентифікатор користувача
	Email	Email користувача
	PasswordHash	Хеш пароля користувача
FK	RoleTypeId	Ідентифікатор типу ролі

Таблиця 2.8 – Таблиця «RoleType»

Ключ	**Поле**	**Значення**
PK	Id	Ідентифікатор типу ролі

Кінець таблиці 2.8

Ключ	**Поле**	**Значення**
	Title	Назва ролі

Таблиця 2.9 – Таблиця «Doctor»

Ключ	**Поле**	**Значення**
PK	Id	Ідентифікатор лікаря в базі даних
	DoctorCode	Код лікаря в договорі
	FirstName	Ім’я лікаря
	LastName	Прізвище лікаря
	DoctorType	Спеціальність лікаря
	Address	Адреса прописки лікаря
	Country	Країна проживання лікаря
FK	UserId	Ідентифікатор користувача

Таблиця 2.10 – Таблиця «InfoPatient»

Ключ	**Поле**	**Значення**
PK	Id	Ідентифікатор пацієнта в базі даних
	DoctorId	Ідентифікатор лікаря в базі даних
	PatientCode	Код пацієнта в договорі
	FirstName	Ім’я пацієнта
	LastName	Прізвище пацієнта
	MaritalStatus	Сімейний стан пацієнта
	Gender	Стать пацієнта
	SpouseFullName	ПІБ подружжя
	FatherFullName	ПІБ батька пацієнта
	MotherFullName	ПІБ матір пацієнта
	BloodGroup	Група крові пацієнта

Кінець таблиці 2.11

Ключ	**Поле**	**Значення**
	Agreement	Наявність договору
	Remarks	Примітки про пацієнта
	DateOfBirth	Дата народження пацієнта
	Address	Адреса прописки пацієнта
	Country	Країна проживання пацієнта

Таблиця 2.12 – Таблиця «PatientAppointment»

Ключ	**Поле**	**Значення**
PK	Id	Ідентифікатор прийому в базі даних
FK	PatientId	Ідентифікатор пацієнта в базі даних
FK	CheckupId	Ідентифікатор обстеження в базі даних
FK	DoctorId	Ідентифікатор лікаря в базі даних
	SerialNo	Серійний номер обстеження
	AppoinmentDate	Дата прийому пацієнт
	Note	Примітки обстеження

Таблиця 2.13 – Таблиця «PatientTest»

Ключ	**Поле**	**Значення**
PK	Id	Ідентифікатор дослідження тесту пацієнта в базі даних
FK	PatientId	Ідентифікатор пацієнта в базі даних
FK	CheckupId	Ідентифікатор обстеження в базі даних
FK	DoctorId	Ідентифікатор лікаря в базі даних
	TestDate	Дата створення тесту
	DeliveryDate	Дата відправки доставки
	PaymentStatus	Статус платежу за тест

Таблиця 2.14 – Таблиця «PatientTestDetail»

Ключ	**Поле**	**Значення**
PK	Id	Ідентифікатор детального описання лабораторного дослідження пацієнта в базі даних
FK	PatientTestId	Ідентифікатор лабораторного дослідження пацієнта в базі даних
FK	LabTestId	Ідентифікатор лабораторного дослідження в базі даних
	Result	Результат лабораторного дослідження
	UnitPrice	Ціна обраного лабораторного дослідження
	Remarks	Зауваження

Таблиця 2.15 – Таблиця «LabTest»

Ключ	**Поле**	**Значення**
PK	Id	Ідентифікатор лабораторного дослідження в базі даних
FK	LabTestCategoriesId	Ідентифікатор категорії лабораторного дослідження в базі даних

Таблиця 2.16 – Таблиця «LabTest»

Ключ	**Поле**	**Значення**
	LabTestName	Назва лабораторного дослідження
	Status	Статус лабораторного дослідження

Таблиця 2.17 – Таблиця «LabTestCategories»

Ключ	**Поле**	**Значення**
PK	Id	Ідентифікатор категорії лабораторного дослідження в базі даних
	Name	Назва категорії тесту
	Description	Опис тесту

Таблиця 2.18 – Таблиця «CheckupSummaryPatient»

Ключ	**Поле**	**Значення**
PK	Id	Ідентифікатор огляду пацієнта
	PatientType	Тип пацієнта. Амбулаторний чи Стаціонарний

Таблиця 2.19 – Таблиця «CheckupSummaryPatient»

Ключ	**Поле**	**Значення**
	Symptoms	Симтоми пацієнта
	Diagnosis	Діагностика захворювання пацієнта
	PhysicalExamination	Висновок медогляду
	CheckupDate	Дата звернення
	NextCheckupDate	Дата наступного звернення
	Advice	Поради до лікування хвороби
	BPSystolic	Записується систолічний кров’яний тиск пацієнта
	BPDiastolic	Записується артеріальний тиск пацієнта
	RespirationRate	Записується частота дихання
	Notes	Примітки лікаря
FK	PatientId	Ідентифікатор пацієнта в базі даних
FK	DoctorId	Ідентифікатор лікаря в базі даних

Основна мета розробки моделі даних - переконатися, що об'єкти даних, які пропонуються функціональною групою, представлені точно.

2.5 Діаграма компонентів

На рис. 2.15 наведено діаграму компонентів клієнтської частини проекту, де продемонстровано безпосереднє взаємодію браузера з проектом.

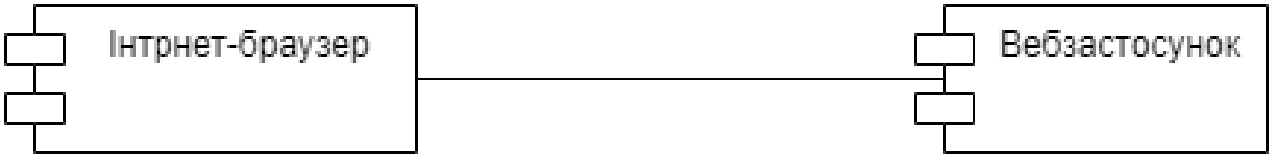

Рисунок 2.15 – Діаграма компонентів клієнтської частини

Діаграма компонентів серверної частини проекту наведена на рис. 2.16, де видно взаємодію проекту, що розробляється, з сервером і з СУБД, з який і на основі яких працює проект.

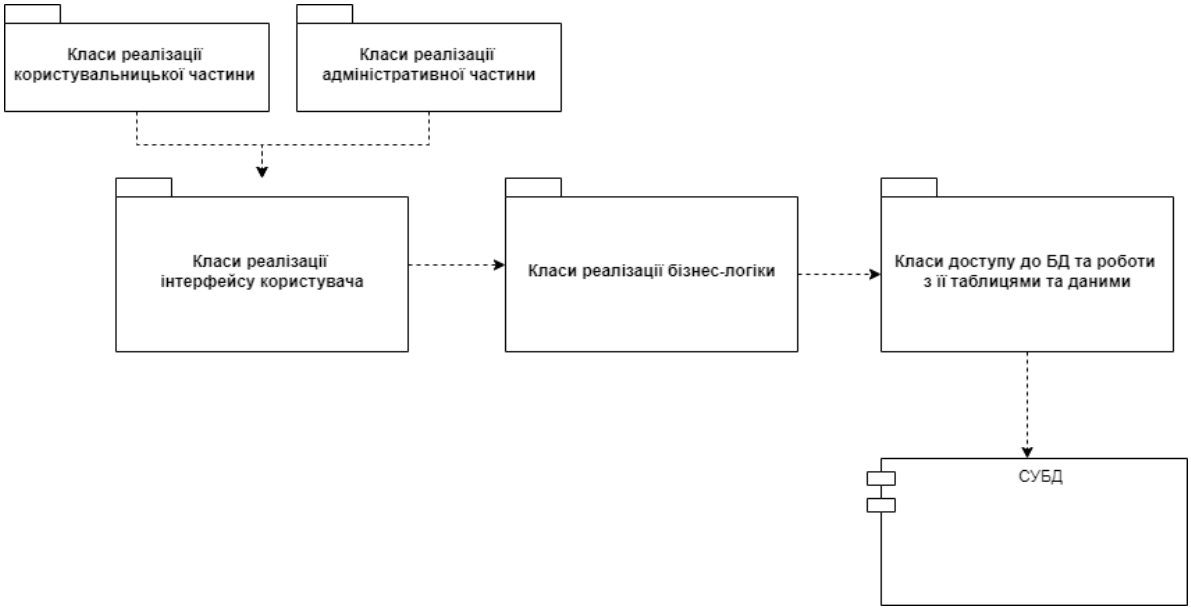

Рисунок 2.16 – Діаграма компонентів серверної частини

В об’єктно-орієнтованій парадигмі діаграма компонентів використовується для групування класів на основі спільної мети, щоб дати проекту розробки програмного забезпечення візуалізації на високому рівні абстракції. Він моделює фізичний вигляд системи, такої як виконувані файли, файли, бібліотеки тощо, яка знаходиться всередині вузла.

2.6 Діаграма розгортання проєкту

Для візуалізації елементів і компонентів системи, наявних лише на етапі її виконання, на рис 2.17 наводиться діаграма розгортання з елементами зображення, з яких повинна складатися система. На діаграмі зображені тільки компоненти-екземпляри програми, що є виконаними файлами або динамічними бібліотеками.

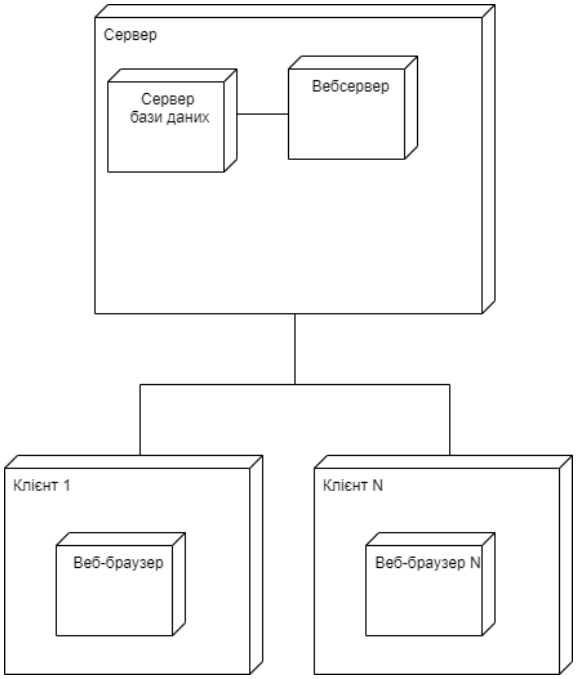

Рисунок 2.17 – Діаграма розгортання проекту

В UML-діаграмі розгортання моделюють фізичну архітектуру системи. Діаграми розгортання показують взаємозв’язки між програмними та апаратними компонентами в системі та фізичний розподіл обробки.

Діаграми розгортання, які розроблюються на етапі розробки впровадження, показують фізичне розташування вузлів у розподіленій системі, артефакти, які зберігаються на кожному вузлі, а також компоненти й інші елементи, які реалізують артефакти. Вузли представляють апаратні пристрої, такі як комп’ютери, датчики та принтери, а також інші пристрої, які підтримують середовище виконання системи. Шляхи зв'язку та відношення розгортання моделюють з’єднання в системі.

2.7 Підбір технологій для ефективної розробки вебзастосунку

Щоб розробити вебзастосунок Обліку медичних карток, треба обрати мову програмування, сукупні технології до вибраної мови, та середовища для розробки. Основними критеріями за яким було обрана мова програмування, це сфера розробки і величина ком’юніті.

2.7.1 Опис обраних середовищ для розробки вебзастосунку

Visual Studio – це інтегроване середовище розробки, створено Microsoft і використовуване для розробки різних типів програмного забезпечення, таких як комп'ютерні програми, вебсайти, вебзастосунки, вебслужби та мобільні програми. Він містить інструменти для компіляторів, компілятори та інші функції, що полегшують процес розробки програмного забезпечення [23].

SQL Server Management Studio (SSMS) – це інтегроване середовище для управління будь-якою інфраструктурою SQL, від SQL Server до бази даних SQL Azure. SSMS надає інструменти для налаштування, моніторингу та адміністрування екземплярів SQL Server та баз даних. Використовується SSMS для розгортання, моніторингу та оновлення компонентів рівня даних, які використовуються програмами, а також для створення запитів та сценаріїв [15].

2.7.2 Технології для розробки серверної частини вебзастосунку

Серверна частина відноситься до розробки на стороні сервера. Основна увага приділяється базам даних, сценаріям, архітектурі вебзастосунку. Він містить приховані дії, які відбуваються під час виконання будь-яких дій у вебзастосунку. Код, написаний внутрішніми розробниками, допомагає браузерам взаємодіяти з інформацією бази даних.

Мова програмування C Sharp(C#)

Для створення вебзастосунку буде використана мова програмування C Sharp (C#). Мова програмування C# базується на практиках об'єктно-орієнтованого програмування (ООП). Ця концепція кодування передбачає, що ви можете визначити тип і структуру даних, щоб застосувати до них набір стандартних функцій. ООП збирає дані в об'єкти, що полегшує розбивку програми на менші частини, які швидше створювати, керувати та об'єднувати. Мови ООП створюють вебзастосунки, які легше тестувати та читати, дають змогу реагувати на будь-які проблеми, що виникають, і загалом означають більш економний підхід до написання коду.

Вебфреймворк ASP.NET Core

Після вибору мови програмування, йде обов’язковим до вибору програмного каркасу, який спрощує розробку, частково за рахунок автоматизації, і позбавляє від необхідності написання рутинного коду.

Найкращім вибором для мови програмування C# став вебфреймворк ASP.NET Core на платформі .NET Core і ось чому:

– висока продуктивність;
– підтримка кросплатформності;
– підтримує паралельне керування версіями.

ASP.NET Core MVC — це багатий фреймворк для створення веб-програм і API за допомогою шаблону проектування Model-View-Controller [8]. Архітектурний шаблон Model-View-Controller (MVC) розділяє програму на три основні групи компонентів: моделі, подання та контролери. Цей шаблон допомагає досягти розділення проблем. Використовуючи цей шаблон, запити користувачів направляються до контролера, який відповідає за роботу з моделлю для виконання дій користувача та/або отримання результатів запитів. Контролер вибирає подання для відображення користувачеві та надає йому будь-які дані моделі, які йому потрібні.

Entity Framework Core

Entity Framework являє собою ORM-технологію (об'єктно-реляційне відображення - зображення даних на реальних об’єктах) від компанії Microsoft для доступу до даних. Entity Framework Core дозволяє абстрагуватися від самої бази даних і її таблиць і працювати з даними, як з об'єктами класу незалежно від типу зберігання [18].

T-SQL

Transact-SQL (так він називається T-SQL) є базою даних (база даних) процедурною мовою програмування, яка відповідає монопольному Microsoft і використовується в SQL Server.

Процедурна мова була створена для розширення можливостей SQL з можливістю добре інтегруватися з SQL. Добавлені деякі функції, як локальні змінні та обробка даних.

Вони так же використовуються для записів у процедурі зберігання: фрагмент коду, що знаходиться на сервері, керує складними бізнес-правилами, які складно або неможливо керувати операціями на основі набору (операції на основі чистих наборів).

2.7.3 Технології для розробки для клієнтської частини вебзастосунку

Клієнтська частина виконується на стороні браузера, з яким користувачі можуть взаємодіяти та контактувати безпосередньо.

JavaScript

JavaScript — це мова сценаріїв, що використовується для створення динамічного та інтерактивного вебконтенту [20]. Це означає, що це мова комп’ютерного програмування, яка працює в Інтернет-браузері (браузер також відомий як веб-клієнт, оскільки він підключається до веб-сервера для завантаження сторінок). На стороні клієнта JavaScript запускається двигуном v8 (Google Chrome).

Bootstrap та AdminLTE

Bootstrap — це безкоштовний інтерфейсний фреймворк для швидшої та простішої веброзробки [9]. Bootstrap включає в себе шаблони дизайну на основі HTML і CSS для форм, кнопок, таблиць, навігації, каруселей зображень та багатьох інших, а також додаткові плагіни JavaScript.

AdminLTE – популярний шаблон WebApp з відкритим кодом для інформаційних панелей та панелей керування адміністратора. Це адаптивний HTML-шаблон, який базується на фреймворку CSS Bootstrap.

jQuery data table

jQuery DataTable - це потужний та інтелектуальний плагін для покращення HTML-таблиць, що надається бібліотекою jQuery [14]. Це дуже гнучкий інструмент, який в основному створений для відображення інформації в таблицях, а також додавання до них взаємодій, отже, для підвищення доступності даних у таблицях HTML. Таблиці даних jQuery дозволяють створювати динамічні таблиці даних, додаючи до таблиць HTML деякі розширені функції, такі як розбивка на сторінки, сортування, впорядкування, пошук та багато інших з мінімальними зусиллями. Це забезпечує найкраще управління, коли кількість даних величезна. Він підтримує як обробку на стороні клієнта, і обробку на стороні сервера.

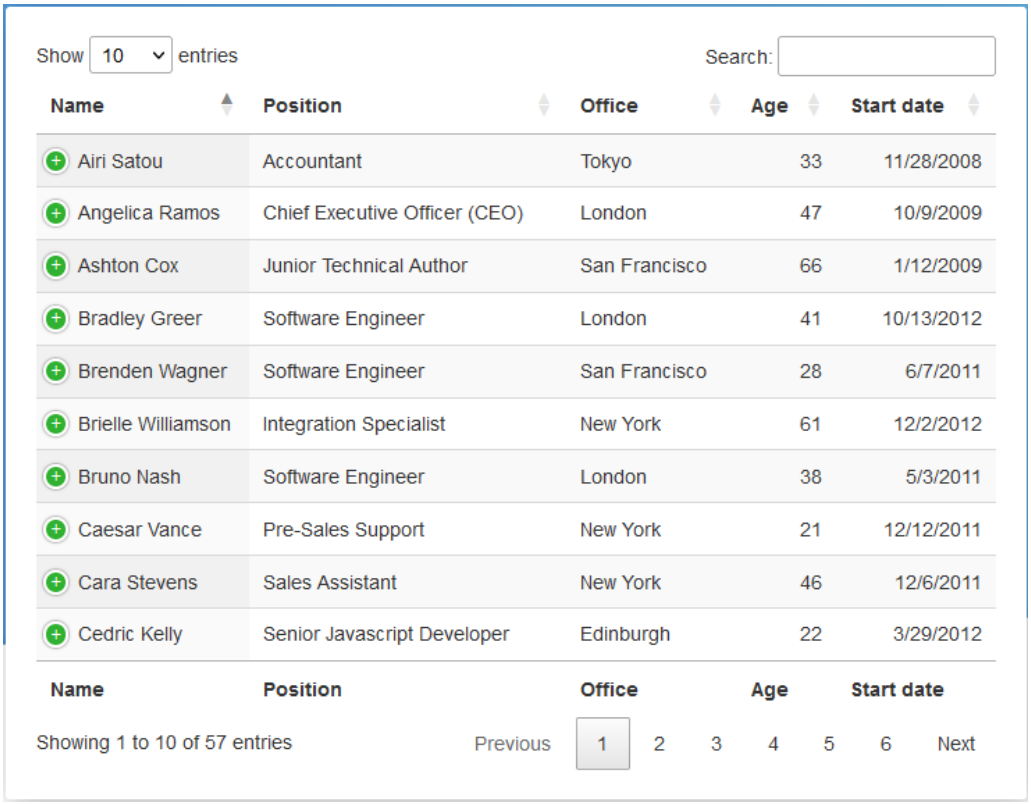

Рисунок 2.18 – Таблиця створена за допомогою плагіна jQuery DataTable

AJAX

Основна робота Ajax полягає в асинхронному оновленні контенту, тобто на веб-сторінці користувача немає необхідності перезавантажувати весь контент, а перезавантажується лише поле. XML — це мова розмітки, що означає, що це

закодовані мови для анотування частин веб-документа, які дають веб-браузерам інструкції з розуміння та відображення контенту користувача.

AJAX зв'язується з сервером за допомогою об'єкта XMLHttpRequest. На рис. 2.19 показано, як взаємодіє AJAX між браузером та вебсервером.

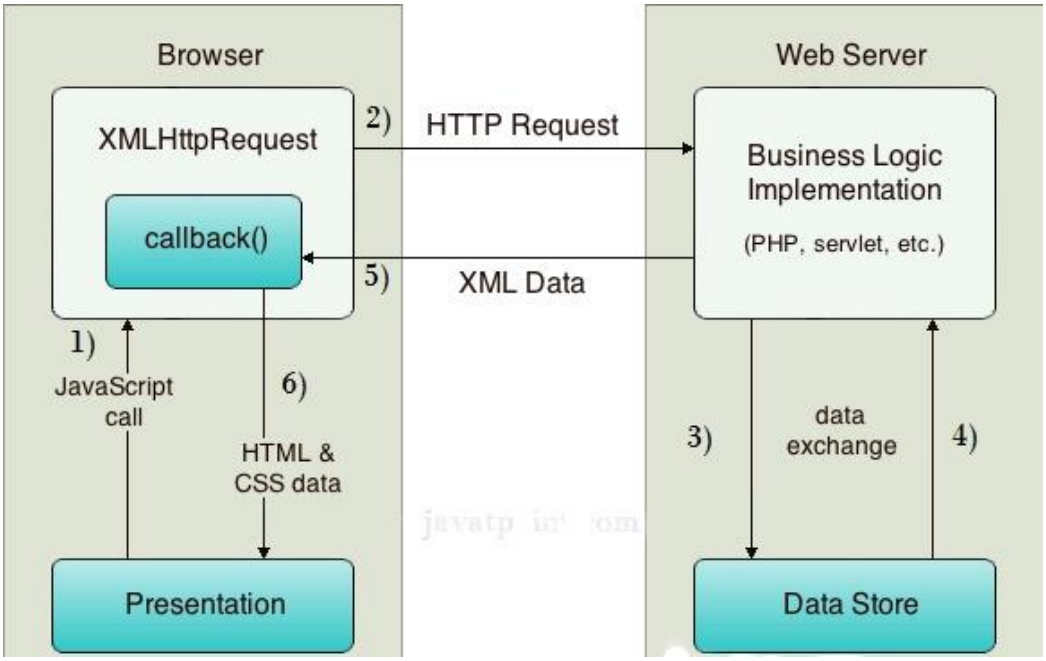

Рисунок 2.19 – Робота AJAX[]

1) користувач відправляє запит з інтерфейсу користувача, і викликає JavaScript;

2) переходить до об’єкта XMLHttpRequest;

3) HTTP-запит надсилається на сервер об'єктом XMLHttpRequest;

4) сервер взаємодіє з базою даних, використовуючи JSP, PHP, Servlet, ASP.net тощо;

5) дані отримані;

6) сервер надсилає дані XML або JSON у функцію зворотного виклику XMLHttpRequest;

7) дані HTML та CSS відображаються у браузері.

Висновки до розділу 2

В другому розділі кваліфікаційної роботи змодельовано діаграму використання, для того щоб показати можливості кожного актора (гість, лікар, адміністратор) в системі. Також створено найважливіші сценарії використання такі як, створення запису до лікаря, створення обстеження, роздрукування історії пацієнта, створення відгука до лабораторного дослідження пацієнта, авторизація користувача.

У другому пункті розроблено макети основних сторінок, для полегшити створення користувацького інтерфейсу. Також визначити візуальні недоліки на перших стадіях роботи над проєктом.

В наступних пунктах відображено:

1) поведінку системи, для якої створено діаграму діяльності, в якій показано розподіл авторизованих користувачів за ролями та їх діяльність у системі;

2) базу даних, для якої спроектовано фізичну модель бази даних, з описом кожної сутності та її атрибутів;

3) компоненти системи, для якої створено діаграми компонентів для клієнтської та серверної частини проєкту;

4) взаємозв’язки між програмними та апаратними компонентами в системі та фізичний розподіл обробки, для якої було створено діаграму розгортання проєкту.

В кінці другого розділу описано технології, які будуть використані для створення вебзастосунку.

На основі другого розділу можна переходити до останньої частини кваліфікаційної роботи, в якій описується розробка та тестування вебзастосунку.

3 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

3.1 Архітектура вебзастосунку та його частини

Для створення вебзастосунку було використано архітектурний шаблон MVC. Архітектура MVC дозволяє розділити код програми на 3 частини: Модель (Model), Вид або Представлення (View) та Контролер (Controller) [22]. Архітектурний шаблон MVC в системі має пасивну модель, це коли модель не має жодних способів впливати на представлення або контролер і користується ними як джерело даних для відображення. Всі зміни моделі відстежуються контролером і він відповідає за копіювання уявлення, якщо це необхідно.

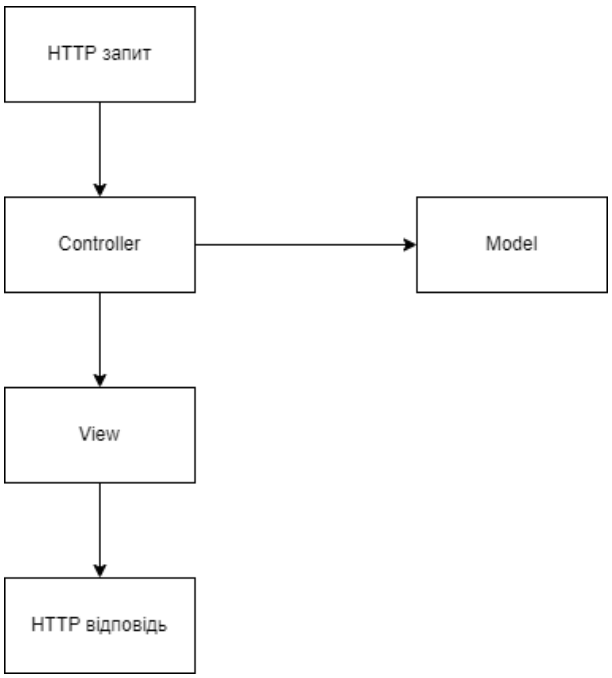

Рисунок 3.1 – Пасивна модель MVC

Models

Модель отримує дані від контролера, які запросив користувач, а також визначає структуру вебзастосунку. Проєкт містить такі моделі, як: CheckupSummaryPatient.cs, CheckupMedicineDetails.cs, PatientAppointment.cs, PatientInfo.cs, PatientTest.cs, PatientTestDetail.cs, Doctor.cs, DefaultIdentityOptions.cs, UserProfile.cs.

Таблиця 3.1 – Опис моделей

Назва моделі	**Опис моделі**
CheckupSummaryPatient.cs	Містить дані про обстеження пацієнта для
CheckupMedicineDetails.cs	Містить дані про виписані ліки для пацієнта
PatientAppointment.cs	Містить дані про запис на прийом до лікаря
PatientInfo.cs	Містить дані про пацієнта
PatientTest.cs	Містить дані про лабораторне дослідження пацієнта
PatientTestDetail.cs	Містить дані про результат лабораторних досліджень
Doctor.cs	Містить дані про лікаря
DefaultIdentityOptions.cs	Містить дані про параметри ідентифікації за замовчуванням
UserProfile.cs	Містить дані про обліковий запис користувача

Вихідний код головних моделей знаходиться в додатку В.

Views

Компонент View використовується для всієї логіки інтерфейсу користувача вебзастосунку. Проєкт містить папку «Views», в якій знаходяться перегляди для інтерфейсу користувача.

Таблиця 3.2 – Опис переглядів

Папка	Вид	Опис
Account	Login	Форми авторизації
CheckupSummary	_AddEdit	Форма для створення та змінення обстеження
	_AddLabTests	Форма з таблицею для створення направлення на лабораторні дослідження
	_AddMedicine	Форми з таблицею для випису ліків пацієнту
	_CheckupHeader	Шапки історії обстеження пацієнта

Продовження таблиці 3.2

Папка	Вид	Опис
CheckupSummary	_CheckupDeteils	Основна інформація історії обстеження
	_PatientHistory	Дві кнопки для завантаження та друку історії обстеження
	_PatientTestDetail	Висновки на лабораторні дослідження
	_VitalSigns	Створення форми для життєво-важливих функцій
	Index	Сторінка «Історія обстеження», в якій знаходиться історії обстежень
DoctorManagment	_AddDoctorAccount	Форма для додання до системи облікового запису лікаря
	Index	Сторінка «Список лікарів» з таблицею
	_ResetPassword	Форма для скидання пароля лікаря
UserProfile	_EditUserProfile	Форма для редагування профілю користувача
	_ResetPassword	Форма для скидання пароля користувача
	Index	Сторінка для профіля користувача
DownloadMedCardPatient	_MedCardPrintableArea	Сторінка з інформацією про медичну картки пацієнта

Кінець таблиці 3.2

Папка	Вид	Опис
Shared	_Adminlte	Панель керування.
	_AdminlteBlank	
	_AdminlteFooter	
	_AdminlteMainHeader	
	_AdminlteMainSidebar	
PatientAppoinment	_AddEdit	Форма для створення та змінення запису до лікаря
	_Deteils	Вікно з інформацією про запис до лікаря
	Index	Сторінка «Історія записів» з записами пацієнтів до лікаря
PatientTest	_AddEdit	Форма для створення та змінення пакету аналізів
	_AddLabTests	Форма з таблицею для додання відгуку на аналізи пацієнта
	_Details	Вікно з інформацією про пакет лабораторних досліджень
	Index	Сторінка «Історія відгуків на аналізи», в якій знаходиться пакети лабораторних досліджень
PatientInfo	_AddEdit	Форма для додання пацієнта до бази даних пацієнтів
	_Deteils	Вікно з інформацією про пацієнта
	Index	Сторінка «Список пацієнтів», в якій находиться списком пацієнтів

Controllers

Контролери діють як інтерфейс між компонентами Model і View для обробки всієї бізнес-логіки та вхідних запитів, маніпулювання даними за допомогою компонента Model і взаємодії з Views для відтворення кінцевого результату. Проєкт містить такі контролери, як: AccountController., CheckupSummaryController, DownloadReportController, PatientAppoinmentController, PatientInfoController, PatientTestController, DoctorMenegrmentController, UserProfileController, UserRoleControler.

Таблиця 3.3 – Опис контролерів

Контролер	Функція	Опис
Account	Login	Для авторизації користувача
	Logout	Для виходу з облікового запису
CheckupSummary	GetDataTableData	Для роботи з таблицею, яка містить список обстежень пацієнтів
	Details	Для отримання інформації про обстеження
	PatientHistory	Для роботи з історією обстеження
	PrintCheckup	Для друку історії обстеження
	AddFromPatientInfo	Для додання обстеження пацієнта до інформації про пацієнта
	AddEdit	Для створення або змінення обстеження пацієнта
	UpdateCheckupSummary	Для обновлення обстеження пацієнта
	Delete	Для видалення обстеження пацієнта

Продовження таблиці 3.2

Контролер	Функція	Опис
DownloadReport	DownloadCheckupReport	Для завантаження історії обстеження
PatientAppoinment	GetDataTableData	Для роботи з таблицею записів до лікаря
	Details	Для отримання інформації про запис до лікаря
	AddEdit	Для створення або змінення запису до лікаря
	UpdateCheckupSummary	Для обновлення обстеження пацієнта даними про запис до лікаря
	Delete	Для видалення запис до лікаря
PatientInfo	GetDataTableData	Для роботи з таблицею, яка містить список пацієнтів
	Details	Для отримання інформації про пацієнта
	AddEdit	Для створення або змінення інформації про пацієнта
	Delete	Для видалення пацієнта з системи
PatientTest	AddEdit	Для створення або змінення інформації про лабораторне дослідження
	SavePatientTestDetail	Для зберігання інформації про тест пацієнта
	UpdatePatientTestDetail	Для оновлення інформації про тест пацієнта
	Delete	Для видалення тесту пацієнта

Кінець таблиці 3.2

Контролер	Функція	Опис
DoctorMenegrment	GetDataTableData	Для роботи з таблицею, яка містить список облікових записів про лікаря
	AddDoctorAccount	Для додання лікаря до системи
	EditDoctorProfile	Для змінення облікового запису лікаря
	ResetPassword	Для зміни пароля лікаря
	DeleteDoctorAccount	Для видалення облікового запису лікаря
UserRole	Index	Для перевірки ролі користувачу
	ManageRole	Для надання ролі користувачу

Вихідний код головних контролерів знаходиться в додатку Г.

3.2 Огляд вебзастосунку

Кожен користувач повинен авторизуватися до системи. На рис. 3.2 представлено сторінку для авторизації користувача. Сторінка містить форму, в якій користувач вводить свої облікові дані. Якщо користувачу потрібно щоб система запам’ятала його сеанс, він може натиснути на прапорець «Запам'ятати мене», після цього система автоматично авторизує користувача.

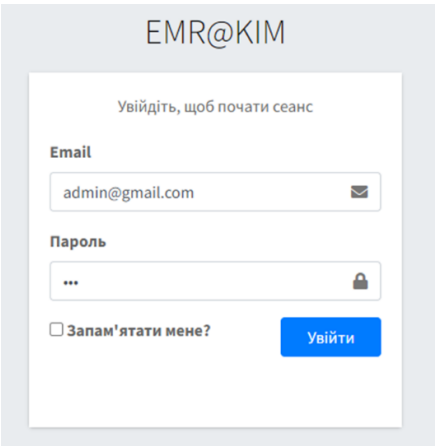

Рисунок 3.2 – Сторінка для авторизації користувача

Після аутентифікації користувача, система визначить роль користувача, та надасть права доступу до ресурсів. Користувачі мають спільні сторінки, такі як, профіль користувача і список записів.

3.2.1 Сторінки адміністратора

Користувач з ролем «Адміністратор» містить так сторінки, як: історія записів, список пацієнтів, список лікарів та профіль користувача.

Профіль користувача

Після авторизації користувача до системи, лікар входить на головну сторінку, в якій знаходяться його персональні дані. На рис.3.3 містить головну інформацію про користувача Також сторінка має дві кнопки «Редагувати профіль користувача» і «Скинути пароль».

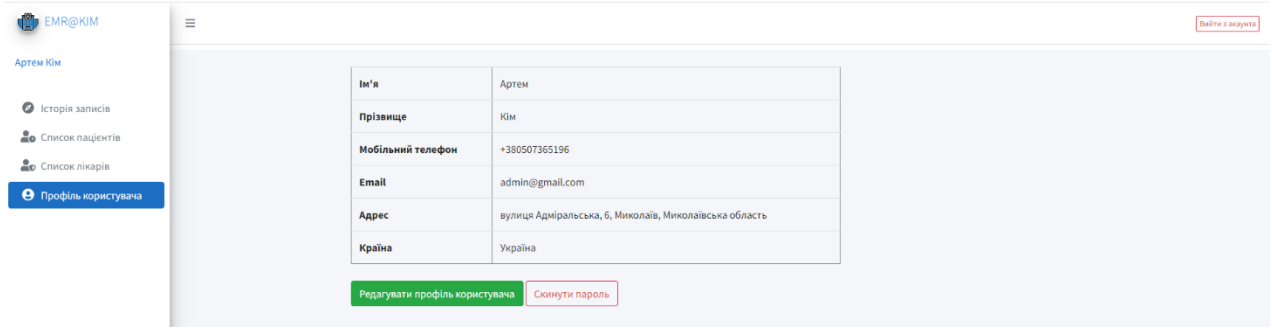

Рисунок 3.3 – Профіль користувача з ролем «Адміністратор»

Історія записів

Ця сторінка створена, щоб адміністратор записував пацієнтів на прийом до лікаря, а лікар дізнавався про час, дату прийому пацієнта і скільки записалося до нього на прийом.

Сторінка на рис 3.4 має такі дії з записом на прийом, як: створити прийом, переглянути прийом, змінити прийом, видалити прийом та знайти запис на прийом за потрібним значенням.

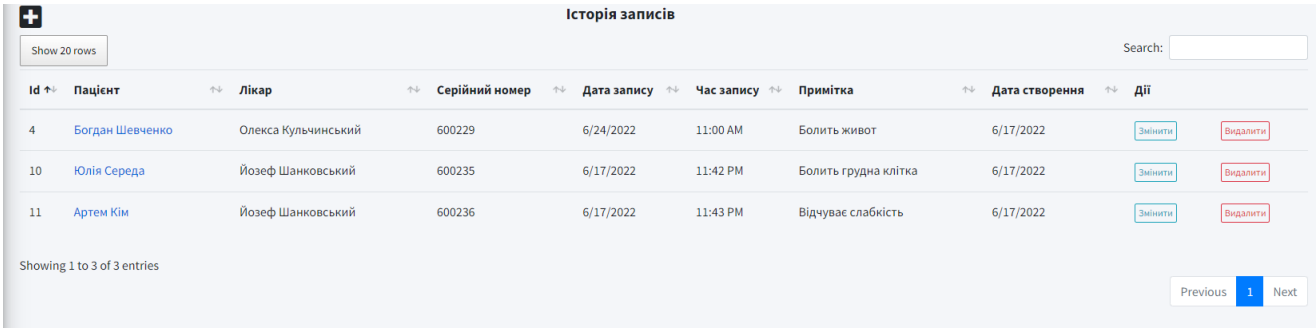

Рисунок 3.4 – Історія записів для адміністратора

На рис. 3.5 містить інформацію про пацієнта, якому лікарю записався пацієнт, серійний номер запису, дату та час обстеження, та примітка про симптоми пацієнта.

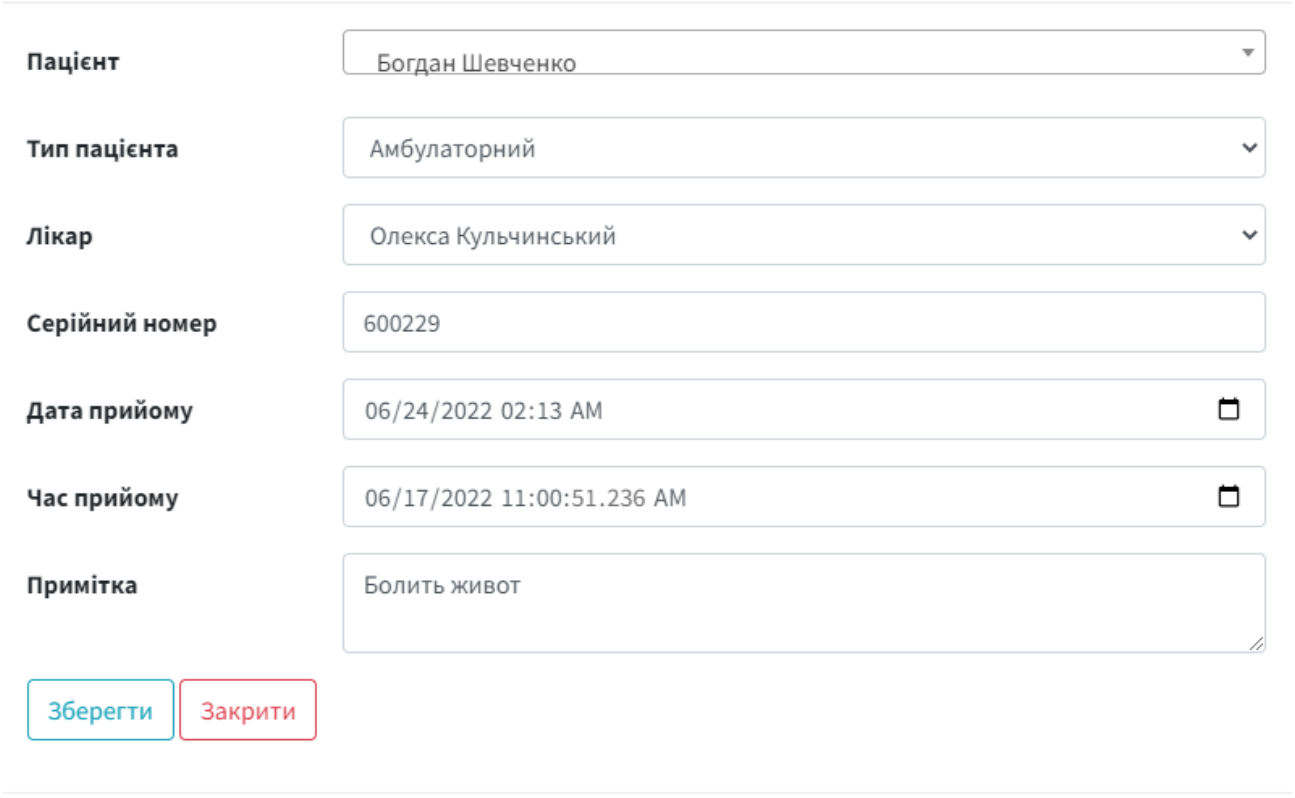

Рисунок 3.5 – Форма для створення запису до лікаря

Щоб додати запис до лікаря, потрібно натиснути кнопку з іконкою «Додати». Після цього система додасть запис до історії записів.

3.2.2 Сторінки лікаря

Історія обстежень

Після запису пацієнта до лікаря, то лікарю потрібно буде вводити інформацію в систему про обстеження. В таблиці на рис. 3.6 знаходиться список обстежень, які провели лікарі над пацієнтами. Лікар може виконувати такі дії з обстеженнями, як: створення обстеження, змінити обстеження, видалити обстеження, роздрукувати обстеження, переглянути обстеження, завантажити обстеження та знайти обстеження за потрібним значенням. Результат завантаження та друку знаходиться в Додатку А. До кожного обстеження генерується ідентифікатор обстеження.

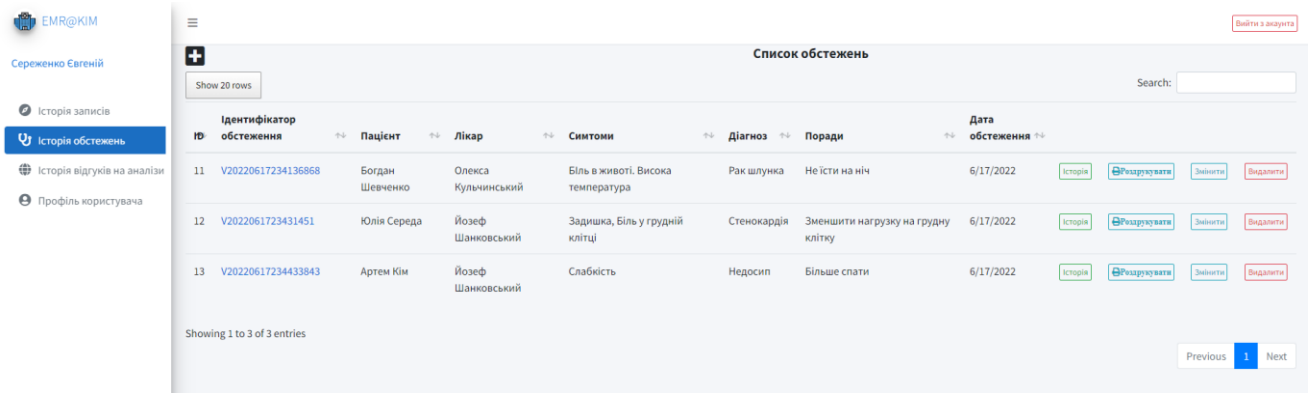

Рисунок 3.6 – Список обстежень пацієнтів

Після того, як адміністратор додасть запис до лікаря, форма на обстеження автоматично створиться у списку обстежень. Перша частина форми складається з загальної частини. Друга частина форми складається з вкладок, яка містить інформацію про обстеження.

Вкладка «Головні скарги» на рис 3.7 створена, щоб лікар документував хворобу пацієнта.

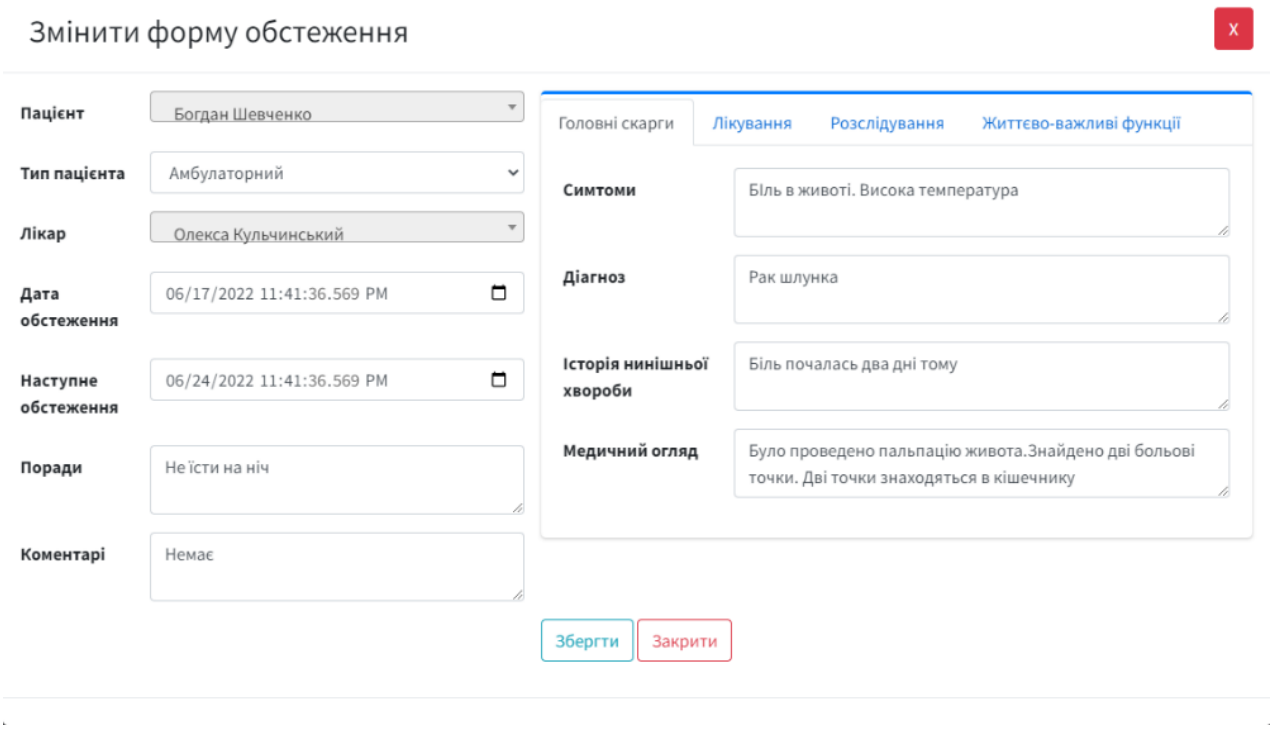

Рисунок 3.7 – Форма обстеження загальної частини та з вкладкою «Головні скарги»

На рис. 3.7 вкладка «Лікування». Ця вкладка створена, щоб лікар міг виписати і додати ліки у медичну картку, та надати необхідну інструкцію, яка допоможе пацієнту вилікуватись.

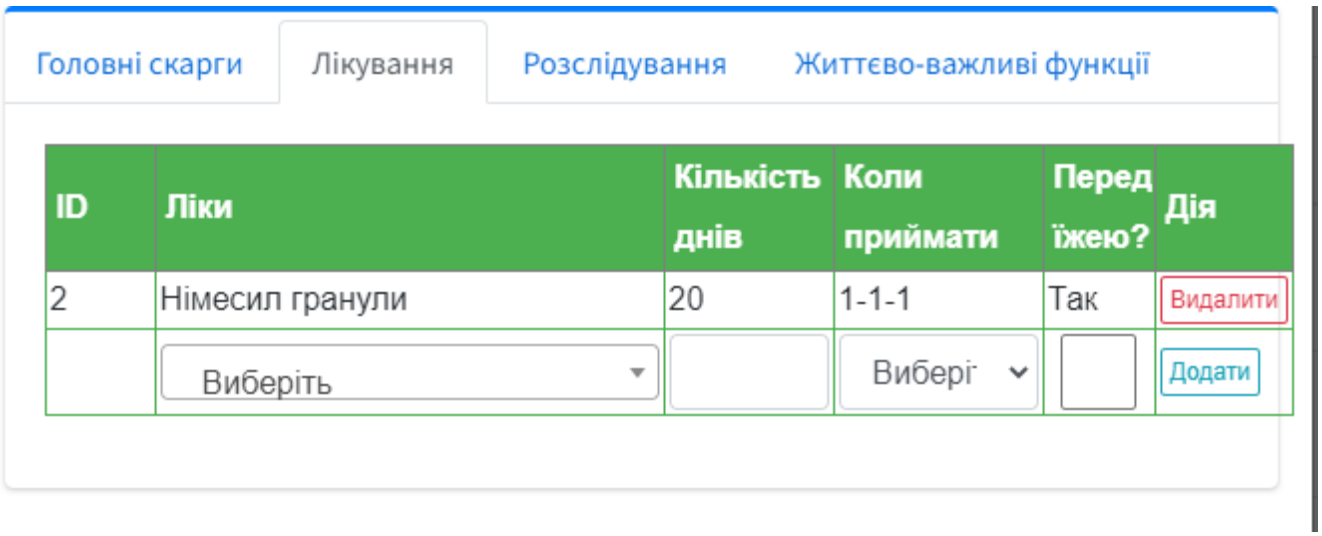

Рисунок 3.7 – Форма обстеження з вкладкою «Лікування»

На рис. 3.8 вкладка «Розслідування». Ця вкладка створена, щоб лікар міг задокументувати направлення на лабораторні дослідження.

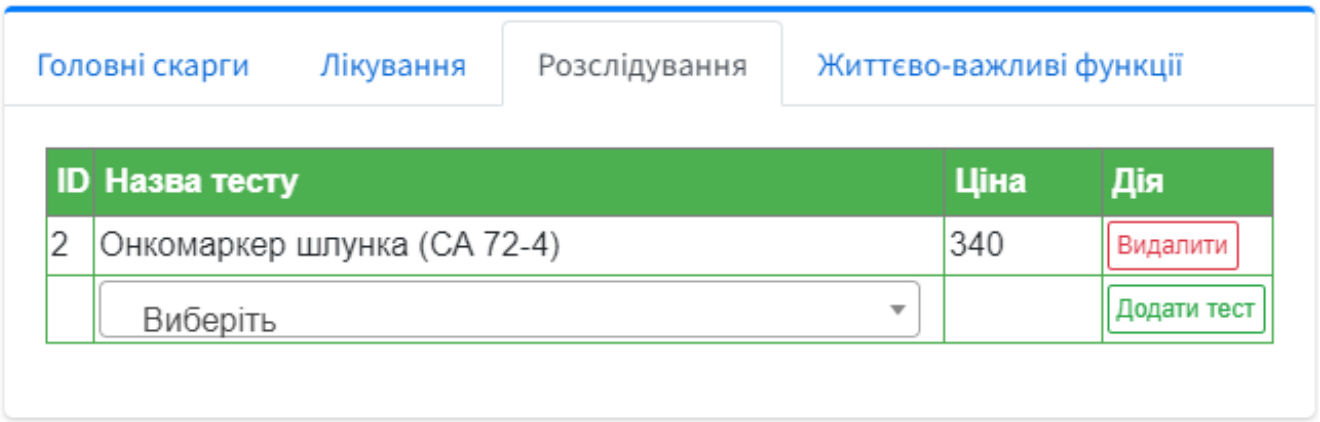

Рисунок 3.8 – Форма обстеження з вкладкою «Лікування»

На рис. 3.9 вкладка «Життєво-важливі функції». Ця вкладка створена, щоб лікар міг додати до медичної картки про стан життєво важливих систем та (або) функцій організму пацієнта.

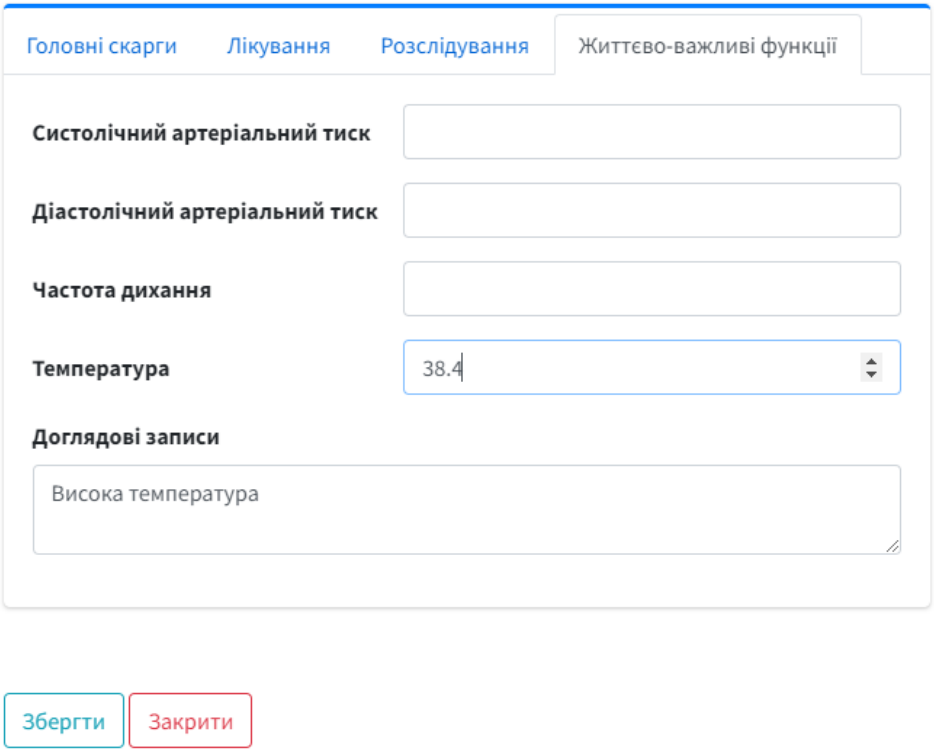

Рисунок 3.9 – вкладка «Життєво-важливі функції»

На рис.3.10 містить вікно з всією інформацією про обстеження. Якщо пацієнт захоче мати копію свого обстеження, він може попросити лікаря завантажити обстеження у форматі PDF на USB-флеш-накопичувач. Щоб завантажити обстеження, лікарю потрібно натиснути кнопку з іконкою «Завантажити».

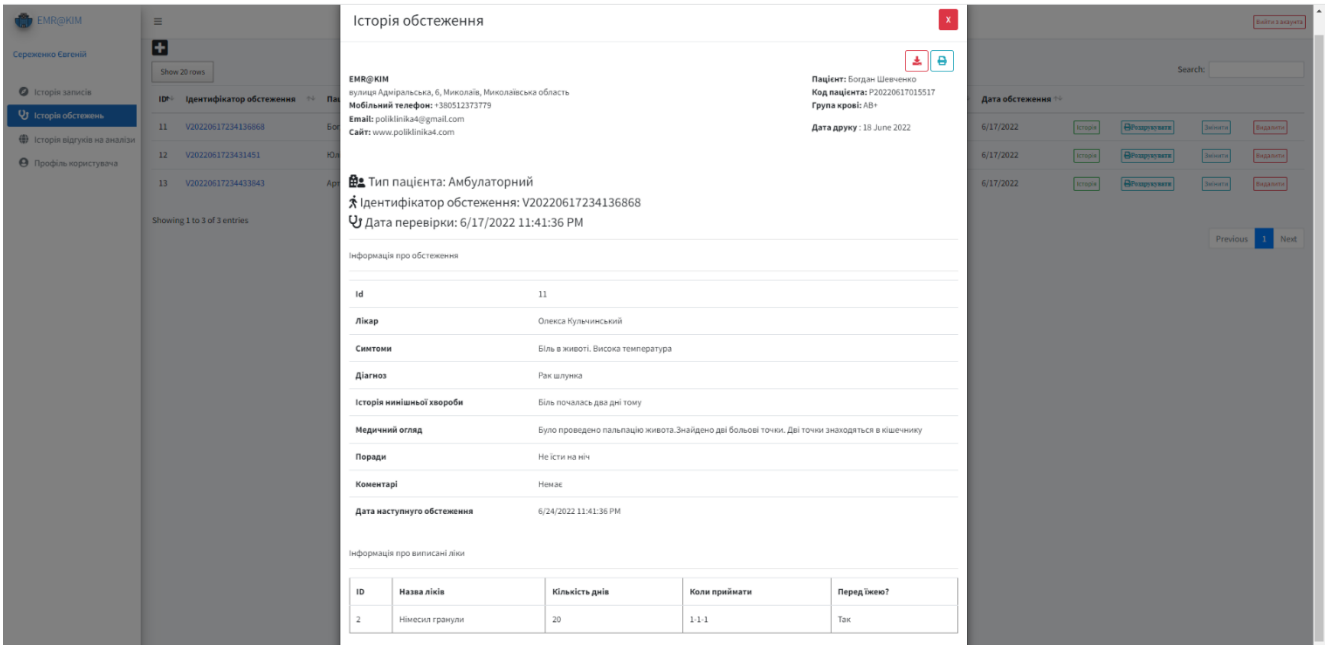

Рисунок 3.10 – Вікно з історією обстеження для завантаження або друку

Історія відгуків на аналізи

Якщо лікар відправить пацієнта на лабораторні дослідження, то йому потрібно буде надати відгуки на аналізи пацієнта. В таблиці на рис. 3.11 знаходиться пакети лабораторних досліджень, на які лікар надав відгуки. Лікар може виконувати такі дії з пакетами лабораторних досліджень, як: створити, змінити, видалити та знайти пакет з аналізами за потрібним значенням.

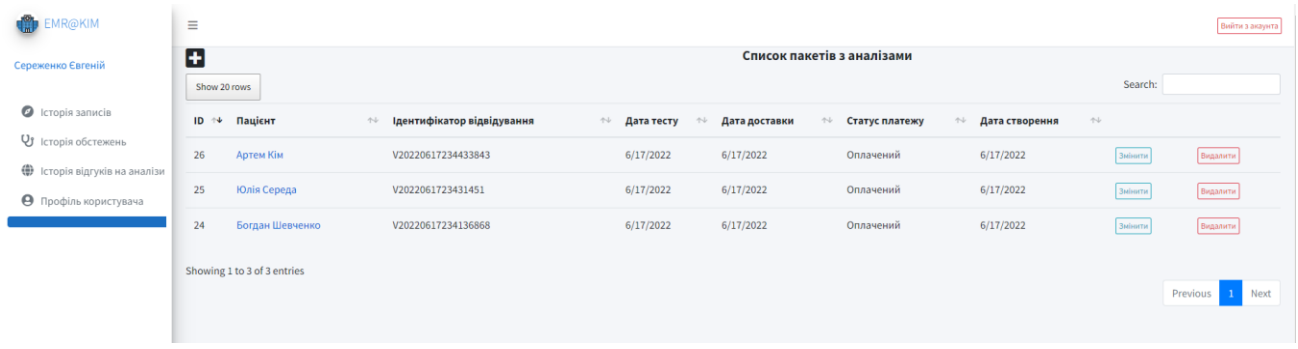

Рисунок 3.11 – Пакети лабораторних досліджень

Після того, як лікар створить направлення на лабораторні дослідження, система автоматично створює пакет з направленими аналізами. На рис. 3.12 форма з направленими аналізами.

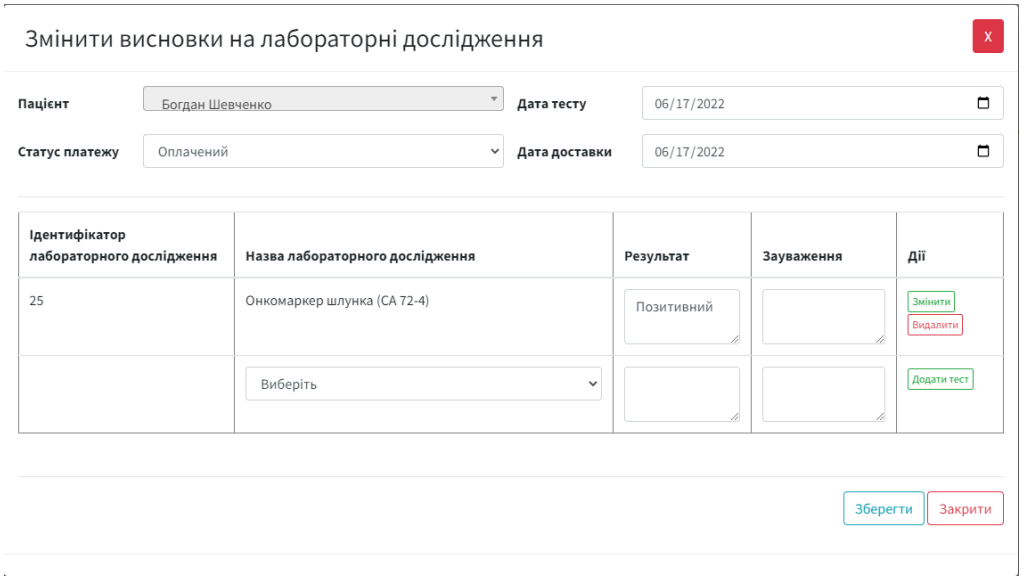

Рисунок 3.12 – Форма з направленими аналізами

Це вікно призначене для зміни даних про пакет лабораторних досліджень, та додавання у таблицю висновки про аналізи.

Висновки до розділу 3

В третьому розділі кваліфікаційної роботи розглянуто і описано архітектуру вебзастосунку, та його компонентів, такі як: модель, вид та контролер. Реалізація компонентів була додана до додатків. Також описано функції системи, та показано в рисунках інтерфейс вебзастосунку.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра було розроблено вебзастосунок для обліку медичних карток, який допоможе лікарям документувати хвороби пацієнтів, та краще керувати своїм робочим часом, а для медустанов спростить створення та зберігання медичних карток. Для виконання роботи відповідно до мети визначено завдання, які успішно виконані.

Вирішено наступні завдання:

- аналіз предметної області та існуючих аналогів;
- сформувати специфікацію вимог до вебзастосунку;
- створити макет вебзастосунку;
- смоделювати вебзастосунок;
- підібрати технології для ефективної розробки вебзастосунку;
- розробити вебзастосунок.

Обрано технології для розробки серверної частини вебзастосунку, де мова програмування CSharp, вебфреймворк ASP.NET Core та фреймворк для бази даних Entity Framework. Для клієнтської частини вебзастосунку обрано, такі технології: мова програмування JavaScript, інтерфейсний фреймворк Bootstrap, для створення панелі AdminLTE, та для створення потужних таблиць бібліотека jQuery data tables. Для створення вебзастосунку було використано архітектурний шаблон MVC.

Результатом проведеної роботи є вебзастосунок для обліку медичних карток, який складається з двух частин: клієнтської частини та серверної. Розроблений вебзастосунок відповідає всім сформованим вимогам до системи.

ПЕРЕЛІК ДЖЕРЕЛ ПИСИЛАННЯ

1. Діаграми UML для моделювання процесів і архітектури проекту. *Evergreen - web розробка і діджиталізація бізнесу за допомогою AI продуктів*. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 14.06.2022).

2. Електронна система охорони здоров”я. *Електронна система охорони здоров”я*. URL: https://ehealth.gov.ua/ (дата звернення: 14.06.2022).

3. Модели баз данных на логическом и физическом уровнях. *Studme*. URL: https://studme.org/380568/informatika/modeli_dannyh_logicheskom_fizicheskom_urovnyah (дата звернення: 14.06.2022).

4. Різниця між автентифікацією та авторизацією - Технологія - 2022. *fondoperlaterra*. URL: https://uk.fondoperlaterra.org/comdifference-between-authentication-and-authorization-21 (дата звернення: 14.06.2022).

5. Точна різниця між SQL та NoSQL (знайте, коли використовувати NoSQL та SQL) - Інший. *Огляди, Ігри, Розваги, Червень 2022*. URL: https://uk.myservername.com/sql-vs-nosql-exact-differences (дата звернення: 21.06.2022).

6. Что такое ER-диаграмма? *Lucidchart*. URL: https://www.lucidchart.com/pages/ru/erd-диаграмма (дата звернення: 14.06.2022).

7. #1 EHR (electronic health records) - easy to use & intuitive. *Award Winning EHR, Telemedicine, Medical Billing & Patient Engagement*. URL: https://www.curemd.com/ehr.asp (date of access: 14.06.2022).

8. ASP.NET core | введение в MVC. *METANIT.COM - Сайт о программировании*. URL: https://metanit.com/sharp/aspnet5/3.1.php (дата звернення: 21.06.2022).

9. Bootstrap. *Bootstrap · The most popular HTML, CSS, and JS library in the world.* URL: https://getbootstrap.com/ (date of access: 21.06.2022).

10. Component diagrams. *Edrawsoft*. URL: https://www.edrawsoft.com/uml-component.html (date of access: 14.06.2022).

11. Create a UML use case diagram. *Microsoft Support*. URL: https://support.microsoft.com/en-gb/office/create-a-uml-use-case-diagram-92cc948d-fc74-466c-9457-e82d62ee1298 (date of access: 14.06.2022).

12. Create better wireframes and prototypes with moqups. *Your virtual Project Manager*. URL: https://yourvirtualprojectmanager.com/create-better-wireframes-and-prototypes-with-moqups/ (date of access: 14.06.2022).

13. CureMD. *Software Selection Tool | Software Selection Management | SelectHub*. URL: https://www.selecthub.com/emr-software/curemd/ (date of access: 21.06.2022).

14. DataTables | Table plug-in for jQuery. *DataTables | Table plug-in for jQuery*. URL: https://datatables.net/ (date of access: 21.06.2022).

15. Download SQL server management studio (SSMS) 19 - SQL server management studio (SSMS). *Developer tools, technical documentation and coding examples | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms-19?view=sql-server-ver16 (date of access: 21.06.2022).

16. Dr. ELEKS. *Dr. ELEKS*. URL: https://doctor.eleks.com/product-ru/system-architecture-ru/ (date of access: 14.06.2022).

17. EHR systems | insync healthcare solutions. *Medical Software by InSync Healthcare Solutions*. URL: https://www.insynchcs.com/emr (date of access: 14.06.2022).

18. Entity Framework documentation. *Developer tools, technical documentation and coding examples | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/ef/ (date of access: 14.06.2022).

19. IBM docs. *IBM - Deutschland | IBM*. URL: https://www.ibm.com/docs/en/rational-soft-arch/9.5?topic=diagrams-deployment (date of access: 14.06.2022).

20. JavaScript - Клієнтською мова програмування, що робить сторінки сайту інтерактивними. *Astwellsoft - A software development company that's easy to work with*. URL: https://astwellsoft.com/uk/blog/tehnology/javascript.html (дата звернення: 21.06.2022).

21. Medical software by insync healthcare solutions. *Medical Software by InSync Healthcare Solutions*. URL: https://www.insynchcs.com/ (date of access: 21.06.2022).

22. MVC framework - introduction. *Online Tutorials Library*. URL: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm (date of access: 21.06.2022).

23. Overview of visual studio. *Developer tools, technical documentation and coding examples | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022 (date of access: 14.06.2022).

24. Trubitsyn A. Статичні та динамічні діаграми. Навіщо та як використовувати їх під час документування архітектури. *ДОУ*. URL: https://dou.ua/lenta/columns/static-and-dynamic-diagrams/ (дата звернення: 14.06.2022).

25. What is visual studio? - incredibuild. *Incredibuild*. URL: https://www.incredibuild.com/integrations/visual-studio (date of access: 14.06.2022).

ДОДАТОК А

Фізична модель бази даних

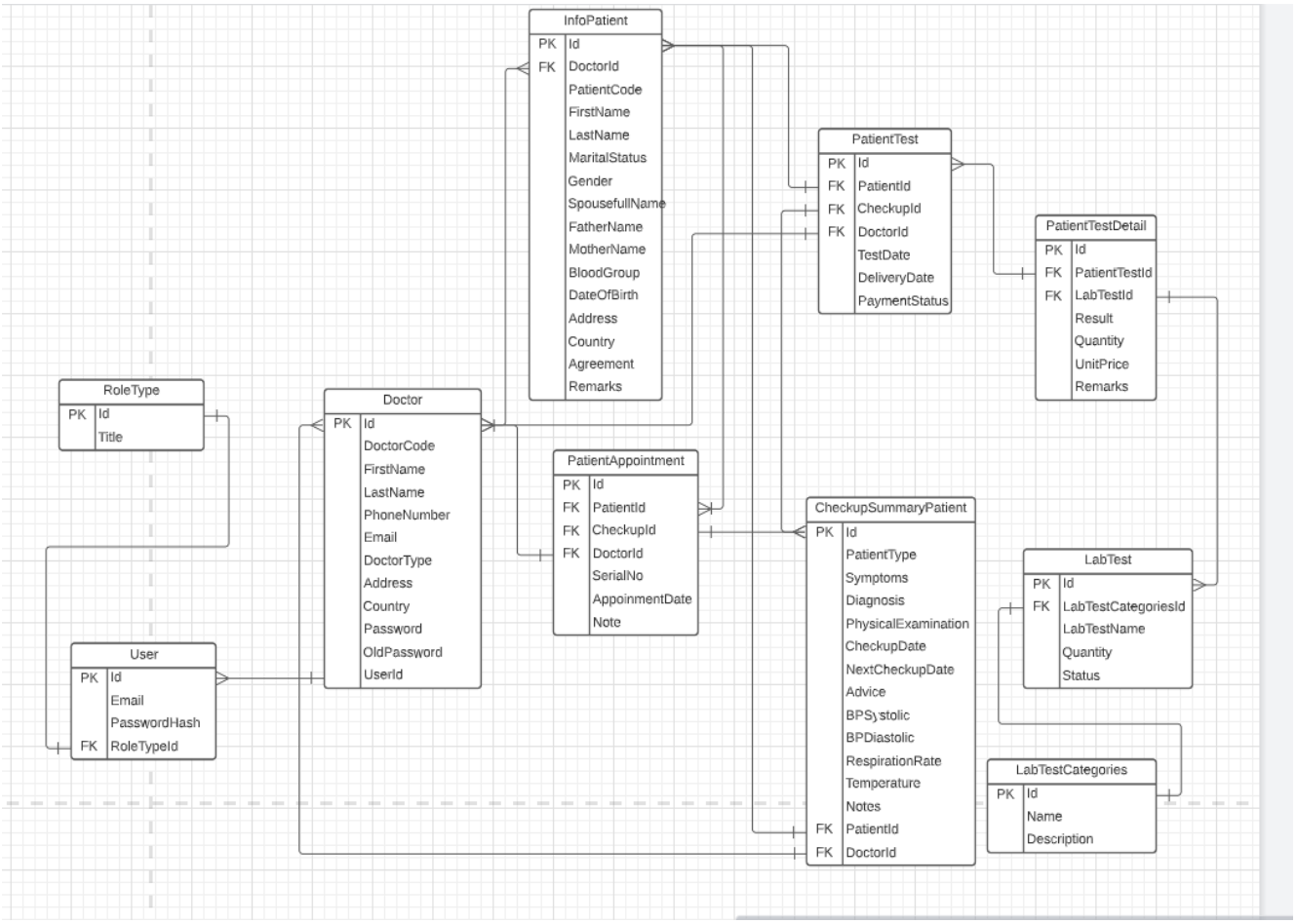

Рисунок 1 – Фізична модель бази даних

ДОДАТОК Б

Друк документів

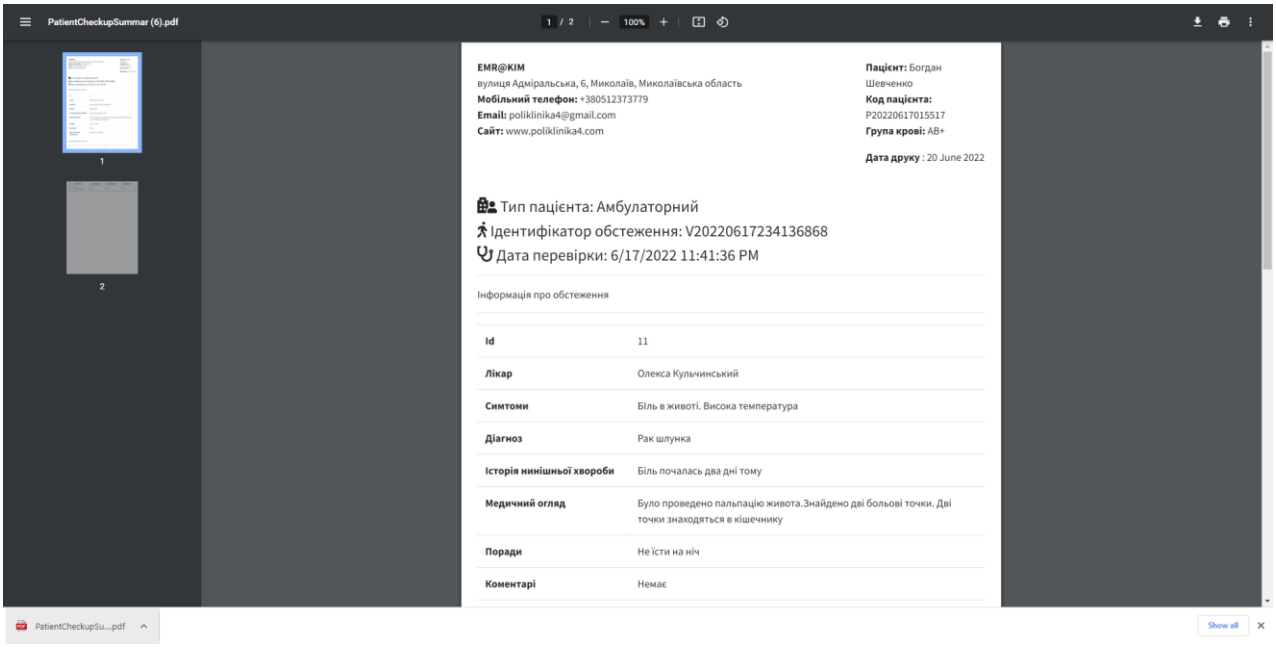

Рисунок 1 – Відкритий файл в браузері

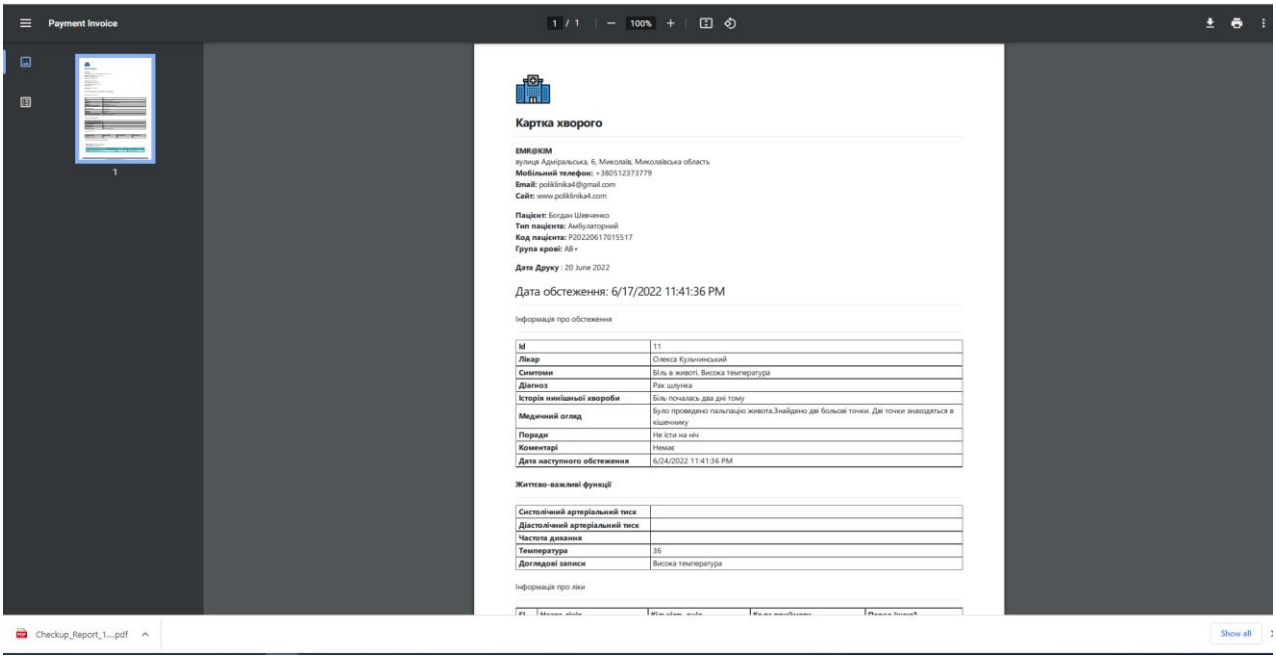

Рисунок 2 – Відкритий завантажений файл з медичною карткою

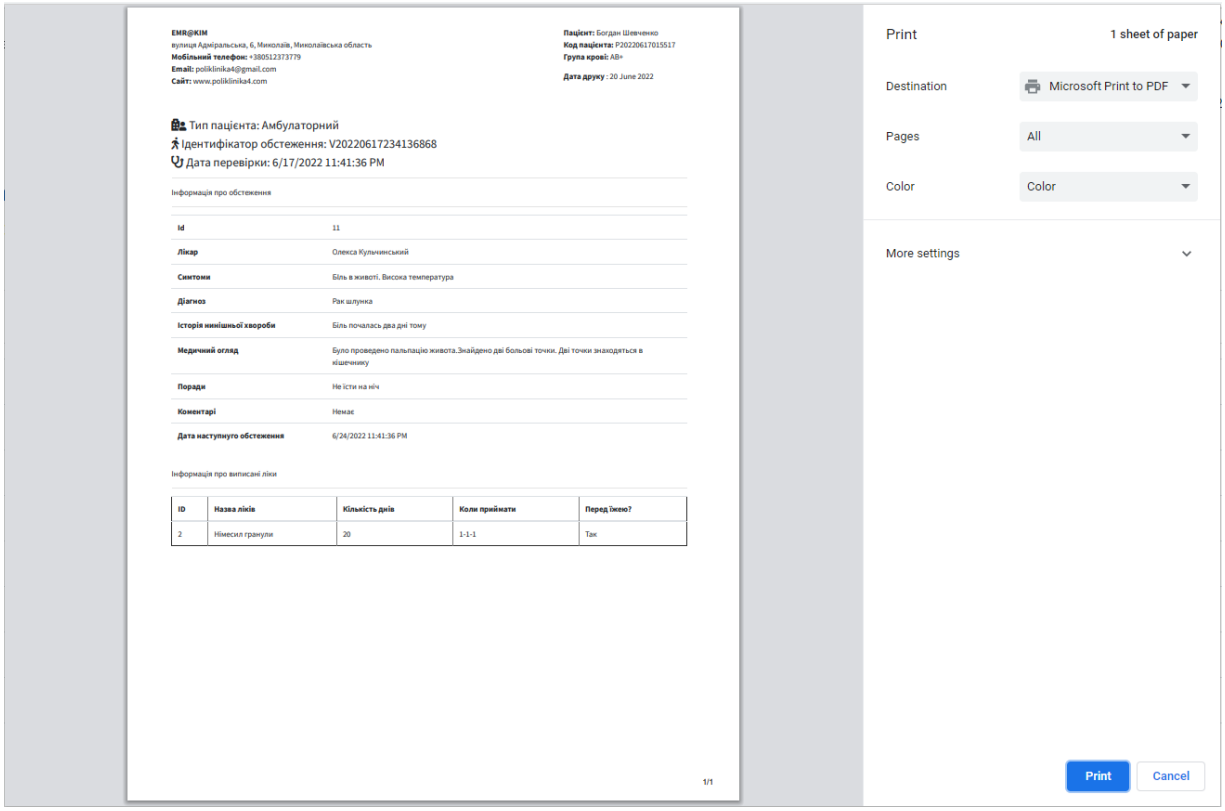

Рисунок 3 – Друк обстеження

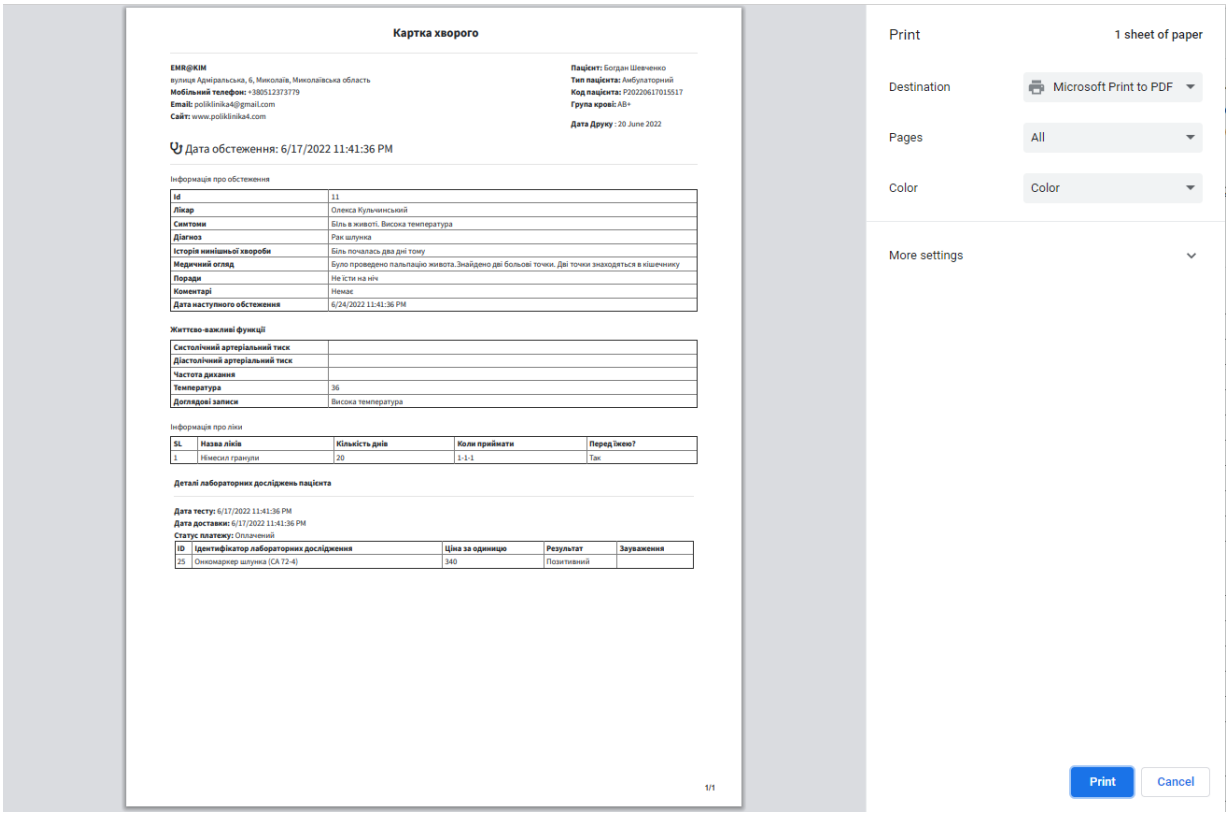

Рисунок 4 – Друк медичної картки пацієнта

ДОДАТОК В

Найголовніші моделі вебзастосунку

PatientAppointmentModel

```
public class PatientAppointment : EntityBase
  {
    public Int64 Id { get; set; }
    public Int64 PatientId { get; set; }
    public string VisitId { get; set; }
    public string PatientType { get; set; }
    public Int64 DoctorId { get; set; }
    public Int64 SerialNo { get; set; }
    public DateTime AppointmentDate { get; set; }
    public DateTime AppointmentTime { get; set; }
    public string Note { get; set; }
  }
```

PatientTestModel

```
 public class PatientTest : EntityBase
  {
    public Int64 Id { get; set; }
    public Int64 PatientId { get; set; }
    public string VisitId { get; set; }
    public string ApplicationUserId { get; set; }
    public DateTime TestDate { get; set; }
    public DateTime DeliveryDate { get; set; }
    public string PaymentStatus { get; set; }
  }
```

PatientInfoModel

```
public class PatientInfo : EntityBase
  {
    public Int64 Id { get; set; }
    public string PatientCode { get; set; }
    public string ApplicationUserId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string MaritalStatus { get; set; }
    public string Gender { get; set; }
    public string SpouseName { get; set; }
    public string BloodGroup { get; set; }
    public string FatherName { get; set; }
    public string MotherName { get; set; }
    public DateTime? DateOfBirth { get; set; }
    public string Phone { get; set; }
    public string Email { get; set; }
    public string Address { get; set; }
    public string Country { get; set; }
    public bool? Agreement { get; set; }
    public string Remarks { get; set; }
  }
```

UserProfileModel

```
public class UserProfile : EntityBase
    {
        public Int64 Id { get; set; }
```

```
    public string ApplicationUserId { get; set; }
    public string UserType { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string PhoneNumber { get; set; }
    public string Email { get; set; }
    public string Address { get; set; }
    public string Country { get; set; }
    public string PasswordHash { get; set; }
    public string ConfirmPassword { get; set; }
    public string OldPassword { get; set; }
  }
```

CheckupSummary

```
public class CheckupSummary : EntityBase
  {
    public Int64 Id { get; set; }
    public Int64 PatientId { get; set; }
    public string VisitId { get; set; }
    public Int64 PaymentId { get; set; }
    public Int64 DoctorId { get; set; }
    public string PatientType { get; set; }
    public string Symptoms { get; set; }
    public string Diagnosis { get; set; }
    public string HPI { get; set; }
    public string PhysicalExamination { get; set; }
    public string Comments { get; set; }
    public DateTime CheckupDate { get; set; }
```

```
    public DateTime NextVisitDate { get; set; }
    public string Advice { get; set; }
    public int? BPSystolic { get; set; }
    public int? BPDiastolic { get; set; }
    public int? RespirationRate { get; set; }
    public int? Temperature { get; set; }
    public string NursingNotes { get; set; }
}
```

ДОДАТОК Г

Найголовніші контролери вебзастосунку

```
public class UserRoleController : Controller
  {
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly RoleManager<IdentityRole> _roleManager;

    public UserRoleController(UserManager<ApplicationUser> userManager,
RoleManager<IdentityRole> roleManager)
    {
      _userManager = userManager;
      _roleManager = roleManager;
    }

    [Authorize(Roles = Pages.MainMenu.UserManagement.RoleName)]
    public async Task<IActionResult> Index(int id)
    {
      var _ApplicationUser = await
_userManager.FindByEmailAsync(HttpContext.User.Identity.Name);
      ViewBag.ApplicationUserId = _ApplicationUser.Id;
      var user = await _userManager.FindByIdAsync(_ApplicationUser.Id);
      if (user == null)
      {
        ViewBag.ErrorMessage = $"User with Id = {_ApplicationUser.Id} cannot be
found";
        return View("NotFound");
      }
      ViewBag.UserName = user.UserName;
      var model = new List<ManageUserRolesViewModel>();
      foreach (var role in _roleManager.Roles)
      {
        var userRolesViewModel = new ManageUserRolesViewModel
        {
          RoleId = role.Id,
          RoleName = role.Name
        };
        if (await _userManager.IsInRoleAsync(user, role.Name))
        {
          userRolesViewModel.Selected = true;
        }
        else
        {
          userRolesViewModel.Selected = false;
```

```
            }
            model.Add(userRolesViewModel);
        }
        return View(model.OrderBy(x => x.RoleName).ToList());
    }


    [HttpPost]
    public async Task<IActionResult>
ManageRole(List<ManageUserRolesViewModel> model)
    {
        string _ApplicationUserId = TempData["ApplicationUserId"].ToString();
        var user = await _userManager.FindByIdAsync(_ApplicationUserId);
        if (user == null)
        {
            return View();
        }
        var roles = await _userManager.GetRolesAsync(user);
        var result = await _userManager.RemoveFromRolesAsync(user, roles);
        if (!result.Succeeded)
        {
            ModelState.AddModelError("", "Cannot remove user existing roles");
            return View(model);
        }
        result = await _userManager.AddToRolesAsync(user, model.Where(x =>
x.Selected).Select(y => y.RoleName));
        if (!result.Succeeded)
        {
            ModelState.AddModelError("", "Cannot add selected roles to user");
            return View(model);
        }
        TempData["successAlert"] = "Role update Successfully. User Name: " +
user.Email;
        return RedirectToAction(nameof(Index));
    }

public class PatientTestController : Controller

  {

    private readonly ApplicationDbContext _context;

    private readonly ICommon _iCommon;

    private readonly IDBOperation _iDBOperation;
```

```
public PatientTestController(ApplicationDbContext context, ICommon iCommon, IDBOperation iDBOperation)
{
    _context = context;
    _iCommon = iCommon;
    _iDBOperation = iDBOperation;
}

[Authorize(Roles = Pages.MainMenu.PatientTest.RoleName)]
public IActionResult Index()
{
    return View();
}

[HttpPost]
public IActionResult GetDataTabelData()
{
    try
    {
        var draw = HttpContext.Request.Form["draw"].FirstOrDefault();
        var start = Request.Form["start"].FirstOrDefault();
        var length = Request.Form["length"].FirstOrDefault();

        var sortColumn = Request.Form["columns[" + Request.Form["order[0][column]"].FirstOrDefault() + "][name]"].FirstOrDefault();
        var sortColumnAscDesc = Request.Form["order[0][dir]"].FirstOrDefault();
        var searchValue = Request.Form["search[value]"].FirstOrDefault();
```

```
            int pageSize = length != null ? Convert.ToInt32(length) : 0;
            int skip = start != null ? Convert.ToInt32(start) : 0;
            int resultTotal = 0;

            var _GetGridItem = _iCommon.GetPatientTestGridItem();
            //Sorting
            if (!(string.IsNullOrEmpty(sortColumn) &&
string.IsNullOrEmpty(sortColumnAscDesc)))
            {
               _GetGridItem = _GetGridItem.OrderBy(sortColumn + " " +
sortColumnAscDesc);
            }

            //Search
            if (!string.IsNullOrEmpty(searchValue))
            {
               searchValue = searchValue.ToLower();
               _GetGridItem = _GetGridItem.Where(obj =>
obj.Id.ToString().Contains(searchValue)
               || obj.PatientName.Contains(searchValue)
               || obj.VisitId.Contains(searchValue)
               || obj.TestDate.ToString().Contains(searchValue)
               || obj.DeliveryDate.ToString().Contains(searchValue)
               || obj.PaymentStatus.Contains(searchValue)
               || obj.CreatedDate.ToString().Contains(searchValue));
            }

            resultTotal = _GetGridItem.Count();
```

```
            var result = _GetGridItem.Skip(skip).Take(pageSize).ToList();
            return Json(new { draw = draw, recordsFiltered = resultTotal, recordsTotal = resultTotal, data = result });
        }
        catch (Exception ex)
        {
            throw ex;
        }

    }

    public async Task<IActionResult> Details(Int64 id)
    {
        var result = await _iCommon.GetByPatientTestDetails(id);
        if (result == null) return NotFound();
        return PartialView("_Details", result);
    }

    public async Task<IActionResult> AddEdit(Int64 id)
    {
        ViewBag._LoadddlPatientName = new SelectList(_iCommon.LoadddlPatientName(), "Id", "Name");
        ViewBag._LoadddlDoctorName = new SelectList(_iCommon.LoadddlDoctorName(), "Id", "Name");
        ViewBag._LoadddlLabTests = new SelectList(_iCommon.LoadddlLabTests(), "Id", "Name");

        ManagePatientTestViewModel _ManagePatientTestViewModel = new ManagePatientTestViewModel();
        if (id > 0)
```

```
        {
            _ManagePatientTestViewModel = await
_iCommon.GetByPatientTestDetails(id);
        }
        return PartialView("_AddEdit", _ManagePatientTestViewModel);
    }

    [HttpPost]
    public async Task<IActionResult> AddEdit(ManagePatientTestViewModel vm)
    {
        try
        {
            PatientTest _PatientTest = new PatientTest();
            if (vm.PatientTestCRUDViewModel.Id > 0)
            {
                vm.PatientTestCRUDViewModel.CreatedDate = _PatientTest.CreatedDate;
                vm.PatientTestCRUDViewModel.CreatedBy = _PatientTest.CreatedBy;
                vm.PatientTestCRUDViewModel.ModifiedDate = DateTime.Now;
                vm.PatientTestCRUDViewModel.ModifiedBy =
HttpContext.User.Identity.Name;

_context.Entry(_PatientTest).CurrentValues.SetValues(vm.PatientTestCRUDViewMod
el);
                await _context.SaveChangesAsync();



                foreach (var item in vm.listPatientTestResultUpdateViewModel)
                {
                    var result = await UpdatePatientTestDetail(item);
```

```
            }

            var _AlertMessage = "Patient Test Updated Successfully. ID: " +
_PatientTest.Id;
            TempData["successAlert"] = _AlertMessage;
            return new JsonResult(_AlertMessage);
          }
          else
          {
            //Bug
            //vm.PatientTestCRUDViewModel.ApplicationUserId =
_context.DoctorsInfo.Where(x => x.Id ==
vm.PatientTestCRUDViewModel.Id).SingleOrDefault().ApplicationUserId;
            _PatientTest = vm.PatientTestCRUDViewModel;
            _PatientTest.CreatedDate = DateTime.Now;
            _PatientTest.ModifiedDate = DateTime.Now;
            _PatientTest.CreatedBy = HttpContext.User.Identity.Name;
            _PatientTest.ModifiedBy = HttpContext.User.Identity.Name;
            _context.Add(_PatientTest);
            await _context.SaveChangesAsync();

            foreach (var item in vm.listPatientTestDetailCRUDViewModel)
            {
              PatientTestDetail _PatientTestDetail = item;
              _PatientTestDetail.PatientTestId = _PatientTest.Id;
              _PatientTestDetail.CreatedBy = HttpContext.User.Identity.Name;
              _PatientTestDetail.ModifiedBy = HttpContext.User.Identity.Name;
              await _iCommon.CreatePatientTestDetail(_PatientTestDetail);
            }
```

```
                TempData["successAlert"] = "Patient Test Created Successfully. ID: " +
_PatientTest.Id;
                return new JsonResult(_PatientTest);
            }
        }
        catch (DbUpdateConcurrencyException ex)
        {
            TempData["errorAlert"] = "Operation failed.";
            if (!IsExists(vm.PatientTestCRUDViewModel.Id))
                return NotFound();
            else
                throw ex;
        }
    }

    [HttpPost]
    public async Task<IActionResult>
SavePatientTestDetail(PatientTestDetailCRUDViewModel vm)
    {
        string strCurrentUserName = HttpContext.User.Identity.Name;
        PatientTestDetail _PatientTestDetail = vm;
        _PatientTestDetail.CreatedBy = strCurrentUserName;
        _PatientTestDetail.ModifiedBy = strCurrentUserName;
        var result = await _iCommon.CreatePatientTestDetail(_PatientTestDetail);

        //Add Payment Details
        if (result != null)
        {
```

```
            AddPaymentViewModel _AddPaymentViewModel = new
AddPaymentViewModel();

            var _VisitId = _context.PatientTest.Where(x => x.Id ==
result.PatientTestId).SingleOrDefault().VisitId;

            var _PaymentId = _context.Payments.Where(x => x.VisitId ==
_VisitId).SingleOrDefault().Id;

            _AddPaymentViewModel.LabTestsId = _PatientTestDetail.LabTestsId;

            _AddPaymentViewModel.PaymentsId = _PaymentId;

            await _iDBOperation.CreatePaymentsDetails(_AddPaymentViewModel,
PaymentItemType.LabTest, strCurrentUserName);

        }


        return new JsonResult(result);

    }

    [HttpPost]

    public async Task<IActionResult>
UpdatePatientTestDetailDB(PatientTestResultUpdateViewModel vm)

    {

        var _PatientTestDetail = await UpdatePatientTestDetail(vm);

        return new JsonResult(_PatientTestDetail);

    }


    private async Task<PatientTestDetail>
UpdatePatientTestDetail(PatientTestResultUpdateViewModel vm)

    {

        var _PatientTestDetail = await _context.PatientTestDetail.FindAsync(vm.Id);

        vm.ModifiedDate = DateTime.Now;

        vm.ModifiedBy = HttpContext.User.Identity.Name;

        _context.Entry(_PatientTestDetail).CurrentValues.SetValues(vm);

        _context.SaveChanges();
```

```
        return _PatientTestDetail;
    }

    [HttpPost]
    public async Task<IActionResult>
DeletePatientTestDetail(PatientTestDetailCRUDViewModel vm)
    {
        try
        {
            var _PatientTestDetail = await _context.PatientTestDetail.FindAsync(vm.Id);
            //var _PatientTestDetail = await _context.PatientTestDetail.Where(x =>
x.LabTestsId == vm.LabTestsId && x.PatientTestId ==
vm.PatientTestId).SingleOrDefaultAsync();
            _PatientTestDetail.ModifiedDate = DateTime.Now;
            _PatientTestDetail.ModifiedBy = HttpContext.User.Identity.Name;
            _PatientTestDetail.Cancelled = true;

            _context.Update(_PatientTestDetail);
            await _context.SaveChangesAsync();
            return new JsonResult(_PatientTestDetail);
        }
        catch (Exception ex)
        {
            throw ex;
        }
    }

    [HttpPost]
    public async Task<IActionResult> Delete(Int64 id)
```

```
        {
            try
            {
                var _PatientTest = await _context.PatientTest.FindAsync(id);
                _PatientTest.ModifiedDate = DateTime.Now;
                _PatientTest.ModifiedBy = HttpContext.User.Identity.Name;
                _PatientTest.Cancelled = true;

                _context.Update(_PatientTest);
                await _context.SaveChangesAsync();
                return new JsonResult(_PatientTest);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        private bool IsExists(long id)
        {
            return _context.PatientTest.Any(e => e.Id == id);
        }
    }
public class UserProfileController : Controller
    {
        private readonly IRoles _roles;
        private readonly ApplicationDbContext _context;
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;
        private readonly ICommon _iCommon;
```

```
    public UserProfileController(UserManager<ApplicationUser> userManager,
        ApplicationDbContext context,
        RoleManager<IdentityRole> roleManager,
        IRoles roles,
        ICommon iCommon)
    {
        _context = context;
        _roleManager = roleManager;
        _userManager = userManager;
        _roles = roles;
        _iCommon = iCommon;
    }

    [Authorize(Roles = Pages.MainMenu.UserProfile.RoleName)]
    [HttpGet]
    public async Task<IActionResult> Index()
    {
        var _ApplicationUser = await
_userManager.FindByEmailAsync(HttpContext.User.Identity.Name);
        UserProfileCRUDViewModel _UserProfileCRUDViewModel = new
UserProfileCRUDViewModel();
        if (_ApplicationUser.Id != null)
        {
            _UserProfileCRUDViewModel = _context.UserProfile.Where(x =>
x.ApplicationUserId == _ApplicationUser.Id).SingleOrDefault();
        }
        return View(_UserProfileCRUDViewModel);
    }

    [HttpGet]
    public IActionResult EditUserProfile(int id)
    {
        return PartialView("_EditUserProfile", _iCommon.GetByUserProfile(id));
    }

    [HttpPost]
    public async Task<IActionResult> EditUserProfile(UserProfileCRUDViewModel
_UserProfileCRUDViewModel)
    {
        try
        {
            UserProfile _UserProfile =
_iCommon.GetByUserProfile(_UserProfileCRUDViewModel.Id);
```

```
            _UserProfile.FirstName = _UserProfileCRUDViewModel.FirstName;
            _UserProfile.LastName = _UserProfileCRUDViewModel.LastName;
            _UserProfile.PhoneNumber = _UserProfileCRUDViewModel.PhoneNumber;
            _UserProfile.Address = _UserProfileCRUDViewModel.Address;
            _UserProfile.Country = _UserProfileCRUDViewModel.Country;

            if (_UserProfileCRUDViewModel.ProfilePictureDetails != null)
                _UserProfile.ProfilePicturePath = "/upload/" +
_iCommon.UploadedFile(_UserProfileCRUDViewModel.ProfilePictureDetails);

            _UserProfile.ModifiedDate = DateTime.Now;
            _UserProfile.ModifiedBy = HttpContext.User.Identity.Name;
            var result2 = _context.UserProfile.Update(_UserProfile);
            await _context.SaveChangesAsync();
            TempData["successAlert"] = "User info Updated Successfully. User Name: "
+ _UserProfileCRUDViewModel.Email;
            return RedirectToAction(nameof(Index));
        }
        catch (Exception ex)
        {
            throw ex;
        }
    }

    [HttpGet]
    public async Task<IActionResult> ResetPassword(string id)
    {
        var _ApplicationUser = await _userManager.FindByIdAsync(id);
        ResetPasswordVM _ResetPasswordVM = new ResetPasswordVM();
        _ResetPasswordVM.ApplicationUserId = _ApplicationUser.Id;
        return PartialView("_ResetPassword", _ResetPasswordVM);
    }

    [HttpPost]
    public async Task<IActionResult> ResetPassword(ResetPasswordVM vm)
    {
        var _ApplicationUser = await
_userManager.FindByIdAsync(vm.ApplicationUserId);
        if (vm.NewPassword.Equals(vm.ConfirmPassword))
        {
            var result = await _userManager.ChangePasswordAsync(_ApplicationUser,
vm.OldPassword, vm.NewPassword);
            if (result.Succeeded)
```

```
                TempData["successAlert"] = "Change Password Succeeded. User name: "
+ _ApplicationUser.Email;
            else
            {
                string errorMessage = string.Empty;
                foreach (var item in result.Errors)
                {
                    errorMessage = errorMessage + " " + item.Description;
                }
                TempData["errorAlert"] = errorMessage;
            }
        }
        return RedirectToAction(nameof(Index));
    }
```