Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Гришко Олени Олегівни*
(ПІБ)

академічної групи *122-20з-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка комп'ютерної системи для медичних лабораторій*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*проф. Лактіонов І.С.*			
розділів:				
спеціальний	*проф. Лактіонов І.С.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20з-1* *Гришко О.О.*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка комп'ютерної системи для медичних лабораторій*

затверджена наказом ректора НТУ «ДП» від *23.05.2024 р.* № *470-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***10.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***14.06.2024 р.***

Завдання видав _____ *проф. Лактіонов І.С*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Гришко О.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *17.06.2024 р*

РЕФЕРАТ

Пояснювальна записка: 93 с., 40 рис., 3 дод., 22 джерел.

Об'єкт розробки: комп’ютерна система для медичних лабораторій.

Мета кваліфікаційної роботи: створення комп’ютерної системи для забезпечення можливості взаємодії клієнта з працівниками лабораторії через мережу Інтернет.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проаналізовано предметну галузь та наявні рішення, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі обрано платформи для розробки, виконано проєктування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає у створенні комп’ютерної системи, що надає можливість автоматизації взаємодії з клієнтами, а також електронного зберігання даних про медичні аналізи, замовлення клієнтів і медичної документації пов’язаної з результатами досліджень здоров’я відвідувачів лабораторії.

Актуальність інформаційної системи визначається великим попитом на подібні розробки, що оптимізують та спрощують комунікацію співробітників лабораторії з її клієнтами, скорочують час обслуговування та покращують досвід користування послугами лабораторії; підвищують ефективність діяльності лабораторії шляхом електронного ведення документації з можливістю аналізу наявних даних і надання необхідних звітів і супутньої ділової документації.

Список ключових слів: ІНФОРМАЦІЙНА СИСТЕМА, ОБЛІК, ІНТЕРФЕЙС, ПРОЄКТУВАННЯ, ПАНЕЛЬ НАВІГАЦІЇ, БАГАТОРІВНЕВИЙ ДОСТУП.

ABSTRACT

Explanatory note: 93 pages., 40 fig, 3 appendixes, 22 sources.

The objective of this bachelor thesis is to create a computer system which enables client interaction with laboratory staff via the Internet.

The introduction examines the analysis and current state of the problem, specifies the objective of the qualification work and its application field, provides justification for the relevance of the topic, and clarifies the task setting.

The first chapter analyzes the subject area and existing solutions, determines the relevance of the task and the purpose of the development, formulates the task setting, specifies the requirements for software implementation, technologies, and software tools.

The second chapter selects platforms for development, designs and develops the program, describes the program's operation, algorithm, and structure, as well as the invocation and loading of the program, defines input and output data, and characterizes the set of technical parameters.

The economic section determines the labor intensity of the developed information system, calculates the cost of the work to create the program, and estimates the time required for its creation.

The practical significance lies in the creation of a computer system that automates client interaction and provides electronic storage of data on medical analyses, client orders, and medical documentation related to health research results of laboratory visitors.

The relevance of the information system is determined by the high demand for such developments, which optimize and simplify communication between laboratory staff and clients, reduce service time, and improve the user experience with laboratory services; enhance laboratory efficiency by electronically maintaining documentation with the ability to analyze existing data and provide necessary reports and related business documentation.

Keywords: INFORMATION SYSTEM, ACCOUNTING, INTERFACE, DESIGN, NAVIGATION PANEL, MULTI-LEVEL ACCESS.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ПК – персональний комп’ютер;

ОС – операційна система;

ЗМІ – засоби масової інформації;

HTML – HyperText Markup Language;

HTTP – HyperText Transfer Protocol;

API – application programming interface;

CSS – Cascading Style Sheets;

DI – Dependency Injection;

ER-модель – entity-relationship model, модель «сутність-зв'язок».

ЗМІСТ

РЕФЕРАТ 3
ABSTRACT 4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 5
ЗМІСТ 6
ВСТУП 8
РОЗДІЛ 1 10
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ 10
1.1. Загальні відомості з предметної галузі 10
1.2 Призначення розробки та галузь застосування 15
1.3. Підстава для розробки 16
1.4. Постановка завдання 16
1.5. Вимоги до програми або програмного виробу 17
1.5.1. Вимоги до функціональних характеристик 17
1.5.2. Вимоги до інформаційної безпеки 19
1.5.3. Вимоги до складу та параметрів технічних засобів 19
1.5.4. Вимоги до інформаційної та програмної сумісності 21
РОЗДІЛ 2 22
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ 22
2.1 Функціональне призначення програми 22
2.2 Опис застосованих математичних методів 22
2.3 Опис використаних технологій та мов програмування 23
2.4 Опис структури програми та алгоритмів її функціонування 25
2.5 Обґрунтування та організація вхідних та вихідних даних програми 32
2.6 Опис розробленої системи 33
2.6.1.Використані технічні засоби. 33
2.6.2 Використані програмні засоби 33
2.6.3 Виклик та завантаження програми 34
2.6.4 Опис інтерфейсу користувача 34

РОЗДІЛ 3 50
ЕКОНОМІЧНИЙ РОЗДІЛ 50
3.1 Розрахунок трудомісткості та вартості розробки програмного продукту 50
3.2. Розрахунок витрат на створення програми 56
ВИСНОВКИ 60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 61
ДОДАТОК А 63
ДОДАТОК Б 92
ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ 92
ДОДАТОК В 93
ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ 93

ВСТУП

На теперішній час неможливо уявити повсякденне життя людини без інформаційних технологій. Кожна людина щодня використовує безліч веб-ресурсів для роботи та відпочинку, таким чином інформаційні технології пришвидшують та полегшують щоденні рутинні задачі людини, покращуючи життя.

Важливою сферою життєдіяльності людини є здоров'я. І необхідно щоб кожен мав змогу вчасно отримати потрібну медичну допомогу. Важливим етапом процесу лікування є діагностика, що включає в себе певний набір досліджень систем організму людини.

Керівники медичних лабораторій розуміють, що клієнту важливо мати можливість онлайн запису на дослідження і вкладають ресурси в розробку веб-додатків з такою функцією. Інтернет сторінка будь-якого закладу або компанії є водночас і візитівкою, що показує головні переваги компанії та допомагає вирізнити її серед конкурентів.

Розроблена комп'ютерна система призначена для застосування у сфері медичного обслуговування.

Метою даної роботи є розробка системи автоматизації обслуговування клієнтів медичної лабораторії.

Головним чином для працівників медичної лабораторії така комп'ютерна система виключає потребу особисто у паперовому вигляді заповнювати дані клієнтів, уточнювати конкретну послугу. Вони можуть приділити більше уваги відвідувачам, що прийшли до клініки безпосередньо для забору матеріалів на дослідження, а також обслуговувати тих, хто з певних причин не може або не хоче користуватися веб-додатком.

Наукові співробітники позбавляються потреби друкувати або писати вручну результати – достатньо завантажити їх у систему, а доставка замовнику здійснюється автоматично. Таким чином медичний спеціаліст має змогу

обробляти більше досліджень за робочий час – збільшується ефективність його праці.

Клієнтам не потрібно чекати у черзі на запис для дослідження та приходити до лабораторії щоб отримати результати, адже вони можуть подивитись їх в особистому кабінеті у додатку. Окрім останніх результатів, можна переглядати попередні записи та моніторити динаміку. Окрім цього людина має змогу швидко знайти потрібне дослідження за назвою та ознайомитись з процесом його проведення детальніше, навіть не виходячи з дому.

Збільшення кількості лабораторій обумовлює приріст попиту на створення комп'ютерних систем для них. Наразі надзвичайно важливо йти в ногу з часом та мати в роботі технології, що створюють комфортне середовище для користувачів та приносять вигоду бізнесу.

Тому задача, розглянута і вирішена в даній кваліфікаційній роботі, є актуальною та має широке практичне значення.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Галузь медичного обслуговування була, є і буде важливою та невід'ємною складовою нашого життя. Кожен, не залежно від віку потребує періодичної перевірки стану здоров'я. Ця галузь характеризується високим рівнем відповідальності та потребою у точних та швидких результатах, що відбивається на вимогах до інформаційних систем, що підтримують діяльність лабораторій.

На сьогоднішній день існують ряд проблем, з якими стикаються як клієнти, так і працівники медичних лабораторій. Першою з них є надмірний час, що займає процес реєстрації нового клієнта в системі. Це стає на шляху до ефективного та швидкого обслуговування, оскільки клієнти витрачають значну кількість часу на заповнення форм та очікування підтвердження реєстрації. Для багатьох клієнтів це може бути надто складним та дратівливим процесом, що призводить до зниження їхньої задоволеності та можливої втрати пацієнтів для лабораторій.

Неможливість онлайн-ознайомлення з послугами та цінами лабораторій є другою серйозною проблемою. Клієнти, які бажають дізнатися про доступні послуги та їхню вартість, змушені звертатися безпосередньо до лабораторій або використовувати інші джерела інформації, що ускладнює їхнє прийняття рішень та може вплинути на вибір конкретного медичного закладу.

Для вирішення цих проблем можна використати інформаційні технології. Наприклад, на ринку існують рішення, що автоматизують процес реєстрації клієнтів за допомогою онлайн-форм, дозволяють перегляд послуг та цін на веб-сайті лабораторії, а також мають функцію замовлення послуг на певний час та дату.

Багато лабораторій на ринку України вже мають власні комп'ютерні системи, що покривають вищеописані проблеми. Розглянемо системи

найкращих лабораторій Києва за Рейтингом Ukrainian Business Award [1] у 2023 році (див. рис. 1.1).

Компанія	Середня оцінка від журі	Місце в рейтингу	Бали	Бали %
Сінево	3,6	1	18,17	100
Діла	4,4	2	14,56	80,11
Ескулаб	3,9	3	10,24	56,37
Invivo	4,3	4	9,04	49,76
CSD Lab	3,6	5	8,37	46,08
DIAGEN	3,7	6	6,79	37,36
NikoLab	3,4	7	6,79	37,35
ДНК-Лабораторія	1,6	8	5,63	30,98
МедЛаб	2,1	9	5,48	30,14
Юнімед	3,8	10	4,04	22,22
Лабораторія Др.Рьодгера	1,5	11	3,22	17,71
IMD	4,2	12	2,11	11,64
HealthyLab	2,3	13	1,05	5,79
Median	1,8	14	1,05	5,76
Нова діагностика	0,8	15	1,04	5,72

Рис. 1.1. Рейтинг «ТОП-15 медичних лабораторій» за версією Ukrainian Business Award

Аналіз відбувався на основі різних показників, таких як: кількість послуг, кількість філій, знання торгової марки (кількість пошукових запитів в Інтернет) та ін.

«Сінево» став лідером ринку завдяки великій кількості відділень, високому рівню обслуговування, широкому спектру послуг та зручним отриманням результатів аналізів через пошту або на веб-сайті компанії.

Розглянувши функції сайту, побачимо що він відповідає складності структури компанії і покриває її протреби в високому рівні автоматизації (див. рис. 1.2).

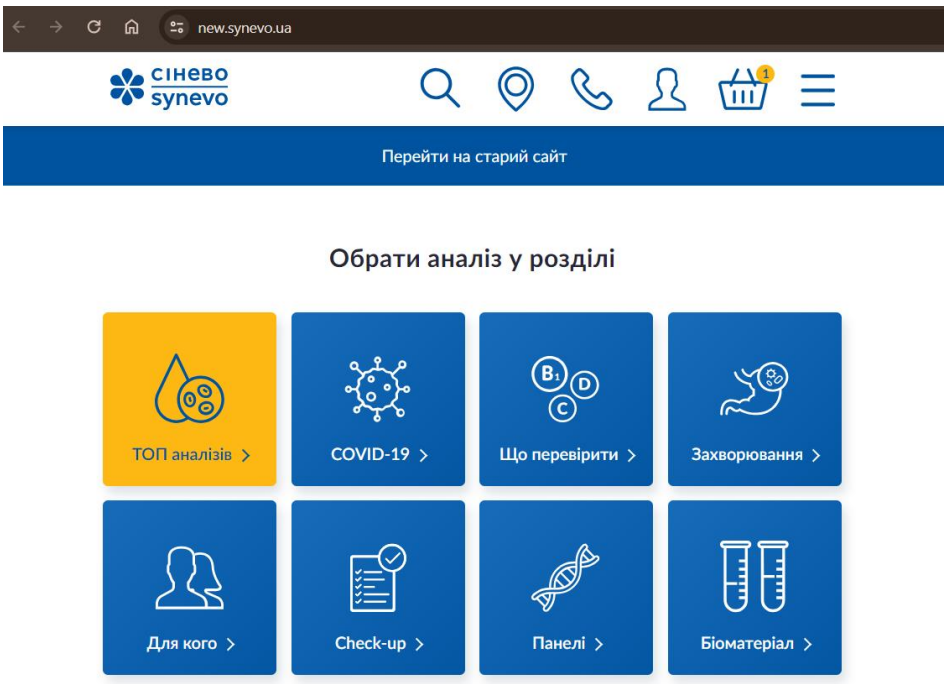


Рис. 1.2. Інтерфейс веб-додатку лабораторії «Сінево» [2]

Велика кількість розділів призначена для полегшення пошуку необхідної послуги. Сайт також грає роль імідж-мейкера: лабораторія розміщує свою офіційну позицію на новини в ЗМІ і публікує новини, пов'язані з її діяльністю (див. рис. 1.3).

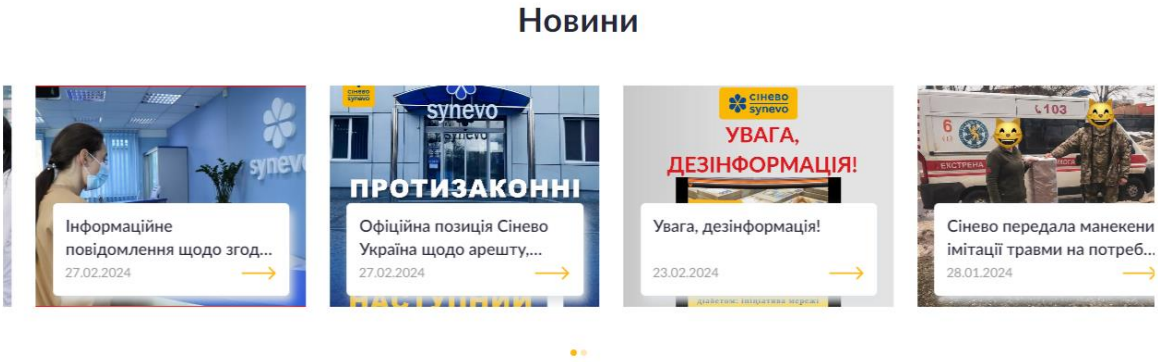


Рис. 1.3. Розділ «Новини» веб-додатку лабораторії «Сінево» [2]

Наступну позицію у рейтингу займає лабораторія «Діла». Інтерфейс її веб-додатку (див. рис. 1.4) більш насичений інформацією, ніж у «Сінево».

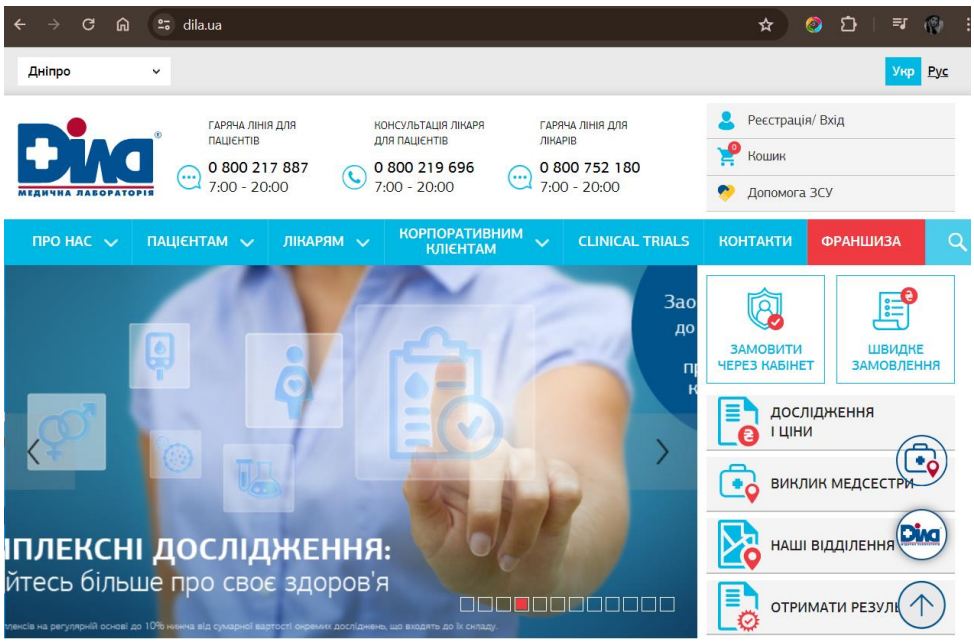


Рис. 1.4. Інтерфейс веб-додатку лабораторії «Діла» [3]

На головній сторінці зібрані різноманітні посилання для широкого кола клієнтів лабораторії: пацієнтів, лікарів, корпоративних клієнтів, франчайзі. Це полегшує комунікацію з бізнес-клієнтами, але може заплутати пацієнтів.

Третя лабораторія в списку – «Ескулаб». Інтерфейс її веб-додатку достатньо схожий на інтерфейс лабораторії «Діла», але є більш мінімалістичним (див. рис. 1.5).

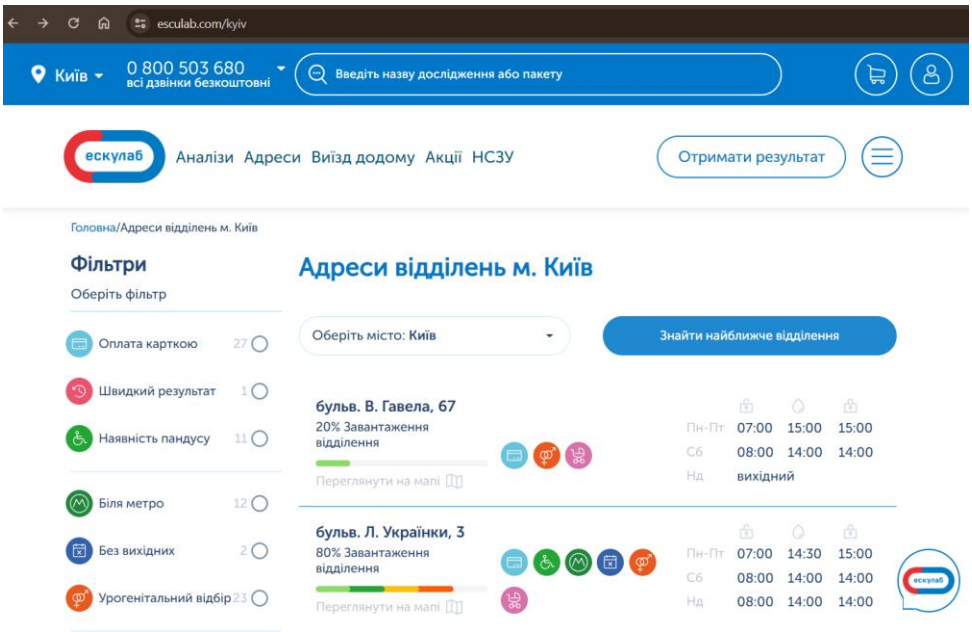


Рис. 1.5. Інтерфейс веб-додатку лабораторії «Ескулаб» [4]

Система простіша для навігації, але нажаль складно зрозуміти на якій сторінці користувач перебуває, бо немає підсвічування поточного розділу сайту.

Цікавою і корисною функцією є інформація про завантаженість відділень лабораторії. Завдяки ній клієнти можуть обрати те відділення, де наразі менше відвідувачів і здати аналізи швидше.

Проаналізувавши існуючі рішення, можемо помітити що вони характеризуються достатньо складним інтерфейсом, не адаптованим до девайсів з невеликими розмірами екрану, таких як телефон і планшет. Це стає перешкодою у зручній та ефективній взаємодії клієнтів із системою. Якщо користувачам важко користуватися вашим сайтом, вони, швидше за все, одразу ж його покинуть, і шанси на те, що вони повернуться, мізерні.

Згідно з дослідженням Global Digital 2023 [5], 92,3% користувачів інтернету заходять з мобільних пристроїв (див. рис. 1.6). Це означає що велика частина користувачів повинна мати зручний доступ до послуг лабораторії прямо зі свого телефону.

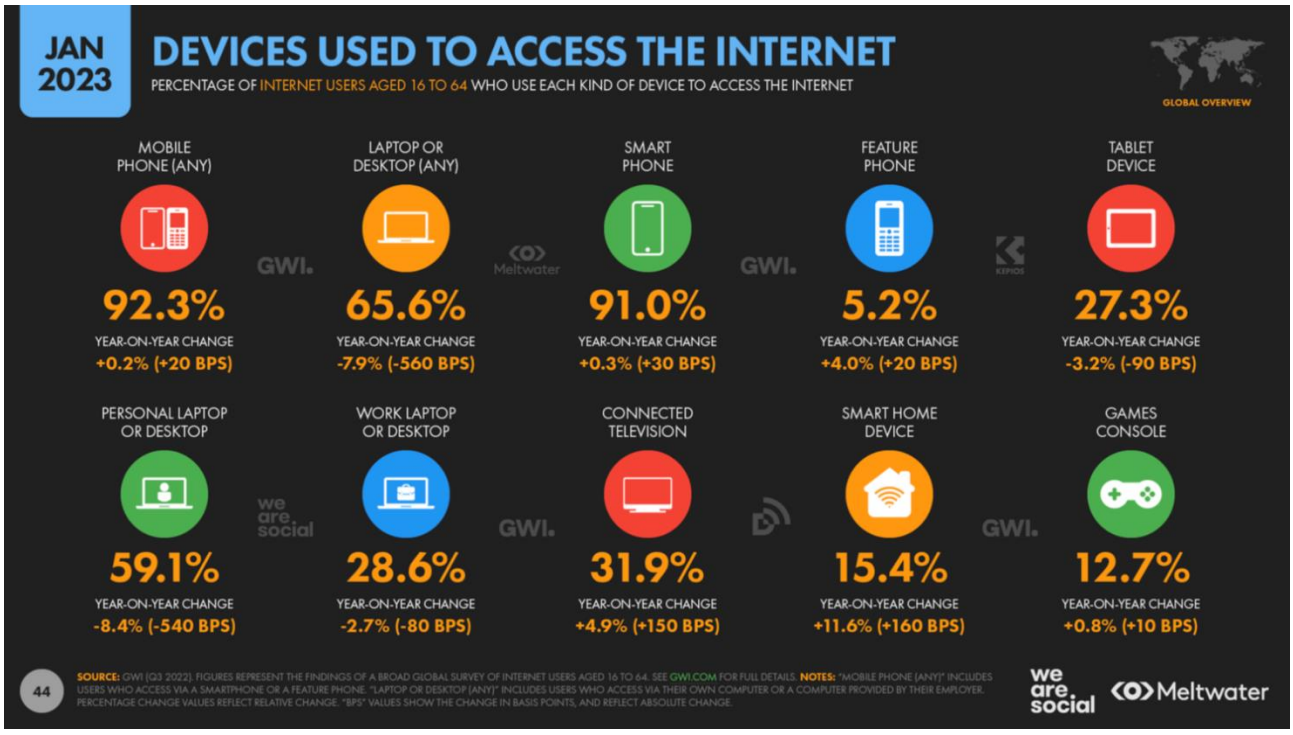


Рис. 1.6. Девайси, з яких користувачі підключаються до інтернету за результатами дослідження Global Digital 2023

Досягти цього дозволяє адаптивний дизайн — це вид верстки сайту, який враховує особливості різних типів пристроїв, щоб сайт завжди відображався правильно для користувача. Такий дизайн має однаково добре підлаштовуватися під розмір екрана ПК, телефона, планшета тощо [6]. Адаптивний дизайн не просто покращує юзабіліті — він відкриває ваш сайт для ширшої аудиторії (користувачів мобільних пристроїв). По суті, хороший адаптивний дизайн не залишає без уваги жодного користувача, незалежно від того, як він потрапив на ваш сайт [7].

Якщо веб-сайт простий у навігації, функціональний і пропонує чудовий користувацький досвід, він сприяє зміцненню довіри серед відвідувачів. Люди схильні повертатися на веб-сайти, які надали їм позитивний досвід. Це не тільки підвищує лояльність, але й збільшує ймовірність конверсії [6].

Тому основним фокусом при розробленні системи буде адаптивний та інтуїтивний інтерфейс, що буде включати всі функції, необхідні для зручного користування послугами лабораторії.

1.2. Призначення розробки та галузь застосування

Основною метою розробки комп’ютерної системи для медичних лабораторій є автоматизація взаємодії клієнтів лабораторії з її працівниками, поліпшення досвіду користування послугами лабораторії.

Комп’ютерна система виникає як реакція на потребу лабораторій у зручному інструменті з простим та інтуїтивним інтерфейсом, що сприятиме прискоренню обробки замовлень та систематизації документообігу.

Після проведення аналізу наявних рішень, було визначено, що вони характеризуються складністю використання, особливо на мобільних пристроях. Тому особливістю розроблюваної системи є простота її дизайну та адаптивність до різних розмірів екрану сучасних пристроїв.

Розроблювана система призначена для застосування у медичній галузі, зокрема в лабораторіях, де вона забезпечить ефективну автоматизацію процесів та покращить якість обслуговування пацієнтів.

1.3. Підстава для розробки

В кінці навчання, студент виконує кваліфікаційну роботу (проєкт). Тема роботи узгоджується з керівником проєкту, випускаючою кафедрою.

Підставою для розробки кваліфікаційної роботи на тему “Розробка комп’ютерної системи для медичних лабораторій” є наказ по Національному технічному університету «Дніпровська політехніка» №470-с від 23.05.2024.

1.4. Постановка завдання

Спроєктувати та розробити комп’ютерну систему для медичних лабораторій для забезпечення можливості взаємодії клієнта з працівниками лабораторії через мережу Інтернет.

Вхідні дані:

– відомості про пацієнта: прізвище, ім’я, по-батькові, дата народження, стать, email, пароль. Всі поля, окрім email та пароль, зашифровані як один документ для більшої безпеки зберігання. Пароль захешовано алгоритмом SCRYPT;

– відомості про дослідження: назва, опис, глобальний код, ціна. Зберігаються у вигляді тексту, бо це відкрита інформація.

Вихідні дані: сформована відомість про результати проведених досліджень. Зберігається у вигляді таблиць та діаграм у консолі MongoDB. Налаштування доступу до консолі відбувається на платформі MongoDB.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Комп’ютерна система для медичних лабораторій повинна мати графічний інтерфейс для багаторівневого доступу: неавторизований користувач, клієнт, співробітник, адміністратор, для кожного з яких передбачені відповідні функціональні можливості. Use Case діаграма з детальним описом функцій додатку наведена на рисунку 1.7.

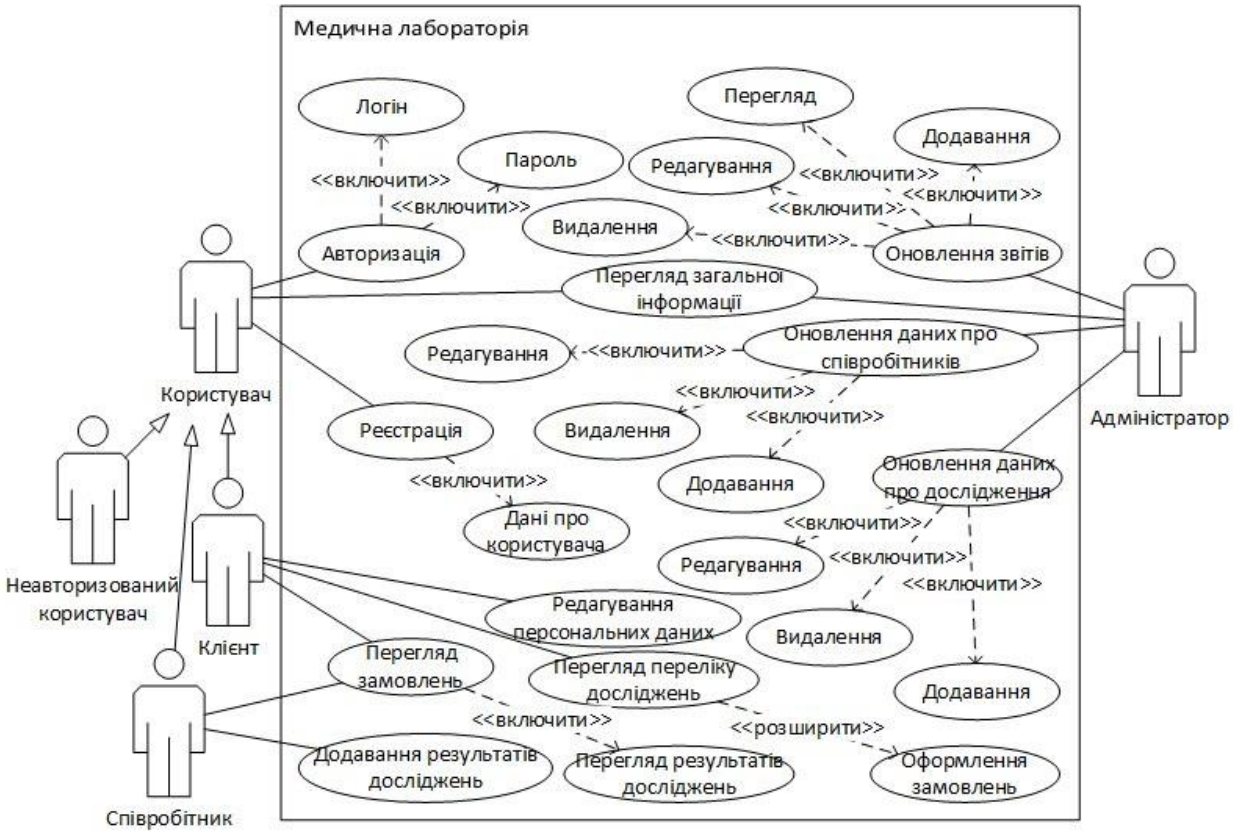


Рис. 1.7. Use Case діаграма функцій системи

Рівень «Неавторизований користувач» матиме наступні функції:

- перегляд сторінки «Головна»;
- авторизація в інформаційній системі;
- створення облікового запису клієнта.

Рівень «Клієнт» матиме наступні функції:

– перегляд сторінки «Головна»;

– авторизація в інформаційній системі;

– створення облікового запису клієнта;

– перегляд переліку досліджень, які проводяться медичною лабораторією, та інформації про них;

– вибір досліджень та створення замовлення, для зручності пошуку відповідних досліджень необхідно передбачити фільтр за назвою або кодом;

– перегляд своїх замовлень та отримання їх результатів, з можливістю сортування;

– перегляд та редагування персональної інформації.

Рівень «Співробітник» матиме наступні функції:

– перегляд сторінки «Головна»;

– авторизація в інформаційній системі;

– перегляд переліку досліджень та інформації про них;

– перегляд переліку замовлень клієнтів, з можливістю фільтрації за прізвищем клієнта;

– додавання та отримання результатів замовлень клієнтів.

Рівень «Адміністратор» матиме наступні функції:

– додавання даних щодо користувачів рівня «Співробітник» у систему;

– оновлення переліку досліджень та відомостей про них у системі;

– допомога користувачам рівня «Співробітник» та «Клієнт» при виникненні проблем з авторизацією шляхом зміни їх даних для авторизації у базі даних;

– перегляд статистичних даних та звітів з роботи лабораторії.

Інтерфейс розробленої системи повинен задовольняти таким вимогам:

а) верхній рядок навігації повинен містити посилання на сторінки сайту, що відповідають рівню доступу користувача та кнопку переходу на сторінку авторизації;

б) в центральній частині інтерфейсу системи повинен бути розташований основний контент, що залежить від обраної в рядку навігації сторінки:

1) «Головна» – інформація про медичну лабораторію;

2) «Аналізи» – перелік досліджень та інформація про них;

3) «Профіль» – інформація про клієнта та перелік його замовлень;

4) «Замовлення» – перелік замовлень клієнтів.

Інтерфейс повинен бути інтуїтивно зрозумілим будь-якому клієнту. Оформлення замовлення не повинно викликати труднощів або додаткових запитань.

1.5.2. Вимоги до інформаційної безпеки

Система буде використовуватись широким колом користувачів з відкритим доступом до мережі Інтернет.

Щоб забезпечити належний рівень інформаційної безпеки, буде впроваджена система багаторівневого доступу. Доступ поділяється на ролі:

- неавторизований користувач;
- клієнт (авторизований користувач);
- співробітник;
- адміністратор.

Інтерфейс для ролі «Адміністратор» повинен мати окрему веб-адресу ресурсу для зменшення можливості доступу неавторизованого користувача до конфіденційної інформації.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для користування розроблюваною системою потрібен девайс із встановленим браузером та можливістю користування мережею Інтернет. Це може бути:

а) персональний комп'ютер, клавіатура, миша або тачпад;

б) мобільний пристрій.

Для належного функціонування серверної частини необхідна обчислювальна машина. Її характеристики залежать від кількості запитів від клієнтського додатку.

Мінімальні вимоги до обчислювальної машини, на якій розгортається серверна частина програмного забезпечення:

- кількість вільного місця на диску – 1 ГБ;
- об’єм оперативної пам’яті – 16 ГБ;
- кількість ядер – 4;
- ОС: Linux або Microsoft Windows 10/11;
- доступ в інтернет зі швидкістю від 3 Мбіт/сек.

Також можливо розмістити серверну частину додатку на хмарному сервісі. Цей підхід називається Cloud hosting, або хмарний хостинг, і має ряд привілеїв:

- Масштабованість. Гнучкість хмарного хостингу гарантує, що ваші ресурси завжди відповідають попиту. Провайдер хмарного хостингу може виділити більше або менше ресурсів, коли вони потрібні;
- Доступність. Багато провайдерів хмарного хостингу мають центри обробки даних по всьому світу, що зменшує затримку та підвищує доступність. Вони мають додаткові механізми відмов для захисту своїх служб;
- Економічна ефективність. Найбільша вартість локального хостингу – це інвестиції в нове обладнання та необхідну інфраструктуру. Навіть після цих початкових витрат вам все одно доведеться платити за поточне обслуговування, яке може бути дорогим. Хмарний хостинг пропонує систему оплати за використані ресурси та усуває потребу у витратах на обслуговування;
- Безпека. Хмарні постачальники інвестують значні кошти у надійні системи безпеки та пропонують високий рівень безпеки для всіх своїх клієнтів. Постачальник оновлює системи, щоб вони завжди відповідали найновішим вимогам безпеки [8].

Трьома провідними хмарними провайдерами за даними Statista [9] є Amazon Web Services (AWS), Microsoft Azure і Google Cloud Platform (GCP).

При виборі конкретного провайдера необхідно врахувати такі фактори:

- регіони хостингу, які необхідно підтримувати;
- вимоги до безпеки і захисту даних;
- подальші плани розвитку інфраструктури комп'ютерної системи;
- інтеграція з поточними системами;
- ціна послуг.

Зважаючи на те що розроблювана система не потребує специфічних нативних хмарних рішень, а тільки віртуальні машини, є можливість вибору будь-якого хмарного провайдера з розміщенням серверів на території України або Європи. Плюсом, але не обов'язковою умовою, буде інтеграція з MongoDB. Перелік та порівняння хмарних сервісів наведено на ресурсі [10].

При розміщенні серверної частини на хмарних сервісах необхідно обирати план послуг відштовхуючись від мінімальних вимог до системи описаних вище. Ці вимоги будуть застосовуватись для віртуальних машин, на яких буде розміщуватись серверна частина програмного забезпечення. Також необхідно налаштувати масштабування щоб кількість ресурсів, виділених для програми змінювалась залежно від кількості запитів на сервер.

1.5.4. Вимоги до інформаційної та програмної сумісності

Система буде складатись з клієнтської та серверної частини.

Для функціонування клієнтської частини – веб-інтерфейсу – необхідно використати такі технології:

- HTML;
- CSS and SCSS;
- TypeScript;
- Angular.

Серверна частина буде реалізована на мові TypeScript з використанням середовища виконання Node.js та фреймворку express.js. База даних MongoDB буде розміщена з використанням хмарних технологій.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

На основі аналізу існуючих рішень на ринку, сформулюємо функціональне призначення програми. Воно полягає у наданні клієнтам та співробтникам лабораторії зручного та інтуїтивно-зрозумілого інтерфейсу, що має такі функції:

– відображення переліку послуг. Надання повного та актуального переліку послуг лабораторії, включаючи різноманітні види медичних аналізів та досліджень з їх описом;

– автоматизація замовлення послуг. Забезпечення можливості клієнтам швидко та зручно замовляти необхідні послуги через інтерфейс програми;

– відображення інформації про лабораторію. Надання інформації про місце розташування та розклад роботи лабораторії, ознайомлення клієнтів з перевагами лабораторії у порівнянні з конкурентами;

– автоматизація документообігу та надання статистики з роботи лабораторії. Спрощення документообігу та надання співробітникам інструменти для генерування та перегляду статистичних звітів щодо діяльності лабораторії;

– отримання результатів досліджень. Забезпечення можливості клієнтам-отримувати результати досліджень онлайн з будь-якої точки світу.

2.2. Опис застосованих математичних методів

При написані даної кваліфікаційної бакалаврської роботи використовувалися наступні математичні методи:

- чисельні методи;
- статистичний аналіз ціноутворення замовлень;
- методи булевої алгебри.

2.3. Опис використаних технологій та мов програмування

Для розробки комп’ютерної системи для медичних лабораторій застосовуються такі технології та засоби розробки:

– HTML;
– SASS;
– TypeScript;
– Angular;
– Node.js;
– Express;
– MongoDB;
– Webstorm.

HTML та SASS використовуються у проєкті для створення клієнтського інтерфейсу.

HTML – стандартизована мова розмітки веб-сторінок. Код HTML інтерпретується браузерами, отримана в результаті інтерпретації сторінка відображається на екрані монітора комп'ютера або мобільного пристрою [11].

SASS (Syntactically Awesome Stylesheets) – препроцесор, що під час компіляції проєкту перетворюється на CSS – мову опису стилізації елементів HTML.

SASS дозволяє використовувати функції недоступні в самому CSS, наприклад, змінні, вкладеності, міксини, успадкування та ін., які повертають зручність написання CSS [12].

Для обробки дій користувача на сторінці були використані фреймворк Angular та мова програмування TypeScript.

JavaScript – динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript.

TypeScript – мова програмування, що є типізованим розширенням мови JavaScript. Головна перевага TypeScript полягає в тому, що він може виділити несподівану поведінку у коді, знижуючи ймовірність помилок.

Angular – це платформа для розробки додатків та створення ефективних та складних односторінкових програм [13]. Завдяки цій платформі відбувається інтеракція користувача з системою без перезавантаження браузера при переході між сторінками системи. Це пришвидшує роботу системи і покращує досвід користувача.

Для взаємодії з користувачем використовуються різноманітні кнопки, поля для вводу, панелі меню та інші компоненти. Щоб не створювати їх власноруч та мати усі елементи керування в одному стилі, було використано бібліотеку Angular Material. Це бібліотека компонентів, створена командою Angular для легкої інтеграції з Angular [14].

Для створення бекенд-частини системи були використані платформа Node.js, мова програмування TypeScript і фреймворк Express. Для зберігання даних було обрано базу даних MongoDB.

Node.js надає можливості для створення сервер-частини системи, що забезпечує доступ до бази даних та передачу даних на клієнт (машину, на якій виконується програма).

Express – це гнучкий веб-фреймворк для додатків Node.js, що надає великий набір функцій для мобільних і веб-додатків.

Маючи в своєму розпорядженні безліч службових методів HTTP і проміжних оброблювачів, він здатен створити надійний програмний інтерфейс додатку – API – швидко і легко.

Окрім цього Express використовується для роботи з базами даних [15]. При розробці веб-додатків, для зберігання даних досить часто обирають документо-орієнтовані бази даних, адже вони мають гнучку структуру і непогану сумісність – документ бази даних, при обробці за допомогою Express, співвідноситься до JSON-об’єкту, який широко використовується у веб-розробці.

MongoDB – це документо-орієнтована база даних із необхідною масштабованістю та гнучкістю, із запитами та індексацією.

MongoDB зберігає дані у гнучких документах, схожих на JSON, тобто поля можуть відрізнятися від документа до документа, а структура даних може

змінюватися з часом.

Однією з функцій розроблюваної системи є зберігання файлів. Для цього використовується технологія GridFS. GridFS – це специфікація, визначена в базі даних MongoDB для зберігання файлів.

GridFS розбиває великі файли на частини. Ці частини зберігаються в одну колекцію (fs.chunks), а метадані про файл в іншу (fs.files). Коли відбувається запит до файлу, GridFS робить запит в колекцію з частинами файлу і потім повертає файл цілком [16].

Середовищем розробки було обрано IDE Webstorm від компанії JetBrains, що має безліч інструментів для роботи, як з клієнтською, так і серверною частиною застосунку, а тому підходить для розробки веб-застосувань. Окрім того, Webstorm має підтримку систем контролю версій та ведення локальної історії, що допомагають контролювати етапи розробки та створювати невеликі, але повністю робочі частини функціоналу [17].

2.4. Опис структури програми та алгоритмів її функціонування

Узагальнену схему роботи системи наведено на рисунку 2.1.

Рис. 2.1. Узагальнена схема роботи програми

Взаємодія клієнтського інтерфейсу з сервером відбувається за допомогою АПІ-запитів через протокол HTTP. HTTP відповідає класичній моделі клієнт-сервер, коли клієнт відкриває з’єднання, щоб зробити запит, а потім чекає, поки не отримає відповідь. HTTP є протоколом без стану, тобто сервер не зберігає жодних даних між двома запитами [18]. Вся інформація, необхідна для отримання відповіді, передається в рамках одного запиту.

Для комунікації «Сервер – База даних» використовується спеціальний протокол «mongodb», що працює через TCP/IP. Він оптимізований для операцій з базою даних: має більш гнучку конфігурацію, порівняно з HTTP. А бібліотека mongoose забезпечує підтримання активних з'єднань з БД і повторні спроби запитів при помилках. Це збільшує швидкість та надійність отримання даних.

База даних є важливою складовою системи. Вона забезпечує централізоване, ефективне та безпечне зберігання і управління великими обсягами даних, що дозволяє підтримувати цілісність, продуктивність і масштабованість. Вона також надає механізми для аналітики, резервного копіювання, відмовостійкості та контролю доступу.

Щоб зобразити концептуальну схему БД, використаємо узагальнені конструкції блоків ER-моделі (див. рис. 2.2).

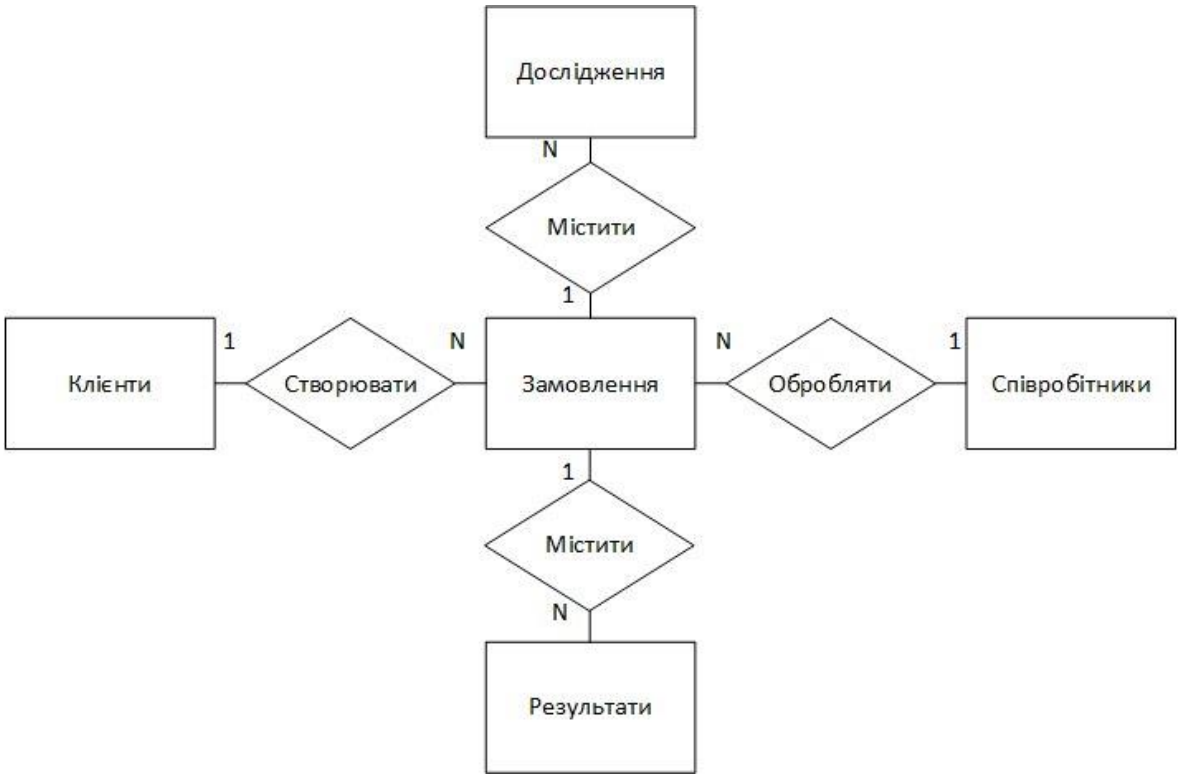


Рис. 2.2. ER-діаграма бази даних

Основні переваги ER-моделей:

– є легкими для перетворення в схему БД;

– надають можливість моделювання реальних об'єктів, що робить їх дуже корисними;

– не вимагають технічних знань та апаратної підтримки;

– прості для розуміння і створення;

– забезпечують стандартне рішення для логічної візуалізації даних [19].

На основі ER-діаграми можна зобразити схему БД (див. рис. 2.3).

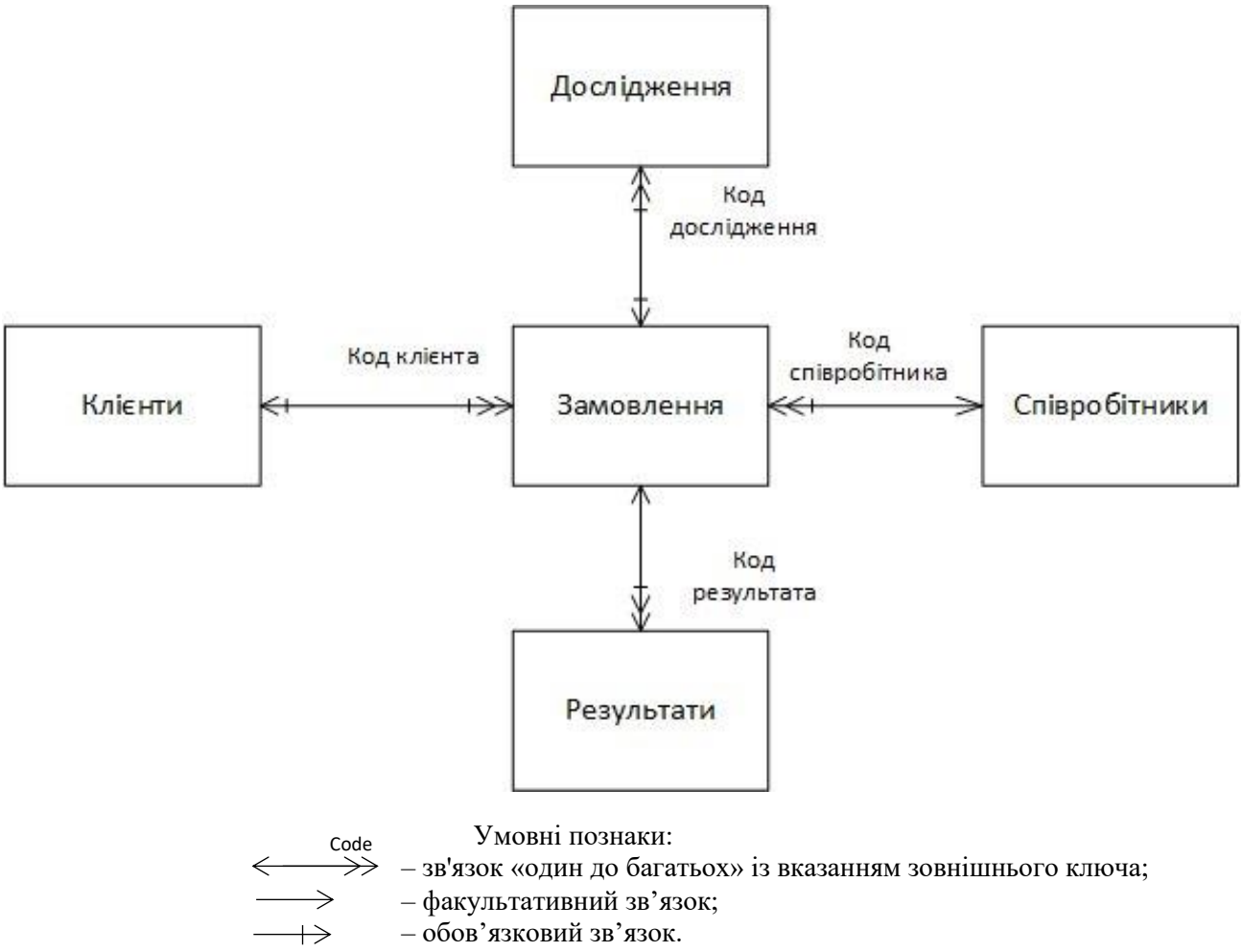


Рис. 2.3. Схема бази даних системи

База даних MongoDB представлена у вигляді фізичного сховища колекцій. Кожна БД має свій власний набір файлів в файлову систему.

Колекція – це група документів MongoDB, є еквівалентом простої таблиці в реляційній базі даних.

Документ – це набір пар "ключ – значення". Документ має динамічну схему. Це означає, що документ в одній і тій же колекції не зобов'язаний мати один однаковий набір полів або структуру, а загальні поля в колекції можуть мати різні типи даних.

Структура бази даних в MongoDB є гнучкою і не має схем, що дозволяє динамічно змінюватись і адаптуватись по мірі розвитку системи [16].

Співставимо сутності ER-діаграми з колекціями БД, зв’язки ER-діаграми зі зв’язками бази даних.

База даних системи складається з 5 колекцій:

– users (Клієнти);
– employees (Співробітники);
– orders (Замовлення);
– analyzes (Дослідження);
– results (Результати).

Зв’язки між колекціями БД:

– «Клієнти» – «Замовлення»: тип зв’язку – один до багатьох; для зв’язку документів в колекції «Замовлення» міститься поле «userId», що містить первинний ключ колекції «Клієнти»;

– «Співробітники» – «Замовлення»: тип зв’язку – один до багатьох; для зв’язку документів в колекції «Замовлення» міститься поле «employeeId», що містить первинний ключ колекції «Співробітники»;

– «Замовлення» – «Дослідження»: тип зв’язку – один до багатьох; для зв’язку документів в колекції «Замовлення» міститься поле «analyzes», що містить масив первинних ключів колекції «Дослідження»;

– «Замовлення» – «Результати»: тип зв’язку – один до багатьох; для зв’язку документів в колекції «Замовлення» міститься поле «results», що містить масив первинних ключів колекції «Результати».

Однією з основних задач при розробці системи було створення інтуїтивного інтерфейсу для спрощення її використання. Нижче наведено блок-схеми алгоритмів головних сценаріїв використання системи:

– фільтрація досліджень (див. рис 2.4);
– реєстрація користувача (див. рис 2.5);
– створення замовлення на проведення досліджень (див. рис 2.6);
– додавання результатів замовлення (див. рис 2.7).

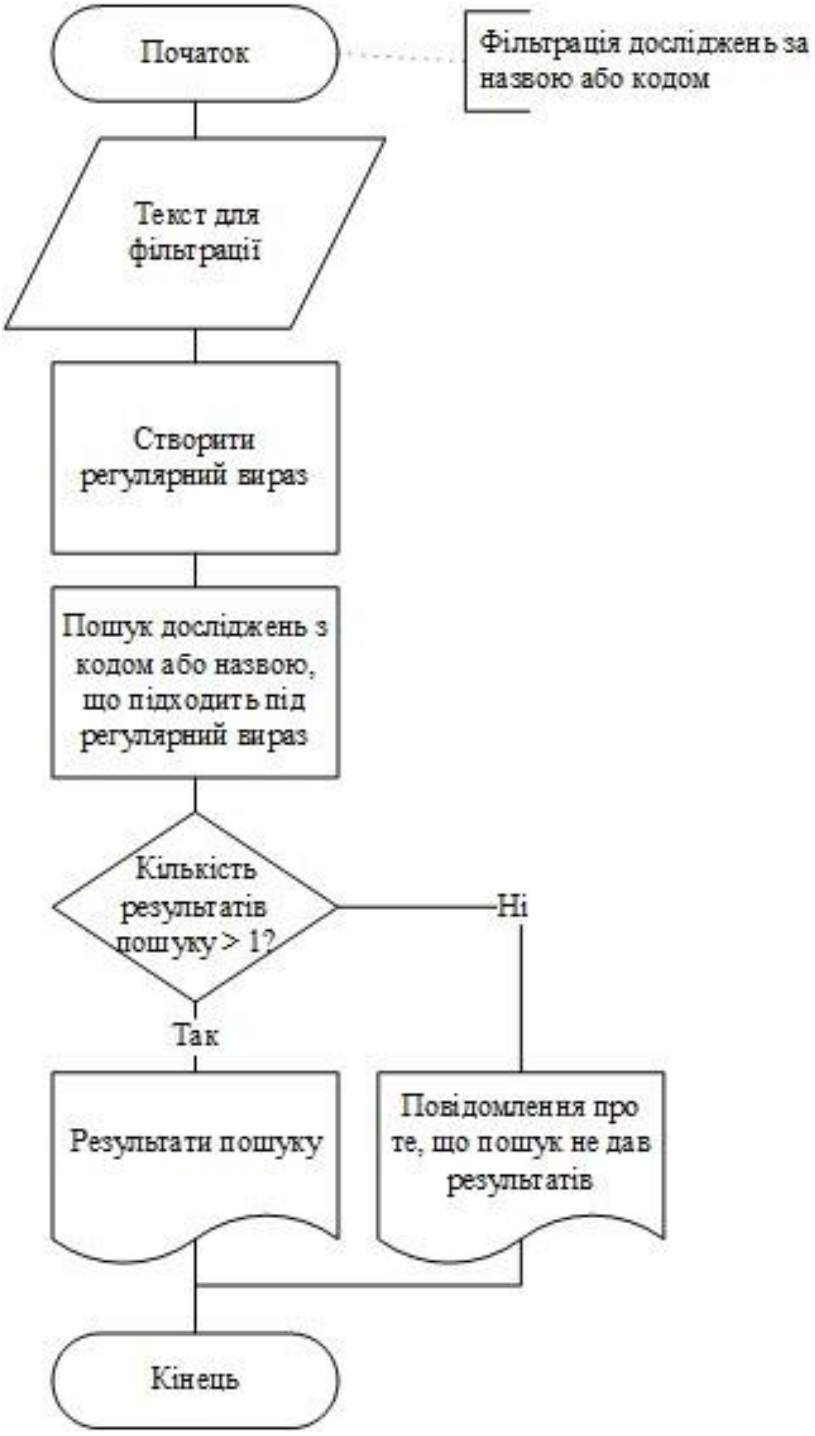


Рис. 2.4. Блок-схема алгоритму фільтрації досліджень

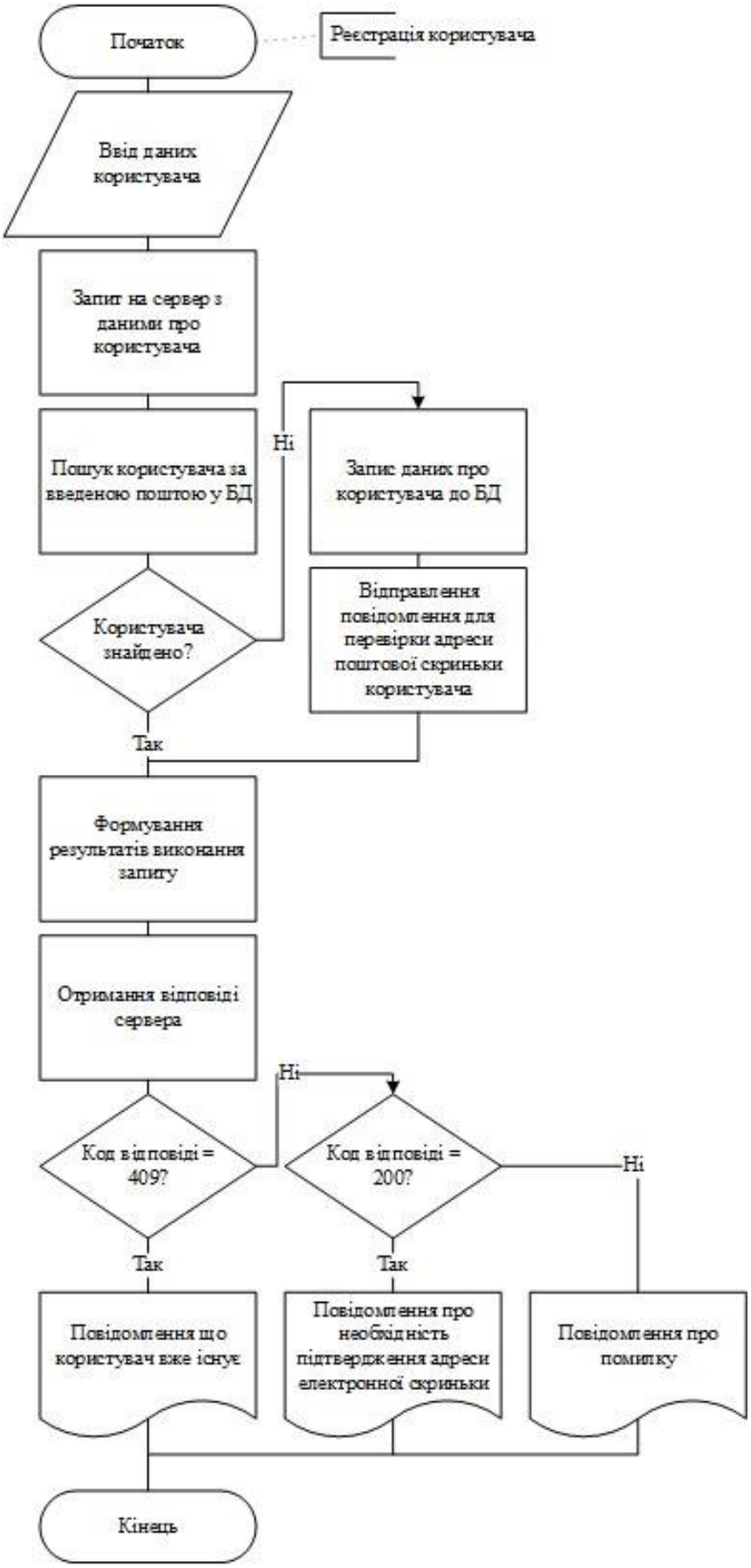


Рис. 2.5. Блок-схема алгоритму реєстрації користувача

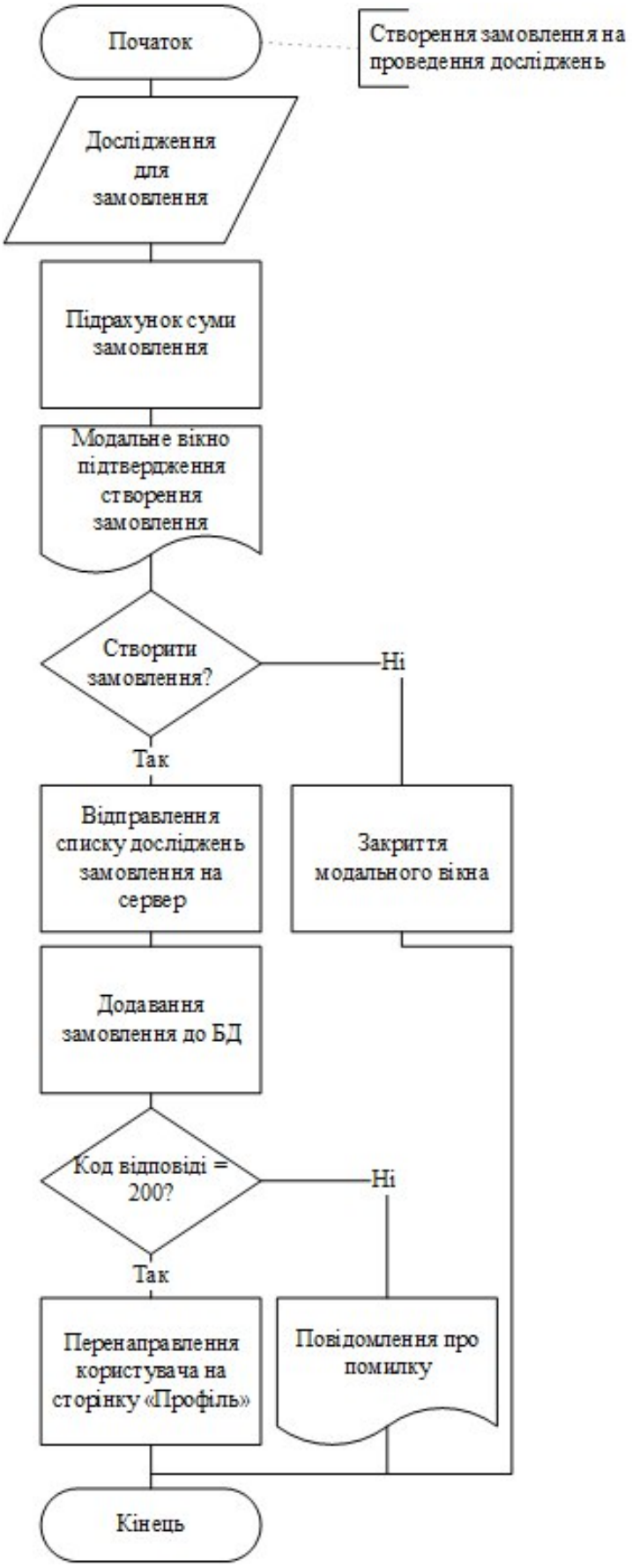


Рис. 2.6. Блок-схема алгоритму створення замовлення на проведення досліджень

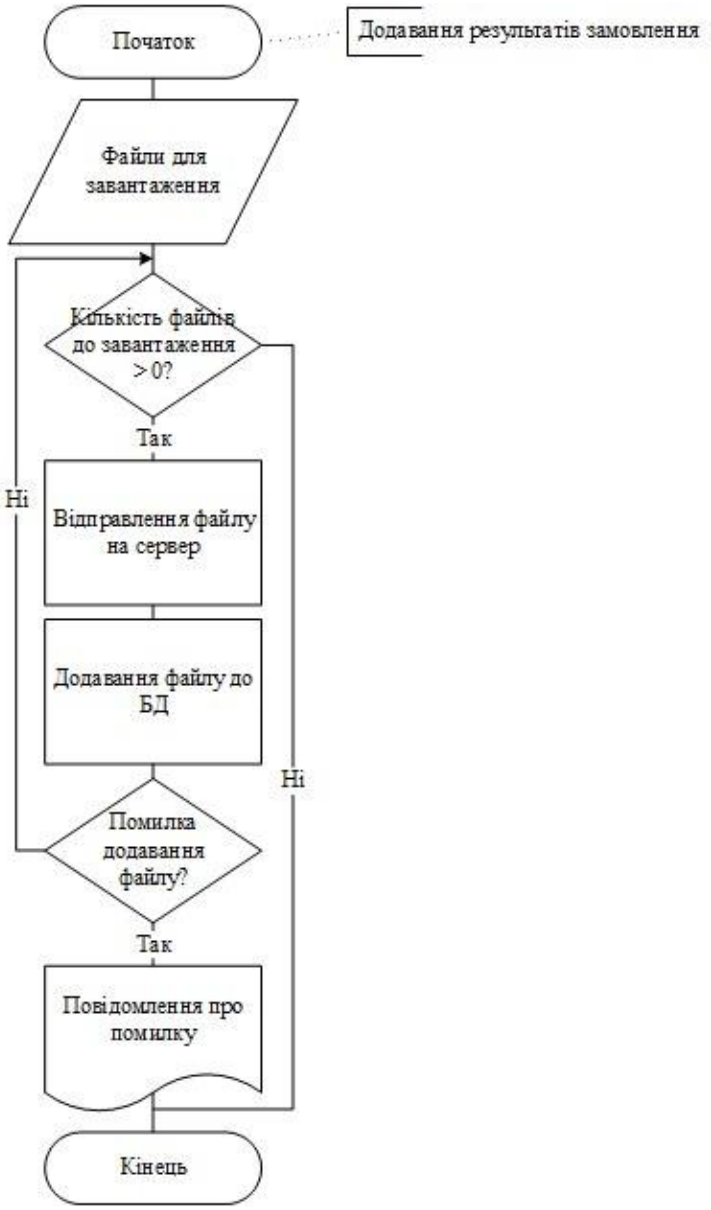


Рис. 2.7. Блок-схема алгоритму додавання результатів замовлення

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними у створеній системі є ті, що вводяться користувачем через графічний інтерфейс:

– відомості про пацієнта: прізвище, ім’я, по-батькові, дата народження, стать, адреса електронної пошти, пароль;

– відомості про дослідження: назва, опис, унікальний код, ціна;

– файли результатів проведених досліджень.

Вихідними даними є відомості та візуалізовані графіки:

– сформована відомість про результати проведених досліджень;

– графік результатів діяльності лабораторії протягом року.

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Розроблена система складається з веб-інтерфейсу, серверу та бази даних, розміщеної в хмарному сховищі. Вона виконується у веб-браузері на локальному сервері та потребує мінімальних ресурсів ПК. Для роботи системи необхідні такі технічні засоби:

– персональний комп’ютер;

– клавіатура;

– миша або тачпад.

2.6.2. Використані програмні засоби

Для розробки системи були використані програмні засоби стеку MEAN у поєднанні з мовою програмування TypeScript та хмарними технологіями.

MEAN стек – це фреймворк на основі JavaScript для розробки масштабованих веб-додатків. Термін MEAN є абревіатурою від MongoDB, Express, Angular та Node – чотирьох ключових технологій, що складають шари технологічного стеку.

– MongoDB: NoSQL, об'єктно-орієнтована база даних, призначена для використання з хмарними додатками.

– Express(.js): Фреймворк для веб-додатків на Node(.js), який підтримує взаємодію між фронтендом (наприклад, клієнтською стороною) та базою даних.

– Angular(.js): Часто називають "фронтендом"; клієнтський JavaScript-фреймворк, що використовується для створення динамічних веб-додатків з інтерактивними інтерфейсами користувача.

– Node(.js): Прем'єрний JavaScript веб-сервер, що використовується для побудови масштабованих мережевих додатків [18].

База даних MongoDB розміщена на хмарному сервісі, тому для роботи системи необхідне стабільне підключення до мережі Інтернет.

Для роботи клієнтської частини системи необхідний будь-який браузер, що працює з операційною системою (Linux, Windows, MacOS, Android), встановленою на девайсі, з якого відбувається доступ до програми.

У процесі розробки використовувався редактор коду Webstorm від компанії JetBrains.

2.6.3. Виклик та завантаження програми

Для запуску програми необхідно в консолі терміналу виконати команди:

1) в корневій папці med-lab:

а) npm i – для встановлення пакетів, необхідних для роботи програми;

б) npm start – для запуску клієнтської частини. Після виконання цієї команди, в браузері на сторінці http://localhost:4200/ відкриється інтерфейс клієнтської частини, через який можна користуватись програмою.

2) в папці server:

а) npm i - для встановлення пакетів, необхідних для роботи програми;

б) npm run start:server – для запуску серверної частини.

2.6.4. Опис інтерфейсу користувача

Авторкою цієї бакалаврської роботи було розроблено систему для медичних лабораторій, що складається з клієнтського інтерфейсу та серверної частини; а також використано можливості веб-консолі MongoDB.

Розглянемо функціонал клієнтського інтерфейсу та веб-консолі MongoDB.

Робота з комп’ютерною системою починається з Головної сторінки (див. рис. 2.8).

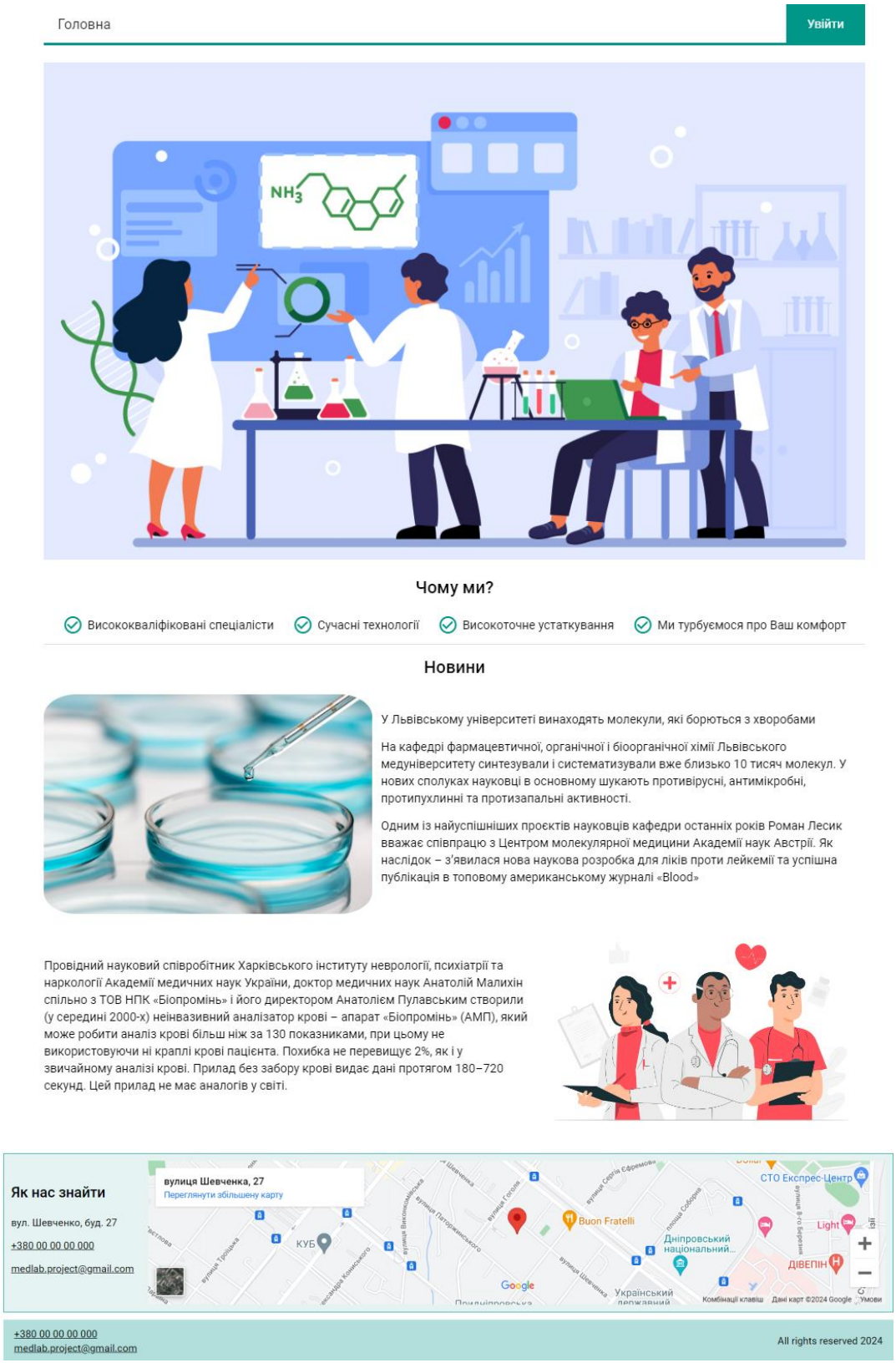


Рис. 2.8. Головна сторінка комп'ютерної системи для медичних лабораторій

Головна сторінка містить:

– шапку, що містить навігаційне меню та кнопку «Увійти»;

– банер лабораторії;

– секцію привілеїв та особливостей лабораторії;

– секцію новин;

– секцію контактних даних, що містить телефон, електронну та фізичну адреси лабораторії, інтерактивну карту з місцем розташування лабораторії;

– футер, що містить основні контактні дані.

Система має багаторівневий режим доступу: неавторизований користувач, адміністратор, клієнт та співробітник, для кожного з яких передбачені відповідні функціональні можливості. Розглянемо їх детально.

Неавторизований користувач може переглядати тільки головну сторінку додатку. Щоб авторизуватись у системі, йому необхідно натиснути на кнопку «Увійти» у шапці інтерфейсу, відкриється сторінка авторизації, яка зображена на рисунку 2.9.

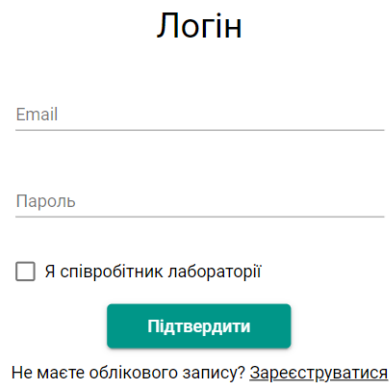


Рис. 2.9. Сторінка «Логін» для авторизації користувача

Для авторизації користувачеві необхідно заповнити поля: «Email» та «Пароль». А також обрати роль користувача «Клієнт» або «Співробітник», з якою він хоче увійти в систему, за допомогою прапорця «Я співробітник лабораторії». Якщо обліковий запис користувача не знайдено у системі або

пароль невірний, буде виведено інформаційне повідомлення, яке зображене на рисунку 2.10.

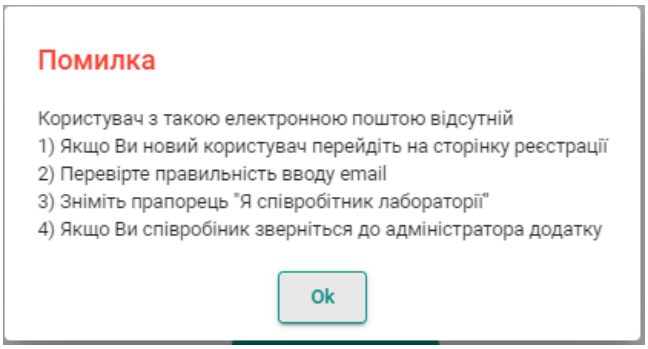


Рис. 2.10. Модальне вікно з повідомленням про помилку авторизації користувача у системі

Зі сторінки «Логін» користувач може повернутись на головну сторінку, натиснувши на кнопку «На головну», або перейти на сторінку реєстрації за посиланням «Зареєструватися», що знаходиться внизу сторінки «Логін». Сторінка реєстрації зображена на рисунку 2.11.

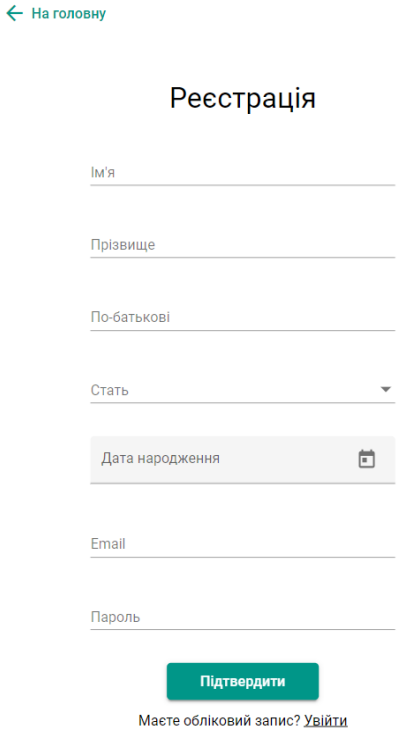


Рис. 2.11. Сторінка «Реєстрація» для реєстрації користувача у системі

Для реєстрації користувачеві необхідно заповнити усі поля на сторінці «Реєстрація», що включають: «Ім’я», «Прізвище», «По-батькові», «Стать», «Дата народження», «Email» та «Пароль». Всі ці поля обов’язкові для заповнення, деякі мають правила валідації, при недотриманні яких поле буде мати червоне обрамлення, а під ним буде виведене відповідне повідомлення про помилку. Приклад валідації полів вікна для реєстрації користувача зображено на рисунку 2.12.

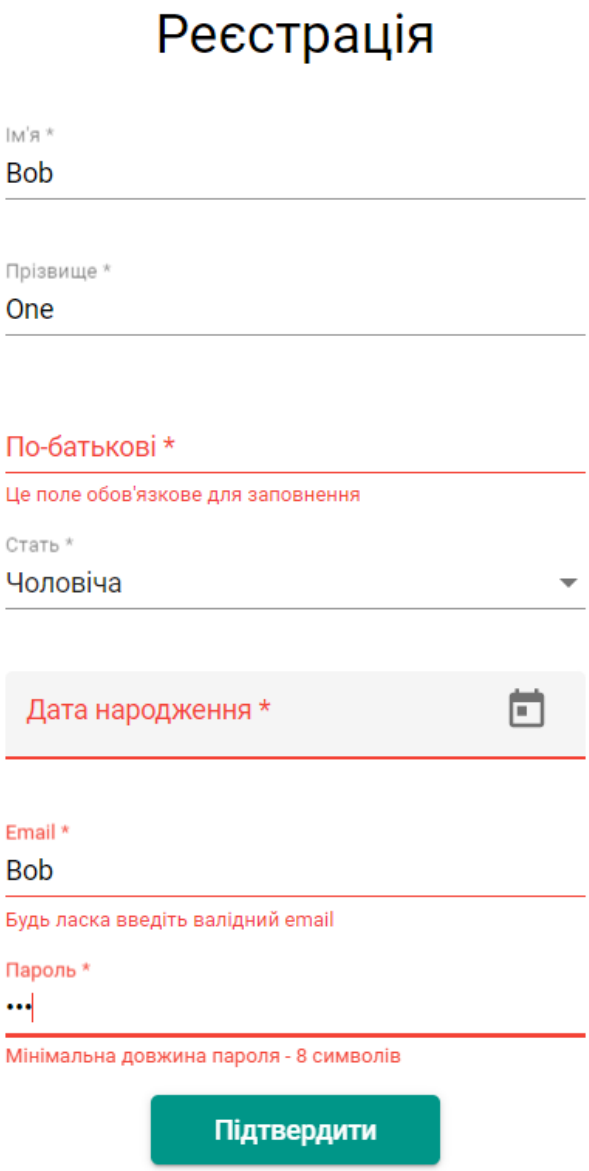


Рис. 2.12. Сторінка «Реєстрація». Приклад валідації полів

Після натискання на кнопку у вікні реєстрації «Підтвердити» користувачу буде виведено модальне вікно з повідомленням про необхідність перевірки вхідних листів в електронній скриньці, яке зображено на рисунку 2.13.

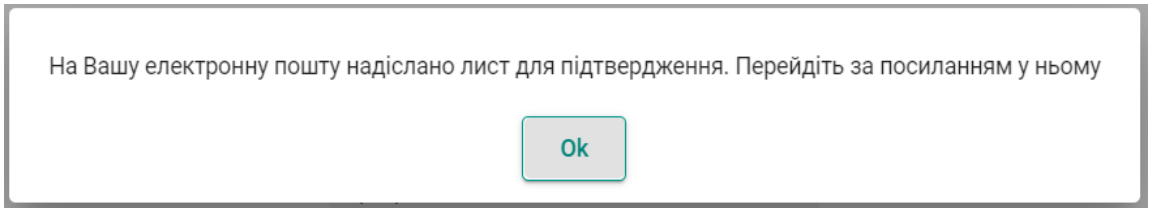


Рис. 2.13. Інформаційне модальне вікно програми щодо необхідності перевірки електронної скриньки користувача

Лист для підтвердження адреси електронної скриньки клієнта зображений на рисунку 2.14. Для продовження роботи, користувачу необхідно натиснути на посилання у ньому.

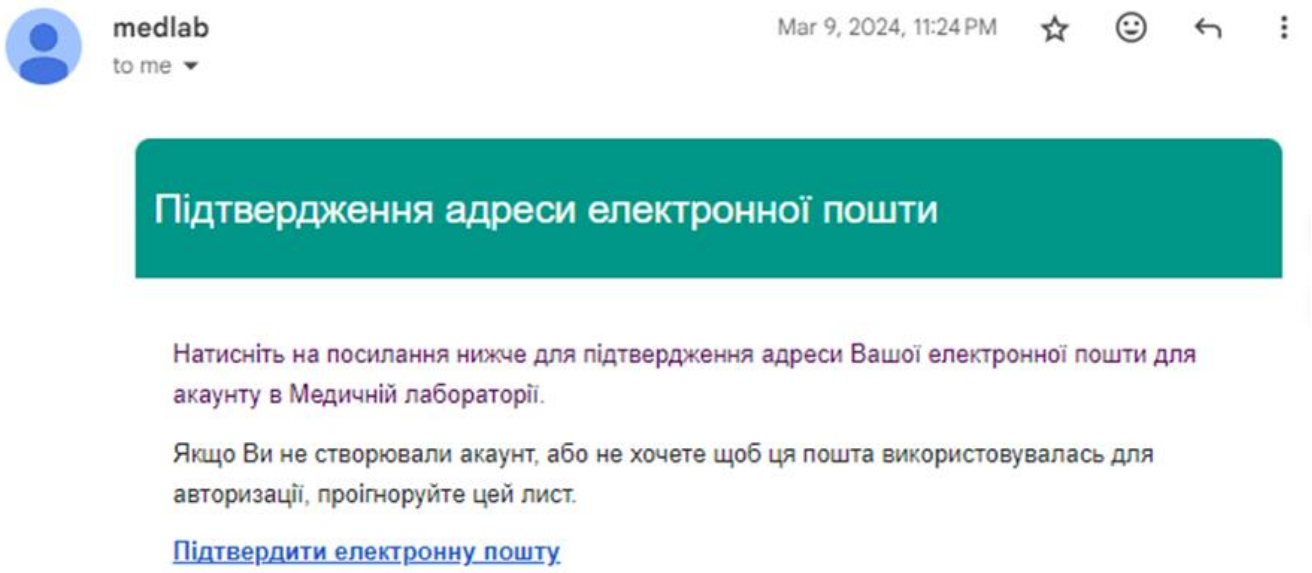


Рис. 2.14. Лист в електронній скриньці клієнта з посиланням для підтвердження адреси цієї скриньки

Після натискання на посилання у листі від комп'ютерної системи лабораторії клієнт перенаправляється на головну сторінку програми, що

зображена на рисунку 2.8, звідки за допомогою панелі навігації він може перейти на сторінку переліку досліджень, що зображена на рисунку 2.15.

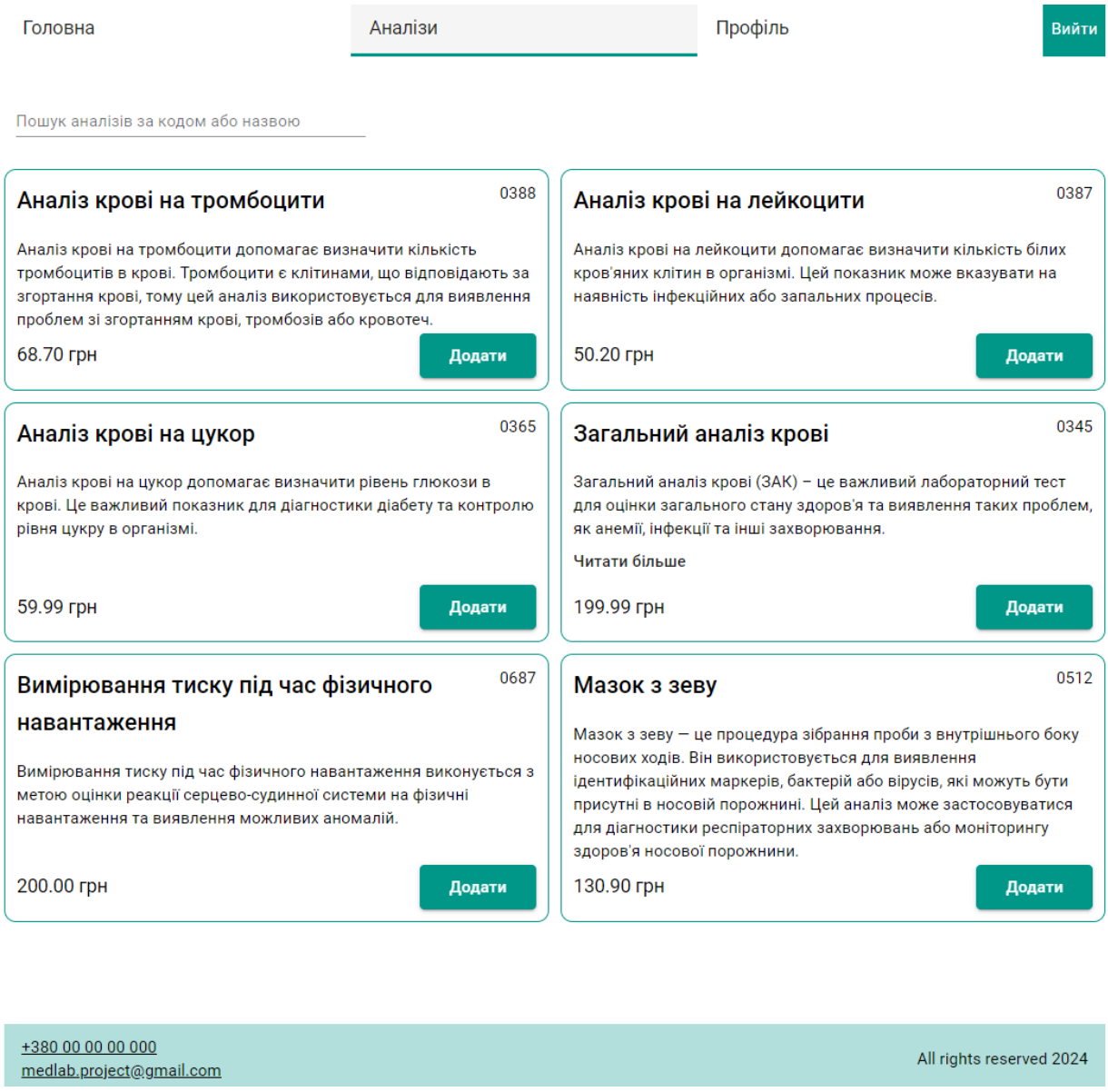


Рис. 2.15. Сторінка «Аналізи» програми для медичних лабораторій з переліком досліджень

На цій сторінці клієнт може переглядати перелік досліджень, для зручності перегляду використовувати пошук за назвою або загальним номером дослідження, а також робити їх вибір для створення замовлення. Приклад фільтрації досліджень за назвою наведено на рисунку 2.16, а приклад вибору наведено на рисунку 2.17.

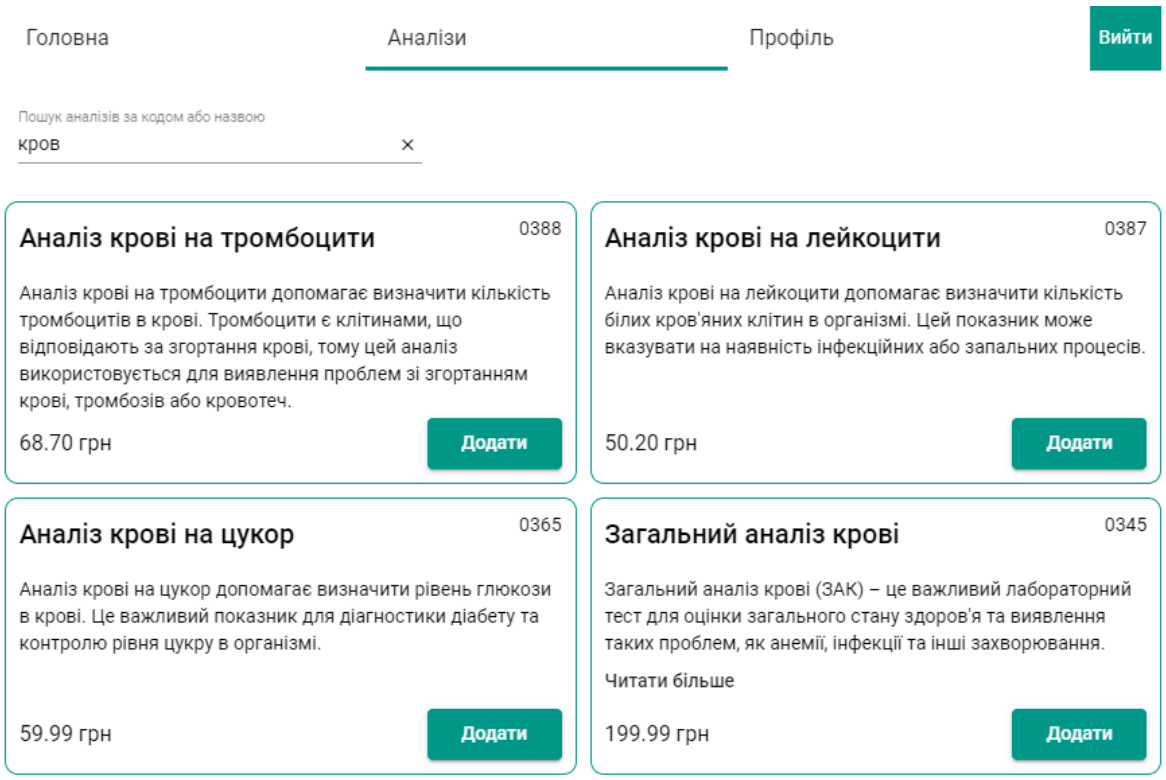


Рис. 2.16. Приклад фільтрації досліджень за назвою на сторінці «Аналізи»

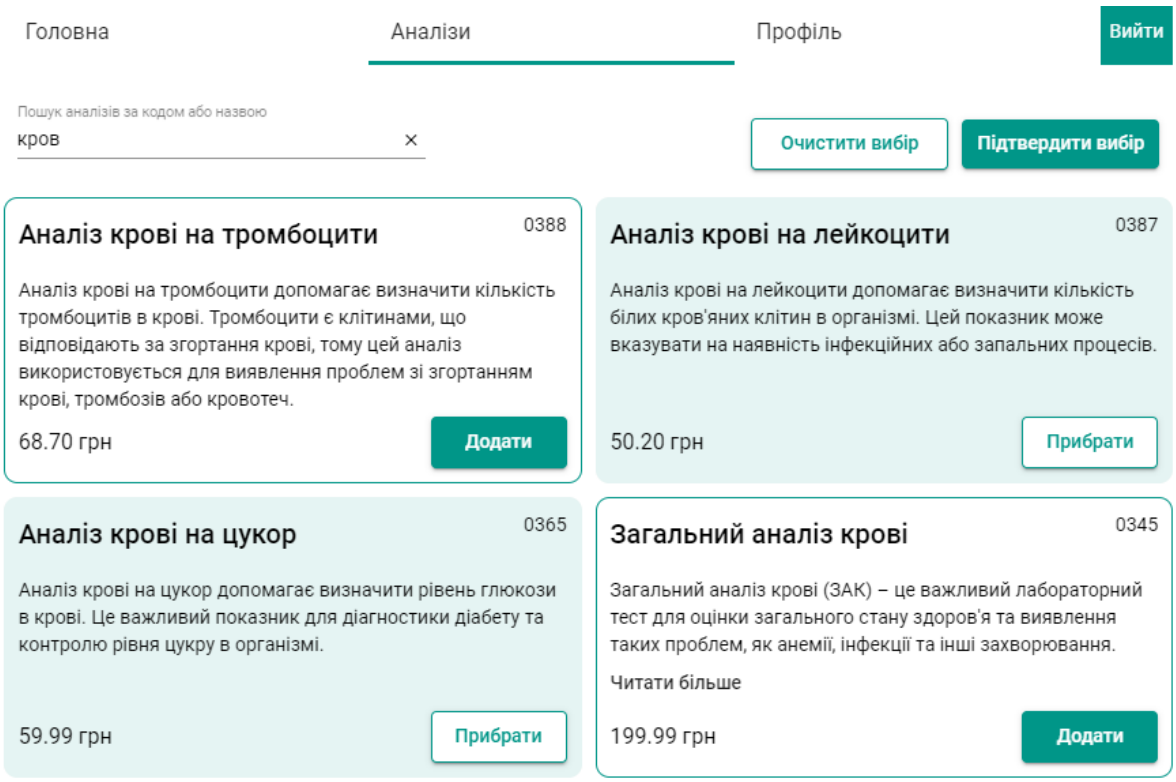


Рис. 2.17. Процес додавання досліджень до замовлення

Після натискання на кнопку «Підтвердити вибір» відкривається модальне вікно зі списком обраних досліджень, зображене на рисунку 2.18.

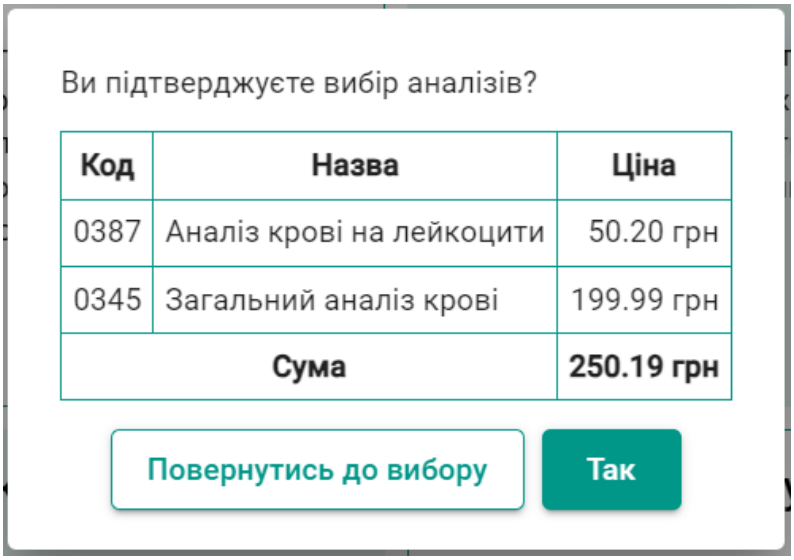


Рис. 2.18. Модальне вікно для підтвердження створення замовлення

Після натискання на кнопку «Так» у цьому вікні, відбувається перенаправлення користувача на сторінку «Профіль», що зображена на рисунках 2.19 та 2.20.

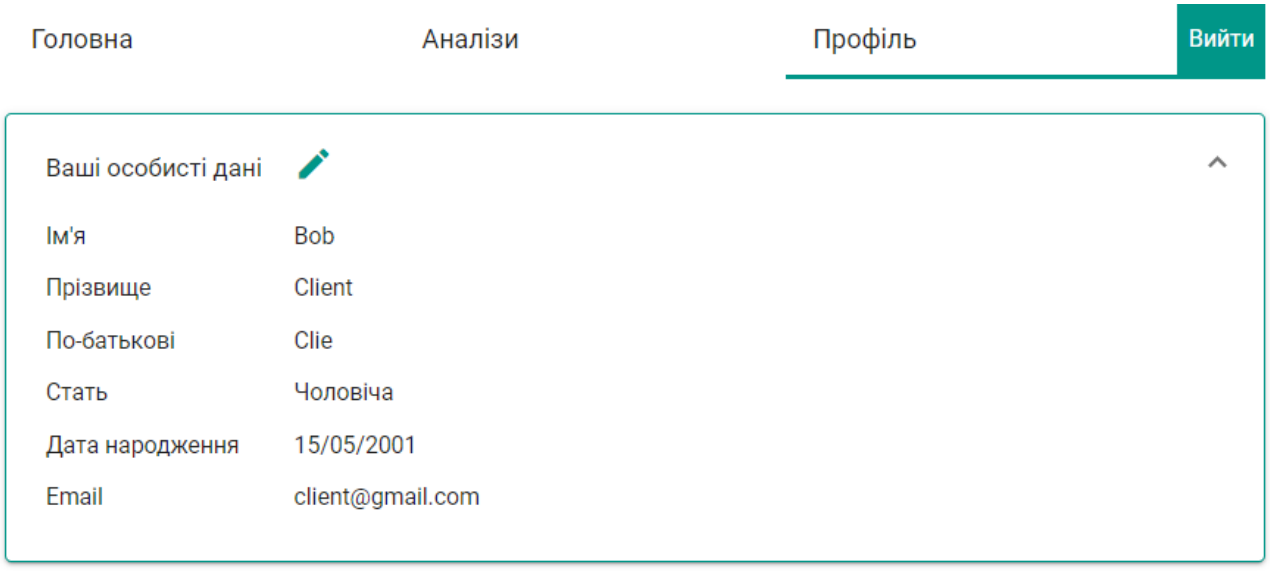


Рис. 2.19. Сторінка «Профіль». Частина 1 – інформація про клієнта

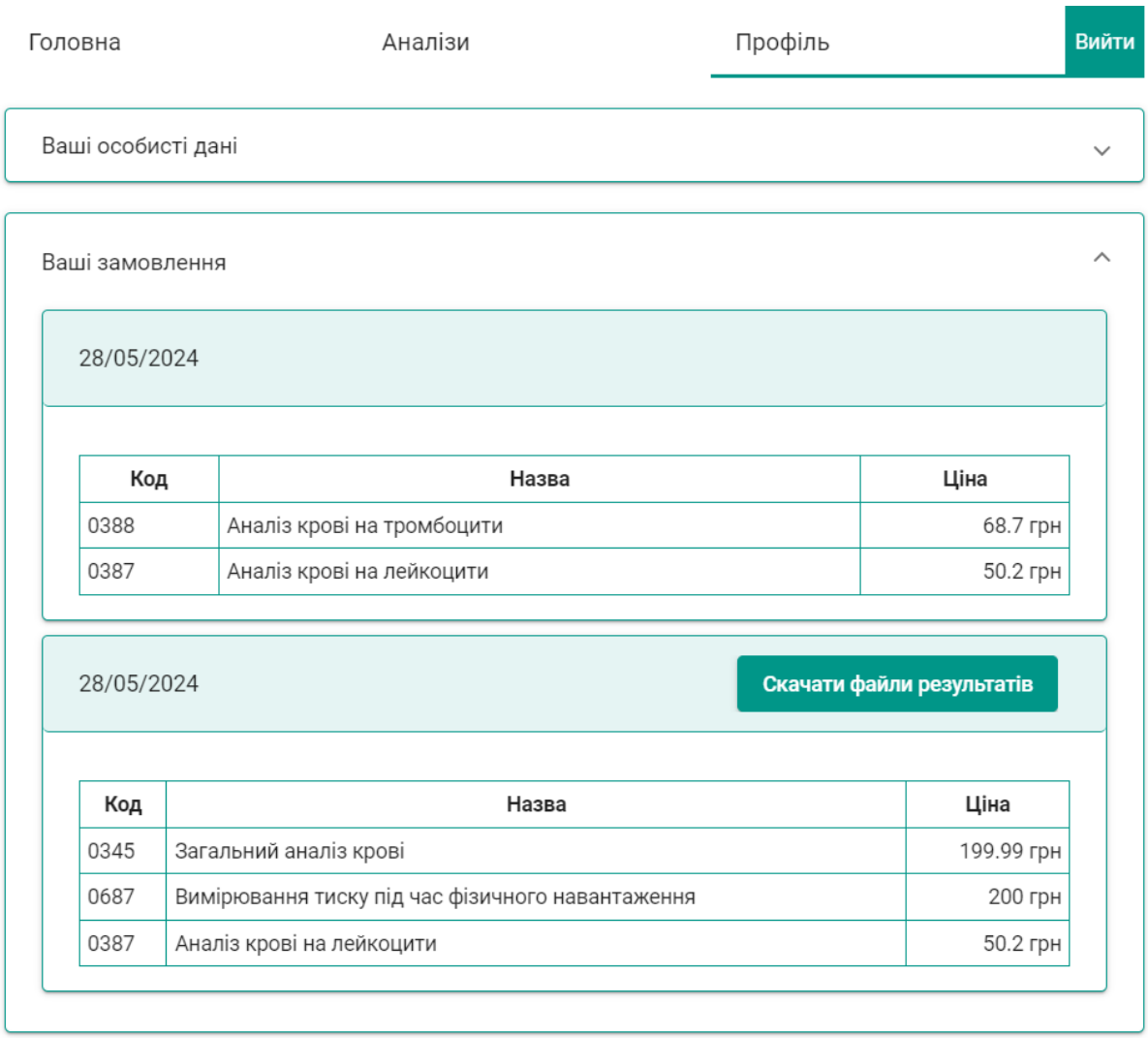


Рис. 2.20. Сторінка «Профіль». Частина 2 – замовлення клієнта

Для перегляду результатів досліджень клієнт може натиснути кнопку «Завантажити результати» на панелі дослідження, після чого розпочнеться завантаження файлів результатів на девайс користувача, де їх можна переглянути та/або надрукувати за допомогою програми для роботи з файлами формату *.pdf або браузеру.

Розглянемо функції, що може виконувати співробітник під час роботи з комп'ютерною системою.

Після авторизації у системі співробітник має можливість переглядати головну сторінку, що зображена на рисунку 2.8, звідки за допомогою панелі навігації, як і клієнт, може перейти на сторінку переліку досліджень, що

зображена на рисунку 2.15. Співробітник не має можливості створювати замовлення, ця сторінка є каталогом-довідником для нього.

Співробітнику доступна сторінка перегляду переліку замовлень створених клієнтами, яка зображена на рисунку 2.21.

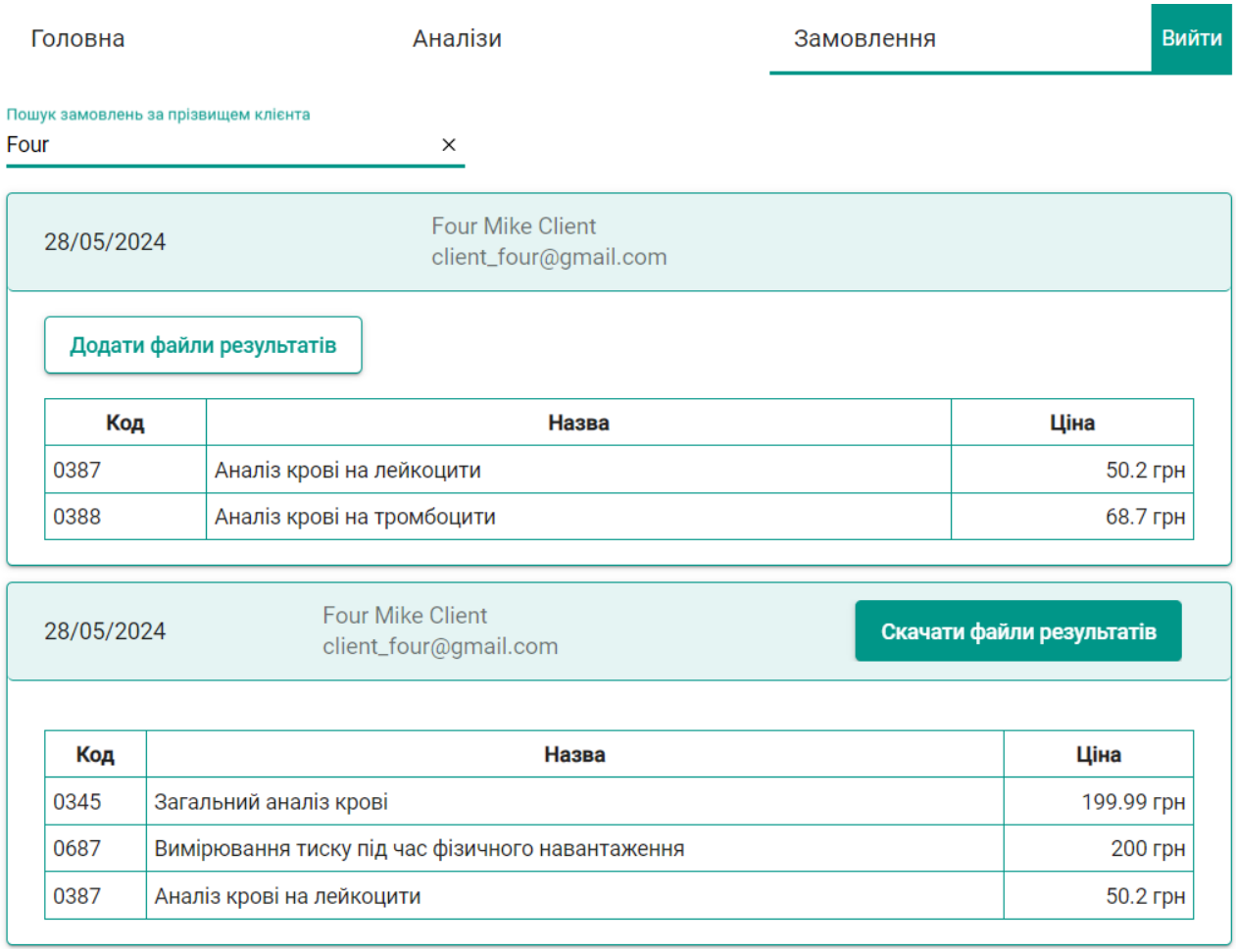


Рис. 2.21. Сторінка «Замовлення» з переліком замовлень клієнтів

Основна функція співробітника, виконувана у системі – це перегляд замовлень клієнтів і додавання до них результатів досліджень, тобто більшість робочого часу він проводить саме на цій сторінці. Щоб виконати додавання результатів досліджень до відповідного замовлення необхідно натиснути кнопку «Додати файли результатів» поряд з переліком досліджень замовлення та обрати потрібні файли у стандартному діалоговому вікні, яке зображене на рисунку 2.22.

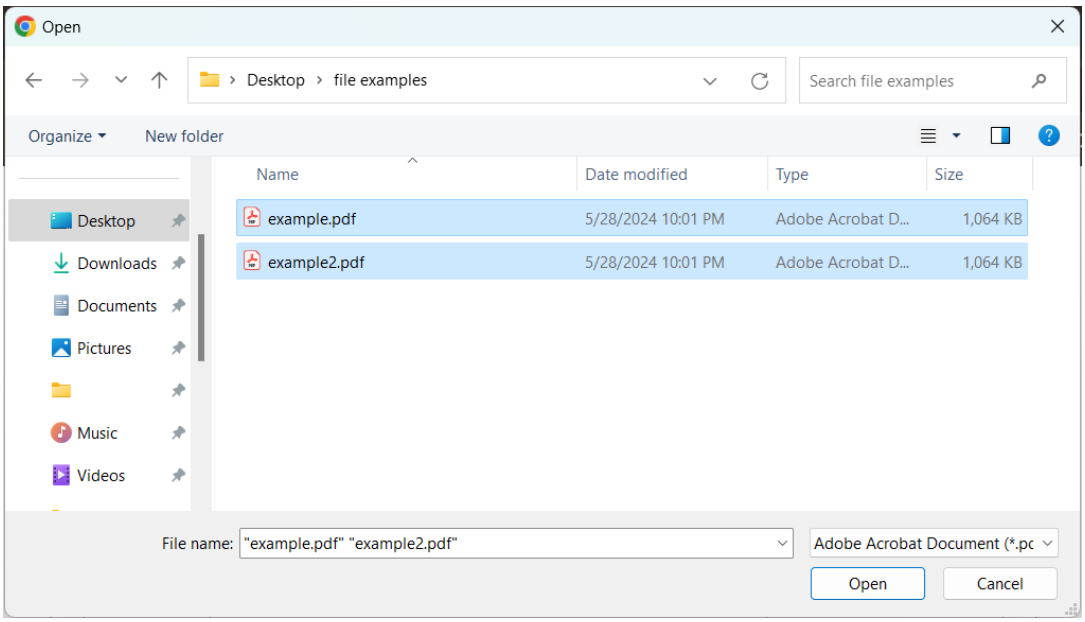


Рис. 2.22. Модальне вікно додавання файлів

Після вибору необхідних файлів, співробітник може натиснути на кнопку «Завантажити файли до бази даних», яка зображена на рисунку 2.23, обрані файли будуть відправлені на сервер та збережені до БД. Клієнт зможе переглянути ці файли на сторінці «Профіль» після авторизації.

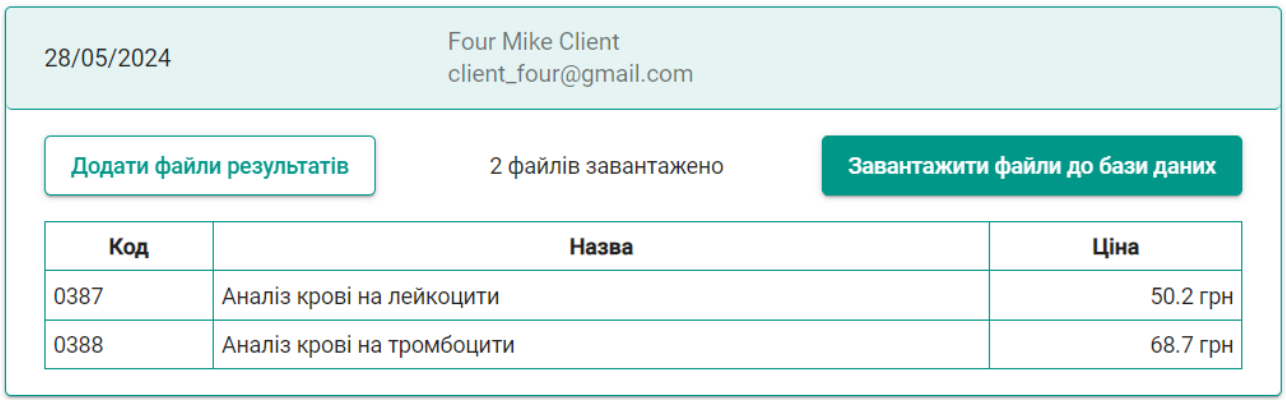


Рис. 2.23. Замовлення, що містить файли, готові до завантаження

Як видно на рисунках 2.24 – 2.25, інтерфейс користувача і співробітника лабораторії є адаптивним. Це означає що вони мають можливість зручно використовувати програму як на комп’ютері, так і на телефоні або планшеті.

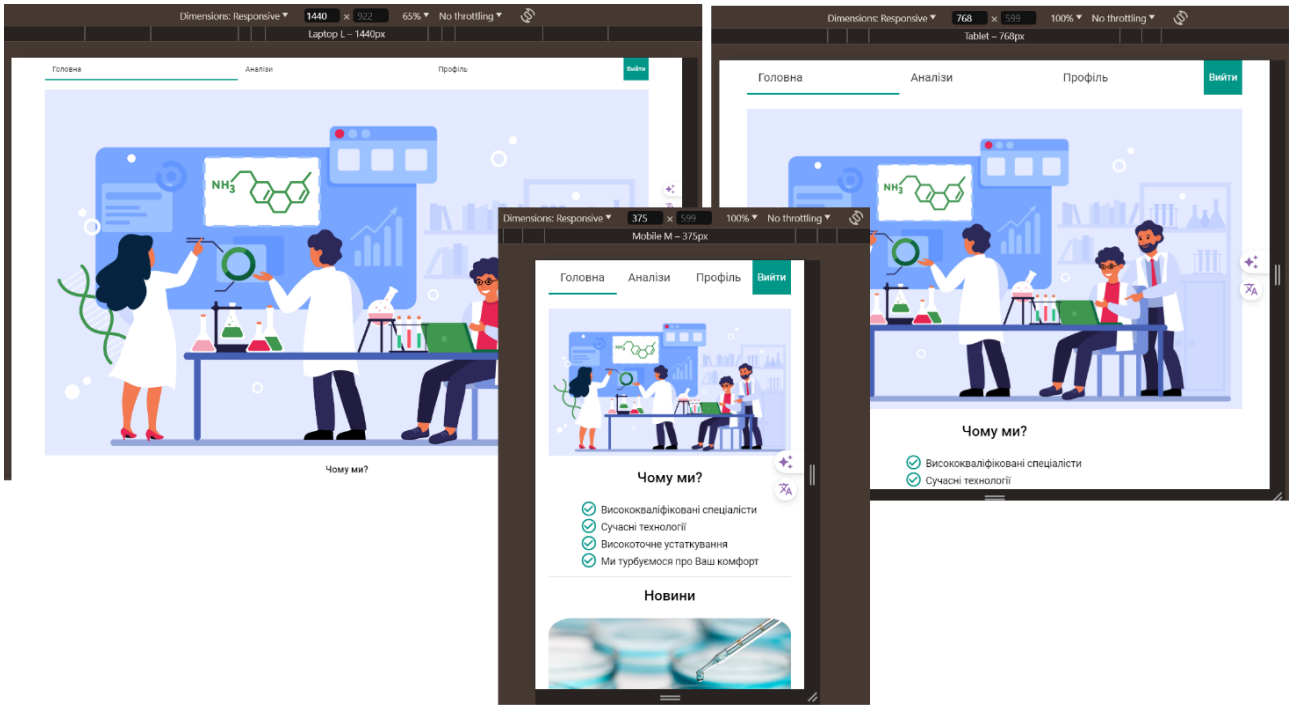


Рис. 2.24. Сторінка «Головна». Вигляд на різних девайсах

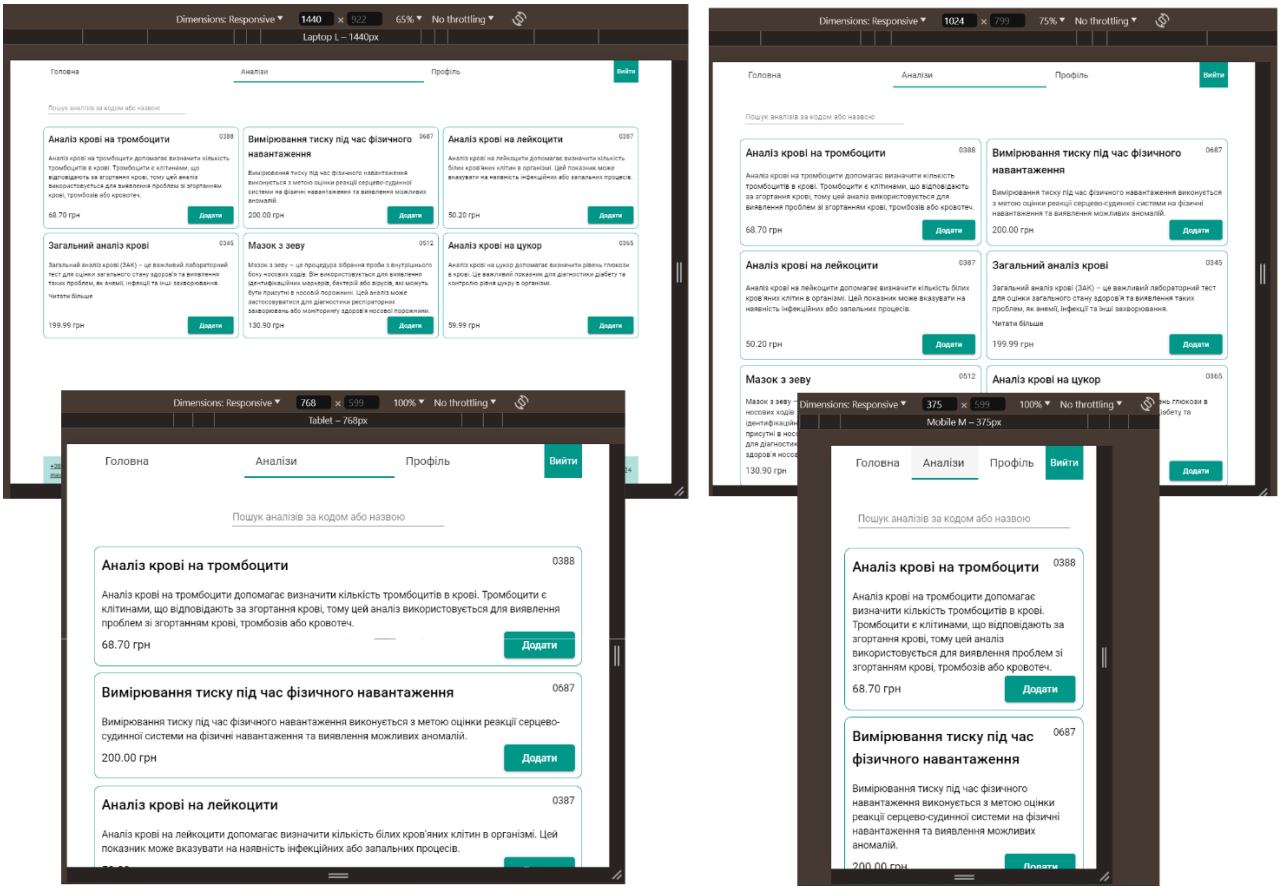


Рис. 2.25. Сторінка «Аналізи». Вигляд на різних девайсах

Функції адміністратора системи зосереджені у веб-інтерфейсі СКБД MongoDB, що зображений на рисунках 2.26 – 2.29. Щоб відкрити його, необхідно перейти за посиланням https://cloud.mongodb.com/ та авторизуватись.

Інтерфейс MongoDB Atlas, що зображений на рисунку 2.28, надає можливість зручно працювати з колекціями БД. Адміністратор має можливість переглядати та оновлювати колекції. Він може додавати нові документи до колекцій users, employees та analyzes, а також редагувати документи в усіх колекціях, окрім results. Для швидкого вибору необхідного документу у колекції, можна використати рядок пошуку, що розташований над переліком документів колекції. За допомогою спеціальних операторів у ньому можна створювати складні запити до БД.

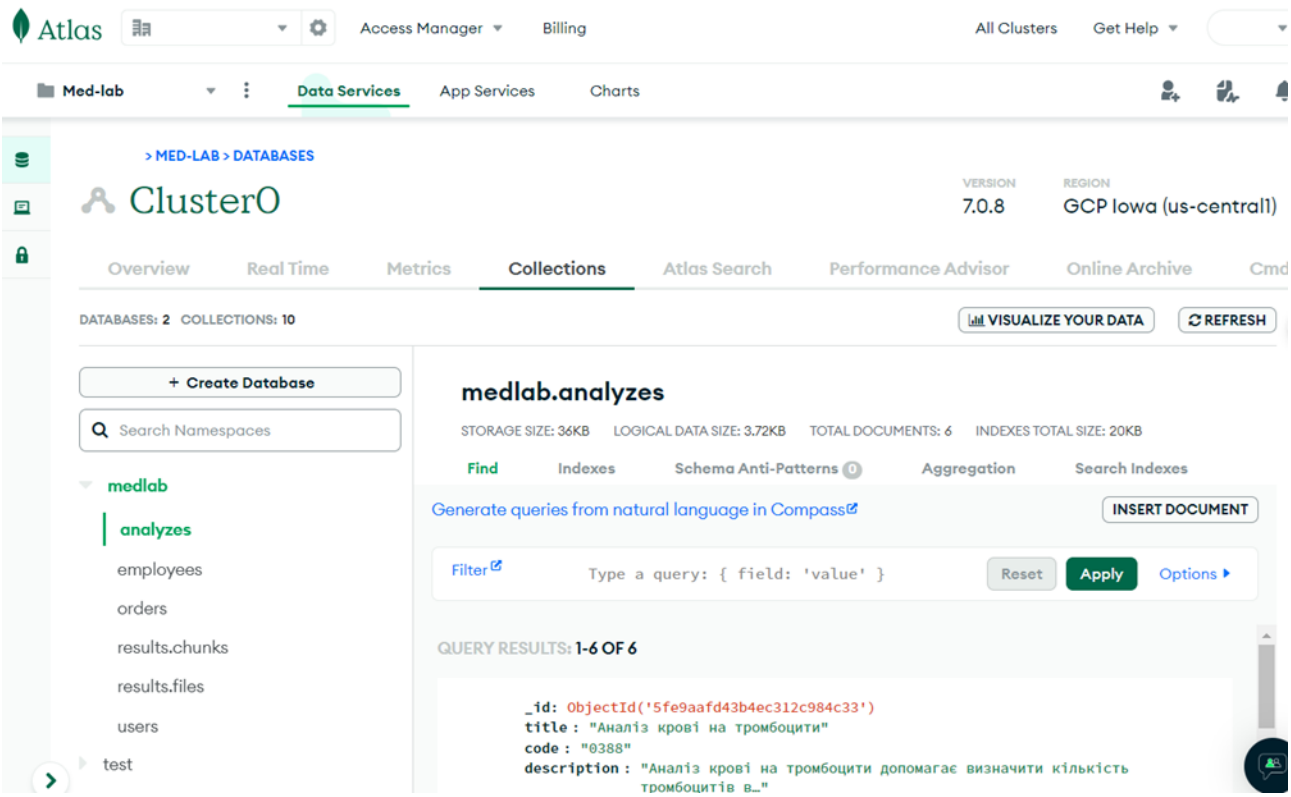


Рис. 2.26. Веб-інтерфейс MongoDB Atlas

Інтерфейс MongoDB Charts, що зображений на рисунку 2.27, надає інструменти для створення та роботи з графічними та табличними даними, що стосуються БД системи.

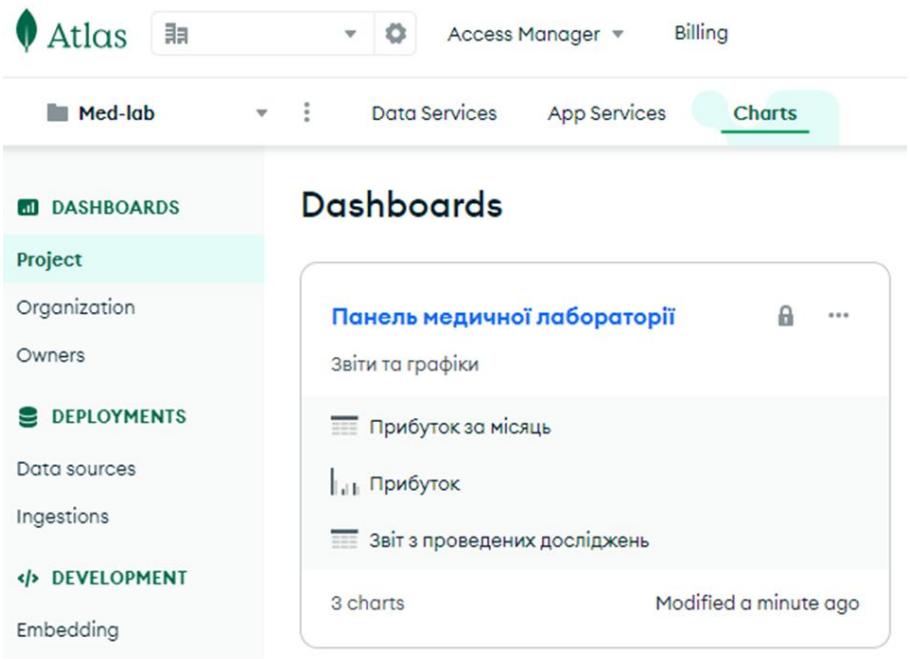


Рис. 2.27. Веб-інтерфейс MongoDB Charts

При переході до панелі «Панель медичної лабораторії» адміністратор має можливість переглянути «Прибуток» у графічному представленні та звіти «Прибуток за місяць», «Звіт з проведених досліджень за рік» з роботи медичної лабораторії, які наведені на рисунках 2.28 – 2.29.

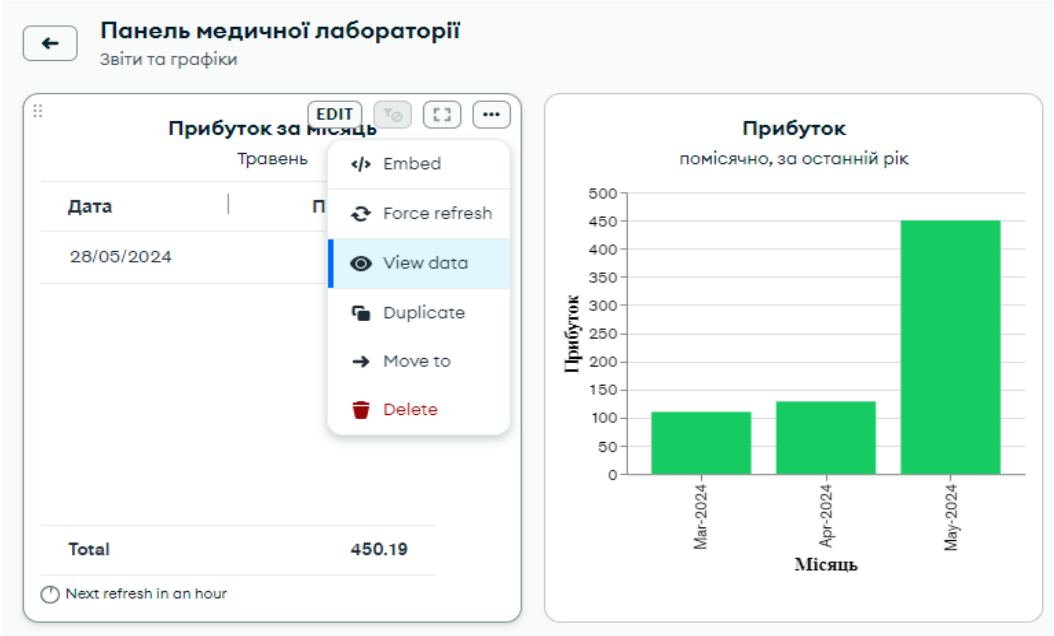


Рис. 2.30. «Панель медичної лабораторії» у інтерфейсі MongoDB Charts (ліва частина)

Адміністратор має можливість редагувати, видаляти та копіювати звіти, використовуючи меню роботи з документом, що зображене на рисунку 2.31, та додавати нові за допомогою кнопки «Add chart», а також виконувати фільтрацію наявних звітів, використовуючи вікно у правій частині інтерфейсу.

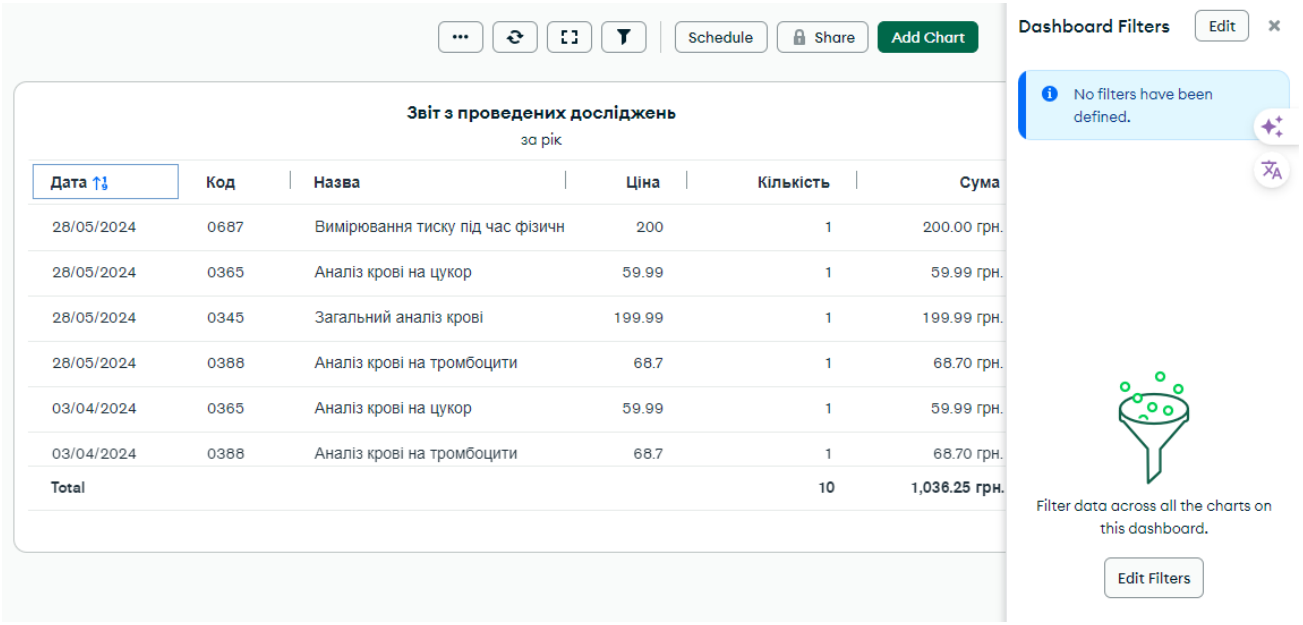


Рис. 2.31. «Панель медичної лабораторії» у інтерфейсі MongoDB Charts (права частина)

Документи, що знаходяться на панелі (наприклад «Звіт з проведених досліджень за рік»), можна завантажити у форматі *.csv або *.json, натиснувши кнопку меню «…», потім «View Data», що знаходяться у меню роботи з документом, яке зображено на рисунку 2.30.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості розробки програмного забезпечення

Вхідні дані:

1. q – передбачуване число операторів – 1100;
2. p – коефіцієнт корекції програми в ході її розробки – 0,1;
3. C – коефіцієнт складності програми – 1,4.
4. годинна заробітна плата розробника – 569,07 грн/год;

Відповідно до статистики на грудень 2023 року на порталі DOU [21], медіанна заробітна плата програміста на позиції Middle Software Engineer становить 2500 доларів (див. рис. 3.1).

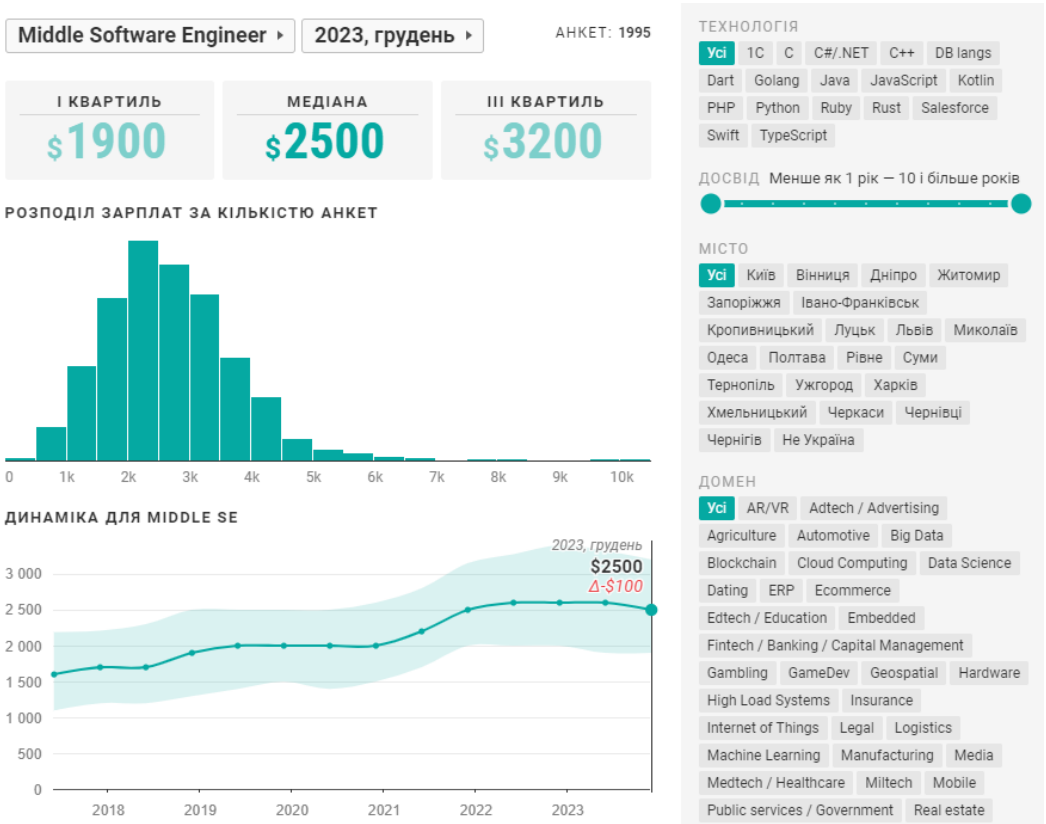
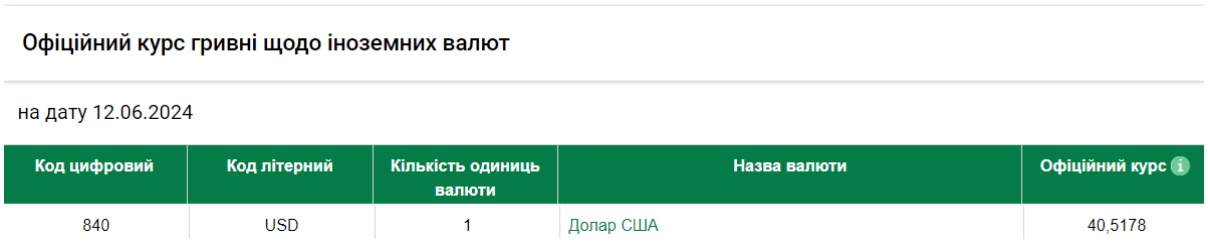


Рис. 3.1. Заробітна плата програміста

На момент написання цієї записки, Офіційний курс гривні Національного банку України щодо долара [22] становить 40,5178 гривень за 1 Долар США (див. рис. 3.2).



Офіційний курс гривні щодо іноземних валют

на дату 12.06.2024

Код цифровий	Код літерний	Кількість одиниць валюти	Назва валюти	Офіційний курс
840	USD	1	Долар США	40,5178

Рис. 3.2. Курс долар/гривня

Тобто, при заробітній платі в розмірі 2500 дол. США, це еквівалентно 101 294,5 грн. на місяць. Стандартний робочий графік становить 176 год/місяць. Отже, погодинна ставка становитиме:

$$101\ 294{,}5 / 176 = 569{,}07 \text{ грн/год.}$$

5. B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;

6. k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,15;

7. вартість машино-години ЕОМ – 1,79 грн/год.

Для розрахунку вартості машино-годин ЕОМ були враховані витрати на електроенергію та домашній інтернет. Згідно з даними Мінфіну на момент написання роботи тариф на електроенергію для населення становила 4,32 грн/кВт·год. Домашній інтернет коштував 200,00 грн. на місяць, тобто 9,23 грн в день, 1,15 грн за годину. Ноутбук споживав 65 Вт на годину. Таким чином вартість 1 години роботи на зазначеному ноутбуці становить $0{,}065 \cdot 4{,}32 = 0{,}2808$ грн. За розрахунками 0, 2808 + 1,15 = 1,4308 (грн/год). Отже, вартість машино-години ЕОМ – 1,43 грн/год;

8. B_k – кількість розробників – 1.

Беручи до уваги, творчий характер роботи розробника, нормування праці в процесі розробки програмного забезпечення не можна розглядати стандартно. Тому, трудомісткість розробки сайту може бути розрахована з використанням системи моделей, що пропонують різні рівні точності оцінки. Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_{\text{и}} + t_{\text{а}} + t_{\text{п}} + t_{\text{отл}} + t_{\text{д}} , \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

$t_{\text{и}}$ – витрати праці на дослідження алгоритму рішення задачі;

$t_{\text{а}}$ – витрати праці на розробку блок-схеми алгоритму;

$t_{\text{п}}$ – витрати праці на програмування по готовій блок-схемі;

$t_{\text{отл}}$ – витрати праці на налагодження програми на ЕОМ;

$t_{\text{д}}$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \times C \times (1 + p) , \qquad (3.2)$$

де q – передбачуване число операторів (1100);

C – коефіцієнт складності програми (1,4);

p – коефіцієнт корекції програми в ході її розробки (0,1).

Звідси за формулою (3.2) умовне число операторів в програмі:

$$Q = 1100 \times 1{,}4 \times (1 + 0{,}1) = 1694.$$

Витрати праці на вивчення опису задачі t_uвизначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \times B}{(75..85) \times k}, \qquad (3.3)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. Стаж роботи 4 роки, тому коефіцієнт кваліфікації програміста = 1,15.

Визначимо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1,4). Враховуючи коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності k = 1,15, отримуємо витрати праці на вивчення опису завдання за формулою (3.3):

Після підставлення значень маємо:

$$t_u = \frac{1694 \times 1{,}4}{85 \times 1{,}15} = 97{,}75 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20..25)\times k}\text{, людино-годин.} \qquad (3.4)$$

Підставивши відповідні значення в формулу (3.4), отримаємо:

$$t_a = \frac{1694}{25 \times 1{,}15} = \ 58{,}92 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі

$$t_{\text{п}} = \frac{Q}{(20..25) \times k}\text{, людино-годин.} \qquad (3.5)$$

За формулою (3.5) та значенням параметру Q, обчислюємо витрати праці на програмування по готовій блок-схемі:

$$t_{\text{п}} = \frac{1694}{25 \times 1{,}15} = \ 58{,}92 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4..5) \times k}\text{, людино-годин.} \qquad (3.6)$$

Витрати праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання за формулою (3.6):

Підставивши значення:

$$t_{\text{отл}} = \frac{1694}{5 \times 1{,}15} = 294{,}6 \text{ людино-годин.}$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{k} = 1{,}5 \times t_{\text{отл}} \text{ , людино-годин.} \qquad (3.7)$$

Витрати праці на налагодження програми на ЕОМ за умови комплексного налагодження завдання за формулою (3.7):

$$t_{\text{отл}}^{k} = 1{,}5 \times 294{,}6 = 441{,}91 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}} \text{ , людино-годин,} \qquad (3.8)$$

де $t_{\text{др}}$ – трудомісткість підготовки матеріалів і рукопису:

$$t_{\text{др}} = \frac{Q}{(15..20) \times k} \text{ , людино-годин,} \qquad (3.9)$$

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \times t_{др}\text{ , людино-годин.} \qquad (3.10)$$

За формулами (3.8), (3.9) та (3.10) обчислюємо витрати праці на документацію:

$t_{др} = \frac{1694}{20 \times 1{,}15}$ = 73,65 людино-годин,

$t_{до}$= 0,75 × 73,65 = 55,24 людино-годин,

$t_{д}$= 73,65 + 55,24 = 128,89 людино-годин.

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

t = 50 + 97,75 + 58,92 + 58,92 + 294,6 + 128,89 = 689,08 людино-години.

3.2. Розрахунок витрат на створення програмного забезпечення

Витрати на створення комп'ютерної системи включають витрати на заробітну плату виконавця програми $З_{зп}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ $З_{мв}$.

$$К_{по} = З_{зп} + З_{мв} \qquad (3.11)$$

Заробітна плата виконавців визначається за формулою:

$$З_{зп} = t \times С_{пр}, \text{ грн}, \quad (3.12)$$

де t - загальна трудомісткість, людино-годин;

$С_{пр}$ - середня годинна заробітна плата програміста, грн/година

Середня плата за одну годинну роботи розробника становить 569,07 грн., тому:

$$З_{зп} = 689{,}08 \times 569{,}07 = 392\ 134{,}7556 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \times С_{мч}, \text{ грн}, \quad (3.13)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год (294,6 год);

$С_{мч}$ - вартість машино-години ЕОМ, грн/год (1,43 грн/год).

Підставивши в формулу (3.13) відповідні значення, обчислюємо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 294{,}6 \times 1{,}43 = 421{,}278 \text{ грн.}$$

Звідси, за формулою (3.11), витрати на створення програмного продукту:

$$К_{по} = 392\,134{,}7556 + 421{,}278 = 392\,556{,}0336 \text{ грн.}$$

Очікуваний період створення програмного застосунку:

$$T = \frac{t}{B_k \times F_p}, \text{ місяців,} \qquad (3.14)$$

де B_k– число виконавців (дорівнює 1);

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

Звідси, за формулою (3.14), очікуваний період створення програмного застосунку:

$$T = \frac{689{,}08}{1 \times 176} = 3{,}9 \text{ місяців}$$

На основі вхідних даних, таких як передбачуване число операторів, коефіцієнти кореляції та складності програми, середня заробітна плата розробника, а також витрати на електроенергію та інтернет, було визначено загальні трудові витрати та витрати на створення програмного продукту.

Трудомісткість розробки програмного забезпечення, враховуючи всі етапи розробки, від підготовки та опису задачі до налагодження та підготовки документації, склала 689,08 людино-годин. Ця оцінка включає в себе всі основні компоненти витрат праці, розраховані на основі кількості операторів та різних коефіцієнтів, що впливають на процес розробки.

Загальні витрати на створення програмного забезпечення склали 392 556,03 грн, з яких основну частину становила заробітна плата виконавця – 392

134,75 грн. Додаткові витрати включали вартість машинного часу для налагодження програми, що становила 421,28 грн.

Очікуваний період створення програмного застосунку, враховуючи місячний фонд робочого часу, склав приблизно 3,9 місяців.

Таким чином, проведені розрахунки показали, що розробка програмного забезпечення є досить трудомістким і витратним процесом, що вимагає значних людських та матеріальних ресурсів. Отримані дані можуть бути використані для планування ресурсів і розрахунку бюджету при реалізації подібних проєктів у майбутньому.

ВИСНОВКИ

Під час роботи над кваліфікаційною роботою було спроєктовано та розроблено комп'ютерну систему для медичних лабораторій. Вона призначена для зв’язку клієнтів лабораторії з її співробітниками та зберігання результатів роботи лабораторії.

Під час розробки системи було проаналізовано подібні системи та з позиції користувача визначено їх переваги та недоліки. У порівнянні з іншими, дана система має спрощений, більш зручний інтерфейс, що дозволяє працювати з нею недосвідченим користувачам.

Особливістю розробленої системи є те, що нею однаково зручно користуватись з будь-якого девайсу від персонального комп’ютера з роздільною здатністю монітора 1920 px до смартфона з роздільною здатністю 320 px, а отже користувачі мають можливість відкрити його як вдома, так і в дорозі, користуючись браузером у смартфоні або планшеті.

Система може використовуватись у приватних та державних медичних лабораторіях. Проведені дослідження у рамках даної кваліфікаційної роботи можуть слугувати основою для подальших розробок у сфері медичного обслуговування.

У процесі виконання кваліфікаційної роботи було визначено трудомісткість розробленої системи, а з використанням середньої зарплати розробника була розрахована вартість роботи, яка склала 392 556,03 гривень. Також було оцінено час, необхідний для розробки системи, який склав 689,08 людино-годин, що приблизно відповідає 3,9 місяцям роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Найкращі Медичні Лабораторії Києва за Рейтингом Ukrainian Business Award: URL https://uba.top/medical-laboratories/ (дата звернення 21.04.2024).
2. Лабораторія «Сінево». Сторінка «Головна». URL https://new.synevo.ua/ (дата звернення 21.04.2024).
3. Лабораторія «Діла». Сторінка «Головна». URL https://dila.ua/ (дата звернення 21.04.2024).
4. Лабораторія «Ескулаб». Сторінка «Адреси». URL https://esculab.com/kyiv (дата звернення 21.04.2024).
5. Головні висновки звіту Global Digital 2023: URL https://www.linkedin.com/pulse/головні-висновки-звіту-global-digital-2023-75min-club/ (дата звернення 21.04.2024).
6. Що таке адаптивний дизайн сайту та як його зробити: URL https://hostiq.ua/blog/ukr/adaptive-design/ (дата звернення 21.04.2024).
7. Як веб-дизайн впливає на користувацький досвід та конверсію: URL https://www.ranktracker.com/uk/blog/how-web-design-impacts-user-experience-and-conversions/ (дата звернення 21.04.2024).
8. What is Cloud Hosting?: URL https://aws.amazon.com/what-is/cloud-hosting/ (дата звернення 21.04.2024).
9. Cloud infrastructure market: URL https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/ (дата звернення 21.04.2024).
10. Cloud Computing in 2024: URL https://tridenstechnology.com/cloud-service-providers/ (дата звернення 21.04.2024).
11. HTML – official documentation. URL: https://developer.mozilla.org/en-US/docs/ (дата звернення 09.05.2024).
12. SASS – official documentation. URL: https://sass-lang.com/documentation (дата звернення 09.05.2024).

13. Angular – official documentation. URL: https://angular.io/docs (дата звернення 09.05.2024).
14. Angular Material – official documentation. URL: https://material.angular.io/ (дата звернення 09.05.2024)
15. Brown E. Web Development with Node and Express, Second edition. O'Reilly Media, Inc., 2019. 409 с.
16. MongoDB – official page. URL: https://www.mongodb.com/ (дата звернення 09.05.2024)
17. Webstorm – official page. URL: https://www.jetbrains.com/webstorm/ (дата звернення 09.05.2024)
18. HTTP – official documentation. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (дата звернення 29.05.2024)
19. Introduction of ER Model. URL: https://www.geeksforgeeks.org/introduction-of-er-model/ (дата звернення 29.05.2024)
20. What Is the MEAN Stack? URL: https://www.mongodb.com/resources/languages/mean-stack (дата звернення 29.05.2024)
21. Статистика заробітної плати на порталі DOU. URL: https://jobs.dou.ua/salaries/ (дата звернення 12.06.2024)
22. Офіційний курс гривні щодо іноземних валют. URL: https://bank.gov.ua/ua/markets/exchangerates (дата звернення 12.06.2024)

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Файл «app.component.ts»:

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class AppComponent implements OnInit {
  title = 'med-lab';

  constructor(private dialog: MatDialog) {}

  public ngOnInit(): void {
    window.onerror = _ => {
      this.dialog.open(DialogComponent, {
        data: {
          title: UserErrorMessage.error,
          content: `<p>${UserErrorMessage.unhandledError}</p>`,
          buttons: [{ role: 'close', text: 'Ok' }],
        },
      });
      return EMPTY;
    };
  }
}
```

Файл «app-routing.module.ts»:

```
const routes: Routes = [
  {
    path: 'login',
    component: LoginComponent,
  },
  {
    path: '',
    component: PageContainerComponent,
    children: [
      {
        path: '',
        component: HomeComponent,
      },
      {
        path: 'analyzes',
        component: AnalyzesComponent,
        canActivate: [AuthGuard],
      },
      {
        path: 'profile',
        component: ProfileComponent,
        canActivate: [AuthGuard],
      },
      {
        path: 'orders',
        component: UserOrdersComponent,
        canActivate: [AuthGuard],
      },
    ],
  },
```

```
  {
    path: '**',
    component: NotFoundComponent,
  },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Файл «constants.ts»:

```
export const ControlErrorMessages = {
  required: `Це поле обов'язкове для заповнення`,
  email: 'Будь ласка введіть валідний email',
  minlength: 'Значення поля закоротке',
  minlengthPassword: 'Мінімальна довжина пароля - 8 символів'
};

export enum UserErrorMessage {
  exists = 'Користувач з такою електронною поштою вже існує. Перейдіть на сторінку логіну будь
ласка',
  wrongPass = 'Неправильна електронна пошта або пароль',
  wrongEmail = `Користувач з такою електронною поштою відсутній<br>
  1) Якщо Ви новий користувач перейдіть на сторінку реєстрації<br>
  2) Перевірте правильність вводу email<br>
  3) Зніміть прапорець "Я співробітник лабораторії"<br>
  4) Якщо Ви співробіник зверніться до адміністратора додатку<br>`,
  unhandledError = 'Вибачте, виникла помилка. Спробуйте перезавантажити сторінку або перейти на
іншу',
  error = 'Помилка',
  warning = '',
  emailNotVerified = 'Ваша електронна пошта присутня у базі, але не підтверджена. Знайдіть лист
з темою Account verification від medlab project та перейдіть за посиланням у ньому'
}

export const baseUrl: string = 'http://localhost:3000';
export const currency: string = 'грн';

export enum UserRole {
  client = 'client',
  employee = 'employee'
}

export enum TabNames {
  home = 'Головна',
  analyzes = 'Аналізи',
  profile = 'Профіль',
  orders = 'Замовлення',
  exit = 'Вийти',
  enter = 'Увійти'
}

export enum AnalyzesLabel {
  search = 'Пошук аналізів за кодом або назвою',
  thCode = 'Код',
  thTitle = 'Назва',
  thPrice = 'Ціна',
  processed = 'Результати готові',
}

export enum AnalyzesMessage {
  reset = 'Ви впевнені, що хочете скинути всі обрані аналізи?',
  submit = 'Ви підтверджуєте вибір аналізів?',
```

```
  sum = 'Сума',
}

export enum ResultsMessage {
  addFiles = 'Додати файли результатів',
  upload = 'Завантажити файли до бази даних',
  download = 'Скачати файли результатів',
  quantityFilesUploaded = 'файлів завантажено'
}

export enum UserInfoLabels {
  email = 'Email',
  password = 'Пароль',
  newPassword = 'Новий пароль',
  confirmPassword = 'Підтвердіть новий пароль',
  name = 'Ім\'я',
  surname = 'Прізвище',
  middlename = 'По-батькові',
  gender = 'Стать',
  genderMale = 'Чоловіча',
  genderFemale = 'Жіноча',
  birthDay = 'Дата народження',
  login = 'Логін',
  cancel = 'Скасувати',
  submit = 'Підтвердити',
  submitPasswordChange = 'Підтвердити зміну пароля',
  registration = 'Реєстрація',
  suggestLogin = 'Маєте обліковий запис?',
  toEnter = 'Увійти',
  suggestRegistration = 'Не маєте облікового запису?',
  toRegister = 'Зареєструватися',
  userInfo = 'Ваші особисті дані',
  userOrders = 'Ваші замовлення',
  changePassword = 'Змінити пароль',
  noNewOrders = 'Нові замовлення відсутні',
  noOrdersFromYou = 'Ви ще не зробили замовення на виконання досліджень. Це можна зробити на
вкладці "Аналізи"',
  loadMore = 'Переглянути більше',
  iAmEmployee = 'Я співробітник лабораторії',
  toMain = 'На головну'
}

export enum OrderLabel {
  search = 'Пошук замовлень за прізвищем клієнта'
}

export enum Message {
  verifyEmail = 'На Вашу електронну пошту надіслано лист для підтвердження. Перейдіть за
посиланням у ньому'
}
```

Файл «page-container.component.ts»:

```
@Component({
  selector: 'app-page-container',
  templateUrl: './page-container.component.html',
  styleUrls: ['./page-container.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class PageContainerComponent {
  public readonly spinnerDiameter = 50;
  public isLoading$: Observable<boolean> = this.loaderService.isLoading$;
```

```
  public userRole: UserRole = this.authService.userRole;


  constructor(private loaderService: LoaderService, private authService: AuthService, private
cdr: ChangeDetectorRef) {
    this.isLoading$.subscribe(() => setTimeout(() => this.cdr.detectChanges(), 0));
  }
}
```

Файл «auth.guard.ts»:

```
@Injectable({
  providedIn: 'root',
})
export class AuthGuard  {
  constructor(private authService: AuthService, private router: Router) {}

  canActivate(route: ActivatedRouteSnapshot): boolean {
    if (
      this.authService.userId &&
      this.authService.userRole === UserRole.client && ['analyzes',
'profile'].includes(route.routeConfig.path) ||
      this.authService.userRole === UserRole.employee && ['analyzes',
'orders'].includes(route.routeConfig.path)
    ) {
      return true;
    }

    this.router.navigateByUrl('/404');
    return false;
  }
}
```

Файл «dialog.component.ts»:

```
@Component({
  selector: 'app-dialog',
  templateUrl: './dialog.component.html',
  styleUrls: ['./dialog.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class DialogComponent implements AfterViewInit {
  @ViewChild('actions') buttonsWrapper: ElementRef;

  public DialogStyle = DialogStyle;

  constructor(@Inject(MAT_DIALOG_DATA) public data: DialogConfig) {}

  public ngAfterViewInit(): void {
    if (document.activeElement === document.querySelector('mat-dialog-container')) {
      this.buttonsWrapper?.nativeElement.querySelector('.mat-primary')?.focus();
    }
  }
}
```

Файл «search.component.ts»:

```
@Component({
  selector: 'app-search',
  templateUrl: './search.component.html',
  styleUrls: ['./search.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
```

```
export class SearchComponent implements OnInit, OnDestroy {
  @Input() label: string = 'Пошук';
  @Output() valueChanged$: Subject<string> = new Subject<string>();

  public searchControl: FormControl = new FormControl();
  private unsubscribe$: Subject<boolean> = new Subject<boolean>();

  constructor() {}

  public ngOnInit(): void {
    this.searchControl.valueChanges
      .pipe(takeUntil(this.unsubscribe$), debounceTime(500))
      .subscribe(value => this.valueChanged$.next(value));
  }

  public ngOnDestroy(): void {
    this.unsubscribe$.next(true);
  }

  public reset(): void {
    this.searchControl.patchValue('');
  }
}
```

Файл «validation-message.directive.ts»:

```
@Directive({
  // tslint:disable-next-line:directive-selector
  selector: '[formControl], [formControlName]',
})
export class ValidationMessageDirective implements OnInit, OnDestroy {
  private submit$: Observable<void | Event>;
  private unsubscribe$: Subject<void> = new Subject<void>();
  @Input() customErrors = {};

  constructor(
    private elementRef: ElementRef,
    private controlDir: NgControl,
    private cdr: ChangeDetectorRef,
    @Optional() @Host() private form: FormSubmitDirective,
  ) {
    this.submit$ = this.form ? this.form.submit$ : EMPTY;
  }

  public ngOnInit(): void {
    this.submit$.subscribe(() => console.log(this.form));
    merge(this.submit$, this.control.valueChanges)
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe(() => {
        const controlErrors: ValidationErrors | null = this.control.errors;
        if (controlErrors) {
          const firstKey = Object.keys(controlErrors)[0];
          const text = this.customErrors[firstKey] || ControlErrorMessages[firstKey];
          this.setError(text);
        }
      });
  }

  public ngOnDestroy(): void {
    this.unsubscribe$.complete();
  }

  private get control(): AbstractControl {
    return this.controlDir.control;
  }

  private setError(text: string): void {
    if (text) {
```

```
      this.cdr.detectChanges(); // to add to DOM mat-error component
      const errorContainer: HTMLElement = this.elementRef.nativeElement
        .closest('.mat-form-field-wrapper')
        ?.querySelector('mat-error');
      if (errorContainer) {
        errorContainer.innerHTML = text;
      }
    }
  }
}
```

Файл «auth.service.ts»:

```
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private userID: string = null;
  private userROLE: UserRole = null;
  private unsubscribe$: Subject<boolean> = new Subject<boolean>();

  constructor(private http: HttpClient, private router: Router, private dialog: MatDialog) {
    this.getUserInfoFromLocaleStorage();
  }

  public get userId(): string {
    return this.userID;
  }

  public get userRole(): UserRole {
    return this.userROLE;
  }

  public get canCreateOrder(): boolean {
    return this.userROLE === UserRole.client;
  }

  public loginUser(userData): void {
    this.http
      .post(`${baseUrl}/api/user/login`,
        {
          data: btoa(escape(userData.email + ':' + userData.password)),
          isEmployee: userData.isEmployee
        }
      )
      .pipe(
        takeUntil(this.unsubscribe$),
        catchError((err: HttpErrorResponse) => {
          switch (err.status) {
            case 404:
              this.dialog.open(DialogComponent, {
                data: {
                  title: UserErrorMessage.error,
                  content: `<p>${UserErrorMessage.wrongEmail}</p>`,
                  buttons: [{ role: 'close', text: 'Ok' }],
                  style: DialogStyle.error
                },
              });
              return EMPTY;
            case 401:
              this.dialog.open(DialogComponent, {
                data: {
                  title: UserErrorMessage.error,
                  content: `<p>${UserErrorMessage.emailNotVerified}</p>`,
                  buttons: [{ role: 'close', text: 'Ok' }],
                  style: DialogStyle.error
                },
```

```
            });
            return EMPTY;
          default:
            throw err;
        }
      }),
    )
    .subscribe((res: HttpResponse<any>) => {
      this.userID = res.body.userId;
      this.userROLE = userData.isEmployee ? UserRole.employee : UserRole.client;
      this.setUserInfoToLocaleStorage();

      this.router.navigateByUrl('');
    });
}

public registerUser(userData): void {
  userData = {
    ...userData,
    password: btoa(escape(userData.email + ':' + userData.password)),
  };
  this.http
    .post(`${baseUrl}/api/user/register`, userData)
    .pipe(
      takeUntil(this.unsubscribe$),
      catchError((err: HttpErrorResponse) => {
        switch (err.status) {
          case 409:
            this.dialog.open(DialogComponent, {
              data: {
                title: UserErrorMessage.error,
                content: `<p>${UserErrorMessage.exists}</p>`,
                buttons: [{ role: 'close', text: 'Ok' }],
                style: DialogStyle.error
              },
            });
            return EMPTY;
          default:
            throw err;
        }
      }),
    )
    .subscribe(() => {
      this.dialog.open(DialogComponent, {
        data: {
          content: `<p>${ Message.verifyEmail }</p>`,
          buttons: [{ role: 'close', text: 'Ok' }],
        },
      });
    });
}

public resetUser(): void {
  this.userID = null;
  this.userROLE = null;
  this.removeUserInfoFromLocaleStorage();
}

private getUserInfoFromLocaleStorage(): void {
  this.userID = localStorage.getItem('userId');
  this.userROLE = localStorage.getItem('userRole') as UserRole;
}

private setUserInfoToLocaleStorage(): void {
  localStorage.setItem('userId', this.userID);
  localStorage.setItem('userRole', this.userROLE);
}
```

```
  private removeUserInfoFromLocaleStorage(): void {
    localStorage.removeItem('userId');
    localStorage.removeItem('userRole');
  }
}
```

Файл «loader.service.ts»:

```
@Injectable({
  providedIn: 'root'
})
export class LoaderService {
  public loading$: BehaviorSubject<boolean> = new BehaviorSubject<boolean>(false);
  public isLoading$: Observable<boolean> = this.loading$.pipe(distinctUntilChanged()) as
Observable<boolean>;

  public startLoading(): void {
    this.loading$.next(true);
  }

  public endLoading(): void {
    this.loading$.next(false);
  }
}
```

Файл «login.component.ts»:

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class LoginComponent implements OnInit {
  public isLoginForm: boolean = true;
  public loginForm: FormGroup;
  public registerForm: FormGroup;
  public placeholders = UserInfoLabels;
  public ControlErrorMessages = ControlErrorMessages;

  constructor(private fb: FormBuilder, private authService: AuthService, private router:
Router) {}

  public ngOnInit(): void {
    this.authService.resetUser();
    this.initForms();
  }

  public switchForm(): void {
    this.isLoginForm = !this.isLoginForm;
  }

  public toMain(): void {
    this.router.navigateByUrl('');
  }

  public onSubmit(): void {
    if (this.isLoginForm) {
      this.authService.loginUser(this.loginForm.getRawValue());
    } else {
      this.authService.registerUser(this.registerForm.getRawValue());
    }
  }

  private initForms(): void {
    this.loginForm = this.fb.group({
      email: ['', [Validators.required, Validators.email]],
```

```
      password: ['', Validators.required],
      isEmployee: [false]
    });
    this.registerForm = this.fb.group({
      name: ['', Validators.required],
      surname: ['', Validators.required],
      middlename: ['', Validators.required],
      birthDay: ['', Validators.required],
      gender: [null, Validators.required],
      email: ['', [Validators.required, Validators.email]],
      password: ['', [Validators.required, Validators.minLength(8)]],
    });
  }
}
```

Файл «analyzes.component.ts»:

```
@Component({
  selector: 'app-analyzes',
  templateUrl: './analyzes.component.html',
  styleUrls: ['./analyzes.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class AnalyzesComponent implements OnInit, OnDestroy {
  public analyzesList$: BehaviorSubject<Array<Analysis>> = new
BehaviorSubject<Array<Analysis>>([]);
  public searchLabel: string = AnalyzesLabel.search;
  public itemsSelected: boolean = false;
  public labels = UserInfoLabels;
  public canCreateOrder$: BehaviorSubject<boolean> = new BehaviorSubject(true);

  private searchString$: Subject<string> = new Subject<string>();
  private unsubscribe$: Subject<boolean> = new Subject<boolean>();

  constructor(
    private analyzesService: AnalyzesService,
    private dialog: MatDialog,
    private router: Router,
    private loaderService: LoaderService,
    private authService: AuthService
  ) {}

  public ngOnInit(): void {
    this.loadAnalyzes();
    this.searchString$
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe(str => this.loadAnalyzes(str));

    this.itemsSelected = !!this.analyzesList$.value.find(item => item.selected);
    this.canCreateOrder$.next(this.authService.canCreateOrder);
  }

  public ngOnDestroy(): void {
    this.unsubscribe$.next(true);
    this.unsubscribe$.complete();
    this.loaderService.endLoading();
  }

  public loadAnalyzes(titleOrCode: string = ''): void {
    this.analyzesService
      .getAnalyzes(titleOrCode)
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe(res =>
        this.analyzesList$.next(res)
      );
  }

  public onAnalysisToggle(code: string): void {
```

```
    this.analyzesList$.value.forEach(item => {
      if (item.code === code) {
        item.selected = !item.selected;
      }
    });

    this.itemsSelected = !!this.analyzesList$.value.find(item => item.selected);
  }

  public onResetClick(): void {
    this.dialog.open(DialogComponent, {
      data: {
        content: `<p>${AnalyzesMessage.reset}</p>`,
        buttons: [
          { role: 'close', text: 'Повернутись до вибору' },
          { role: 'submit', text: 'Так', primary: true, handler: () => this.resetSelection() },
        ],
      },
      autoFocus: false,
    });
  }

  public onSubmitClick(): void {
    const result: Array<Analysis> = this.analyzesList$.value.filter(item => item.selected);
    let totalPrice = 0;
    result.forEach(item => (totalPrice += item.price));
    this.dialog.open(DialogComponent, {
      data: {
        content: this.createSubmissionMessage(result, totalPrice),
        buttons: [
          { role: 'close', text: 'Повернутись до вибору' },
          { role: 'submit', text: 'Так', primary: true, handler: () =>
this.submitSelection(result) },
        ],
      },
      autoFocus: false,
    });
  }

  public setSearchValue(searchStr: string): void {
    this.searchString$.next(searchStr);
  }

  public analyzesTrackBy(_, analyze): number {
    return analyze.selected;
  }

  private resetSelection(): void {
    const list = this.analyzesList$.value;
    list.forEach(item => (item.selected = false));
    this.analyzesList$.next(list);
    this.itemsSelected = false;
  }

  private submitSelection(selectedItems: Array<Analysis>): void {
    this.analyzesService
      .submitOrder({ analyzes: selectedItems.map(item => item._id) })
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe(() => {
        this.resetSelection();
        this.router.navigate(['profile']);
      });
  }

  private createSubmissionMessage(selectedItems: Array<Analysis>, totalPrice: number): string {
    const tableHeader =
`<tr><th>${AnalyzesLabel.thCode}</th><th>${AnalyzesLabel.thTitle}</th><th>${AnalyzesLabel.thPri
ce}</th></tr>`;
```

```
    let selectedItemsTable = selectedItems.reduce((accum, item) => {
      accum += `<tr>
        <td>${item.code}</td>
        <td>${item.title}</td>
        <td class="align-right">${item.price.toFixed(2)} ${currency}</td>
        </tr>`;
      return accum;
    }, tableHeader);
    selectedItemsTable += `<tr><th colspan="2">${AnalyzesMessage.sum}</th><th class="align-
right">${totalPrice} ${currency}</th></tr>`;
    return `<p>${AnalyzesMessage.submit}</p>
    <table class="dialog__table">${selectedItemsTable}</table>`;
  }
}
```

Файл «analysis.component.ts»:

```
@Component({
  selector: 'app-analysis',
  templateUrl: './analysis.component.html',
  styleUrls: ['./analysis.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class AnalysisComponent implements AfterViewInit, OnDestroy {
  public analysisItem: Omit<Analysis, 'description'> & { description: string[] };
  @Input() canSelect: boolean = true;
  @Output() toggled: EventEmitter<string> = new EventEmitter<string>();

  @ViewChild('button') matButtonElem: MatButton;
  @ViewChild('wrapper') wrapperElem: ElementRef;
  public showFullDescription: boolean = false;
  public readonly linebreakSymbol = '<br>';

  @Input() set analysis(item: Analysis) {
    this.analysisItem = {
      ...item,
      description: item.description.split('\\n')
    };

    if (this.matButtonElem && this.wrapperElem) {
      this.toggleSelectionStyles();
    }
  }

  private unsubscribe$: Subject<boolean> = new Subject<boolean>();

  public ngAfterViewInit(): void {
    if (this.canSelect) {
      this.toggleSelectionStyles();
    }
  }

  public ngOnDestroy(): void {
    this.unsubscribe$.next(true);
    this.unsubscribe$.complete();
  }

  public onBtnClick(): void {
    this.toggled.emit(this.analysisItem.code);
    this.toggleSelectionStyles();
  }

  public toggleDescription() {
    this.showFullDescription = !this.showFullDescription;
  }

  private toggleSelectionStyles(): void {
```

```
    if (this.analysisItem?.selected) {
      this.matButtonElem?._elementRef.nativeElement.classList.remove('mat-primary');
      this.wrapperElem?.nativeElement.classList.add('analysis__wrapper--primary');
    } else {
      this.matButtonElem?._elementRef.nativeElement.classList.add('mat-primary');
      this.wrapperElem?.nativeElement.classList.remove('analysis__wrapper--primary');
    }
  }
}
```

Файл «analyzes.service.ts»:

```
@Injectable()
export class AnalyzesService implements OnDestroy {
  private analyzesCached: Array<Analysis> = null;
  private unsubscribe$: Subject<boolean> = new Subject<boolean>();

  constructor(private http: HttpClient, private loaderService: LoaderService, private
authService: AuthService) {}

  public ngOnDestroy(): void {
    this.unsubscribe$.next(true);
  }

  public getAnalyzes(titleOrCode: string): Observable<Array<Analysis>> {
    if (!this.analyzesCached) {
      this.loaderService.startLoading();
      return this.getAnalyzesFromServer().pipe(
        tap(res => (this.analyzesCached = res)),
        finalize(() => this.loaderService.endLoading())
        );
    }
    if (titleOrCode) {
      const regExpStr: string = titleOrCode
        .split(' ')
        .reduce((acc, word, index, arr) => acc + word + (index < arr.length - 1 ? ')(?=.*' :
').*'), '(?=.*');
      const regExp: RegExp = new RegExp(regExpStr, 'i');
      return of(this.analyzesCached.filter(analysis => regExp.test(analysis.code) ||
regExp.test(analysis.title)));
    }
    return of(this.analyzesCached);
  }

  public submitOrder(order: { analyzes: Array<string> }): Observable<any> {
    return this.http.post(`${baseUrl}/api/orders`, { userId: this.authService.userId, ...order
});
  }

  private getAnalyzesFromServer(): Observable<Array<Analysis>> {
    return this.http.get<Array<Analysis>>(`${baseUrl}/api/analyzes`);
  }
}
```

Файл «profile.component.ts»:

```
@Component({
  selector: 'app-profile',
  templateUrl: './profile.component.html',
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class ProfileComponent implements OnDestroy {
  constructor(public loaderService: LoaderService) {}

  ngOnDestroy(): void {
    this.loaderService.endLoading();
  }
```

```
}
```

Файл «user-info.component.ts»:

```
@Component({
  selector: 'app-user-info',
  templateUrl: './user-info.component.html',
  styleUrls: ['./user-info.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class UserInfoComponent implements OnInit, OnDestroy, AfterViewInit {
  public labels = UserInfoLabels;
  public data$: Subject<UserInfo> = new Subject<UserInfo>();
  public isEditMode: boolean = false;
  public userInfoForm: FormGroup;
  public isUserInfoFormChanged$: Subject<boolean> = new Subject();

  private unsubscribe$: Subject<boolean> = new Subject<boolean>();
  private initialUserInfo: {};

  @ViewChild('editBtn') private editBtn: ElementRef;

  constructor(private userInfoService: UserInfoService, private fb: FormBuilder) {}

  public ngOnInit(): void {
    this.initForms();

    this.data$
      .pipe(takeUntil(this.unsubscribe$), tap(console.log))
      .subscribe(data => {
        this.isUserInfoFormChanged$.next(false);
        this.initialUserInfo = data;
        this.userInfoForm.patchValue(data);
      });

    this.userInfoForm.valueChanges.subscribe(v =>
      this.isUserInfoFormChanged$.next(Object.keys(v).some(key => {
        const curVal = v[key] instanceof Date
        ? v[key].toISOString()
        : v[key];
        return curVal !== this.initialUserInfo[key];
      })
      )
    );
  }

  public ngAfterViewInit(): void {
    this.userInfoService.getUserInfo()
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe(res => this.data$.next(res));
  }

  public ngOnDestroy(): void {
    this.unsubscribe$.next(true);
    this.unsubscribe$.complete();
  }

  public toggleEditMode(): void {
    this.isEditMode = !this.isEditMode;
    if(this.isEditMode) {
      this.userInfoForm.patchValue(this.initialUserInfo);
    }
  }

  public hideEditBtn(): void {
    this.editBtn?.nativeElement.classList.add('user-info-edit--hidden');
  }
```

```
  public showEditBtn(): void {
    this.editBtn?.nativeElement.classList.remove('user-info-edit--hidden');
  }

  public updateUserInfo(): void {
    const body = Object.entries(this.userInfoForm.getRawValue())
      .reduce((acc, [key, value]) => {
        if (value !== this.initialUserInfo[key]) {
          acc[key] = value;
        }
        return acc;
      }, {});

    this.userInfoService.updateUserInfo(body).subscribe(res => {
      this.userInfoService.resetCachedUserInfo();
      this.data$.next(res);
      this.toggleEditMode();
    });
  }

  private initForms(): void {
    this.userInfoForm = this.fb.group({
      name: ['', Validators.required],
      surname: ['', Validators.required],
      middlename: ['', Validators.required],
      birthDay: ['', Validators.required],
      gender: [null, Validators.required],
      email: ['', [Validators.required, Validators.email]]
    });
  }
}
```

Файл «orders.component.ts»:

```
@Component({
  selector: 'app-orders',
  templateUrl: './orders.component.html',
  styleUrls: ['./orders.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class OrdersComponent implements OnInit, OnDestroy {
  public orders$: Subject<Array<Order>> = new Subject();
  public labels =  UserInfoLabels;

  private unsubscribe$: Subject<boolean> = new Subject<boolean>();

  constructor(private ordersService: OrdersService) {}

  public ngOnInit(): void {
    this.ordersService
      .getOrders()
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe(orders => this.orders$.next(orders));
  }

  public ngOnDestroy(): void {
    this.unsubscribe$.next(true);
    this.unsubscribe$.complete();
  }
}
```

Файл «user-info.service.ts»:

```
@Injectable()
export class UserInfoService {
  public userInfo: UserInfo;
  private userId: string;
```

```
  constructor(private http: HttpClient, private authService: AuthService, private dialog:
MatDialog) {}

  public getUserInfo(): Observable<UserInfo> {
    if (this.userId !== this.authService.userId) {
      this.userId = this.authService.userId;
      this.resetCachedUserInfo();
    }
    return this.userInfo
      ? of(this.userInfo)
      : this.http
          .get<UserInfo>(`${baseUrl}/api/user/${this.userId}`)
          .pipe(
            tap(info => (this.userInfo = info)),
            catchError(() => {
              this.dialog.open(DialogComponent, {
                data: {
                  title: UserErrorMessage.error,
                  content: `<p>${UserErrorMessage.unhandledError}</p>`,
                  buttons: [{ role: 'close', text: 'Ok' }],
                },
              });
              return EMPTY;
            })
          );
  }

  public updateUserInfo(body: {}): Observable<UserInfo> {
    return this.http
      .put<UserInfo>(`${baseUrl}/api/user/${this.authService.userId}`, body)
      .pipe(
        tap(info => (this.userInfo = info)),
        catchError(() => {
          this.dialog.open(DialogComponent, {
            data: {
              title: UserErrorMessage.error,
              content: `<p>${UserErrorMessage.unhandledError}</p>`,
              buttons: [{ role: 'close', text: 'Ok' }],
            },
          });
          return EMPTY;
        })
      );
  }

  public resetCachedUserInfo(): void {
    this.userInfo = null;
  }
}
```

Файл «orders.service.ts»:

```
@Injectable()
export class OrdersService {
  constructor(private http: HttpClient, private authService: AuthService, private
loaderService: LoaderService) {}

  public getOrders(): Observable<Array<Order>> {
    this.loaderService.startLoading();
    return this.http
    .get<Array<Order>>(`${baseUrl}/api/user/${this.authService.userId}/orders`)
    .pipe(finalize(() =>  this.loaderService.endLoading()));
  }
}
```

Файл «order.component.ts»:

```
@Component({
  selector: 'app-order',
  templateUrl: './order.component.html',
  styleUrls: ['./order.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class OrderComponent {
  public labels = AnalyzesLabel;
  public currency = currency;
  public ResultsMessage = ResultsMessage;
  public filesToUpload$: BehaviorSubject<Array<File>> = new BehaviorSubject([]);

  @Input('order') order: Order;
  @Input('isExpanded') isExpanded: boolean = false;
  @Input('canAddResults') canAddResults: boolean = false;
  @Input('userData') userData: UserInfo;

  @ViewChild('fileLoader') fileLoader: ElementRef;

  constructor(private fileUploadService: FileUploadService) {}

  public fileInputChange(fileInputEvent: any): void {
    this.filesToUpload$.next(fileInputEvent.target.files);
  }

  public uploadFiles(): void {
    for (const file of this.filesToUpload$.value) {
      this.fileUploadService.postFile(this.order._id, file)
        .subscribe(() => console.log('file uploaded: ' + file.name));
    }
  }

  public downloadFiles(e): void {
    e.stopPropagation();

    this.order.results.forEach(id =>
      this.fileUploadService.loadFie(id)
        .subscribe(res => {
          // It is necessary to create a new blob object with mime-type explicitly set
          // otherwise only Chrome works like it should
          const newBlob = new Blob([res.body], { type: 'application/pdf' });

          // IE doesn't allow using a blob object directly as link href
          // instead it is necessary to use msSaveOrOpenBlob
          if (window.navigator && window.navigator.msSaveOrOpenBlob) {
            window.navigator.msSaveOrOpenBlob(newBlob);
            return;
          }

          // For other browsers:
          // Create a link pointing to the ObjectURL containing the blob.
          const data = window.URL.createObjectURL(newBlob);

          const filename = this.getFilename(res);
          const link = document.createElement('a');
          link.href = data;
          link.download = filename;
          // this is necessary as link.click() does not work on the latest firefox
          link.dispatchEvent(new MouseEvent('click', { bubbles: true, cancelable: true, view:
window }));

          setTimeout(() => {
            // For Firefox it is necessary to delay revoking the ObjectURL
            window.URL.revokeObjectURL(data);
            link.remove();
          }, 100);
        })
```

```
    );
  }

  private getFilename(res: HttpResponse<any>): string {
    let filename;
    try {
      const contentDisposition: string = res.headers.get('content-disposition');
      const r = /(?:filename=")(.+)(?:")/;
      filename = r.exec(contentDisposition)[1];
    } catch (err) {
      filename = 'result.pdf';
    }
    return filename;
  }
}
```

Файл «file-upload.service.ts»:

```
@Injectable()
export class FileUploadService implements OnDestroy {

  constructor(private http: HttpClient, private loaderService: LoaderService) {}

  ngOnDestroy(): void {
    this.loaderService.endLoading();
  }

  public postFile(orderId: string, file: File): Observable<boolean> {
    this.loaderService.startLoading();

    const formData: FormData = new FormData();
    formData.append('result', file, file.name);

    return this.http.post<boolean>(`${baseUrl}/api/orders/${orderId}/result`, formData)
      .pipe(
        catchError((err: HttpErrorResponse) => {
          console.log(err);
          return EMPTY;
        }),
        finalize(() => this.loaderService.endLoading())
      );
  }

  public loadFie(id: string): Observable<HttpResponse<any>> {
    this.loaderService.startLoading();
    return this.http.get(`${baseUrl}/api/result/${id}`, { responseType: 'blob', observe:
'response' })
      .pipe(
        catchError((err: HttpErrorResponse) => {
          console.log(err);
          return EMPTY;
        }),
        finalize(() => this.loaderService.endLoading())
      );
  }
}
```

Файл «user-orders.component.ts»:

```
@Component({
  selector: 'app-user-orders',
  templateUrl: './user-orders.component.html',
  styleUrls: ['./user-orders.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class UserOrdersComponent implements OnInit, OnDestroy {
  public labels = UserInfoLabels;
```

```
  public orders$: BehaviorSubject<Array<Order>> = new BehaviorSubject<Array<Order>>([]);
  public canLoadMoreOrders: boolean;
  public searchLabel: string = OrderLabel.search;

  private searchString$: BehaviorSubject<string> = new BehaviorSubject<string>('');
  private ordersIndex = 0;
  private ordersCount = 0;
  private readonly ordersQuantity = 5;
  private unsubscribe$: Subject<boolean> = new Subject<boolean>();

  constructor(private userOrdersService: UserOrdersService, private loaderService:
LoaderService) {}

  public ngOnInit(): void {
    this.searchString$
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe(str => this.loadOrders(str));
  }

  public ngOnDestroy(): void {
    this.unsubscribe$.next(true);
    this.unsubscribe$.complete();
    this.loaderService.endLoading();
  }

  public loadOrders(searchSurname: string = ''): void {
    if (searchSurname) { // search query changed
      this.ordersIndex = 0;
      this.ordersCount = 0;
      this.orders$.next([]);
    }

    if (!this.ordersCount || this.canLoadMoreOrders) {
      this.userOrdersService.getOrders(
        this.ordersIndex,
        this.ordersQuantity,
        searchSurname || this.searchString$.value // search query changed || load more click
      )
        .pipe(takeUntil(this.unsubscribe$))
        .subscribe(res => {
        this.orders$.next([...this.orders$.value, ...res.orders]);
        this.ordersIndex += this.ordersQuantity;
        this.ordersCount = res.count;
        this.canLoadMoreOrders = this.ordersCount > this.orders$.value.length;
      });
    }
  }

  public trackByOrders(index, order: Order): string {
    return index || order._id;
  }

  public setSearchValue(searchStr: string): void {
    this.searchString$.next(searchStr);
  }
}
```

Файл «user-orders.service.ts»:

```
@Injectable()
export class UserOrdersService {
  constructor(private http: HttpClient, private loaderService: LoaderService) { }

  public getOrders(index: number, quantity: number, searchSurname: string = ''):
    Observable<{ orders: Array<Order>, count: number }>
  {
    this.loaderService.startLoading();
    let params: HttpParams = new HttpParams()
```

```
      .set('start', `${index}`)
      .set('num', `${quantity}`);

    if (searchSurname) {
      // HttpParams is intended to be immutable. The set and append methods don't modify the
existing instance.
      // Instead they return new instances.
      params = params.set('surname', `${searchSurname}`);
    }

    return this.http.get<{ orders: Array<Order>, count: number }>(`${baseUrl}/api/orders`, {
params })
      .pipe(
        finalize(() => this.loaderService.endLoading())
      );
  }
}
```

Файл «verify_email.html»:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta content="IE=edge" http-equiv="X-UA-Compatible">
  <meta content="width=device-width, initial-scale=1.0" name="viewport">
  <title>Email Confirmation</title>
  <style>
    body {
      background-color: white;
      font-family: 'Arial', sans-serif;
      color: black;
      margin: 0;
      padding: 0;
      text-align: center;
    }

    .container {
      background-color: white;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0, 150, 136, 0.1);
      max-width: 600px;
      margin: 20px auto;
      /*padding: 20px;*/
    }

    .header {
      background-color: #009688;
      color: white;
      font-size: 1.6em; /* 2px bigger than main text */
      padding: 10px;
      border-radius: 8px 8px 0 0;
    }

    a {
      color: #009688;
      font-weight: bold;
    }

    p {
      line-height: 1.6;
      margin: 10px auto; /* Remove default top margin */
    }

    .wrapper {
      padding: 20px;
    }
```

```
  </style>
</head>
<body>
<div class="container">
  <div class="header">
    <p>Підтвердження адреси електронної пошти</p>
  </div>
  <div class="wrapper">
    <p>Натисніть на посилання нижче для підтвердження адреси Вашої електронної пошти для
акаунту в Медичній лабораторії.</p>
    <p>Якщо Ви не створювали акаунт, або не хочете щоб ця пошта використовувалась для
авторизації, проігноруйте цей лист.</p>
    <p><a href="{{linkPlaceholder}}">Підтвердити електронну пошту</a></p>
  </div>
</div>
</body>
</html>
```

Файл «config.js»:

```
import dotenv from 'dotenv';

dotenv.config();

export default {
  PORT: process.env.PORT,
  DB: {
    connectionStr: process.env.DB_CONNECTION_STR,
    name: process.env.DB_NAME
  },
  APP_BASE_URL: process.env.APP_BASE_URL,
  MAILER: {
    service: process.env.MAILER_SERVICE,
    credentials: {
      user: process.env.MAILER_URL,
      pass: process.env.MAILER_PASS
    }
  },
};

module.exports = nconf;
```

Файл «log.js»:

```
import { createLogger, transports } from 'winston';

export default createLogger({
  transports: [
    new transports.Console({
      colorize: true,
      level: 'debug',
    }),
  ],
});
```

Файл «mongoose.js»:

```
import mongoose from 'mongoose';
import log from './log.js';
import config from './config.js';

const { Schema, connect, model } = mongoose;

connect(config.DB.connectionStr, { dbName: config.DB.name })
```

```
  .then(() => log.info('Connected to DB!'))
  .catch(err => {
    log.error('DB connection error:', err);
    process.exit(1);
  });


const User = new Schema(
  {
    name: String,
    surname: String,
    middlename: String,
    birthDay: Date,
    gender: String,
    email: String,
    password: String,
    verified: Boolean
  },
  {
    versionKey: false,
  },
);

const Employee = new Schema(
  {
    name: String,
    surname: String,
    middlename: String,
    email: String,
    password: String
  },
  {
    versionKey: false,
  },
);

const Analysis = new Schema(
  {
    title: String,
    code: String,
    price: Number,
    description: String,
  },
  {
    versionKey: false,
  },
);

const Order = new Schema(
  {
    dateCreated: Date,
    dateProcessed: Date,
    analyzes: [Schema.ObjectId],
    results: [Schema.ObjectId],
    userId: Schema.ObjectId,
    employeeId: Schema.ObjectId,
  },
  {
    versionKey: false,
  },
);

export const UserModel = model('users', User);
export const EmployeeModel = model('employees', Employee);
export const AnalysisModel = model('analyzes', Analysis);
export const OrderModel = model('orders', Order);
```

Файл « orders-query-builder.js»:

```
import { OrderModel } from "./mongoose.js";

export default class OrdersQueryBuilder {
  #aggregationPipeline = [];

  /**
   * result format { paginatedResults, count: [{ total }] }
   * @returns {Aggregate<Array<{ [string]: any }>>}
   */
  getAggregation() {
    return OrderModel.aggregate(this.#aggregationPipeline);
  }

  /**
   * Merges data from analyzes and users tables
   * @returns {OrdersQueryBuilder}
   */
  addBaseQuery() {
    this.#aggregationPipeline.push(
      {
        $lookup: {
          from: 'analyzes',
          localField: 'analyzes',
          foreignField: '_id',
          as: 'analyzesData',
        }
      },
      {
        $lookup: {
          from: 'users',
          localField: 'userId',
          foreignField: '_id',
          as: 'userData',
        }
      },
      {
        $project: {
          "analyzes": 1,
          "analyzesData": 1,
          "dateCreated": 1,
          "dateProcessed": 1,
          "results": 1,
          "userId": 1,
          "userData": { $arrayElemAt: [ "$userData", 0 ] }
        }
      }
    );

    return this;
  }

  /**
   *
   * @param {Array<{ [string]: any }>} pipeline
   * @returns {OrdersQueryBuilder}
   */
  addCustomPipeline(pipeline) {
    this.#aggregationPipeline.push(...pipeline);

    return this;
  }

  addFilteringByUser(query = '') {
    this.#aggregationPipeline.push(
      {
```

```
        $match: {
          $or: [
            { "userData.surname": new RegExp(`(.*)${query}(.*)`) },
            { "userData.email": new RegExp(`(.*)${query}(.*)`) }
          ]
        }
      }
    );

    return this;
  }

  addSortingByCreationDateDesc() {
    this.#aggregationPipeline.push({ $sort: { dateCreated: -1 } });

    return this;
  }

  /**
   *
   * @param {number} starting
   * @param {number} count
   * @returns {OrdersQueryBuilder}
   */
  addPagination(starting = 0, count = 100) {
    const $skip = isNaN(+starting) ? 0 : +starting;
    const $limit = isNaN(+count) ? 100 : +count;

    this.#aggregationPipeline.push({
      $facet: {
        paginatedResults: [{ $skip }, { $limit }],
        count: [{ $count: 'total' }]
      }
    });

    return this;
  }
}
```

Файл «api.js»:

```
import config from './libs/config.js';
import { EmployeeModel, UserModel, OrderModel, AnalysisModel } from './libs/mongoose.js';
import log from './libs/log.js';
import mongodb from 'mongodb';
import jwt from 'jsonwebtoken';
import nodemailer from 'nodemailer';
import { Router } from 'express';
import multer from 'multer';
import { Readable } from 'stream';
import Grid from 'gridfs-stream';
import mongoose from 'mongoose';
import OrdersQueryBuilder from "./libs/orders-query-builder.js";
import fs from 'fs';
import handlebars from 'handlebars';

const router = Router();

const storage = multer.memoryStorage()
const upload = multer({ storage: storage, limits: { fields: 1, fileSize: 6000000, files: 1,
parts: 2 }});

let verifyMailTemplate;

eval(`Grid.prototype.findOne = ${Grid.prototype.findOne.toString().replace('nextObject',
'next')}`);
```

```
const { connection, Types, mongo } = mongoose;

// api/users
router.post('/user/login', (req, res) => {
  const model = req.body.isEmployee ? EmployeeModel : UserModel;

  model.findOne({ password: req.body.data }, (err, user) => {
    if (err) {
      log.error('==Error login user/employee==', err);
      return res.status(500).json({ message: 'Internal server error' });
    }

    if (user) {
      if (!req.body.isEmployee && !user.verified) {
        log.error('==Email not verified error==');
        return res.status(401).json({ message: 'Email not verified' });
      }

      res.status(200);
      res.write(JSON.stringify({ body: { userId: user._id } }));
      log.info('==Login user/employee by email and password==');
    } else {
      log.error('==User not found==');
      res.status(404);
      log.info('==User/employee not found==');
    }
    res.end();
  });
});

router.post('/user/register', async (req, res) => {
  let user;

  try {
    user = await UserModel.findOne({ email: req.body.email });
  } catch (err) {
    log.error('==Error registering user - find user==', err);
    return res.status(500).json({ message: 'Error registering user' });
  }

  if (user) {
    return res.status(409);
  }

  try {
    await UserModel.create({ ...req.body, verified: false });
  } catch (err) {
    log.error('==Error registering user - create user==', err);
    return res.status(500).json({ message: 'Error registering user' });
  }

  try {
    const result = await sendVerificationMail(req.body.email);
    log.info('==Register user==');
    log.info('==Message sent successfully!==');
    log.info(result);
    res.status(200);
  } catch (err) {
    log.error('==Error registering user - send verification mail==', err);
    return res.status(500).json({ message: 'Error registering user' });
  }

  res.end();
});

router.post('/user/resendVerificationEmail', async (req, res) => {
  try {
    log.info('==Resend verification mail==');
```

```
    const result = await sendVerificationMail(req.body.email);
    log.info('==Message sent successfully!==');
    log.info(result);
    res.sendStatus(200);
  } catch (err) {
    res.status(500).json({ message: 'Error resending verification mail' });
  }
});

router.get('/user/:id', (req, res) => {
  UserModel.findOne({ _id: req.params.id }, (err, user) => {
    if (err) {
      log.error('==Error find user==');
    } else {
      log.info('==Get user==');
      delete user._doc.password;
      res.json(user);
    }
    res.end();
  });
});

router.put('/user/:id', (req, res) => {
  UserModel.findOneAndUpdate({ _id: req.params.id }, req.body, (err) => {
    if (err) {
      log.error('==Error find user while updating==');
      res.end();
    } else {
      log.info('==Update user==');
      UserModel.findOne({ _id: req.params.id }, (err, user) => {
        if (err) {
          log.error('==Error find user after updating==');
        } else {
          log.info('==Get user==');
          delete user._doc.password;
          res.json(user);
        }
        res.end();
      });
    }
  });
});

// analyzes
router.get('/analyzes',  (req, res) => {
  AnalysisModel.find((err, analyzes) => {
    if (err) {
      log.error('==Error get analyzes==');
    } else {
      log.info('==Get analyzes==');
      res.json(analyzes);
    }
    res.end();
  });
});

// orders
router.post('/orders',  (req, res) => {
  const analyzes = req.body.analyzes.map(id => Types.ObjectId(id));
  const userId = Types.ObjectId(req.body.userId);
  if (!analyzes.length) {
    return res.status(400).json({ message: 'No analyzes selected' });
  } else if (!userId._id) {
    return res.status(400).json({ message: 'No userId passed' });
  }
  const order = new OrderModel({ analyzes, userId, dateCreated: new Date() });
  order.save(err => {
    if (err) {
```

```
      log.error('==Error save order in Mongo==', err);
    } else {
      log.info('==Save order==');
    }
    res.end();
  });
});

router.post('/orders/:id/result',  (req, res) => {
  OrderModel.findOne({ _id: req.params.id }).exec((err, order) => {
    if (err) {
      return res.status(404).json({ message: 'Order not found' });
    }

    upload.single('result')(req, res, (err) => {
      if (err) {
        return res.status(400).json({ message: 'Upload Request Validation Failed' });
      }

      // Covert buffer to Readable Stream
      const readableStream = new Readable();
      readableStream.push(req.file.buffer);
      readableStream.push(null);

      const bucket = new mongodb.GridFSBucket(connection.db, {
        bucketName: 'results'
      });
      const uploadStream = bucket.openUploadStream(req.file.originalname);
      const id = uploadStream.id;
      readableStream.pipe(uploadStream);

      uploadStream.on('error', err => {
        log.error('==Error uploading file==', err.message);
        return res.status(500).json({ message: 'Error uploading file' });
      });

      uploadStream.on('finish', () => {
        log.info('==File uploaded successfully==');

        if (!order.results) {
          order.results = [];
        }
        order.results.push(Types.ObjectId(id));
        order.dateProcessed = new Date();
        order.save((err) => {
          if (err) {
            log.error('==Error update order with result file id==', err);
            return res.status(500).json({ message: 'File uploaded successfully. Error update
order with result file id', fileId: id });
          }
          return res.status(201).json({ message: 'File uploaded successfully', fileId: id });
        });
      });
    });
  });
});

// userId in params only in GET requests
router.get('/user/:id/orders',  (req, res) => {
  const ordersAggregation = new OrdersQueryBuilder()
    .addBaseQuery()
    .addCustomPipeline([{ $match: { userId: Types.ObjectId(req.params.id) } }])
    .addSortingByCreationDateDesc()
    .addPagination()
    .getAggregation();

  ordersAggregation.exec((err, result) => {
    if (err) {
```

```
      log.error('==Error get orders==', err);
    } else {
      log.info('==Get orders==');

      return res.json({ orders: result[0]?.paginatedResults, count: result[0]?.count[0]?.total
});
    }
    res.end();
  });
});

router.get('/orders', (req, res) => {
  const ordersAggregation = new OrdersQueryBuilder()
    .addBaseQuery()
    .addFilteringByUser(req.query.surname)
    .addSortingByCreationDateDesc()
    .addPagination(req.query.start, req.query.num)
    .getAggregation();

  ordersAggregation.exec(async (err, result) => {
    if (err) {
      log.error('==Error get orders==', err);

      return res.status(500).json({ message: 'Error get orders' });
    } else {
      log.info('==Get orders==');

      const { paginatedResults, count: [{ total }] } = result[0];

      return res.json({ orders: paginatedResults, count: total });
    }
  });
});

router.get('/result/:id',  (req, res) => {
  const gfs = new Grid(connection.db, mongo);
  gfs.findOne({ _id: req.params.id, root: 'results' }, (err, file) => {
    if (err || !file) {
      log.error('==File not found==', err);
      return res.status(404).send({ message: 'File not found' });
    }

    res.set('Content-Type', file.contentType);
    res.set('Access-Control-Expose-Headers', 'Content-Disposition');
    res.set('Content-Disposition', `attachment; filename="${encodeURI(file.filename)}"`);

    const readStream = gfs.createReadStream({ _id: req.params.id, root: 'results' });

    readStream.on('error', err => {
      log.error('==File download error==', err);
      return res.status(500).json({ message: 'File download error' });
    });
    readStream.pipe(res);
  });
});

router.get('/verify', (req, res) => {
  const token = req.query.id;
  if (!token) {
    return res.sendStatus(403)
  }

  try {
    jwt.verify(token, 'secret', (e, decoded) => {
      if (e) {
        log.error('', e);
        return res.sendStatus(403)
      }
```

```
      const email = decoded.id;
      UserModel.updateOne({ email }, { verified: true }, (err) => {
        if (err) {
          log.error('==Error update user==', err);
          return res.end();
        }

        log.info('==Update user==');
        UserModel.findOne({ email }, (err, user) => {
          if (err) {
            log.error('==Error find user==', err);
            return res.end();
          }

          log.info('==Get user==');

          try {
            res.cookie('userId', user._doc._id.toString());
            res.cookie('userRole', user.role || 'client');
            res.writeHead(302, { 'Location': config.APP_BASE_URL }); // redirect
          } catch (err) {
            log.error('', err);
          }

          return res.end();
        });
      });
    });
  } catch (err) {
    log.error('', err)
    return res.sendStatus(403)
  }
})

function getVerifyEmailContent(linkPlaceholder) {
  if (!verifyMailTemplate) {
    const verifyMailBuffer = fs.readFileSync('./assets/verify_email.html', { encoding: 'utf8'
});
    const verifyMailText = verifyMailBuffer.toString();
    verifyMailTemplate = handlebars.compile(verifyMailText);
  }

  return verifyMailTemplate({ linkPlaceholder });
}

async function sendVerificationMail(email) {
  const date = new Date();
  const mail = {
    id: email,
    created: date.toString()
  }
  const token_mail_verification = jwt.sign(mail, 'secret', { expiresIn: '365d' });
  const url = `http://localhost:3000/api/verify?id=${token_mail_verification}`;
  const html = getVerifyEmailContent(url);
  const mailConfig = {
    from: `Medlab Project`,
    to: email, // list of receivers seperated by comma
    subject: 'Підтвердження адреси електронної пошти',
    html,
  };

  let transporter = nodemailer.createTransport({
    service: config.MAILER.service,
    auth: config.MAILER.credentials,
  });

  // send mail with defined transport object
```

```
  try {
    const result = await transporter.sendMail(mailConfig);
    transporter.close();
    return result;
  } catch (err) {
    log.error('error sending verification email', err)
    throw err;
  }
}

export default router;
```

Файл «server.js»:

```
import express from 'express';
import cors from 'cors';
import config from './libs/config.js';
import log from './libs/log.js';
import apiConfig from './api.js';

const app = express();
app.use(cors());
app.use(express.json()); //initialize json parser
app.use(express.urlencoded({ extended: true })); //parse url
app.use('/api', apiConfig);

app.listen(config.PORT, () => {
  log.info('Server start running on port ' + config.PORT);
});
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

на кваліфікаційну роботу бакалавра на тему:

«Розробка комп'ютерної системи для медичних лабораторій»

студентки групи 122-20з-1 Гришко Олени Олегівни

Керівник економічного розділу

доц. каф. ПЕП та ПУ, к.е.н Л.В. Касьяненко

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Пояснювальна_записка_Гришко.docx	Пояснювальна записка до дипломного проєкту. Документ Word.
Пояснювальна_записка_Гришко.pdf	Пояснювальна записка до дипломного проєкту в форматі PDF
Програма	
med-lab.zip	Архів. Містить коди програми
Додаткові файли	
Example.pdf	Приклад файлу результатів досліджень клієнта
Презентація	
Презентація_Гришко.pptx	Презентація дипломного проєкту