Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

**МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА**

на тему:

# ІНФОРМАЦІЙНА WEB-СИСТЕМА ОБСЛУГОВУВАННЯ ТА ДІАГНОСТУВАННЯ ПАЦІЄНТІВ У МЕДИЧНОМУ ЗАКЛАДІ З ВИКОРИСТАННЯМ ДЕРЕВА РІШЕНЬ

Виконав: студент 2 курсу, групи 2КІ-22м
спеціальності 123 — «Комп'ютерна інженерія»
_____ Степанов О. Д.

Керівник: д.т.н., проф. каф. ОТ
_____ Азаров О. Д.
«15» 12 2023 р.

Опонент: к.т.н., доц. каф. МБІС
_____ Грицак А. В.
«18» 12 2023 р.

**Допущено до захисту**

Завідувач кафедри ОТ
_____ д.т.н., проф. Азаров О. Д.
«15» 12 2023 р

ВНТУ 2023

**ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Галузь знань — Інформаційні технології
Освітній рівень — магістр
Спеціальність — 123 Комп'ютерна інженерія
Освітня програма — Комп'ютерна інженерія

**ЗАТВЕРДЖУЮ**

Завідувач кафедри обчислювальної техніки

О.Д. Азаров

"26" вересня 2023 р.

**З А В Д А Н Н Я**

**НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ**

студенту Степанову Олексію Дмитровичу

1 Тема роботи «Інформаційна web-система обслуговування та діагностування пацієнтів у медичному закладі з використанням дерева рішень» керівник роботи Азаров Олексій Дмитрович д.т.н., професор, затверджено наказом вищого навчального закладу від **18.09.23** року **№ 247.**

2 Строк подання студентом роботи **19.12.23**.

3 Вихідні дані до роботи: призначення : обслуговування та діагностування пацієнта, засоби — середовище програмування Visual Studio Code , мови програмування React, JavaScript, Node.js, база даних mongoDB.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз технологій обслуговування та діагностування, проектування моделі та принципів роботи web-додатку, розробка функціоналу web-додатку, тестування web-системи медичного закладу.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): блок-схема алгоритму роботи web-додатку, блок-схема алгоритму дерева рішень.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1— Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1-4 | Азаров Олексій Дмитрович д.т.н., професор | | |
| 1-4 | Кадук Олександр Володимирович | | |
| 5 | Небава Микола Іванович проф., к.е.н | | |

7 Дата видачі завдання **19.09.2023** .

8 Календарний план виконання МКР приведений в таблиці 2.

Таблиця 2 — Календарний план

| № з/п | Назва етапів МКР | Строк виконання | Підпис |
|---|---|---|---|
| 1 | Постановка задачі | **19.09.23** | |
| 2 | Огляд існуючих рішень | **20.09.23** | |
| 3 | Розробка структури рішення | **28.09.23** | |
| 5 | Проектування дерева рішень | **5.10.23** | |
| 6 | Вибір ПЗ для розробки | **15.10.23** | |
| 7 | Розробка роботи системи | **23.10.23** | |
| 8 | Розрахунок економічної частини | **2.11.23** | |
| 9 | Оформлення пояснювальної записки та ілюстративного матеріалу | **12.11.23** | |
| 10 | Виконання магістерської кваліфікаційної роботи | **25.11.23** | |
| 11 | Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків | **4.12.23** | |
| 12 | Підписи супроводжувальних документів у керівника, опонента, нормоконтролера | **8.12.23** | |
| 13 | Перевірка «антиплагіат» | **8.12.23** | |
| 14 | Попередній захист | **12.12.23** | |

Студент Степанов Олексій Дмитрович

Керівник д.т.н., проф. Азаров Олексій Дмитрович

# АНОТАЦІЯ

УДК 004

Степанов О. Д. Інформаційна web-система обслуговування та діагностування пацієнтів у медичному закладі з використанням дерева рішень. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна Інженерія, Вінниця: ВНТУ, 2023 — 105 с. На укр. мові. Бібліогр.: 24 назв; рис.: 30; табл. 6.

У роботі розглянуто технології для обслуговування та діагностування, зпроектовано дерево рішень, вибрано та обгрунтувано підходи до організаціїї web-системи, розроблено клієнтську та серверну частину web-системи обслуговування та діагностування, написані рекомендації з розгортання системи.

Ключові слова: обслуговування, дерево рішень, діагностування, web-система.

# ABSTRACT

УДК 004

Stepanov, O. D. Information web-system for servicing and diagnosing patients in a medical institution using a decision tree. Master's thesis in the field of 123 — Computer Engineering, Vinnitsa: VNTU, 2023 - 105 p. In Ukrainian. Bibliography: 24 titles; figures: 30; tables: 6.

The paper considers technologies for maintenance and diagnostics, designs a tree of decisions, selects and justifies approaches to the organization of the web-system, develops the client and server parts of the web-system of maintenance and diagnostics, writes recommendations for deploying the system.

Keywords: service, decision tree, diagnostics, web-system..

**ЗМІСТ**

# ВСТУП

Створення системи діагностування для пацієнтів лікарні є важливим етапом в розвитку технологічних рішень для збору та аналізу даних від пацієнта та видачі йому правильного діагнозу для системи охорони здоров'я.

Основними функціями такого додатку буде вибір лікаря, запис в реєстратурі, збір даних пацієнта щодо симптомів та видача діагнозу пацієнту. Це зекономить час пацієнта і чергового або сімейного лікаря, зменшить кількість живих черг і мінімізує актуальну можливість захворіти у лікарні. Хоч діагноз і приблизний, завчасний збір анамнезу та попередній діагноз пацієнта в подальшому полегшить взаємодію та пришвидчить час прийому пацієнта вже безпосередньо в лікарні, тобто цей додаток зможе легко діагностувати легкі хвороби, а у випадках хвороб, що не є дуже поширеними, додаток звузить коло допустимих діагнозів до мінімуму, і цим полегшить подальшу роботу лікаря.

Створення інформаційних систем для діагностування — це надважливе завдання, тому що у часи пандемій та сезонних вірусів надзвичайно важливо не підхопити ще одне додаткове захворювання просто перебуваючи у черзі до свого сімейного лікаря, чи лікаря будь-якої іншої практики. А також при сезонних напливах вірусів такий додаток допоможе уникнути переповнення у лікарнях, чим дуже допоможе лікарям.

**Актуальність** дослідження полягає в тому, що завдяки web технологіям користувачі мають можливість здійснювати запис до лікаря та попередньо діагностуватися онлайн. Це не лише економить час користувача на поїздку до лікарні, а й зменшує вірогідність заразитися або заразити когось у лікарні.

**Метою** роботи є розширення функціональних можливостей web-додатку для обслуговування та діагностування пацієнтів за рахунок проектування дерева рішень для проведення адаптивного збору анамнезу пацієнта, що дозволить ефективно його попередньо діагностувати.

Для досягнення поставленої мети слід виконати такі **задачі**:

— проаналізувати аналоги веб-додатку;

— сформулювати основні вимоги до веб-додатку;

— спроектувати дерево рішень для діагностування;

— обрати архітектуру та технології для розробки веб-додатку;

— розробити структуру та визначити особливості реалізації інформаційного забезпечення web-додатку;

— спроектувати інтерфейс та функціонал системи.

**Об’єктом дослідження** є інформаційне обслуговування лікарень та оптимізація їхнього функціонування для забезпечення ефективного попереднього діагностування.

**Предметом дослідження** є технології та функції, що пов’язані з інформаційним обслуговуванням лікарень, запис до лікаря та діагностування хвороби пацієнта відносно його симптомів.

**Методи дослідження**: методи системного та статистичного аналізу для розрахунку статистичних даних, методи алгоритмічного проектування для розробки алгоритму та структурної схеми web-системи.

**Наукова новизна одержаних результатів** полягає в тому, що набув подальшого розвитку метод діагностування за допомогою дерева рішень, в якому проводиться адаптивний збір анамнезу пацієнта на основі його відповідей, чим підвищує ефективність діагностування та економить ресурси серверу.

**Практичне значення** роботи полягає в тому, що впровадження запропонованої системи дозволяє пацієнту отримати ефективне попереднє діагностування онлайн, чим полегшить роботу лікаря та зекономить час пацієнта.

**Апробація** результатів роботи здійснена у доповіді на Міжнародній науково-практичній Інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи» (2023)

# 1 АНАЛІЗ ТЕХНОЛОГІЙ ОБСЛУГОВУВАННЯ ТА ДІАГНОСТУВАННЯ

## 1.1 Огляд сучасного стану обробки персональних даних в медичних закладах

Сучасний стан обробки персональних даних в медичних закладах є надзвичайно важливим аспектом у сфері охорони здоров'я. З розвитком технологій і збільшенням обсягів збирання, зберігання та обробки медичних даних стали виникати нові виклики та питання щодо конфіденційності, безпеки та етики. Давайте розглянемо деякі з цих аспектів більш детально.

Законодавча база грає критичну роль у цьому контексті. В багатьох країнах у світі прийнято закони та нормативні акти, що регулюють обробку персональних даних в медичних закладах. Зазвичай, це є загальні закони про захист даних та закони, специфічні для сфери охорони здоров'я.

Медичні заклади збирають великі обсяги даних про пацієнтів, включаючи особисті дані, історію захворювань та результати обстежень. Забезпечення безпеки та конфіденційності цих даних важливо для пацієнтів і закладу. В цьому контексті важливо враховувати інформаційні технології, які допомагають у збереженні та організації цих даних.

Дозвільний режим та контроль доступу до даних також є суттєвими аспектами. Медичні заклади повинні встановити чіткі правила щодо доступу до персональних даних і дотримуватися їх. Зазвичай, лише обмежена кількість медичних фахівців має доступ до конфіденційної інформації.

Запровадження електронних медичних записів (EMP) і систем обробки даних стало нормою, полегшуючи доступ до інформації, але вимагаючи додаткових заходів забезпечення кібербезпеки. Ризик кібератак зростає разом з цифровими технологіями, тому важливо вживати заходи забезпечення кібербезпеки для захисту медичних даних від несанкціонованого доступу.

Питання етики обробки медичних даних набувають на важливості. Лікарі та медичні працівники повинні дотримуватися коду деонтології та етичних норм у відношенні збору та використання даних пацієнтів.

Законодавство в деяких країнах надає пацієнтам право отримувати копії своїх медичних даних, і медичні заклади повинні бути готові відповідати на такі запити та забезпечувати доступ до даних.

Штрафи та відповідальність за порушення законодавства щодо обробки даних стимулюють медичні заклади дотримуватися норм та стандартів. Порушення може призвести до серйозних наслідків.

Важливим аспектом є також використання медичних даних для досліджень та покращення методів лікування. У цьому важливо забезпечити анонімізацію даних та дотримання етичних норм у цілях досліджень.

Загальною тенденцією є зростання уваги до правових, етичних та технічних аспектів обробки медичних даних в медичних закладах. Забезпечення безпеки, конфіденційності та ефективного використання даних є ключовим завданням для забезпечення якості медичної допомоги та дотримання прав пацієнтів.

## 1.2 Застосування інформаційних технологій у медичному секторі

Застосування інформаційних технологій (IT) в медичному секторі має велике значення для покращення якості медичної допомоги, ефективності та доступності медичних послуг, а також для збереження та обробки медичних даних.

Одною з основних сфер застосування інформаційних технологій в медицині є електронні медичні записи (EMP) [1] — це цифрові версії медичних записів пацієнтів, які включають історію захворювань, рецепти, результати обстежень та інші. Вони полегшують доступ до інформації, покращують координацію медичної допомоги та зменшують ризик помилок (рисунок 1.1).

ІТ дозволяють здійснювати консультації та обстеження віддалено (телемедицина) [2]. Це особливо важливо для пацієнтів, які мешкають в віддалених регіонах або мають обмежену фізичну можливість для відвідування лікаря.

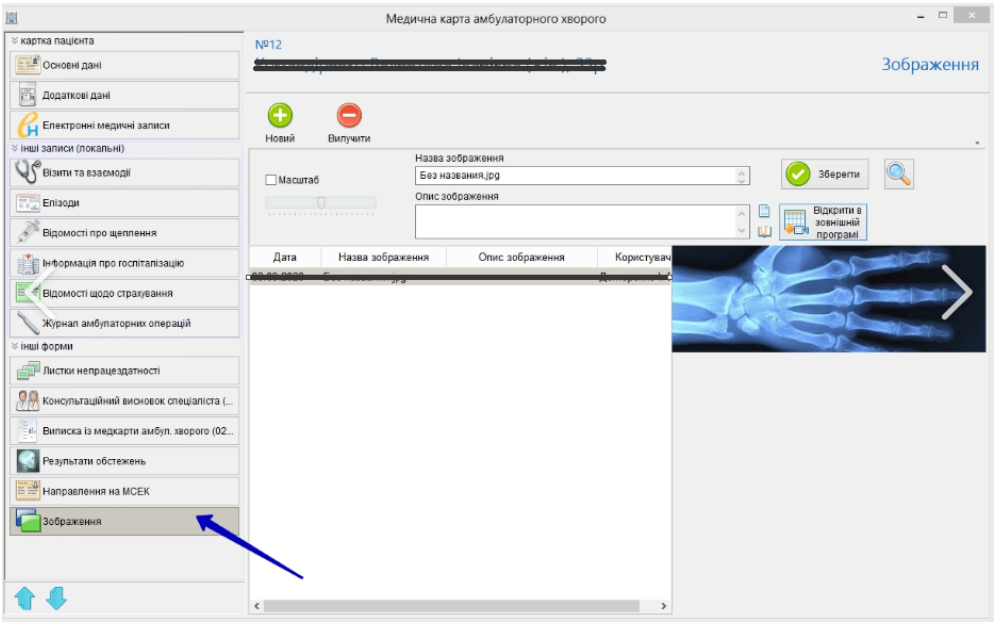



Рисунок 1.1 — Приклад EMP

ІТ розвивають медичні додатки для моніторингу здоров'я, прийому рецептів, нагадування про прийом ліків та багато іншого. Ці додатки полегшують пацієнтам дотримання медичних рекомендацій (рисунок 1.2).

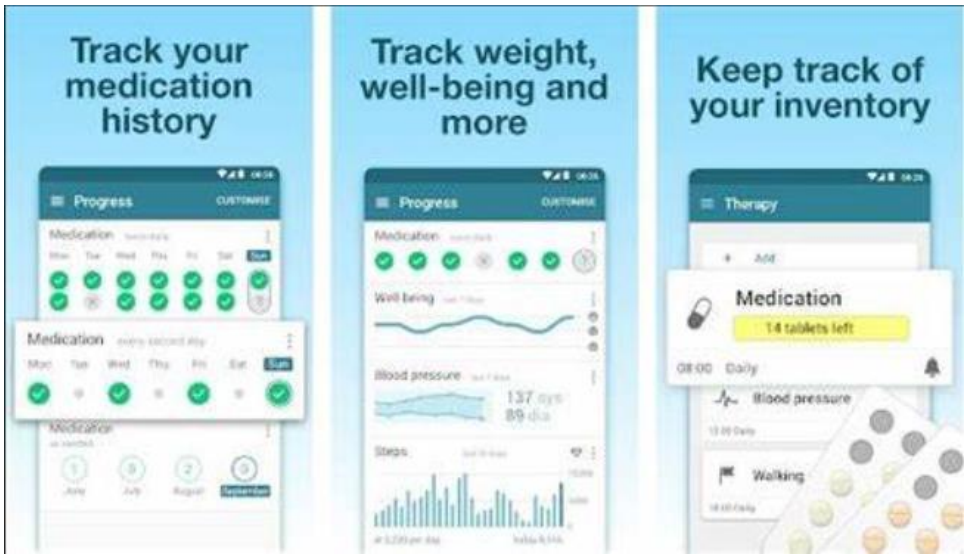



Рисунок 1.2 — Додаток MyTherapy

За допомогою аналізу даних і штучного інтелекту (ШІ), лікарі можуть виявити тенденції, діагнози та лікування на основі великих обсягів медичної інформації. ШІ також використовується для розробки прогностичних моделей та покращення точності діагнозів (рисунок 1.3).

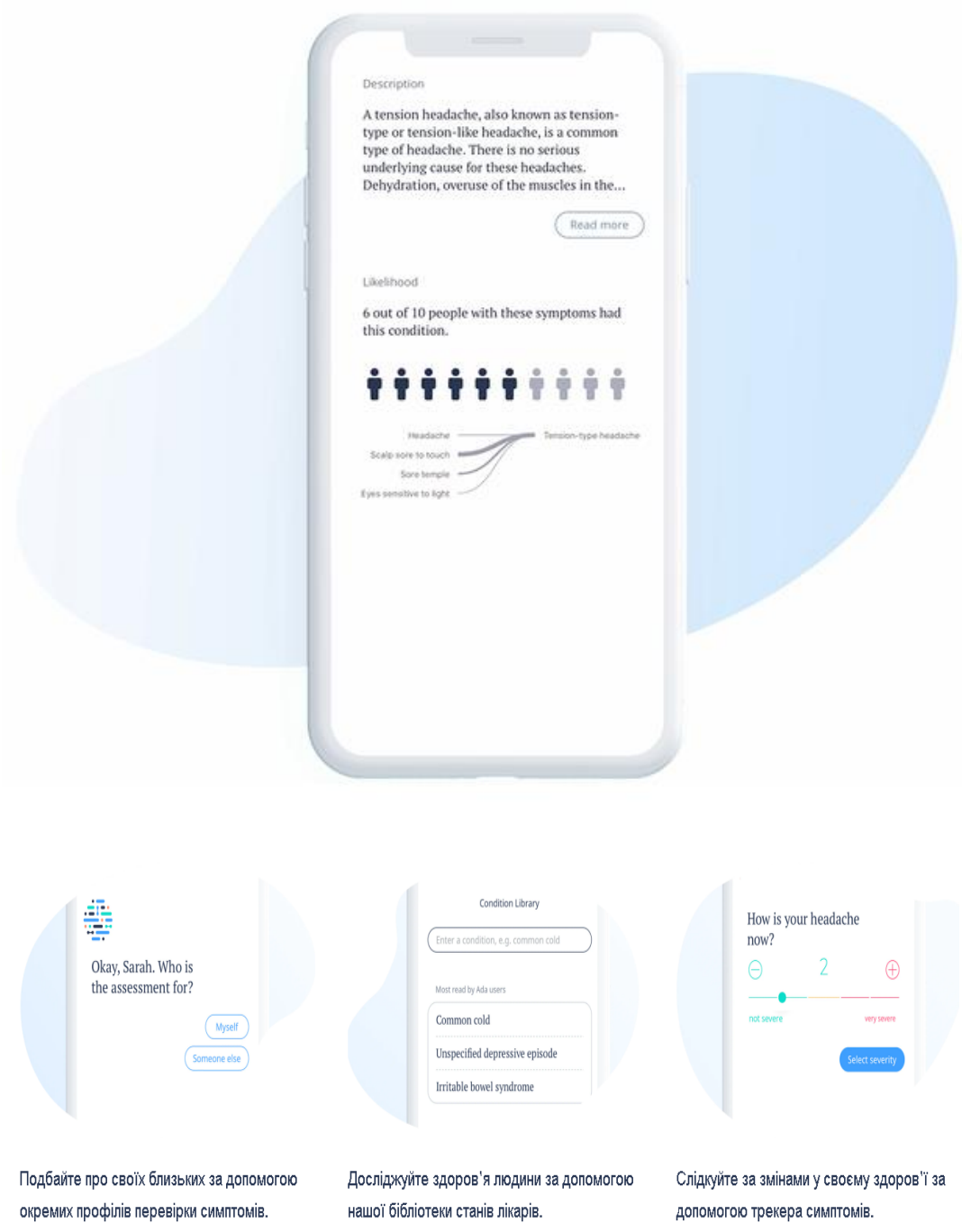



Рисунок 1.3 — Мобільний додаток Ada

ІТ грають ключову роль у забезпеченні безпеки та конфіденційності медичних даних. Застосування шифрування, автентифікації та інших технологій допомагає запобігти несанкціонованому доступу до даних.

Інтернет речей (IoT)-пристрої, такі як зв'язані медичні пристрої та датчики, дозволяють моніторити стан здоров'я в реальному часі та передавати дані лікарям (рисунок 1.4) [3].

Рисунок 1.4 — Інсулінова помпа (IoT)

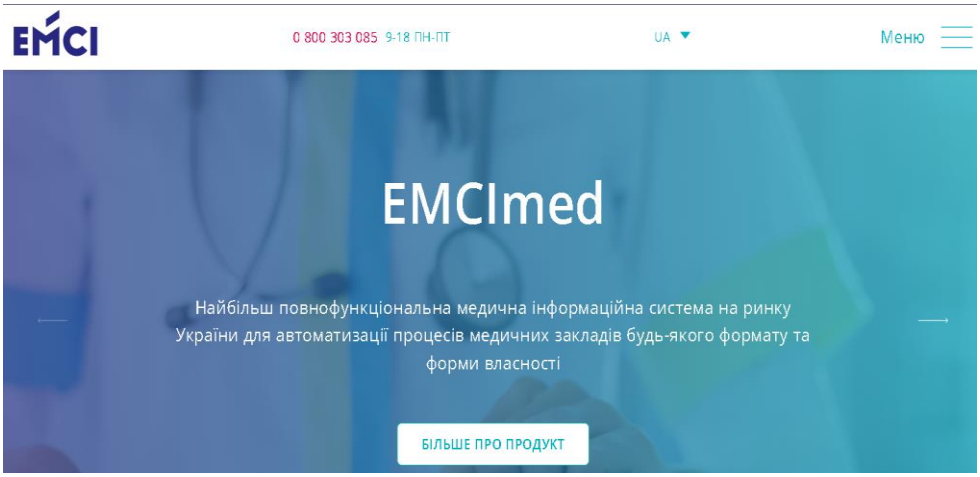
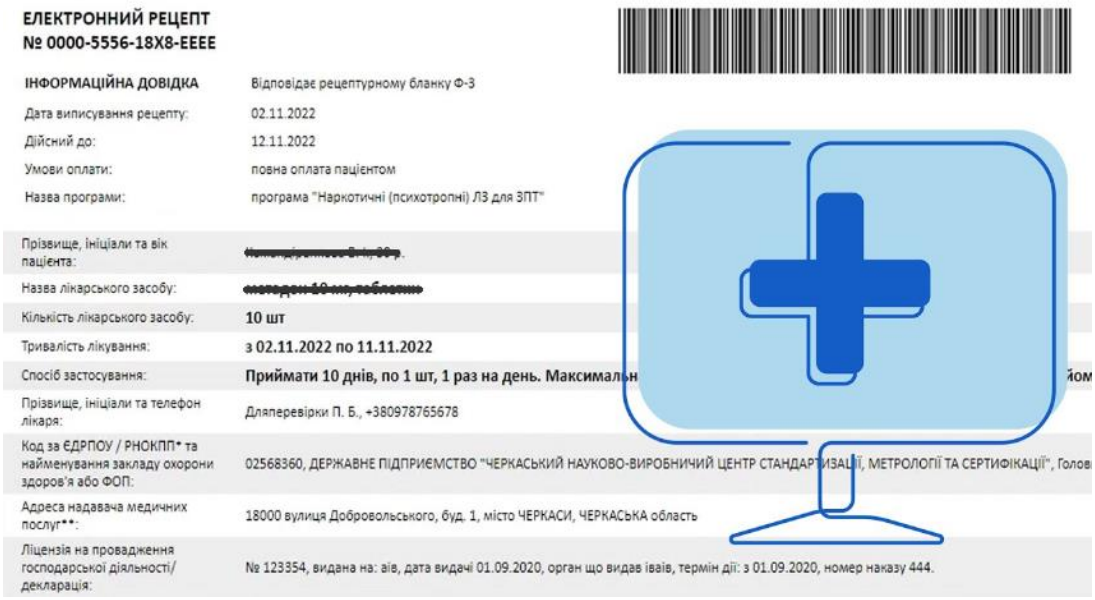



Рисунок 1.5 — Інформаційна система EMCI

ІТ використовуються для автоматизації бізнес-процесів в медичних закладах, включаючи прийом пацієнтів, управління запасами та фінансовий облік (рисунок 1.5) [4].

Цифрові системи рецептури спрощують виписування ліків та сприяють підвищенню безпеки та ефективності лікування (рисунок 1.6) [5].

ІТ використовуються для навчання медичних працівників, симуляційного навчання та віддаленої освіти (рисунок 1.7) [6].



**ЕЛЕКТРОННИЙ РЕЦЕПТ**
**№ 0000-5556-18X8-EEEE**

| | |
|---|---|
| **ІНФОРМАЦІЙНА ДОВІДКА** | Відповідає рецептурному бланку Ф-3 |
| Дата виписування рецепту: | 02.11.2022 |
| Дійсний до: | 12.11.2022 |
| Умови оплати: | повна оплата пацієнтом |
| Назва програми: | програма "Наркотичні (психотропні) ЛЗ для ЗПТ" |
| Прізвище, ініціали та вік пацієнта: | [redacted] |
| Назва лікарського засобу: | [redacted] |
| Кількість лікарського засобу: | **10 шт** |
| Тривалість лікування: | **з 02.11.2022 по 11.11.2022** |
| Спосіб застосування: | **Приймати 10 днів, по 1 шт, 1 раз на день. Максимальн** |
| Прізвище, ініціали та телефон лікаря: | Дляперевірки П. Б., +380978765678 |
| Код за ЄДРПОУ / РНОКПП* та найменування закладу охорони здоров'я або ФОП: | 02568360, ДЕРЖАВНЕ ПІДПРИЄМСТВО "ЧЕРКАСЬКИЙ НАУКОВО-ВИРОБНИЧИЙ ЦЕНТР СТАНДАРТИЗАЦІЇ, МЕТРОЛОГІЇ ТА СЕРТИФІКАЦІЇ", Голов |
| Адреса надавача медичних послуг**: | 18000 вулиця Добровольського, буд. 1, місто ЧЕРКАСИ, ЧЕРКАСЬКА область |
| Ліцензія на провадження господарської діяльності/ декларація: | № 123354, видана на: ав, дата видачі 01.09.2020, орган що видав іваів, термін дії: з 01.09.2020, номер наказу 444. |

Рисунок 1.6 — Приклад цифрового рецепту

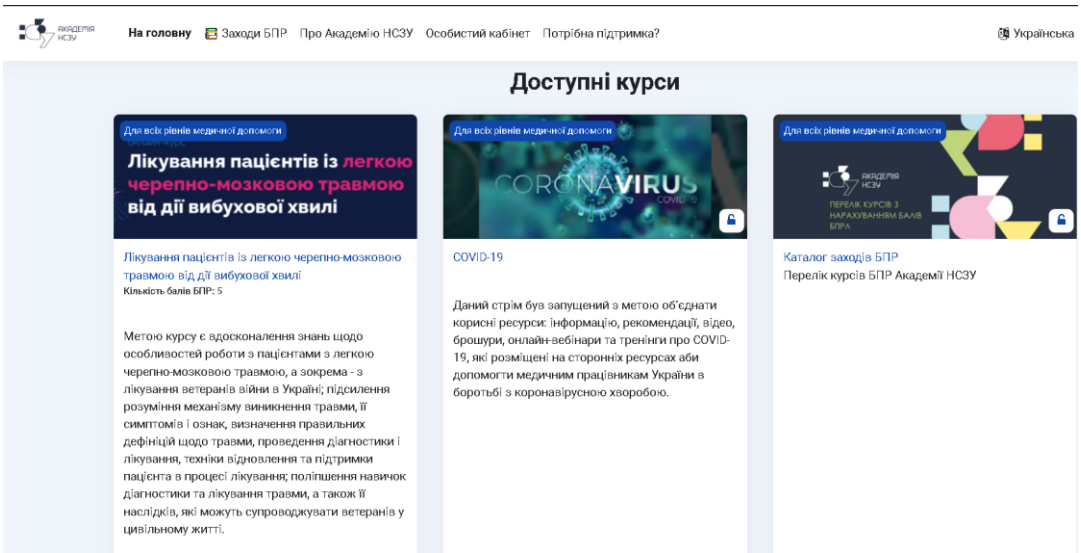



Рисунок 1.7 — Ресурс для навчання медичних працівників

## 1.3 Роль web-систем у покращенні обробки та доступу до медичної інформації

Web-системи грають ключову роль у покращенні обробки та доступу до медичної інформації в сучасній медицині. Вони надають багато переваг для пацієнтів, медичних працівників та медичних закладів.

Вони дозволяють створювати та зберігати цифрові версії медичних записів пацієнтів. Це полегшує доступ до інформації про стан здоров'я пацієнта для лікарів і медичного персоналу в будь-якому місці з доступом до Інтернету.

Можна легко адаптувати web-системи для мобільних пристроїв, що дозволяє лікарям та медичному персоналу отримувати доступ до медичної інформації під час надання медичної допомоги в лікарнях, вдома або навіть на віддалених місцях.

Web-системи надають можливість створювати пацієнтські портали, де пацієнти можуть переглядати свої EMP, результати обстежень, призначені рецепти та інші медичні дані. Це підвищує обізнаність пацієнтів та дозволяє їм брати активну участь у власному здоров'ї.

Для проведення віддалених консультацій та обстежень саме web-системи дозволяють використовувати відповідні технології. Лікарі можуть спілкуватися з пацієнтами через відеозв'язок та розглядати їхні медичні дані в реальному часі.

Web-системи дозволяють збирати, обробляти та аналізувати великі обсяги медичних даних для виявлення тенденцій та розробки нових методів діагностики та лікування.

Для забезпечення безпеки медичної інформації та дотримання вимог щодо конфіденційності Web-системи допомагають встановлювати механізми автентифікації та шифрування.

Web-системи використовуються для автоматизації процесів управління медичними закладами, включаючи прийом пацієнтів, облік ліків, фінансове планування та багато інших аспектів.

Також web-системи допомагають знижувати час обробки даних, запитів та обміну інформацією, що полегшує роботу медичних закладів та сприяє швидкішому наданню медичної допомоги.

В медицині web-системи стали невід'ємною частиною сучасної медичної практики, полегшуючи доступ до медичної інформації, покращуючи якість медичних послуг та сприяючи більш ефективному управлінню медичними закладами.

## 1.4 Аналіз сучасних web-систем для медичних закладів

Сучасний ринок web-систем для медичних закладів дуже різноманітний і постійно зростає.

Epic Systems Corporation, або Epic — американська приватна компанія-розробник програмного забезпечення для охорони здоров'я. За даними компанії, у 2022 році лікарні, які використовують її програмне забезпечення, мали медичні записи 78% пацієнтів у Сполучених Штатах і понад 3% пацієнтів у всьому світі [7].

Epic в основному розробляє, виробляє, ліцензує, підтримує та продає власне програмне забезпечення для електронних медичних записів, відоме як «Epic» або Epic EMR. Програмне забезпечення компанії для охорони здоров'я зосереджено на системі управління базами даних Chronicles. Додатки Epic підтримують функції, пов'язані з доглядом за пацієнтами, включаючи реєстрацію та планування; клінічні системи для лікарів, медсестер, персоналу швидкої допомоги та інших постачальників медичної допомоги; системи для лаборантів, фармацевтів та радіологів; білінгові системи для страховиків.

Epic також пропонує хмарний хостинг для клієнтів, які не бажають обслуговувати власні сервери; а також короткострокові консультанти з оптимізації та впровадження через їхню дочірню компанію Boost, Inc.

NextGen Healthcare, Inc. — американська компанія з розробки програмного забезпечення та послуг зі штаб-квартирою в Атланті, штат Джорджія. Компанія розробляє та продає програмне забезпечення для електронних медичних карток (EHR) та системи управління практикою для галузі охорони здоров'я. NextGen Healthcare також забезпечує здоров'я населення, фінансовий менеджмент та клінічні рішення[модне слово] для медичної та стоматологічної практики [8].

Електронна медична картка NextGen Enterprise. NextGen EHR — це, перш за все, єдина, всеосяжна база даних, яка поєднує в собі управління користувачами, EHR і EPM, ця база даних може працювати під управлінням MS SQL Server 2005, 2008 або Oracle.

Управління практикою NextGen, портал пацієнтів NextGen, центр даних про здоров'я NextGen та заходи якості здоров'я NextGen використовують той самий сервер бази даних, що й основний продукт EHR.

McKesson Corporation — американська компанія, яка займається дистрибуцією фармацевтичних препаратів і надає інформаційні технології в галузі охорони здоров'я, медичні товари та інструменти управління доглядом. Компанія постачає третину всіх фармацевтичних препаратів, що використовуються в Північній Америці [9].

Компанія McKesson займається дистрибуцією систем охорони здоров'я, медичних товарів та фармацевтичної продукції. Крім того, McKesson забезпечує розгалужену мережеву інфраструктуру для галузі охорони здоров'я; Крім того, він був першим, хто застосував такі технології, як сканування штрих-кодів для розповсюдження, аптечна робототехніка та RFID-мітки.

McKesson Medical-Surgical (MMS) пропонує великий вибір національних брендів охорони здоров'я, а також ексклюзивний бренд медичних виробів McKesson. Їхня онлайн-платформа для замовлення медичних товарів обслуговує потреби кабінетів лікарів, хірургічних центрів,

агентств охорони здоров'я на дому, DME, лабораторій та закладів довгострокового догляду.

Zocdoc — це платформа, яка дозволяє пацієнтам знаходити лікарів та записувати на прийом. Вона також надає користувачам можливість вводити свої симптоми та отримувати рекомендації стосовно лікарських спеціалістів [10]. Zocdoc надає систему планування на основі платної підписки для медичного персоналу. Система планування може бути доступна абонентам як у вигляді онлайн-сервісу, так і через розгорнуте програмне забезпечення офісного календаря або інтегрована з їхніми веб-сайтами. Користувачі можуть перейти до Zocdoc для пошуку, використовуючи симптом, спеціальність, причину візиту або ім'я лікаря, а також інформацію про своє місцезнаходження та страхування, щоб переглянути доступні місця для запису. Потім вони можуть звузити коло пошуку, переглянувши розташування офісів, фотографії, інформацію про персонал, освіту та відгуки, надіслані користувачами (рисунок 1.8).

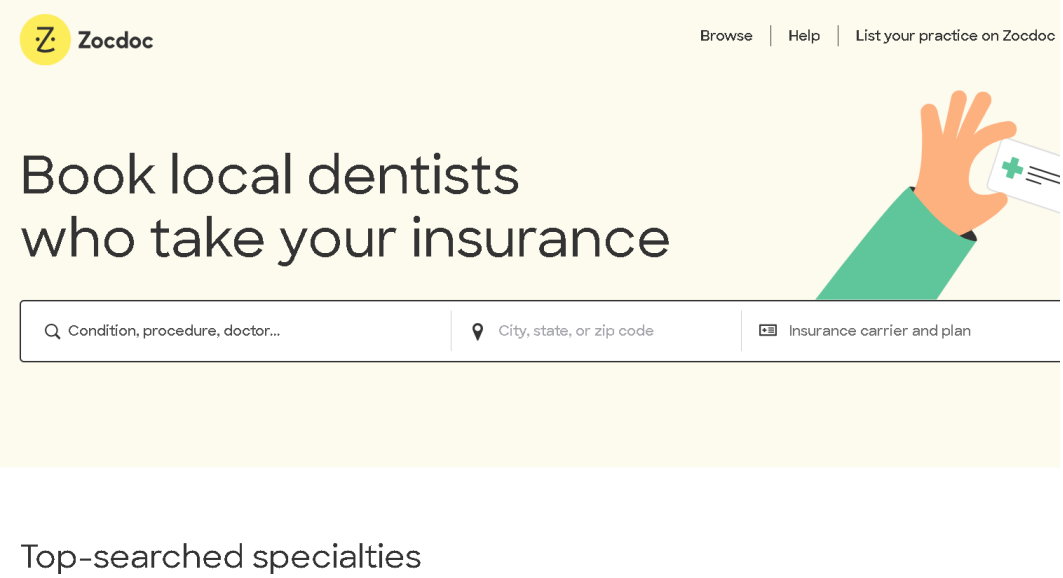



Рисунок 1.8 — Платформа Zocdoc

HealthTap — це платформа для телемедицини, яка дозволяє пацієнтам консультуватися з лікарями в онлайні та отримувати медичні поради та діагнози за допомогою введеного анамнезу та опису симптомів [11] (рисунок

1.9). HealthTap пропонує можливість негайно зв'язатися з лікарем або записатися на прийом до лікаря для консультації за допомогою відеоконференції, телефонного дзвінка або текстового чату через веб- або мобільний додаток. Медична інформація в службі надається в інтерактивному режимі мережею з майже 140 000 ліцензованих лікарів з хорошою репутацією (ліцензовані лікарі, які вчинили проти них активні дисциплінарні стягнення, не допускаються до мережі). HealthTap також надає можливість експертної оцінки, яка складається з оцінювання лікарів один одного та самовизначення спеціалізацій.

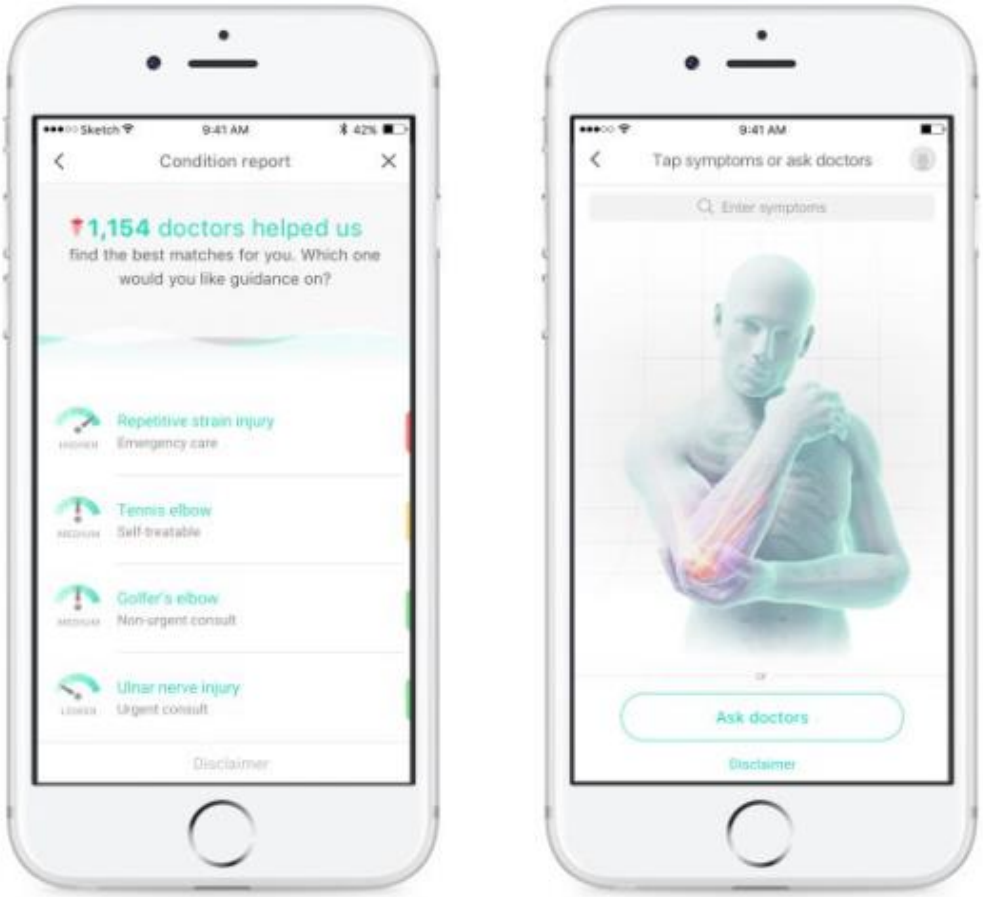



Рисунок 1.9 — Додаток HealthTap

Buoy Health — це інтерактивний додаток для самодіагностики. Він допомагає користувачам з'ясувати можливі причини своїх симптомів та отримувати рекомендації для дій [12]. Компанія Buoy розробила та пропонує інтерактивний веб-інструмент охорони здоров'я, який використовує штучний

інтелект, обробку природної мови та машинне навчання для оцінки відповідей користувача на низку запитань, пов'язаних зі здоров'ям та симптомами, що передаються через чат-бота (рисунок 1.10).

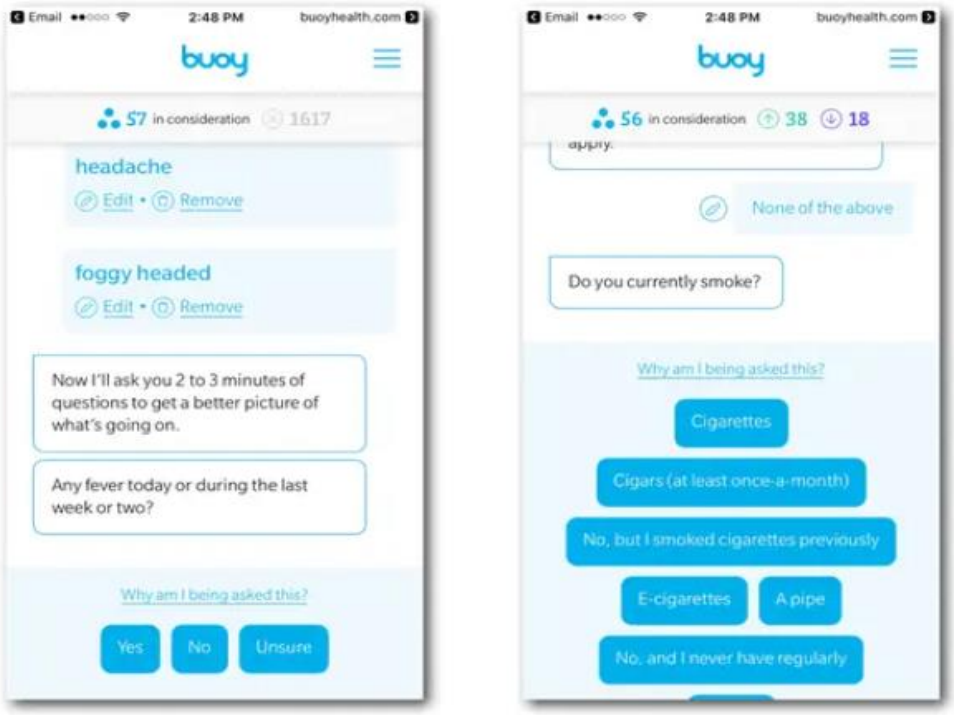



Рисунок 1.10 — Додаток Buoy Health

Your.MD — це платформа для самодіагностики, яка використовує штучний інтелект для надання порад стосовно здоров'я за допомогою чат-бота (рисунок 1.11) [13].

Ви можете говорити з ботом про свої захворювання в повній конфіденційності, і бот зрозуміє, поставить відповідні запитання відповідно до ваших симптомів, особистих факторів (стать, вік тощо) та історії здоров'я, щоб надати вам інформацію про найбільш ймовірну причину ваших симптомів. Це робиться за допомогою найкращого математичного підходу для моделювання розумового процесу лікаря;

Ви також можете здійснити пошук на сайті Your.MD від А до Я, щоб знайти інформацію за будь-яким запитом про здоров'я. Вся надана інформація отримана безпосередньо з NHS Choices і регулярно оновлюється за ліцензією.

Розробники також локалізували контент NHS Choices, щоб зробити його зручним для користувачів.

Якщо вам потрібно більше, ніж просто інформація, Your.MD зможете порекомендувати найбільш підходящі послуги та продукти, що відповідають вашим індивідуальним потребам. OneStop Health від Your.MD допоможе вам знайти найактуальніші медичні послуги у вашому регіоні та дозволить вам негайно отримати направлення до найбезпечніших, найбільш підходящих спеціалістів і служб. Незалежно від того, чи потрібно вам знайти місцевого лікаря або ви хочете отримати консультацію по відеозв'язку; Шукаєте інтернет-аптеку або хочете записатися на сеанс терапії, OneStop Health™ — єдине місце, куди вам потрібно піти. * Зверніть увагу, що деякі послуги, перелічені в OneStop Health, є преміум-класом, а деякі безкоштовні.

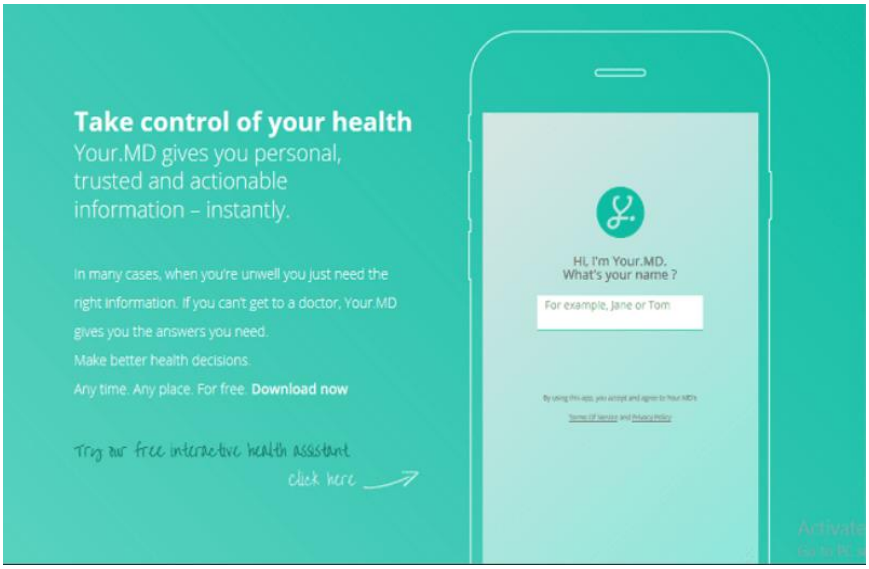



Рисунок 1.11 — Додаток Your.MD

# 2 ПРОЕКТУВАННЯ МОДЕЛІ ТА ПРИНЦИПІВ РОБОТИ WEB-ДОДАТКУ

## 2.1 Постановка задач розробки

Після огляду та аналізу сучасного стану обробки персональних даних в медичних закладах, важливої ролі web-систем у покращенні обробки даних та взаємодії з пацієнтом було визначено завдання, які необхідно виконати для розробки web-додатку:

— обрати та спроектувати загальну модель системи;

— розробити алгоритм роботи web-додатку;

— спроектувати принцип роботи web-додатку;

— провести тестування реалізованого додатку.

## 2.2 Інтеграція дерева рішень у web-додаток для діагностування

Дерево рішень — це графічна модель прийняття рішень, яка виглядає як дерево з різними гілками, представляючи послідовність рішень та їхні наслідки. Цей інструмент широко використовується в області штучного інтелекту, аналізу даних та управління проектами. Основна ідея полягає у візуалізації процесу прийняття рішень та визначенні оптимального шляху на основі визначених критеріїв.

Основними компонентами дерева рішень є:

— корінь (Root) — це початковий вузол, що представляє початкове становище або початкове питання;

— вузли рішень (Decision Nodes) — це вузли, в яких приймається рішення, зазвичай вони мають декілька гілок, які вказують на альтернативні варіанти рішень;

— гілки (Branches) — це певні відгалуження вузлів, що представляють різні альтернативи чи можливі наслідки;

— вузли випробувань (Chance Nodes) — це вузли, які представляють невизначеність чи випадкові події і кожна гілка виходить з такого вузла з певною ймовірністю;

— листки (Leaf Nodes) — це кінцеві вузли дерева, які представляють кінцеві результати або вихідні значення.

Процес прийняття рішень розглядається шляхом просування від кореня до листя, вибираючи альтернативи на кожному вузлі. Кожен шлях від кореня до листя представляє конкретний сценарій рішень та їхні наслідки.

Задачі прийняття рішень можуть бути дуже складними через велику кількість альтернатив, можливих наслідків і невизначеність. Дерева рішень надають структурований і візуальний спосіб аналізувати ці складність і допомагають приймати кращі рішення.

Реалізації роботи дерева рішень відбувається у кілька етапів:

— побудова дерева рішень;

— управління невизначеністю;

— оцінка ризиків і вигод;

— прийняття рішення;

— моніторинг і контроль.

Під час побудови дерева рішень слід чітко визначити проблему або завдання, до якого потрібно прийняти рішення. Потрібно встановити можливі варіанти рішень і дій які можна взяти під розгляд, а також проаналізувати можливі наслідки кожної альтернативи та їх вплив на кінцевий результат.

На етапі управління невизначеністю аналізуються невизначені аспекти, та використовуються вузли випробувань для моделювання ймовірності виникнення певних подій.

На третьому етапі оцінюються можливі негативні наслідки кожної альтернативи і розроблюються стратегії управління ризиками, а також враховуються можливі позитивні наслідки і визначається потенційна вигода кожного варіанту.

На етапі прийняття рішення проводиться аналіз всіх факторів та вибір альтернативи, яка найбільше відповідає поставленим цілям та обмеженням.

Під час моніторингу і контролю відбувається реалізація обраного рішення і введення його у життя. Після цього за виконанням даного рішення слідкують, і при необхідності, вносять корективи.

Дерева рішень застосовуються в багатьох галузях, включаючи бізнес, медицину, фінанси та інші. Вони допомагають аналізувати складні ситуації, узгоджувати різні аспекти та визначати оптимальні рішення з урахуванням багатьох факторів.

Для інтеграції дерева рішень у дану систему необхідно створити базу даних діагнозів та їх симптомів та реалізувати алгоритм побудови дерева. Стартовою точкою виконання алгоритму буде початковий вузол (корінь) дерева рішень, це буде перший симптом першого діагнозу. Залежно від відповідності, або невідповідності кореневого симптому відносно симптому пацієнта будуються вузли рішень.

Згідно алгоритму, якщо кореневий симптом відповідає симптому пацієнта, то вузлом відгалуження стане другий симптом першого діагнозу, якщо і другий симптом відповідає, то вузлом стає 3 симптом діагнозу. У іншому випадку, тобто коли початковий вузол не відповідає симптому, то вузлом відгалуження стане симптом наступного діагнозу у базі, що не включає кореневого симптому, так усі діагнози що мають кореневий симптом стають не валідними. Такі розгалуження будуть будуватися, до поки користувацькі симптоми не співпадуть повністю з симптомами діагнозу в базі даних.

## 2.3 Проектування загальної моделі системи

Інформаційна система — це комплекс взаємопов'язаних елементів, які збирають, обробляють, зберігають і надають інформацію для досягнення певних цілей організації чи користувача. Ця інформація може включати дані, процеси обробки, апаратні та програмні засоби, а також людей, які взаємодіють з системою. Інформаційні системи можуть бути використані у різних сферах, включаючи бізнес, науку, охорону здоров'я, освіту та інші.

Експертні системи — це тип інформаційної системи, яка використовується для моделювання та автоматизації прийняття рішень у певній галузі, яка вимагає експертного знання. Головною метою експертних систем є розробка програм, які можуть відтворювати знання та досвід експертів у конкретній області, і використовувати ці знання для розв'язання складних завдань.

Основні характеристики експертних систем включають:

— базу знань, яка складається з фактів, правил і евристичних правил, що відображають експертне знання у конкретній галузі;

— інференційний механізм для логічного виведення та прийняття рішень на основі доступних знань;

— система обговорення, в якій користувач може взаємодіяти з експертною системою, вводячи запити та надаючи інформацію, а система може запитувати уточнюючі питання;

— експертні системи можуть враховувати нечітку або невизначену інформацію, що робить їх корисними в галузях, де немає чітких правил.

Приклади використання експертних систем включають системи діагностики медичних захворювань, системи підтримки прийняття рішень в фінансовій аналітиці, системи управління технічним обслуговуванням у виробництві та багато інших сферах. Експертні системи допомагають автоматизувати процес прийняття рішень на основі експертних знань, що робить їх корисними для оптимізації складних завдань і підвищення продуктивності.

Методологія об’єктно-орієнтованої моделі припускає конструювання програмного рішення з готових об’єктів, для яких визначаються правила їх взаємодії, що переводять об’єкти з одного стану в інший. Така модель, що передбачає повну відповідність процесу розробки положенням об’єктно-орієнтованої методології (об’єктно-орієнтований аналіз, проектування, програмування), ефективна у великих проектах.

Архітектура інформаційних систем — це структурна організація компонентів та їх взаємодія, яка визначає, як інформаційна система буде побудована і як вона буде функціонувати. Вона включає в себе планування розташування компонентів, комунікацію між ними, обробку даних і контроль доступу.

До основних видів архітектури інформаційних систем належать:

— монолітна архітектура;

— трьохрівнева архітектура;

— мікросервісна архітектура;

— спостережувана архітектура;

— клієнт-серверна архітектура;

— розподілена архітектура;

— хмарні інформаційні системи.

Монолітна архітектура інформаційної системи — це традиційний підхід до розробки програмного забезпечення, при якому всі компоненти та функції системи розташовані в одному єдиному блоку або програмі. В інших словах, всі компоненти програми розташовані в одному "моноліті", і вони взаємодіють один з одним без використання окремих служб або модулів.

Монолітні системи можуть стати великими та складними, що ускладнює розробку та підтримку. Відсутність чіткого розділення функціональності на окремі модулі робить важкою зміну чи розширення системи без значних зусиль. Всі зміни вносяться в один великий блок, що може призвести до проблем зі сумісністю та версіонуванням. Масштабування монолітних систем може бути складним завданням, оскільки всі компоненти об'єднані в одному процесі.

Хоча монолітні архітектури використовуються досить давно і є дієвими для деяких застосувань, вони також мають свої обмеження. Вони можуть стати проблематичними при роботі з великими і складними проектами, оскільки ускладнюють розробку, підтримку та масштабування. У сучасному програмуванні широко використовуються більш розгалужені архітектурні

підходи, такі як мікросервіси, що дозволяють розділити функціональність на окремі модулі для полегшення розробки та підтримки системи.

Трьохрівнева архітектура — це підхід до організації інформаційних систем, який розділяє систему на три основних рівні або компоненти, кожен з яких виконує конкретні функції. Ця архітектурна модель спроектована для забезпечення більшої модульності, розширюваності та підтримки у розробці та обслуговуванні інформаційних систем (рисунок 2.1).

Рівнями трьохрівневої архітектури є:

— презентаційний рівень;

— бізнес-логічний рівень;

— рівень доступу до даних.

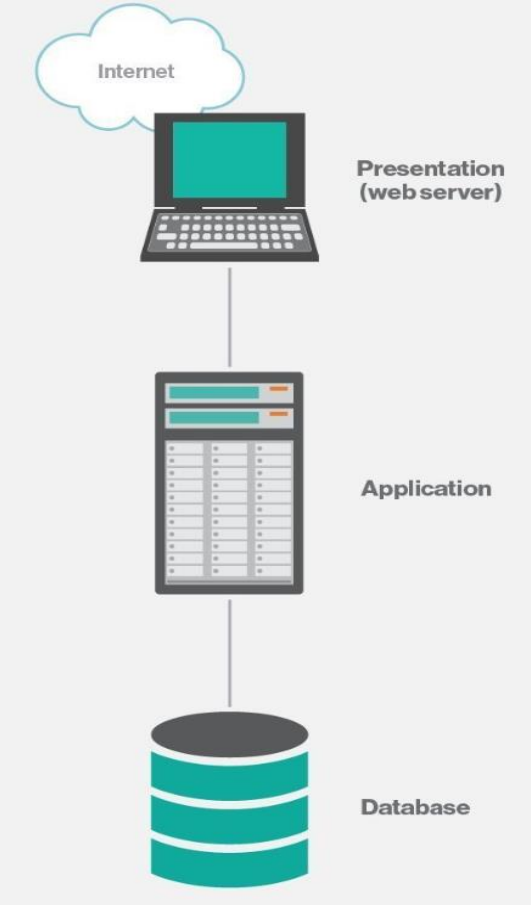



Рисунок 2.1 — Трьохрівнева архітектура

Презентаційний рівень відповідає за інтерфейс користувача і взаємодію з ним, він включає в себе компоненти, що забезпечують відображення

інформації, а саме такі компоненти як веб-сторінки, мобільні додатки, графічний інтерфейс користувача.

На бізнес-логічному рівні розміщені компоненти, що керують бізнес-логікою програми, яка виконується на сервері та правила обробки даних. Цей рівень відповідає за обробку запитів, взаємодію з базою даних та виконання бізнес-процесів.

Рівень доступу до даних відповідає за роботу з даними і забезпечує доступ до бази даних або інших сховищ даних, він реалізує запити до бази даних та взаємодію між ними. На цьому рівні розташовані компоненти, що забезпечують зберігання та доступ до даних.

Кожен рівень виконує свої функції, що дозволяє легше розуміти та керувати системою, також рівень можна розробляти, оновлювати та підтримувати окремо і незалежно від інших рівнів.

Використовуючи цю архітектуру, систему можна масштабувати шляхом додавання нових серверів або ресурсів для кожного рівня окремо і використовувати стандартизовані підходи для реалізації кожного рівня.

Попри все вищесказане у цієї архітектури є і недоліки:

— передача даних між рівнями може призвести до зайвого обсягу даних, особливо в веб-додатках, це може призвести до затрати часу та ресурсів на передачу та перетворення даних;

— під час взаємодії між рівнями інформаційна система може зазнавати затримок через обмін даними та обробку запитів на кожному рівні, це може впливати на продуктивність системи;

— з часом трьохрівнева архітектура може стати складною для підтримки, оскільки кожен рівень може вимагати окремих зусиль та ресурсів для підтримки та оновлення;

— взаємодія між рівнями може вимагати створення та керування з'єднаннями, що додатково ускладнює розробку та підтримку системи;

— помилки на одному рівні можуть впливати на всю систему, особливо якщо вони не виявляються та не обробляються належним чином;

— трьохрівнева архітектура може бути складною для інтеграції з іншими архітектурними підходами, такими як мікросервіси або архітектура, заснована на подіях.

Незважаючи на ці недоліки, трьохрівнева архітектура всеодно залишається популярним вибором для багатьох систем, особливо в сфері веб-розробки та підприємницьких додатків. Багато з цих недоліків можна подолати правильним плануванням та розробкою, а також застосуванням оптимізацій та технік для поліпшення продуктивності та ефективності.

Мікросервісна архітектура — це архітектурний підхід для розробки інформаційних систем, де програмне забезпечення розбивається на невеликі незалежні компоненти, відомі як мікросервіси. Кожен мікросервіс виконує обмежену функцію і взаємодіє з іншими мікросервісами через мережу. Такий підхід спрощує розробку, розгортання, масштабування та керування системою, оскільки кожен мікросервіс може бути розроблений, тестований і підтримуваний окремо (рисунок 2.2).

Ця система складається з невеликих вузлів або мікросервісів, які можуть розташовуватися на різних серверах чи контейнерах. Кожен мікросервіс може бути розроблений ізольовано і незалежно від інших мікросервісів. Вони можуть бути написані на різних мовах програмування та використовувати різні технології. Оскільки кожен мікросервіс виконує обмежену функцію, відмова в одному мікросервісі не вплине на роботу інших, що робить систему більш відмовостійкою. Вони можуть бути масштабовані незалежно від інших, що дозволяє використовувати ресурси ефективно. При внесенні змін в один мікросервіс, система не вимагає перекомпіляції. Також мікросервіси можуть бути керовані окремо, що спрощує процеси підтримки та моніторингу.

Управління багатьма мікросервісами може бути важким завданням. Для протидії складнощам потрібні ефективні механізми для оркестрації, моніторингу, логування та безпеки кожного мікросервісу.

При взаємодії між різними мікросервісами можуть виникати проблеми з синхронізацією та забезпеченням консистентності даних. Залежно від

топології системи, велика кількість мікросервісів може призвести до інтенсивного мережевого обміну даними, що може призвести до затримок та перевантаження мережі та ускладнює їх керування та моніторинг, запуск та підтримка багатьох мікросервісів може вимагати значних інвестицій у інфраструктуру та обладнання. Кожен мікросервіс може використовувати різні технології та мови програмування, що ускладнює розробку та підтримку системи, а також забезпечити безпеку в мікросервісній архітектурі може бути складно, оскільки кожен мікросервіс повинен бути захищений окремо.

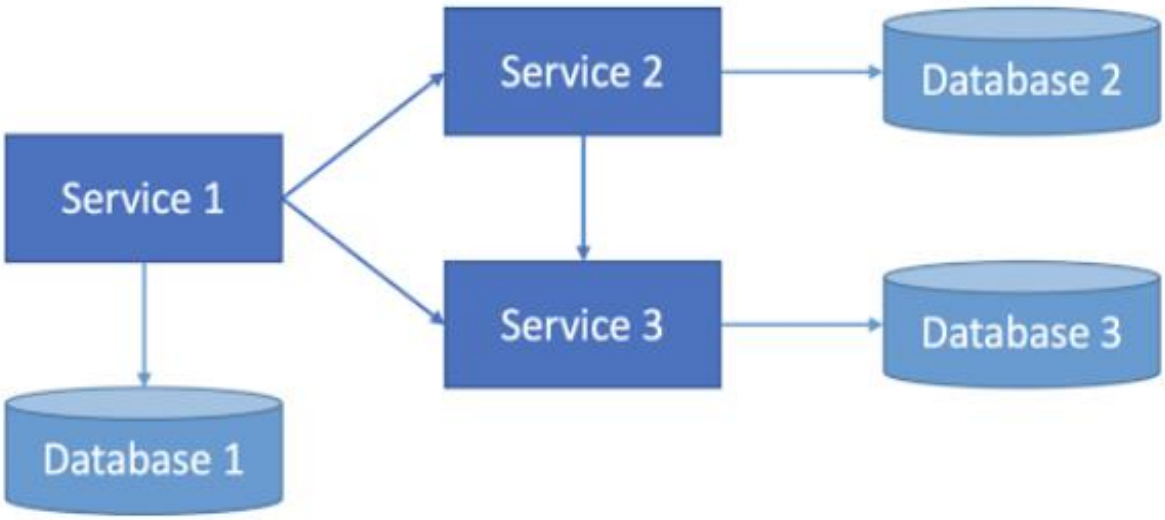



Рисунок 2.2 — Мікросервісна архітектура

Хоча мікросервісна архітектура дозволяє побудувати гнучкі та масштабовані системи, вона не підходить для всіх застосувань і вимагає обгрунтованого підходу до розробки та управління. Вирішення проблем, пов'язаних з мікросервісами, може вимагати значних зусиль та ресурсів, але для деяких проектів вона може бути вельми ефективною.

Спостережувана архітектура, (Event-Driven Architecture (EDA)) — це архітектурний підхід до розробки інформаційних систем, де система побудована навколо обміну подіями (event-driven) між різними компонентами.

В цій архітектурі система реагує на події, винесені в інших компонентах, і відповідає на них відповідними діями. Спостережувана архітектура дозволяє

системі реагувати на зміни в реальному часі та реалізувати асинхронну взаємодію між компонентами (рисунок 2.3).

Обробники подій виконують конкретні дії у відповідь на отримані події. Вони можуть бути реалізовані для обробки та аналізу подій, виконання логіки бізнесу та взаємодії з іншими компонентами системи.

Події включають в себе будь-яку важливу інформацію або зміни в системі, які можуть бути спостережені і використані для прийняття рішень. Події можуть бути створені різними компонентами системи та передаватися іншим. Вони можуть передавати дані від одного компонента до іншого, що дозволяє обмінюватися інформацією між різними частинами системи.

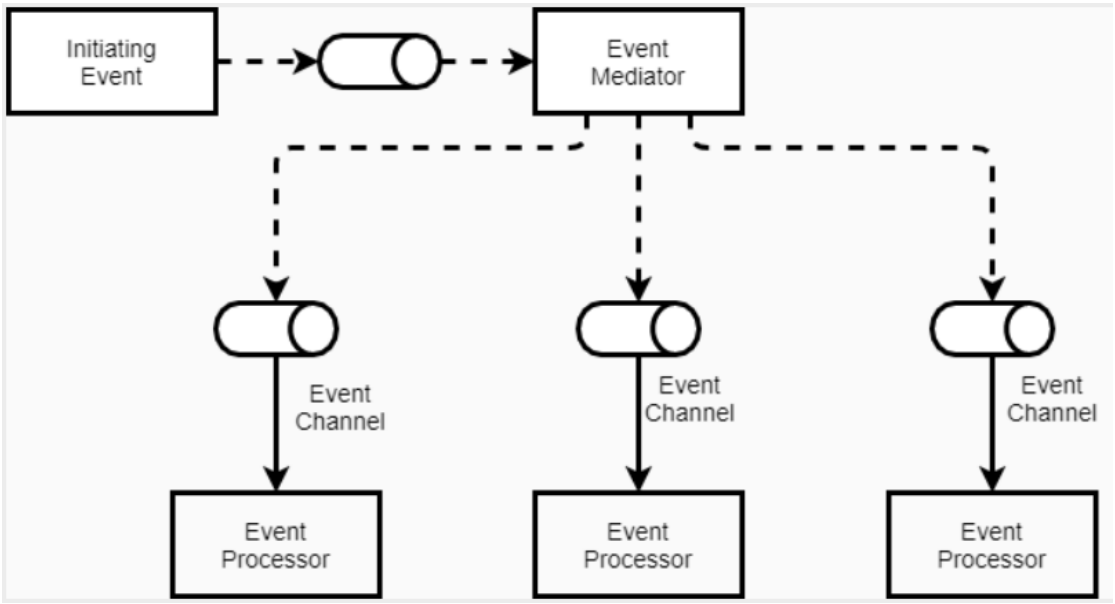



Рисунок 2.3 — Спостережувальна архітектура

Спостерігачі — це компоненти системи, які слідкують за певними подіями та реагують на них. Вони можуть реагувати на одну або декілька категорій подій.

Брокер подій відповідає за передачу подій від спостерігачів до відповідних обробників. Він може використовувати різні механізми для розподілу подій, такі як черги, шини подій та інші.

Обробники подій виконують конкретні дії у відповідь на отримані події. Вони можуть бути реалізовані для обробки та аналізу подій, виконання логіки бізнесу та взаємодії з іншими компонентами системи.

Однією з основних складнощів спостережуваної архітектури є складність відлагодження та відстеження подій, оскільки вони можуть бути викликані різними компонентами в різний час. Це може зробити важким виявлення та розуміння взаємодії між компонентами. Також важливо забезпечити достатню спостережуваність системи, щоб слідкувати за подіями та забезпечити їхню коректну обробку. Брак адекватної спостережуваності може призвести до непередбачуваної поведінки системи.

Синхронізація та забезпечення консистентності даних може бути складним завданням, оскільки події можуть викликатися асинхронно та в різний час. Виявити та усунути помилки в асинхронній спостережуваній архітектурі може бути складно, оскільки вони можуть виникати в різний час та в різних частинах системи.

Управління подіями, їх реєстрація, розподіл, маркування та обробка, може вимагати додаткових зусиль та інфраструктури. Якщо система генерує та обробляє велику кількість подій, це може викликати перевантаження та вплинути на продуктивність. Також використовуючи спостережувану архітектуру можна стикнутися з надмірною асинхронністю, що може ускладнити розробку та зробити систему менш передбачуваною.

Спостережувана архітектура широко використовується в різних сферах, включаючи розробку веб-додатків, систем управління подіями, систем моніторингу та багато інших застосувань. Вона дозволяє побудувати системи, які ефективно реагують на зміни та надають можливість асинхронної обробки подій, що є важливими в сучасних інформаційних системах.

Розподілена архітектура — це архітектурний підхід до розробки інформаційних систем, де компоненти системи розташовані на різних фізичних вузлах або серверах, які можуть знаходитися в різних географічних місцях та взаємодіяти один з одним через мережу. Такий підхід дозволяє

створювати системи, які масштабуються, відмовостійкі та можуть обробляти великі обсяги даних (рисунок 2.4).

Розподілена архітектура включає такі особливості:

— система може бути поділена на дві головні складові — клієнтську частину, яка надає інтерфейс користувача, і серверну частину, яка обробляє запити клієнтів та надає доступ до ресурсів;

— розподілені системи можуть включати різні види служб, такі як служби автентифікації, служби для роботи з базами даних, служби для обробки бізнес-логіки тощо, які можуть розташовуватися на окремих серверах;

— компоненти системи взаємодіють один з одним через мережу, використовуючи мережеві протоколи, такі як HTTP, TCP/IP, REST, SOAP;

— розподілені системи можуть бути легко масштабовані, додаючи нові сервери або вузли для обробки більших обсягів даних та запитів;

— розподілені системи можуть бути відмовостійкими, оскільки відмова в одному компоненті не призведе до повної втрати функціональності системи;

— дані можуть бути збережені на різних серверах і можуть бути репліковані для забезпечення доступу та надійності;

— розподілені системи можуть використовувати різні середовища виконання, такі як сервери додатків, контейнери або хмарні ресурси.

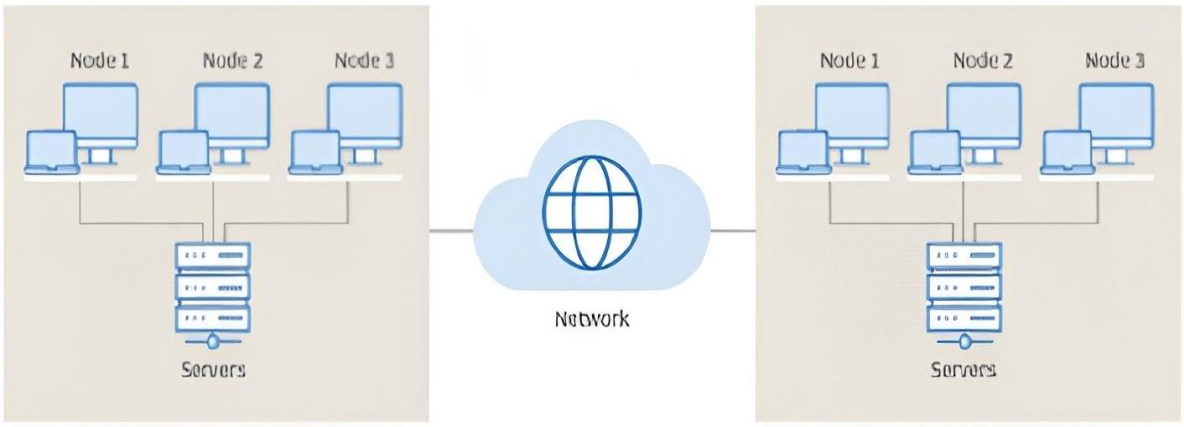



Рисунок 2.4 — Розподілена архітектура

Розподілена архітектура вимагає більше часу та зусиль для розробки в порівнянні з централізованими системами через необхідність обробки великої кількості деталей з мережевою взаємодією та забезпеченням відмовостійкості. Також проблемою цієї архітектури є складність тестування, через потребу у взаємодії з різними компонентами. Розподілені системи залежать від мережевої інфраструктури. Відмови або перебої в мережі можуть призвести до недоступності системи.

Запуск та підтримка розподіленої інфраструктури може вимагати значних витрат на обладнання та ресурси, а також ускладнити управління цією системою.

Хмарна архітектура — це специфікація компонентів та їх взаємодії в обчислювальній хмарі. Ця архітектура використовується для створення інфраструктури, додатків та послуг у хмарному середовищі. Хмарна архітектура дозволяє користувачам отримувати доступ до різних обчислювальних та інших ресурсів через Інтернет, зазвичай з використанням віртуалізації (рисунок 2.5).

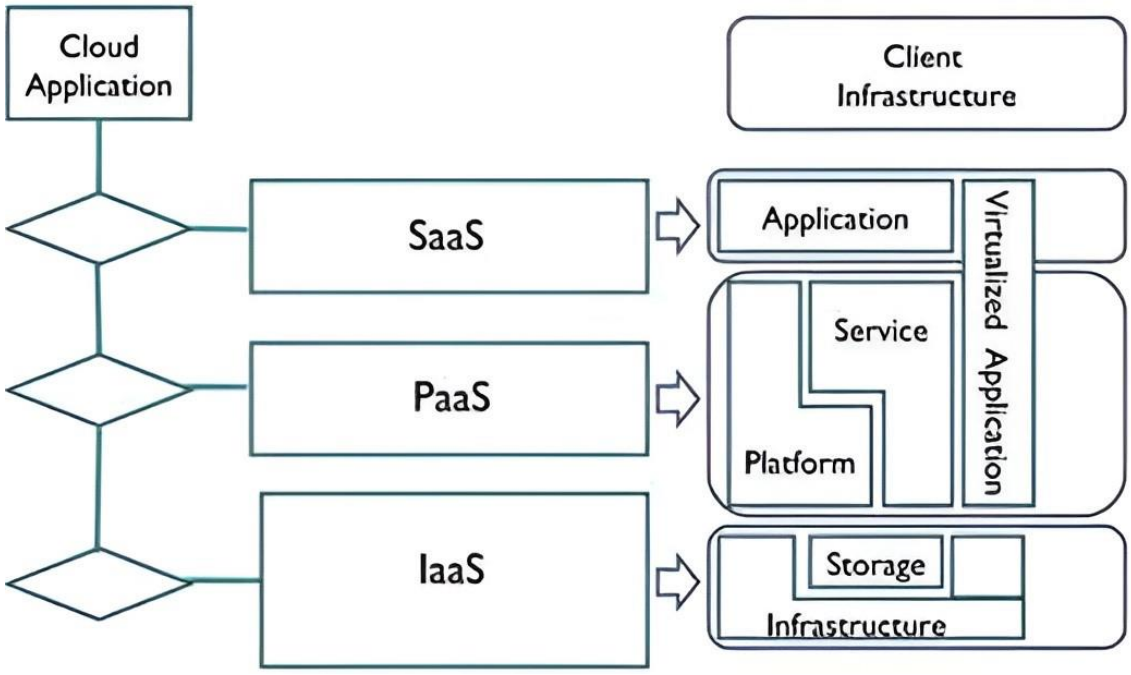



Рисунок 2.5 — Хмарна архітектура

Інфраструктура як послуга (IaaS) надає користувачам доступ до віртуальних серверів, мереж, сховищ та інших обчислювальних ресурсів. Це дозволяє користувачам створювати та керувати віртуальними машинами та іншими обчислювальними ресурсами, не звертаючи уваги на апаратне забезпечення.

Платформа як послуга (PaaS) надає середовище для розробки, тестування та розгортання додатків. Розробники можуть використовувати ці платформи для створення власних додатків без необхідності керувати інфраструктурою.

Програмне забезпечення як послуга (SaaS) надає користувачам доступ до готових додатків через Інтернет. Користувачі можуть використовувати ці додатки без необхідності їх встановлювати або підтримувати.

Хмари надають засоби для зберігання та обробки великих обсягів даних. Це дозволяє компаніям використовувати аналіз даних та машинне навчання для виявлення патернів та отримання цінної інформації.

Хмарні послуги також надають засоби для мережевої взаємодії між компонентами системи та забезпечення безпеки та конфіденційності даних.

Головні переваги хмарної архітектури включають:

— користувачі можуть легко масштабувати ресурси відповідно до потреб проекту;

— хмарні постачальники надають гарантії відмовостійкості та резервування даних;

— хмарні послуги дозволяють оптимізувати витрати на обладнання та обслуговування;

— користувачі можуть отримувати доступ до своїх даних та додатків з будь-якого місця, де є Інтернет;

— розгортання та масштабування додатків у хмарному середовищі може бути значно швидше, ніж на власних серверах.

Слід врахувати і недоліки. Одним із основних недоліків є питання безпеки. Користувачі повинні довіряти постачальникам хмарних послуг щодо

захисту їх даних. Потенційна загроза полягає у витоку конфіденційної інформації або несанкціонованому доступі до даних. Використання хмарних послуг може вимагати збереження даних на серверах постачальників, що може порушити приватність користувачів або підприємств. Закони про захист даних можуть обмежувати місця зберігання даних. Доступність та продуктивність хмарних послуг залежать від якості та доступності Інтернет-з'єднання. Відсутність мережі може призвести до втрати доступу до даних та додатків. Хоча хмарні послуги можуть здатися ефективними зараз, на довгострокову перспективу вони можуть призвести до високих витрат. Місячні чи щорічні оплати можуть накопичуватися та стати значними.

У деяких галузях, таких як фінанси або охорона здоров'я, існують суворі правила та обмеження, які ускладнюють використання хмарних послуг через потребу виконання правових норм та нормативів. Також під час використання хмарних послуг може бути обмежена можливість налаштування та контролю над інфраструктурою. Це може бути недоцільним для деяких вимог.

Не зважаючи на ці недоліки, хмарні послуги залишаються популярними через їх потужність та гнучкість. Вирішення цих питань вимагає ретельного вивчення та розгляду специфічних потреб та вимог проекту перед переходом до хмарної архітектури.

Клієнт-серверна архітектура є одним із найпоширеніших архітектурних підходів у розробці інформаційних систем. Вона передбачає розділення системи на дві головні складові: клієнтів і серверів, які взаємодіють один з одним через мережу (рисунок 2.6).

Клієнти — це користувацькі інтерфейси або додатки, що взаємодіють з користувачем. Вони зазвичай запитують дані або послуги від сервера і відображають їх на екрані користувача. Клієнти можуть бути реалізовані на різних платформах, включаючи веб-браузери, мобільні додатки, настільні програми та інші платформи.

Сервери — це спеціалізовані компоненти системи, які обробляють запити клієнтів і надають їм необхідні дані чи послуги. Сервери можуть

виконувати бізнес-логіку, доступ до баз даних, обробку запитів, аутентифікацію та авторизацію.

Клієнти і сервери взаємодіють один з одним, надсилаючи запити та отримуючи відповіді через мережу. Ця взаємодія може включати методи комунікації, такі як HTTP, TCP/IP, REST, SOAP, інші мережеві протоколи.

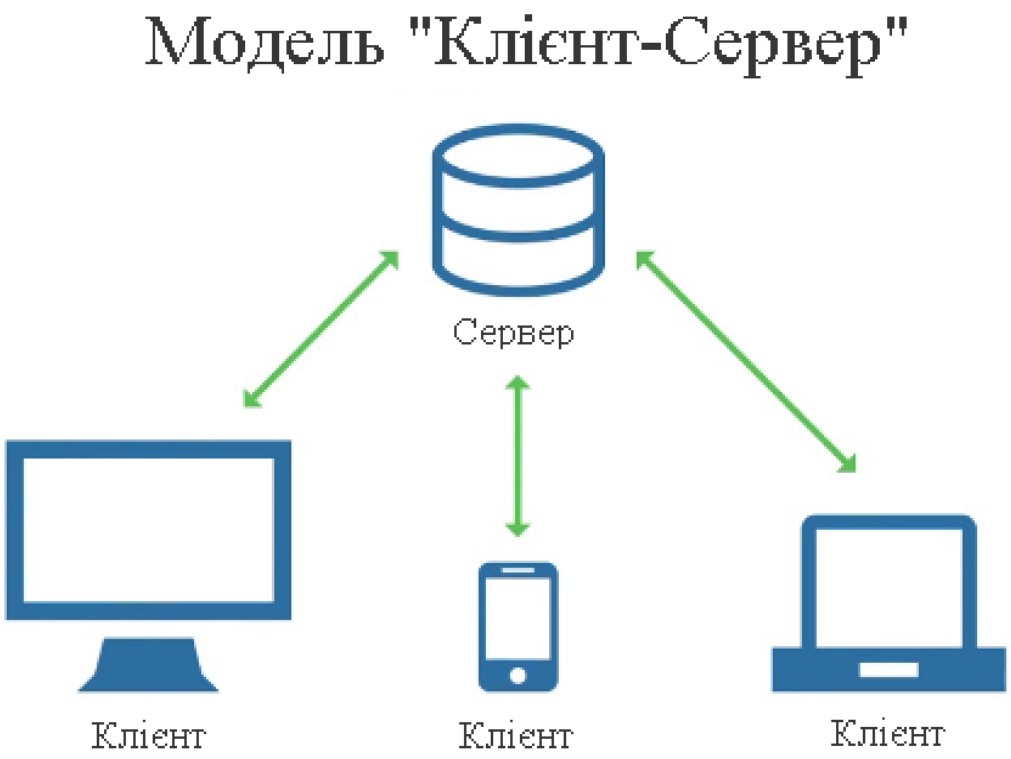



Рисунок 2.6 — Клієнт-серверна архітектура

Головними перевагами клієнт-серверної архітектури є:

— архітектура дозволяє розподіляти завдання між клієнтами та серверами, що спрощує розробку та підтримку;

— використовуючи клієнт-серверну архітектуру, систему можна легко масштабувати, додаючи більше серверів для обробки запитів;

— різні клієнти можуть бути розроблені для різних платформ, так що користувачі можуть отримувати доступ до системи з різних пристроїв.

Але ця архітектура має і недоліки:

— наявність мережі необхідна для взаємодії між клієнтами та серверами, тому відмова мережі може призвести до недоступності системи;

— перевантаження сервера при не збалансованому використанні ресурсів;

— взаємодія між клієнтами та серверами потребує правильного забезпечення безпеки, включаючи аутентифікацію та авторизацію.

Серед розглянутих архітектур інформаційних систем для реалізації було обрано клієнт-серверну архітектуру. Зобразимо загальну модель роботи web-системи (рисунок 2.7).

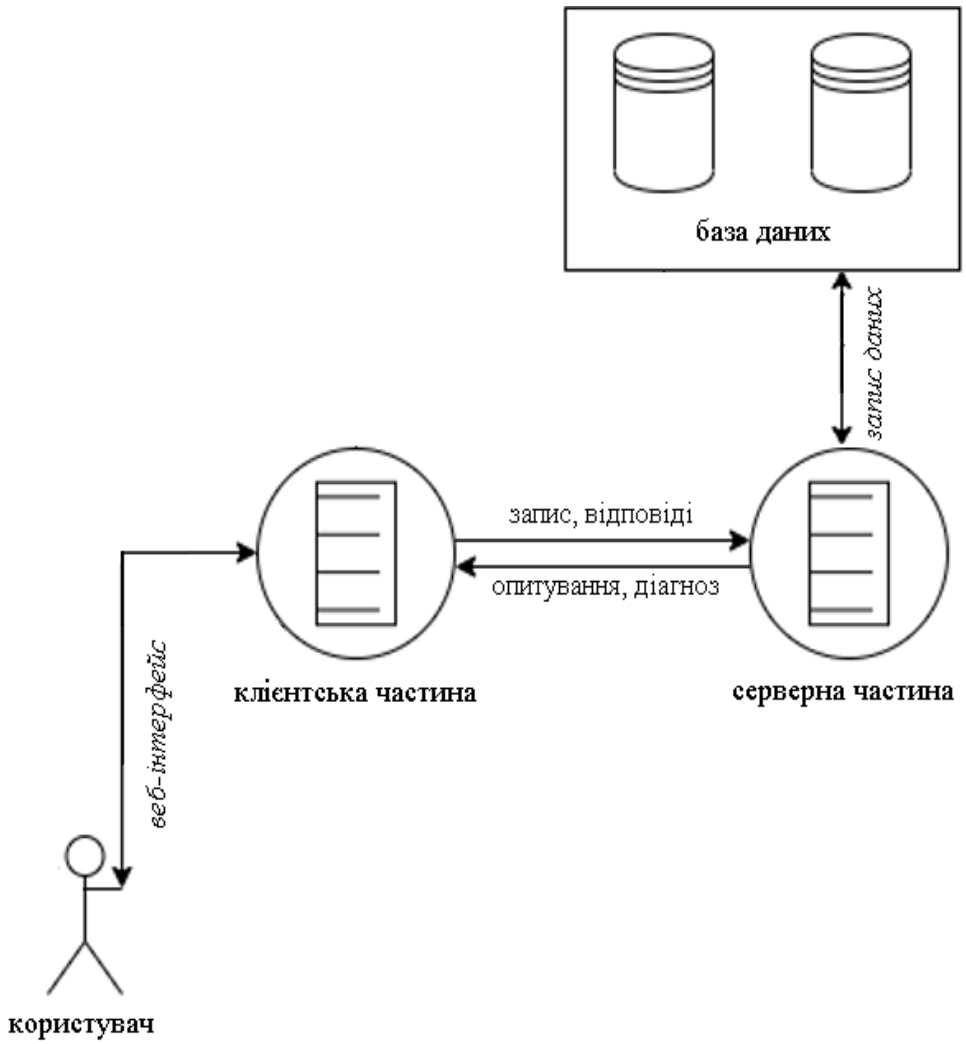



Рисунок 2.7 — Загальна модель роботи системи

API (інтерфейс програмування додатків) представляє собою набір правил і протоколів, що забезпечують взаємодію різних програм. Вони виступають як посередник між програмами та системами, дозволяючи їм обмінюватися інформацією та взаємодіяти. API може бути внутрішнім,

призначеним для використання всередині компанії, або зовнішнім, призначеним для взаємодії зі сторонніми сервісами (рисунок 2.8)

Основна ідея API полягає в тому, що він надає стандартний спосіб для взаємодії з функціональністю програми чи сервісу. Це може бути використано для взаємодії між різними компонентами в межах одного програмного забезпечення або для спілкування між різними програмами та сервісами.

API може приймати різні форми, включаючи бібліотеки, протоколи, веб-сервіси та інші. Наприклад, веб-сервіси дозволяють програмам обмінюватися даними через мережу Інтернет за допомогою стандартних протоколів, таких як HTTP.

API використовується для створення розширень, інтеграцій та взаємодії між різними програмами, що дозволяє розробникам використовувати функціональність вже існуючих систем та розширювати їх можливості без необхідності повного розуміння внутрішньої реалізації цих систем.

Основні типи API-запитів включають:

— запит типу GET — використовується для отримання даних з сервера, наприклад, отримання інформації про користувача;

— запит POST — використовується для надсилання даних на сервер з метою створення нового запису у базі, наприклад, створення нового користувача;

— запит PUT — використовується для оновлення існуючого ресурсу або створення нового, якщо він відсутній, наприклад це може бути використано для оновлення інформації про користувача;

— запит DELETE — використовується для видалення ресурсу на сервері, наприклад, видалення запису користувача.

Серверна складова інформаційної системи є ключовою для забезпечення її функціонування та обслуговування. Цей сегмент відповідає за обробку запитів від користувачів, зберігання інформації, взаємодію з пристроями клієнтів та забезпечення безпеки та доступності системи.

Інформаційна система зазвичай використовує сервери для обробки запитів, які можуть представляти собою фізичні апаратні пристрої або віртуальні машини. Сервери виконують різноманітні операції, обробляють дані та взаємодіють з базами даних. Для забезпечення взаємодії між серверами та клієнтами необхідно реалізувати мережеву інфраструктуру, яка включає комутатори, маршрутизатори, файерволи та інше мережеве обладнання.

Бази даних відіграють важливу роль у зберіганні та управлінні даними, і їх структура та безпека є ключовими аспектами.

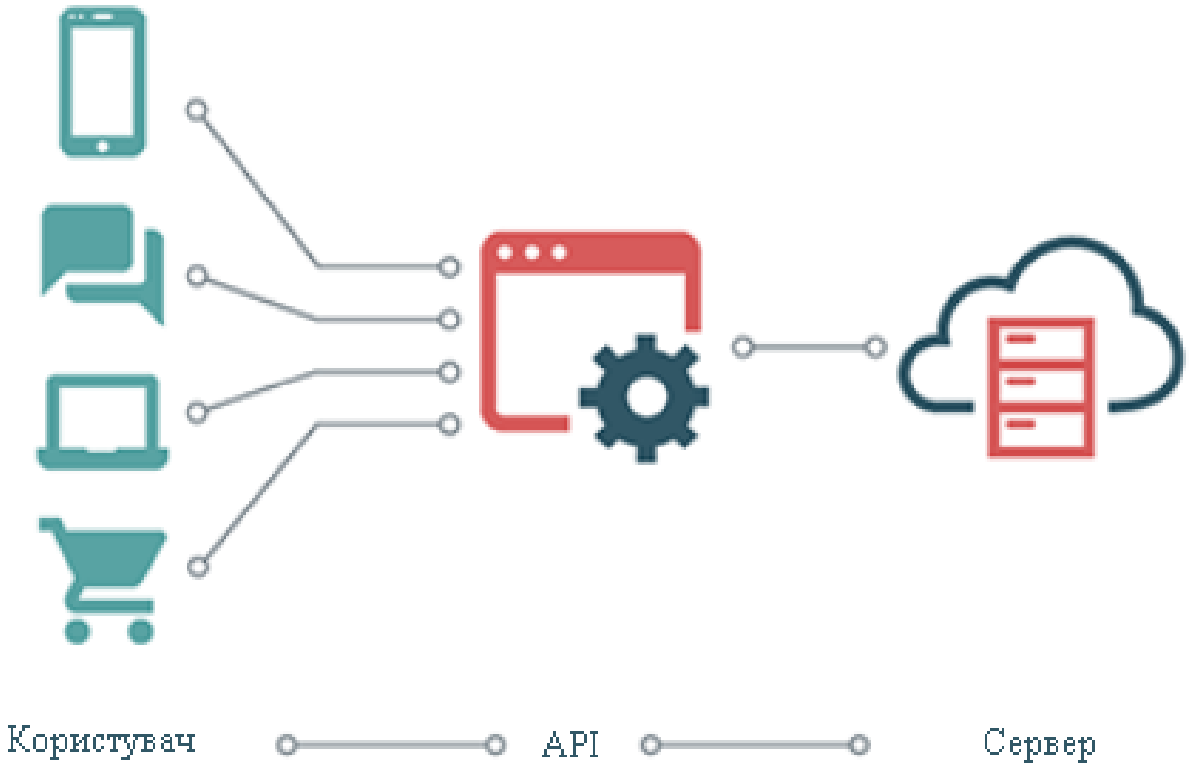



Рисунок 2.8 – Принцип роботи API

Адміністративна складова інформаційної системи грає ключову роль у керуванні та управлінні цією системою. Ця частина системи дозволяє адміністраторам та управлінцям ефективно контролювати та підтримувати систему, гарантуючи її безпеку, надійність та ефективність. Для адміністраторів важливо мати механізми автентифікації та авторизації, щоб визначати права доступу до адміністративних функцій.

Адміністратори повинні мати можливість створювати, блокувати, редагувати та видаляти облікові записи користувачів, а також надавати їм ролі та права доступу до різних частин системи. Окрім того, їм необхідні інструменти для моніторингу стану системи, збору даних про навантаження, продуктивність та безпеку, щоб вчасно виявляти проблеми та вдосконалювати роботу системи.

Адміністраторам надається можливість керувати роботою серверів та програм, включаючи можливість запускати та зупиняти сервери, додатки та інші компоненти системи. Також важливо встановлювати та управляти заходами забезпечення безпеки, такими як брандмауери, антивірусне програмне забезпечення та системи моніторингу безпеки. Регулярне створення бекапів даних та їх відновлення у випадку втрати або пошкодження також є важливим елементом адміністративної частини системи.

Адміністративна складова інформаційної системи допомагає забезпечити ефективну та безпечну роботу системи, а також відповідність системи бізнес-вимогам і стандартам безпеки. З іншого боку, клієнтська складова інформаційної системи представляє інтерфейс для взаємодії користувачів з системою. Ця частина включає різні програми, додатки, веб-сайти та інші інтерфейси, що дають користувачам доступ до функцій та можливостей системи.

Клієнтська частина має графічний інтерфейс, який включає кнопки, меню, форми та інші елементи керування, для взаємодії користувача з системою. Вона може бути реалізована як веб-додаток у браузері чи мобільний додаток для телефону чи планшета.

Клієнтська частина може включати механізми автентифікації, які вимагають від користувачів введення ім'я користувача та пароля, а також систему авторизації для управління правами доступу користувачів. Важливо докладно працювати над розробкою цієї складової системи, забезпечуючи її зручність, безпеку та функціональність, оскільки вона відіграє важливу роль у

взаємодії користувачів з системою і впливає на їхню ефективність та задоволеність використанням системи.

## 2.4 Розробка алгоритму роботи web-додатку

Перш за все на серверній частині слід встановити програмне забезпечення, необхідне для роботи web-додатку, після цього клієнтська та адміністративна частина web-додатку встановлюють усі необхідні залежності для коректної роботи.

Далі необхідно встановити та налаштувати конфігурацію базу даних MongoDB [14], для цього необхідно завантажити MongoDB, створити базу даних та налаштувати доступ до неї, відповідно до ролей.

MongoDB представляє собою систему керування базами даних (СКБД) орієнтовану на документи з відкритим вихідним кодом, яка вражає відсутністю необхідності у визначенні схеми таблиць. Ця система займає унікальне положення між високопродуктивними, масштабованими системами, які обробляють дані у форматі ключ/значення, та реляційними СКБД, що визначаються своєю функціональністю та зручністю в створенні запитів (рисунок 2.9).

MongoDB використовує JSON-подібний формат для зберігання документів, надає гнучку мову для формулювання запитів, здатна створювати індекси для різних атрибутів, досить ефективно обробляє великі бінарні об'єкти, підтримує журналювання операцій зі зміною та додаванням даних в базу даних. Крім того, система відповідає парадигмі Map/Reduce, підтримує реплікацію та конструює надійні конфігурації для опротестування відмовостійкості.

Після відкриття web-додатку клієнт має змогу заповнити форму для запису до певного лікаря, після заповнення генерується запит на збереження на сервері уже у базу даних MongoDB.

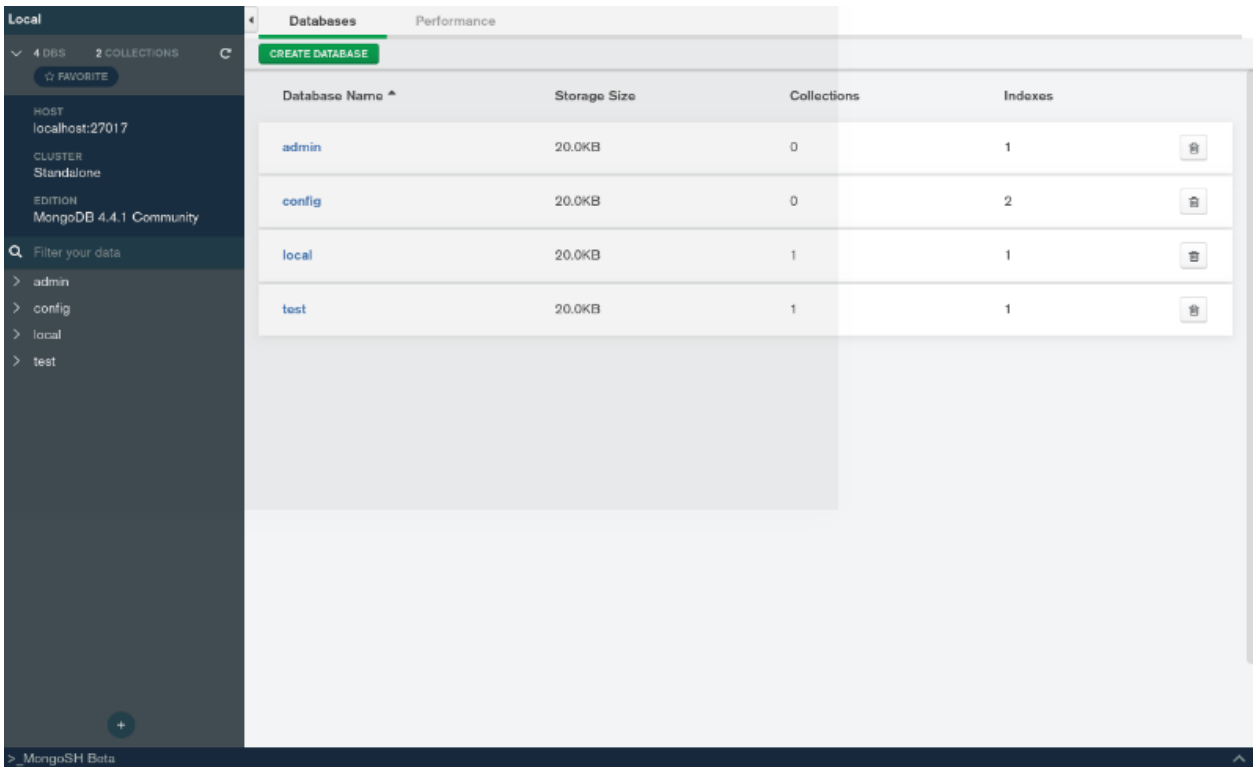



Рисунок 2.9 — Інтерфейс СКБД MongoDB

Після відкриття web-додатку клієнт має змогу заповнити форму для запису до певного лікаря, після заповнення генерується запит на збереження на сервері уже у базу даних MongoDB.

Тоді на серверній частині проводиться перевірка, чи відповідають дані у полях формату та логіці web-додатку. Якщо дані відповідають вимогам, то вони успішно зберігаються в базі даних MongoDB. Інакше, сервер повертає клієнту повідомлення  про помилку.

Адміністратор має можливість перевірити список записаних клієнтів, підчас рендерингу сторінки зі списком сервер надсилає запит на отримання даних з бази даних MongoDB.

Після запису до лікаря, клієнт переходить до сторінки з лікарем, де перевіряється чи він записаний. Якщо це так, то можна переходити до діагностування, а якщо ні, то клієнта направляють на запис до лікаря.

У процесі діагностування система задає питання клієнту (пацієнту), кожне наступне питання звужує область можливих діагнозів, і зводить анамнез пацієнта до максимально вузького кола діагнозів.

Коли діагностування пацієнта завершиться, то клієнт отримує свій діагноз з певною вірогідністю, а дані про діагноз клієнта (пацієнта) додаються до його запису у базі даних.

Адміністратор, на відміну від клієнта, має можливість переглядати дані пацієнтів у будь-якого лікаря, для забезпечення конфіденційності даних.

## 2.5 Проектування принципів роботи web-додатку

Більшість веб-сайтів, веб-систем або веб-додатків включають адміністративні панелі, до яких користувачі отримують доступ після авторизації. Особа, відповідальна за редагування та наповнення інформацією сайту, вводить своє ім'я користувача та пароль через відповідне веб-посилання, отримуючи тим самим доступ до CMS [15]. Більшість адміністративних панелей мають стандартний набір функціональностей, і для ефективного прискорення розробки цифрових рішень зручно використовувати гнучкі багатофункціональні адміністративні панелі, такі як: WordPress , Admin LTE, ArchitectUI, Devias Kit

Сучасний підхід до розробки веб-сайтів та веб-додатків дозволяє подолати вказані обмеження. Тепер широко застосовується використання готових шаблонів адміністративних панелей, які базуються на фреймворках. Ці шаблони надають можливість реалізувати повний функціонал, необхідний для управління контентом, з урахуванням високої якості дизайну UX/UI.

Сучасна адміністративна панель веб-сайту повинна мати привабливий та інтуїтивно зрозумілий дизайн, бути адаптивною для різних браузерів на різних операційних системах і коректно відображатися на різних пристроях.

Крім того, важливо ефективно використовувати ресурси при проєктуванні та створенні адміністративної панелі, оскільки стандартні завдання мають мати стандартні рішення. Відмінним рішенням є використання набору шаблонів та елементів інтерфейсу, які підходять для адміністрування різних цифрових проєктів, включаючи веб-системи для управління бізнесом.

За допомогою адміністративної панелі на основі Admin LTE можна легко керувати веб-сайтами, порталами, CRM- та ERP-системами, серверами мобільних додатків, багатофункціональними веб-системами та іншими цифровими рішеннями.

За допомогою адмін-панелі адміністратор може:

— додавати, редагувати та видаляти контент;

— керувати мультимедійними елементами;

— додавати користувачів і налаштувати права доступу.

Великим проривом у технології створення та розробки сучасних вебпроектів є створення системи керування вмістом (CMS, Content Management System) — це інструментальне середовище для створення і адміністрування сайту без знань мов програмування. Система управління сайтом, або CMS-движок, містить готові шаблони структури, набір функцій для дизайну і наповнення даними. CMS сайту дозволяє зручно обслуговувати ресурс, управляти доступом до контенту, змінювати дані на сторінках, відправляти пошту.

Якщо раніше більшість сайтів були статичними і вимагали внесення змін в їх вміст вручну, зараз динаміка розвитку проектів вимагає готовності швидко реагувати на зміни і впроваджувати їх з максимальною оперативністю. При цьому не усі користувачі хочуть або можуть собі дозволити звертатися до розробників, особливо якщо сайт вимагає постійної роботи над ним.

Сучасні системи керування вмістом широко використовуються на просторах Інтернету при створенні проектів будь-якої складності, при цьому не потребуючи знань в HTML, CSS та інших областях веб-програмування, що є їх головною перевагою. Також дані системи дають можливість швидкого, простого та інтуїтивного додавання, видалення, редагування та форматування контенту, що значно спрощує та полегшує завдання адміністрування сайту.

З використанням CMS можливе не лише додавання текстового контенту, а й різноманітного мультимедійного матеріалу, що дозволяє значно урізноманітнити сайт. Також системи керування вмістом автоматично

генерують панель адміністратора, яка зачіпає всі сфери роботи сайту, що дуже зручно та практично.

Всі CMS розділити розділити на кілька видів:

— CMS з відкритим кодом;

— коробочні CMS;

— самописні CMS;

— фреймворки.

CMS з відкритим кодом означає, що двигун може модифікувати будь-хто. Завдяки цьому в таких CMS регулярно з’являються нові доповнення та теми, а також швидше знаходять та усувають уразливості. Це одна з основних причин, чому WordPress став таким популярним двигуном.

Щодо коробочних CMS, по суті такі двигуни відрізняються тільки тим, що у них закритий код, а значить внести зміни в двигун можуть тільки офіційні розробники. Це не означає, що такі CMS системи менш безпечні або у них гірший функціонал, але кількість тем та доповнень зазвичай дійсно менша.

Самописні ж CMS розробляють на замовлення під конкретний проект. Їх функціонал не такий широкий, як у коробкових CMS або CMS з відкритим кодом, але максимально відповідає поставленим завданням і не містить зайвих інструментів. Єдине, якщо знадобиться розширити функціонал або закрити вразливості, доведеться звертатися до розробника движка або шукати спеціаліста, який з нуля розбиратиметься в коді, а це гроші і час.

Фреймворк — це надбудова над мовою програмування; набір бібліотек, за допомогою яких можна створити сайт для будь-яких завдань. Розробка сайту на фреймворку вимагатиме більше грошей і часу, а крім сайту знадобиться окремо розробляти панель управління сайтом, а це ще один сайт. Але так ви зможете реалізувати будь-який функціонал, який вам потрібен. Тобто для нетипових проектів такий спосіб буде якраз. Плюс продуктивність у добре зробленого сайту на фреймворку буде вищою.

Інформаційна web-система передбачатиме реєстрацію лікаря, його автентифікацію та авторизацію.

Реєстрація — це процес створення облікового запису чи профілю в системі. Під час реєстрації користувач надає необхідну інформацію (таку як ім'я, електронна пошта, пароль тощо), яка використовується для ідентифікації та взаємодії з системою [16].

Автентифікація — це процес підтвердження ідентичності користувача, зазвичай за допомогою комбінації ім'я користувача та пароля. Це забезпечує впевненість в тому, що особа, яка використовує обліковий запис, є тією, за кого вона себе видає [17].

Авторизація — це надання прав доступу конкретному користувачеві після того, як його ідентичність вже була підтверджена. Після успішної автентифікації система визначає, які дії та ресурси користувач може використовувати в рамках системи, базуючись на наданих йому правах або ролях [18].

Сепарація по ролям (RBAC) є концепцією управління доступом, при якій права доступу надаються користувачеві на основі його ролі в системі чи організації. Це означає, що користувачеві присвоюється конкретна роль, а ці ролі визначають, які дії та ресурси користувач може використовувати в системі.

Інформаційна система у даній дипломній роботі передбачає такі ролі, як admin, та client. Адміністратор має доступ до усіх ресурсів і даних системи, в той час як клієнт володіє лише даними про себе і свій попередній діагноз.

При авторизації в ролі адміністратора, web-система надасть можливість переглянути дані про пацієнтів і діагнози кожного з заданих лікарів медичного закладу.

Клієнтська частина веб-ресурсу — це та частина веб-додатка, яка виконується на браузері або клієнтському пристрої користувача. Вона відповідає за те, як користувач взаємодіє з веб-ресурсом, відображення інформації та виконання багатьох фронтенд-операцій [19].

Основні аспекти клієнтської частини веб-ресурсу:

— мови програмування;

— фреймворки та бібліотеки;

— зображення;

— робота з API;

— оптимізація та швидкість.

Серед мов програмування використовуються HTML, CSS та JavaScript. HTML є основною мовою розмітки для створення структури та вмісту веб-сторінок. HTML визначає різні елементи або теги, які вказують браузеру, як правильно відображати вміст на сторінці. CSS використовується для стилізації та оформлення веб-сторінок, які створені за допомогою HTML JavaScript є мовою програмування, яка використовується для надання динамічності та інтерактивності веб-сторінок. Вона виконується безпосередньо на клієнтському браузері користувача і взаємодіє з DOM (Document Object Model), що дозволяє змінювати вміст та поведінку сторінок.

Фреймворки, такі як React, Angular, Vue.js — це фреймворки для розробки інтерфейсу користувача, які спрощують створення складних веб-додатків.. Ці інструменти надають готові рішення для створення інтерфейсу взаємодії з сервером.

Також з клієнтської частини користувач надсилає запити до сервера через API для отримання або відправлення даних.

Веб-сторінки зазвичай включають в себе зображення, відео та інші мультимедійні ресурси, які відображаються в браузері.

Структура сторінок — це організація різних сторінок на веб-сайті, таких як сторінка реєстрації, сторінка запису пацієнта, сторінка діагнозу та інші.

Оптимізація завантаження ресурсів, кешування, компресія зображень – це стратегії для забезпечення швидкої роботи веб-сторінок.

При вході у ролі клієнта, пацієнт записується до певного лікаря у журнал через форму для запису, та переходить до сторінки лікаря. Якщо записаний пацієнт відповідає запису, то проводиться опитування пацієнта за допомогою дерева рішень, кожне наступне запитання звужує коло допустимих діагнозів, по завершенню опитування, опитуваний отримує свій попередній діагноз, але

всеодно отримує рекомендацію відвідати лікаря очно, для уточнення діагнозу та отримання вказівок щодо лікування.

# 3 РОЗРОБКА ФУНКЦІОНАЛУ WEB-ДОДАТКУ

## 3.1 Варіантний аналіз і обґрунтування вибору засобів реалізації web-додатку

Перед початком розробки web-додатку для діагностування важливо визначитися з мовою програмування, яка оптимально підходить для завдання. Оскільки мета полягає в розробці програми, що використовує функції Rest API для безпечного виконання користувацьких рішень, обрана мова програмування повинна підтримувати відповідні виклики. Також важливо, щоб ця мова програмування надавала зручний інтерфейс для роботи з мережевими протоколами для ефективної комунікації з клієнтською частиною.

C# — це об'єктно-орієнтована, типізована мова програмування, яка розроблена компанією Microsoft. Вона входить до складу платформи розробки Microsoft .NET і призначена для створення різноманітних застосунків, включаючи веб-додатки, настільні програми, ігри, мобільні додатки та багато іншого [20]. C# має чистий і зрозумілий синтаксис, що полегшує розробку коду і забезпечує його читабельність. Вона взяла багато синтаксичних елементів із мов C++ та Java, що робить її знайомою для багатьох програмістів. Мова підтримує основні принципи об'єктно-орієнтованого програмування, такі як спадкування, інкапсуляція та поліморфізм. Це дозволяє розробникам створювати ефективні та модульні програми. C# використовується разом з платформою .NET, що надає широкі можливості для розробки різноманітних застосунків. Робота в середовищі .NET дозволяє використовувати багато функцій, таких як управління пам'яттю, безпека коду, серіалізація та інші. C# є однією з основних мов програмування для розробки веб-додатків на платформі ASP.NET. Розробники можуть створювати ефективні та масштабовані веб-застосунки з використанням C#. C# має вбудовану підтримку LINQ, що дозволяє виконувати операції запитів безпосередньо в мові програмування, спрощуючи роботу з колекціями даних. C# надає підтримку асинхронного програмування, що стає важливим для створення

продуктивних та реактивних додатків. З запуском .NET Core, C# став доступним для кросплатформенної розробки, дозволяючи створювати застосунки для різних операційних систем.

C++ — це загальноцільова, компільована мова програмування, яка розширює мову C з новими функціями і об'єктно-орієнтованим підходом. Вона здобула широку популярність завдяки своїй ефективності, гнучкості та можливостям використання в різних областях розробки програмного забезпечення. Мова компілюється в машинний код, що робить виконуваний файл ефективним для виконання на конкретній платформі, підтримує об'єктно-орієнтоване програмування, що дозволяє розробникам створювати класи та об'єкти для структурування коду та покращення його перевикористовуваності [21]. C++ має багатофункціональну та потужну стандартну бібліотеку, яка включає в себе різноманітні класи та функції для роботи з рядками, контейнерами, потоками та алгоритмами. Вона дозволяє розробникам прямий доступ до пам'яті, оптимізацію та низькорівневу роботу з ресурсами, що робить її популярною для системного програмування та розробки високопродуктивного програмного забезпечення. Мова використовується в різних областях, включаючи розробку операційних систем, вбудованих систем, гри, додатки для роботи з графікою, наукові обчислення. Введення шаблонів в C++ дозволяє розробникам писати загальні, універсальні функції та класи, що робить код більш гнучким та перевикористовуваним. Код, написаний на C++, може бути перенесений між різними платформами без значних змін, що робить його придатним для розробки кросплатформенних додатків. Розробники можуть маніпулювати пам'яттю напряму, що дозволяє ефективно використовувати ресурси та оптимізувати продуктивність.

JavaScript — це високорівнева, інтерпретована мова програмування, яка використовується для розробки веб-додатків та взаємодії з користувачем в браузерах. Вона є однією з ключових технологій для фронтенд-розробки і забезпечує динамічну та інтерактивну поведінку веб-сторінок. avaScript має

схожий до інших мов програмування синтаксис, що полегшує вивчення для новачків. Він використовується для взаємодії з DOM (Document Object Model), щоб змінювати структуру та вигляд веб-сторінок. JavaScript є динамічною мовою, тобто типи даних визначаються автоматично під час виконання програми. Основні типи даних включають рядки, числа, булеві значення, масиви та об'єкти [22].

Функції в JavaScript є об'єктами першого класу, і вони можуть бути передані як аргументи, зберігатися в змінних та повертатися з інших функцій. Це дозволяє створювати ефективний та чистий код. JavaScript дозволяє визначати обробники подій для взаємодії з користувачем. Це може бути клік миші, натискання клавіші або інші події. З введенням Promise та async/await, JavaScript надав можливість працювати асинхронно, що робить можливим виконання операцій, які можуть займати час, без блокування виконання інших частин коду. JavaScript підтримує об'єктно-орієнтоване програмування, дозволяючи створювати та використовувати класи та об'єкти для структурування коду.

JavaScript має велику кількість бібліотек та фреймворків, таких як React, Angular, Vue.js, що полегшують розробку веб-додатків та забезпечують перевикористовувані компоненти. Мова може використовуватися не тільки в браузерах, але й на сервері (з використанням Node.js), що робить його вибором для розробки повноцінних веб-додатків.

Для визначення мови враховувалися такі параметри, як наявність об'єктно-орієнтованої парадигми, типізація, використання пам'яті, ефективність, взаємодія з платформами і галузь використання, оскільки вони є основними при виборі мови програмування.

Для більш об’єктивного та наочного порівняння вище наведених мов програмування по визначеним критеріям зведемо усю інформацію у таблицю 3.1.

Таблиця 3.1 — Порівняння мов програмування

| Характеристика | C# | C++ | JavaScript (JS) |
|---|---|---|---|

Продовження таблиці 3.1

| | | | |
|---|---|---|---|
| Стиль мови | Об'єктно-орієнтований | Об'єктно-орієнтований і процедурний | Об'єктно-орієнтований і функціональний |
| Типізація | Сильна, статична | Сильна, статична | Слабка, динамічна |
| Використання пам'яті | Автоматичне керування пам'яттю | Вручну керування пам'яттю | Вручну керування пам'яттю (з можливістю використання автоматичного сбору сміття) |
| Ефективність | Вищий рівень ефективності | Висока ефективність, але важко підтримувати | Зазвичай менш ефективний порівняно з C# і C++ |
| Платформи | Головним чином Windows | Крос-платформеність, але робота з Windows | Веб-програмування, включаючи веб-браузери та сервери |
| Використання | Розробка Windows-додатків, геймдевелопмент, серверні додатки | Системне програмування, геймдевелопмент, вбудовані системи | Веб-розробка, клієнтська і серверна сторони, розширення веб-браузерів |

З врахуванням цільового призначення проекту, можна визначити вагому перевагу мови програмування JavaScript перед C# та C++. Таким чином, обрання JavaScript для реалізації серверної частини та проекту взагалі є обґрунтованим.

При виборі середовища розробки для створення веб-ресурсу проведено аналіз таких рішень, як Visual Studio Code та WebStorm.

Visual Studio Code — це безкоштовний та потужний текстовий редактор, розроблений командою Microsoft. Він призначений для розробки різноманітних програм, зокрема веб-додатків і серверних додатків. VS Code підтримується на різних операційних системах, таких як Windows, macOS і Linux, що робить його універсальним для розробників, які працюють на різних платформах [23]. VS Code має широкий вибір розширень, які дозволяють налаштовувати та розширювати функціонал редактора. Розширення доступні для мов програмування, фреймворків, систем контролю версій, тем оформлення та багатьох інших аспектів. Вбудована підтримка Git дозволяє розробникам взаємодіяти з системою контролю версій безпосередньо з редактора. Це полегшує ведення історії змін та співпрацю в командах. Робочі простори дозволяють групувати пов'язані файли та проекти, щоб полегшити організацію робочого процесу. Розробники можуть легко перемикатися між різними робочими середовищами. VS Code надає автоматичне завершення коду, інтегровані підказки та аналіз коду під час розробки, що робить процес більш продуктивним та ефективним.

Інтегрована консоль дозволяє виконувати команди та скрипти безпосередньо в редакторі, спрощуючи взаємодію з проектами та забезпечуючи швидкий доступ до інструментів командного рядка. Середовище відзначається інтуїтивним і легким інтерфейсом користувача. Він пропонує прості налаштування та можливість швидко адаптуватися під конкретні потреби розробника. Редактор підтримує відладку для різних мов та середовищ, забезпечуючи можливість крокування коду, встановлення точок зупинки та виведення змінних.

Visual Studio Code завдяки своїй легкості використання, багатофункціональності та активній спільноті розробників став одним з найпопулярніших текстових редакторів для роботи з кодом. (рисунок 3.1).

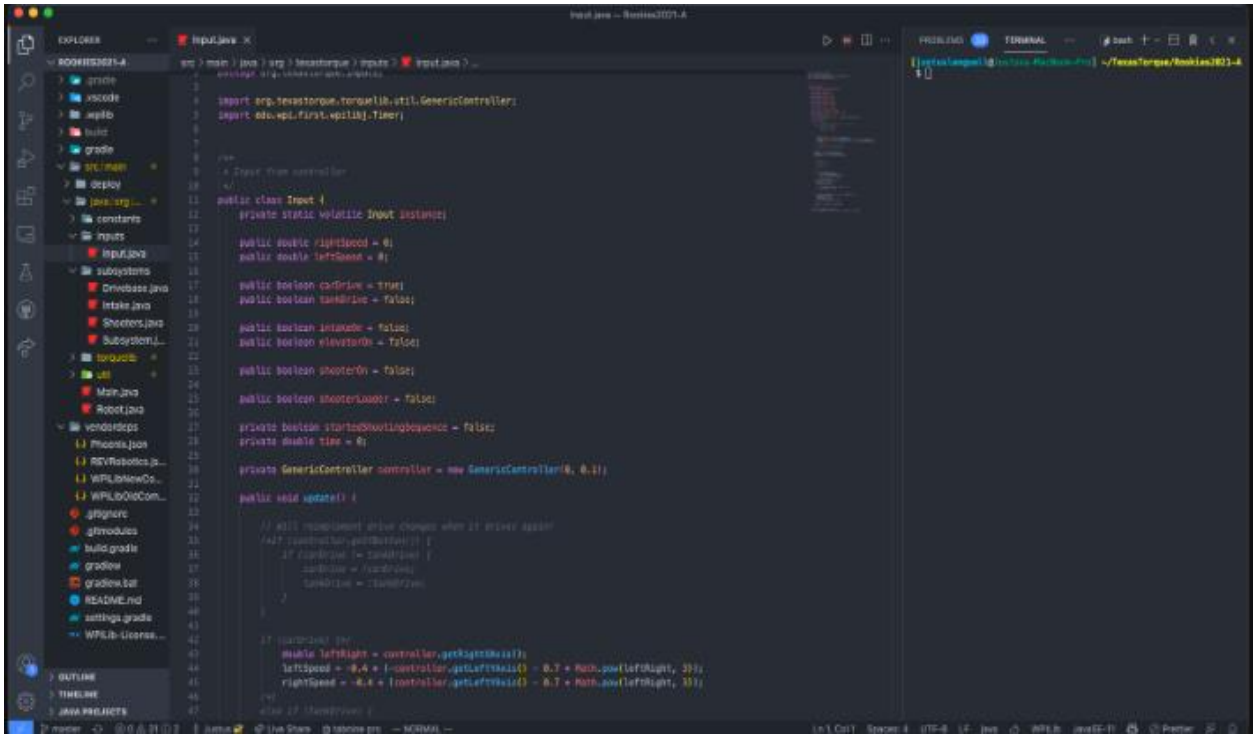


Рисунок 3.1 — Приклад роботи середовища програмування VS Code

WebStorm — це інтегроване середовище розробки (IDE) для веб-розробки, розроблене компанією JetBrains. Це потужний інструмент, спеціалізований на роботі з веб-технологіями, такими як HTML, CSS, JavaScript, та різними фреймворками. WebStorm надає розширену підтримку для різних мов, таких як HTML, CSS, JavaScript, TypeScript, а також для популярних фреймворків, таких як Angular, React, Vue.js, Node.js, і багатьох інших [24]. Середовище надає інтелектуальне автоматичне завершення коду, аналіз коду та інтегровані підказки роблять процес написання коду більш ефективним та допомагають у виявленні помилок. WebStorm підтримує вбудовану відладку для JavaScript, TypeScript, і Node.js, з можливістю встановлення точок зупинки, крокування коду та виведення змінних. Середовище програмування має зручну інтеграцію з популярними системами

контролю версій, такими як Git, SVN, Mercurial, дозволяє ефективно керувати версіями коду та спільно працювати в командах.

WebStorm надає можливість автоматичного форматування коду, що спрощує дотримання стандартів коду та поліпшує читабельність. Засоби рефакторингу дозволяють легко оптимізувати та переробляти код, забезпечуючи його чистоту та оптимальність, інтегрується з популярними фронтенд-інструментами, такими як npm, Yarn, Bower, Gulp, і Webpack, що полегшує управління залежностями та автоматизацію завдань. Live Templates дозволяють використовувати шаблони коду для швидкого написання стандартних конструкцій. Live Edit дозволяє безпосередньо вносити зміни в коді та бачити їх в реальному часі на веб-сторінці без перезавантаження.

WebStorm володіє багатофункціональністю, спрямованою на підвищення продуктивності розробників та полегшення роботи з різними технологіями веб-розробки. (рисунок 3.2).

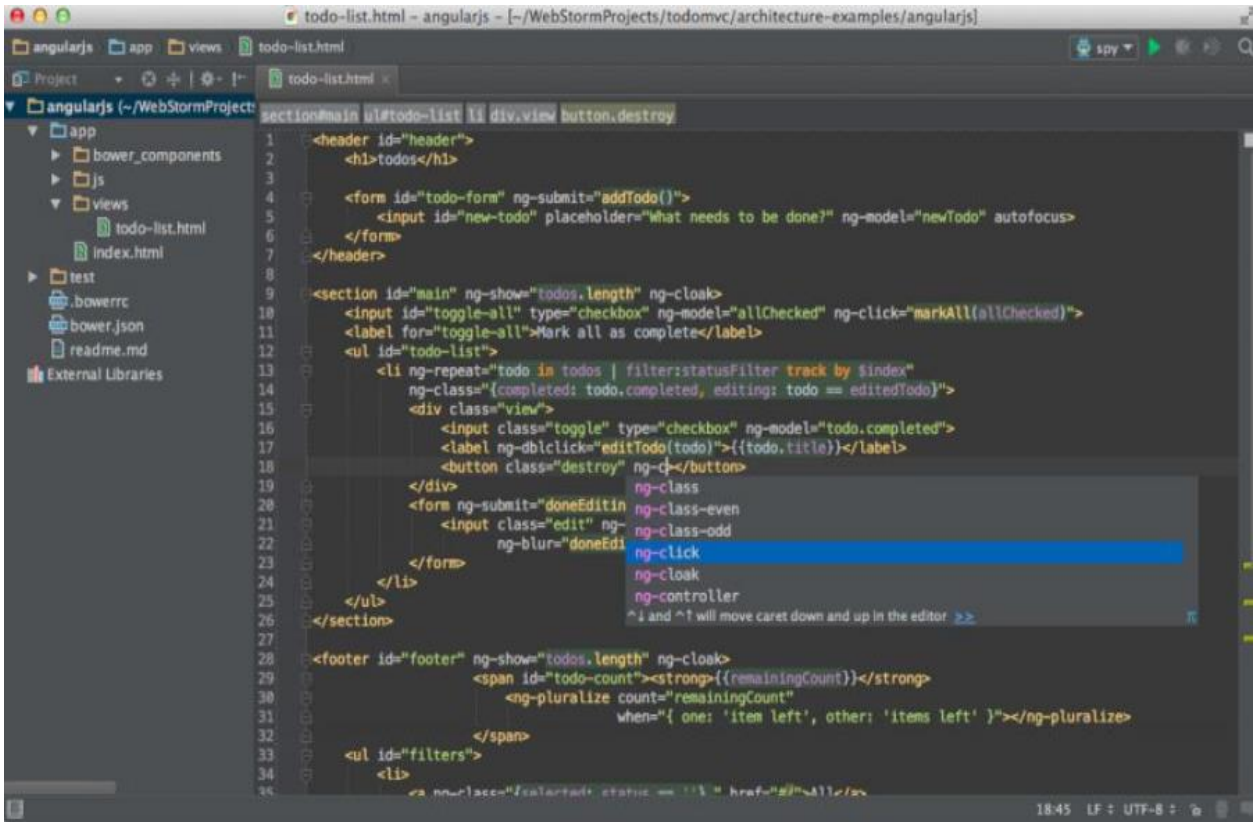



Рисунок 3.2 — Приклад роботи середовища програмування WebStorm

Порівняння розглянутих середовищ програмування за обраними критеріями наведено у таблиці 3.2.

Таблиця 3.2 — Порівняння середовищ програмування

| Характеристика | VS Code | WebStorm |
|---|---|---|
| Вартість | Безкоштовний з відкритим вихідним кодом | Платний, існує безкоштовна пробна версія |
| Розширюваність | Велика кількість розширень та плагінів | Включає в себе багато інтегрованих інструментів та плагінів |
| Мови програмування | Підтримує багато мов програмування, широкий вибір розширень | Основний фокус на підтримці JavaScript, але підтримує інші мови |
| Редагування коду | Добра підтримка автозаповнення коду, IntelliSense | Висока підтримка автозаповнення, інтеграція з інструментами від JetBrains |
| Відлагодження коду | Розширені засоби для відлагодження (за допомогою розширень) | Вбудований відлагоджувач з багатьма функціями |
| Інтеграція з системами керування версіями | Велика підтримка Git, плагіни для інших систем | Вбудована підтримка Git, Mercurial, та інших систем |
| Веб-розробка | Має багато розширень для веб-розробки, підтримка HTML, CSS, та JavaScript | Розроблений як інтегроване середовище для веб-розробки, з підтримкою Angular, |

Продовження таблиці 3.2

| | | |
|---|---|---|
| | | React, інших фреймворків |
| Швидкість та ресурси | Легкий та швидкий, використовує менше ресурсів | Більше вимог до ресурсів, але відмінна продуктивність для великих проектів |

Обрано Visual Studio Code серед інших середовищ програмування через його безкоштовність, швидкість та легкість використання, а також через розширену підтримку веб-розробки завдяки багатим ресурсам розширень.

## 3.2 Розробка web-додатку системи

Розпочнемо створення проекту для клієнтської частини системи обслуговування та діагностування. Відкриймо термінал у Visual Studio Code та введемо команду: npx create-react-app diplomclinic. Ця команда використовує create-react-app для автоматичного створення нового проекту React з усіма необхідними налаштуваннями та структурою каталогів.

Після цього в каталозі автоматично створився файл package.json. Webpack — це інструмент для модульної збірки, який об'єднує ресурси та бібліотеки, необхідні для проекту, у єдиний файл. Для його встановлення використовуємо наступну команду: npm install webpack webpack-cli --save-dev. Таким чином, webpack та його інтерфейс доступу через командний рядок додані до проекту як dev-залежності.

У React існує вбудована система маршрутизації, яка дозволяє призначати компоненти для різних запитів у додатку. Основним інструментом для цього є бібліотека react-router, але для зручності навігації у браузері рекомендується використовувати react-router-dom. Для встановлення потрібних залежностей в проекті використовується команда: npm install react-router react-router-dom. Таким чином, ви матимете можливість

використовувати функціонал маршрутизації в React завдяки бібліотекам react-router і react-router-dom.

Кожен маршрут визначається елементом `Route`, який має кілька атрибутів для конфігурації.

— path — це шаблон адреси, який порівнюється з URL клієнта для визначення, яку компоненту слід показати;

— component — це елемент, який відображатиметься, коли викликається відповідний маршрут, визначений шляхом;

— render — цей атрибут використовується для передачі функціональної компоненти та об'єкта props для цієї компоненти, він дозволяє досягти більш гнучкої конфігурації відображення компоненти;

— exact — якщо цей атрибут встановлено, то порівняння маршрутів відбуватиметься строго, тобто клієнтський URL повинен точно збігатися з шляхом, вказаним у маршруті.

В головному файлі App.js запишемо імпорти інших необхідних компонент та бібліотек:

```
import React from 'react';
import { BrowserRouter as Router, Switch, Route } from 'react-router-dom';
import DecisionTree from './DecisionTree';
import LoginPage from './LoginPage';
import PatientPage from './PatientPage';
const App = () => {
  return (
    <Router>
      <div className="App">
        <header className="App-header">
          <h1>Система діагностики</h1>
        </header>
        <main>
          <Switch>
```

```
          <Route path="/login" component={LoginPage} />
          <Route path="/patient" component={PatientPage} />
          <Route path="/" component={DecisionTree} />
        </Switch>
      </main>
    </div>
  </Router>
 );
};
export default App;
```

У компоненті DoctorRegistrationPage.js код логіки збереження даних лікаря на сервер:

```
import React, { useState } from 'react';
import axios from 'axios';
const DoctorRegistrationPage = () => {
  const [formData, setFormData] = useState({
    firstName: '',
    lastName: '',
    phoneNumber: '',
    email: '',
    username: '',
    password: '',
    confirmPassword: '',
  });
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((prevData) => ({ ...prevData, [name]: value }));
  };
  const handleSubmit = async (e) => {
```

```
    e.preventDefault();
    try {
      const response = await axios.post('/api/doctors/register', formData);
      console.log(response.data);
    } catch (error) {
      console.error('Помилка при реєстрації лікаря:', error);
    }
  };
  return (
    <div>
      <h2>Реєстрація лікаря</h2>
      <form onSubmit={handleSubmit}>
        {/* Форма реєстрації лікаря */}
        {/* ... */}
        <button type="submit">Зареєструватися</button>
      </form>
    </div>
  );
};
export default DoctorRegistrationPage;
```

Компонент PatientPage.jsx відповідає за регістрацію (запис) в журналі відвідувань та вибір лікаря, лістнінг якого представлений у додатку Г.

База даних web-системи складається з 4 таблиць: реєстрація лікарів, журнал запису пацієнтів, журнал для обліку пацієнтів певного лікаря, база даних діагнозів та їх симптомів

Для під’єднання бази даних використовуємо Node.js та у файл Diagnosis.js додаємо такий код:

```
mongoose.connect('mongodb://127.0.0.1:27017/hospital-app');
mongoose.connection
```

```
    .once('open', () => {
        console.log('Mongoose - successful connection...');
        app.listen(PORT, () => {
            console.log(`Server started on port ${PORT}`);
        });
    })
    .on('error', error => console.error(error));
```

Слід також написати логіку виконання дерева рішень для діагностування , для цього у файл DecisionTree.jsx впишемо такий код логіки:

```
const makeDecisionTree = (diagnosesDatabase) => {
  const diagnosisKeys = Object.keys(diagnosesDatabase);
  if (diagnosisKeys.length === 0) {
    return null;
  }
  const rootDiagnosis = diagnosisKeys[0];
  const rootSymptoms = diagnosesDatabase[rootDiagnosis].symptoms;
  return {
    diagnosis: rootDiagnosis,
    symptoms: rootSymptoms,
    question: `Чи є у вас ${rootSymptoms[0]}?`,
    yes_node: makeSubtree(diagnosesDatabase, rootDiagnosis, rootSymptoms[0],
true),
    no_node: makeSubtree(diagnosesDatabase, rootDiagnosis, rootSymptoms[0],
false),
  };
};
const makeSubtree = (diagnosesDatabase, currentDiagnosis, currentSymptom,
isAnswerYes) => {
```

```
  const nextDiagnosis = getNextDiagnosis(diagnosesDatabase, currentDiagnosis,
currentSymptom, isAnswerYes);
  if (nextDiagnosis) {
    const nextSymptoms = diagnosesDatabase[nextDiagnosis].symptoms;
    return {
      diagnosis: nextDiagnosis,
      symptoms: nextSymptoms,
      question: `Чи є у вас ${nextSymptoms[0]}?`,
      yes_node: makeSubtree(diagnosesDatabase, nextDiagnosis, nextSymptoms[0],
true),
      no_node: makeSubtree(diagnosesDatabase, nextDiagnosis, nextSymptoms[0],
false),
    };
  }
  return null;
};
const getNextDiagnosis = (diagnosesDatabase, currentDiagnosis, currentSymptom,
isAnswerYes) => {
  const currentDiagnosisSymptoms =
diagnosesDatabase[currentDiagnosis].symptoms;
  const currentIndex = currentDiagnosisSymptoms.indexOf(currentSymptom);
  if (isAnswerYes) {
    if (currentIndex < currentDiagnosisSymptoms.length - 1) {
      return currentDiagnosis;
    }
  } else {
    const nextDiagnosisIndex =
Object.keys(diagnosesDatabase).indexOf(currentDiagnosis) + 1;
    if (nextDiagnosisIndex < Object.keys(diagnosesDatabase).length) {
      return Object.keys(diagnosesDatabase)[nextDiagnosisIndex];
```

```
    }
  }
  return null;
};
```

Опишемо алгоритм виконання дерева рішень. Початковим вузлом дерева буде визначено перший симптом першого діагнозу у базі даних діагнозів. Система запитуватиме пацієнта чи є в нього цей симптом, наприклад це буде головна біль. Якщо пацієнт відповідає «так», то система будує розгалуження дерева рішень, в результаті якого наступне питання буде з другим по рахунку симптомом того ж самого діагнозу. Якщо відповідь буде «ні», то система будує відгалуження дерева рішень, в результаті якого усі діагнози з цим симптомом відкидуються, і система переходить до пошуку наступного діагнозу з співпадаючим першим симптомом, але без запитуваного другого симптому. З цим алгоритмом відбувається опитування до поки симптоми пацієнта не зійдуться з симптомами відповідного діагнозу у базі даних.

# 4 ТЕСТУВАННЯ WEB-ДОДАТКУ

## 4.1 Опис технологій тестування web-додатків

Веб-тестування — це уважний аналіз веб-сайту з метою виявлення можливих помилок та повна перевірка його функціоналу перед впровадженням. До того, як веб-система стає доступною для кінцевих користувачів, вона пройшла повне тестування від початку до кінця. Процес також включає оцінку зручності та естетичності веб-сайту для користувача, швидкості доступу до необхідної інформації, а також його надійності та безпеки.

Важливою частиною веб-тестування є оцінка того, наскільки веб-сайт відповідає поставленим бізнес-цілям, визначеним на етапі початкового проекту. Головна мета веб-тестування полягає в виявленні можливих проблем, які можуть виникнути під час експлуатації веб-сайту чи програми, таких як помилки чи дефекти. Крім цього, веб-тестування може служити для ідентифікації конкретних областей веб-сайту, які можна оптимізувати для досягнення кращих результатів, таких як збільшення кількості запитів, підвищення обсягу продажів, привертання більше постійних відвідувачів і так далі.

Тестування — це процес перевірки та оцінки характеристик системи чи програмного продукту з метою визначення відповідності вимогам, виявлення помилок або недоліків, а також забезпечення якості та надійності продукту.

Функціональне тестування спрямоване на перевірку функцій програми, щоб забезпечити їх відповідність вимогам. Це включає перевірку роботи коректності, взаємодії та виконання функціональних вимог продукту.

Відмовостійке тестування спрямоване на перевірку того, як система або програмне забезпечення реагує на відмови (помилки) в роботі. Мета — визначити стійкість системи до відмов, виявити проблеми та забезпечити, щоб вони не призводили до серйозних витрат чи непередбачених наслідків.

Інтеграційне тестування спрямоване на перевірку взаємодії між різними компонентами або модулями програмного забезпечення. Його мета —

переконатися, що вони працюють спільно як ціла система і виконують свої функції безпомилково.

Тестування користувацького інтерфейсу — це процес, під час якого перевіряється, чи відповідає веб-застосунок вимогам, з фокусуванням на способі взаємодії користувача з програмою та переконанням у правильності та зручності цієї взаємодії.

Тестування функціональності інтерфейсу включає в себе перевірку правильності відображення всіх елементів, таких як кнопки та поля введення, та їх взаємодії. Важливо переконатися, що елементи інтерфейсу відповідають специфікаціям та дизайну.

Тестування навігації зосереджується на перевірці логічності та ефективності переміщення між сторінками та розділами застосунку. Також важливо перевіряти роботу посилань та кнопок для переходу між частинами додатку.

Відповідність дизайну включає перевірку того, чи відповідають елементи дизайну специфікаціям та стандартам. Тестується відображення інтерфейсу на різних роздільних здатностях екрану та на різних пристроях. Тестування сумісності інтерфейсу проводиться для перевірки його взаємодії з різними браузерами та операційними системами. Також важливо перевірити, як добре інтерфейс адаптується до різних пристроїв.

Адаптивний дизайн передбачає тестування реакції інтерфейсу на зміни розмірів вікна браузера та відображення на різних пристроях. Тестування форм введення даних включає перевірку їхньої валідації та взаємодії з сервером при їхньому відправленні.

Життєвий цикл тестування програмного забезпечення — це процес, що включає в себе планування, розробку, виконання тестів та аналіз результатів з метою забезпечення якості програмного продукту. Цей цикл відображає етапи тестування від початкового планування до завершення проекту.

Починається процес з планування, де визначається мета та обсяг тестування, а також створюється план, включаючи стратегію, ресурси, графік

і критерії завершення. Далі йде дизайн тестових випадків, розробка конкретних випадків тестування, визначення даних та очікуваних результатів.

Після цього настає етап реалізації тестових сценаріїв, де готують тестові дані та налаштовують середовище для виконання тестів. Виконують тест-кейси та збирають результати, при цьому виявляють і документують помилки, які обговорюють з розробниками для виправлення.

Після виконання тестів проводиться аналіз результатів, порівняння фактичних та очікуваних результатів, і визначення рівня якості продукту. Після виявлення та виправлення помилок розпочинається регресійне тестування, щоб переконатися, що зміни не вплинули на інші частини системи.

По завершенню тестування готується звіт, в якому фіксуються виявлені проблеми та можливі покращення. Завершується процес передачею результатів команді розробників або менеджменту проекту.

## 4.2 Тестування роботи web-додатку

Розпочинемо тестування, використовуючи директиви для відкриття web-застосунку: ‘npm run watch’ для автоматичного оновлення компонентів, ‘npm start’ для запуску клієнту та ‘npm run start’ для запуску web-серверу. Зараз проведемо перевірку користувацького інтерфейсу.

Початковою сторінкою є сторінка з вибором ролі на web-додатку (рисунок 4.1).

Якщо користувач обирає роль лікаря, то йому необхідно зареєструвати обліковий запис, заповнивши поля, якщо його ще нема (рисунок 4.2), або увійти в нього (рисунок 4.3).

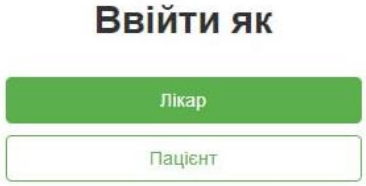



Рисунок 4.1 — Початкова сторінка web-додатку

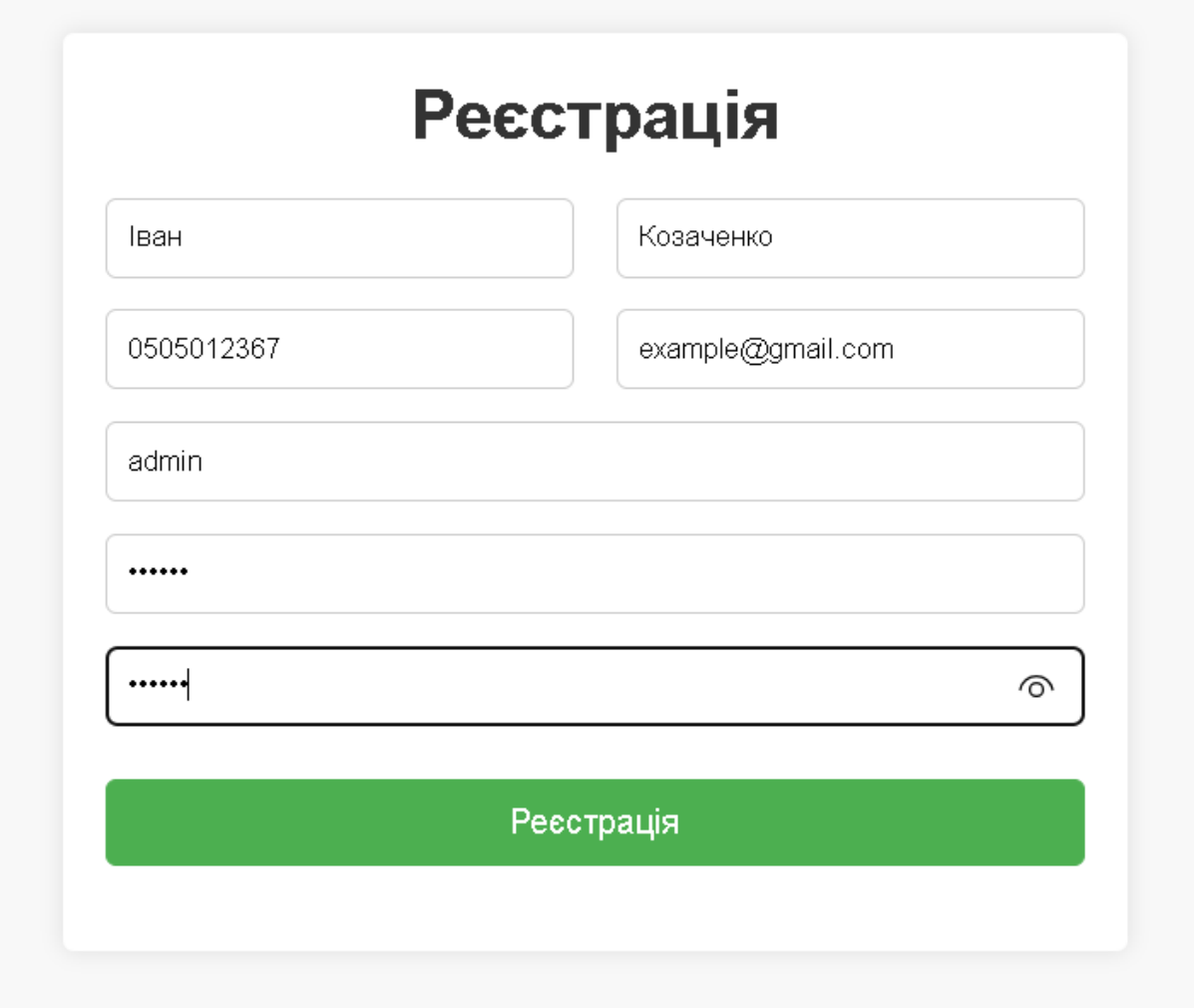



Рисунок 4.2 — Форма реєстрації лікаря

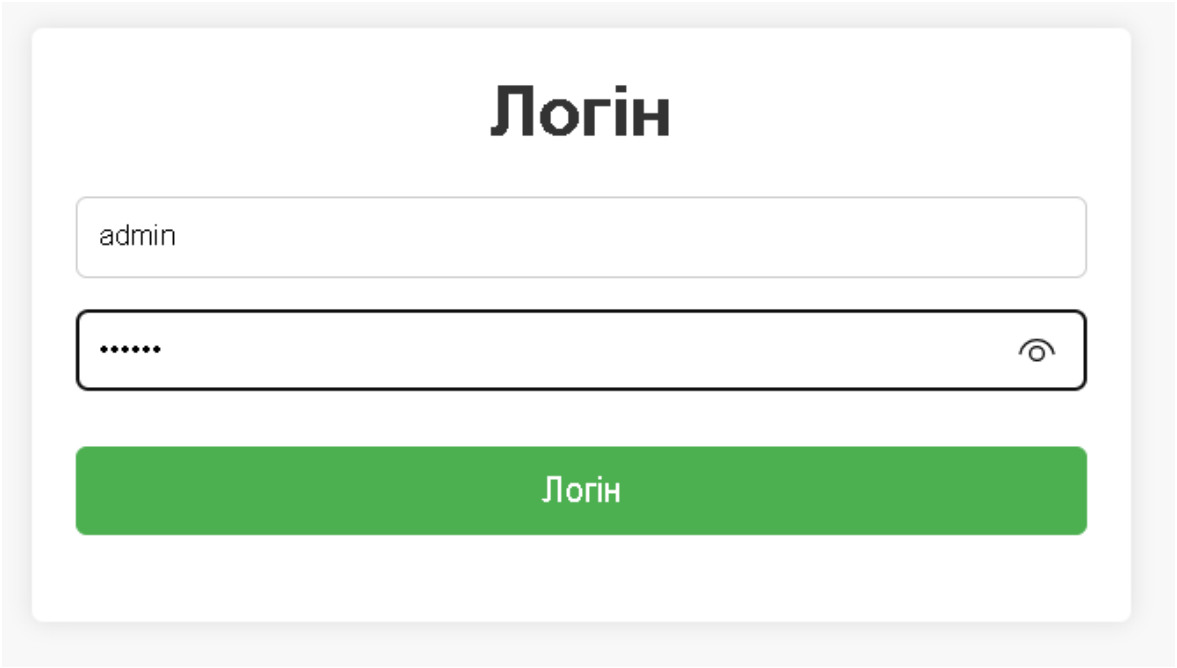



Рисунок 4.3 — Форма авторизації лікаря

Якщо ж користувач обирає роль пацієнта, він натискає на кнопку «Пацієнт», то користувач переходить на сторінку з формою запису до лікаря, де вносить ПІБ, вік, дату запису та лікаря до якого записується (рисунок 4.4).

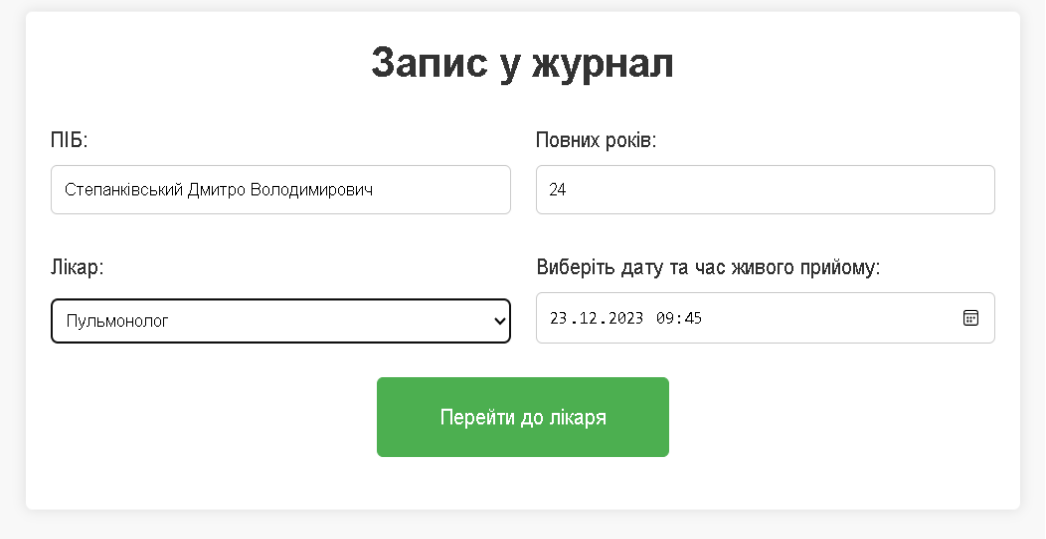



Рисунок 4.4 — Форма запису до лікаря

Після натискання кнопки «Перейти до лікаря», пацієнт переходить на сторінку обраного лікаря де проходить опитування щодо симптомів (рисунок 4.5):

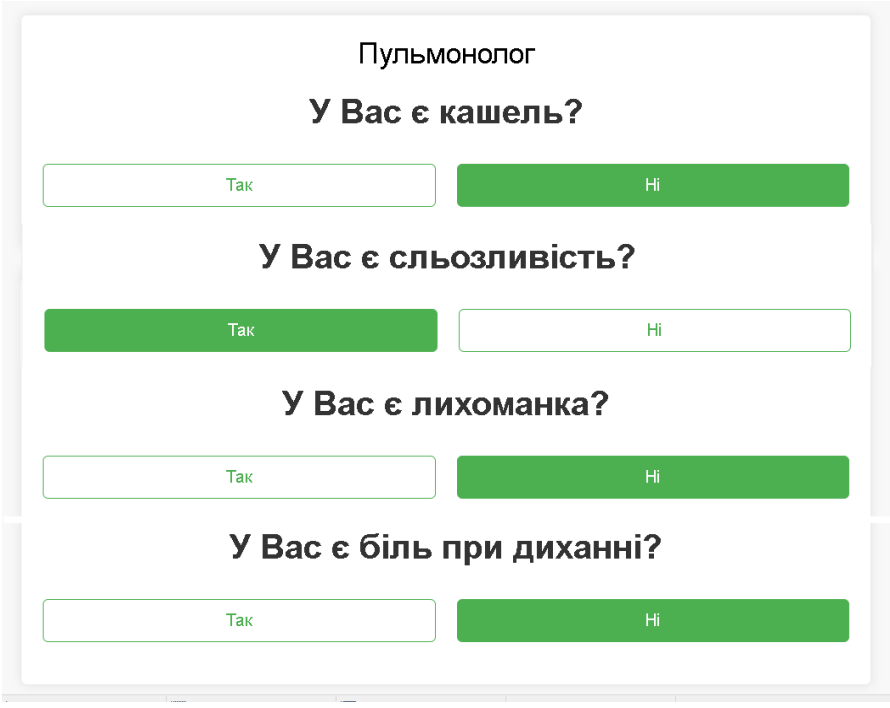



Рисунок 4.5 — Опитування в обраного лікаря

По завершенню опитування, пацієнт отримує свій попередній діагноз та рекомендацію все ж відвідати лікаря (рисунок 4.6)

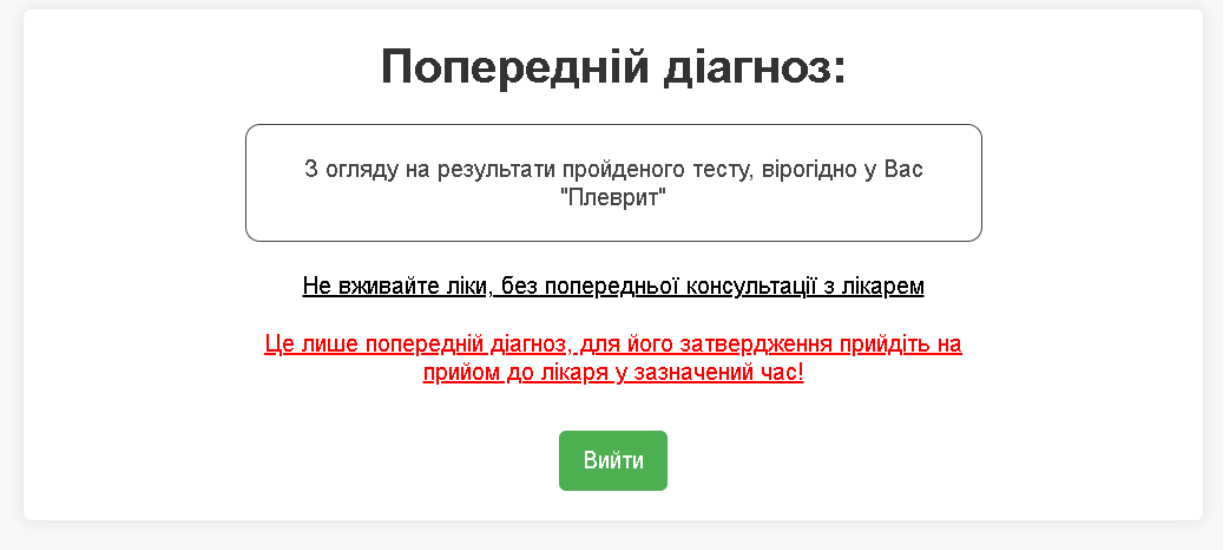



Рисунок 4.6 — Сторінка попереднього діагнозу пацієнта

При вході з ролі адміністратора можна переглянути сторінку список записаних пацієнтів, записаних до лікаря (рисунок 4.7)

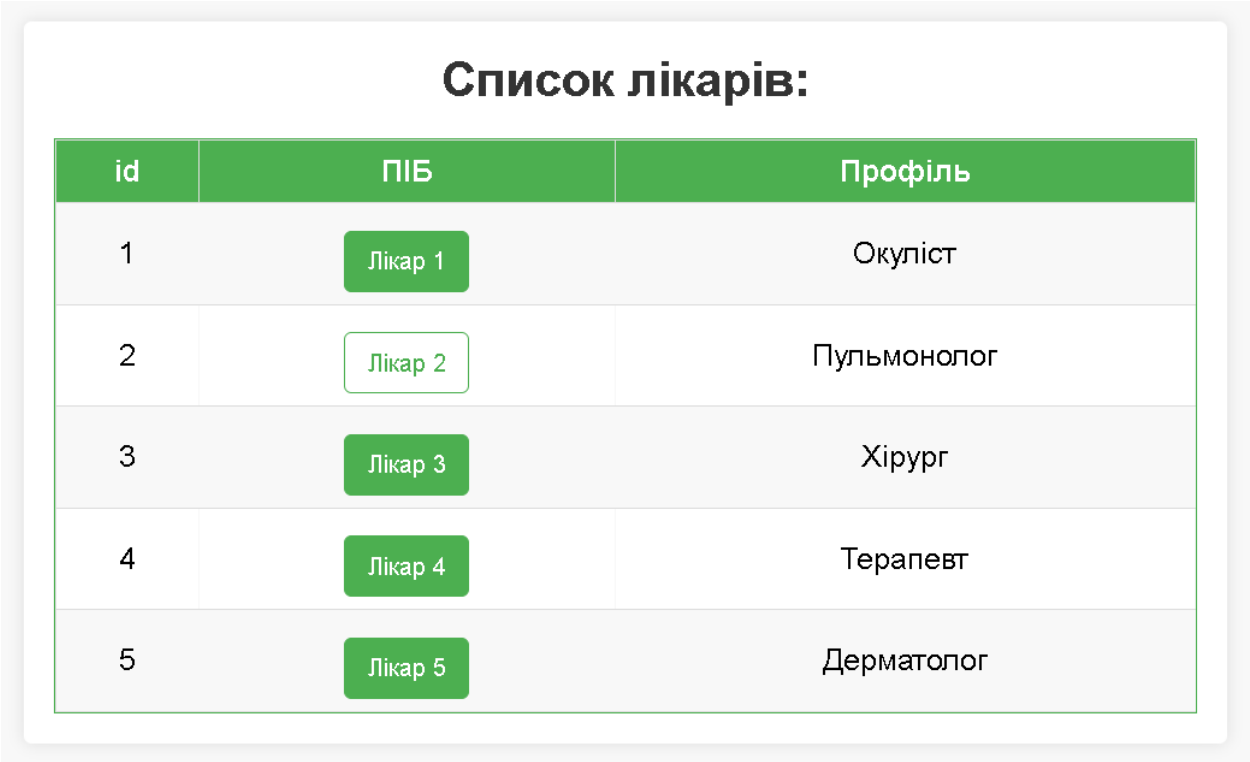
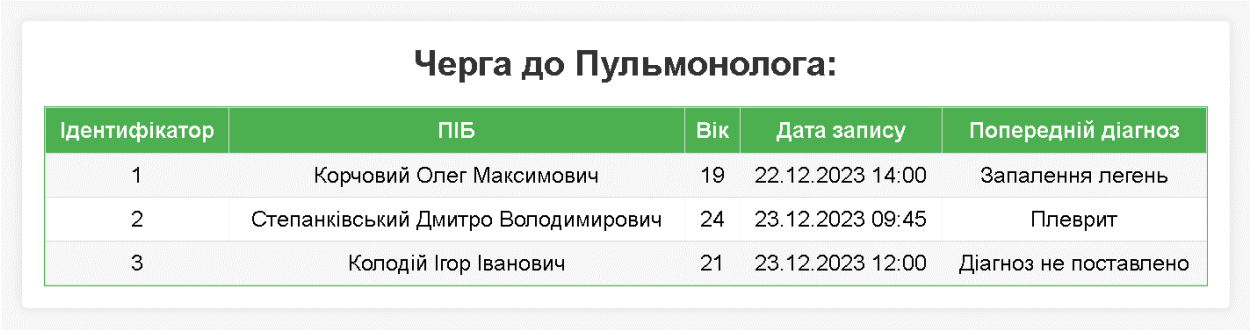



Рисунок 4.7 — Сторінка списку лікарів

Натиснемо кнопку «Лікар 2» та переглянемо записаних до нього пацієнтів та їх попередні діагнози (рисунок 4.8):



**Черга до Пульмонолога:**

| Ідентифікатор | ПІБ | Вік | Дата запису | Попередній діагноз |
|---|---|---|---|---|
| 1 | Корчовий Олег Максимович | 19 | 22.12.2023 14:00 | Запалення легень |
| 2 | Степанківський Дмитро Володимирович | 24 | 23.12.2023 09:45 | Плеврит |
| 3 | Колодій Ігор Іванович | 21 | 23.12.2023 12:00 | Діагноз не поставлено |

Рисунок 4.8 — Список пацієнтів, записаних до пульмонолога

# 5 ЕКОНОМІЧНА ЧАСТИНА

## 5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного та технологічного аудиту є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Магістерська кваліфікаційна робота за темою "Веб-додаток для обслуговування та діагностування пацієнтів у медичному закладі з використанням дерева рішень" передбачає вдосконалення системи діагностування, яка покращує та оптимізує діагностування у лікарні.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету д.т.н. проф. кафедри ОТ Азаров Олексій Дмитрович, к.т.н. доц. кафедри ОТ Богомолов Сергій Віталійович, к.т.н. доц. кафедри ОТ Савицька Людмила Анатоліївна. Для проведення технологічного аудиту було використано таблицю 5.1, в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу розробки.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

| Критерії оцінювання та бали (за 5-ти бальною шкалою) | | | | | |
|---|---|---|---|---|---|
| Кри-терій | 0 | 1 | 2 | 3 | 4 |
| Технічна здійсненність концепції: | | | | | |
| 1 | Достовірність концепції не підтверджена | Концепція підтверджена експертними висновками | Концепція підтверджена розрахунками | Концепція перевірена на практиці | Перевірено роботоздатність продукту в реальних умовах |
| Ринкові переваги (недоліки): | | | | | |
| 2 | Багато аналогів на малому ринку | Мало аналогів на малому ринку | Кілька аналогів на великому ринку | Один аналог на великому ринку | Продукт не має аналогів на великому ринку |

Продовження таблиці 5.1

| | | | | | |
|---|---|---|---|---|---|
| 3 | Ціна продукту значно вища за ціни аналогів | Ціна продукту дещо вища за ціни аналогів | Ціна продукту приблизно дорівнює цінам аналогів | Ціна продукту дещо нижче за ціни аналогів | Ціна продукту значно нижче за ціни аналогів |
| 4 | Технічні та споживчі властивості продукту значно гірші, ніж в аналогів | Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів | Технічні та споживчі властивості продукту на рівні аналогів | Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів | Технічні та споживчі властивості продукту значно кращі, ніж в аналогів |
| 5 | Експлуатаційні витрати значно вищі, ніж в аналогів | Експлуатаційні витрати дещо вищі, ніж в аналогів | Експлуатаційні витрати на рівні експлуатаційних витрат аналогів | Експлуатаційні витрати трохи нижчі, ніж в аналогів | Експлуатаційні витрати значно нижчі, ніж в аналогів |
| Ринкові перспективи | | | | | |
| 6 | Ринок малий і не має позитивної динаміки | Ринок малий, але має позитивну динаміку | Середній ринок з позитивною динамікою | Великий стабільний ринок | Великий ринок з позитивною динамікою |
| 7 | Активна конкуренція великих компаній на ринку | Активна конкуренція | Помірна конкуренція | Незначна конкуренція | Конкурентів немає |
| Практична здійсненність | | | | | |
| 8 | Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї | Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців | Необхідне незначне навчання фахівців та збільшення їх штату | Необхідне незначне навчання фахівців | Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї |

Продовження таблиці 5.1

| | | | | | |
|---|---|---|---|---|---|
| 9 | Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні | Потрібні незначні фінансові ресурси. Джерела фінансування відсутні | Потрібні значні фінансові ресурси. Джерела фінансування є | Потрібні незначні фінансові ресурси. Джерела фінансування є | Не потребує додаткового фінансування |
| 10 | Необхідна розробка нових матеріалів | Потрібні матеріали, що використовуються у військово-промисловому комплексі | Потрібні дорогі матеріали | Потрібні досяжні та дешеві матеріали | Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві |
| 11 | Термін реалізації ідеї більший за 10 років | Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років | Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років | Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років | Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років |
| 12 | Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту | Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу | Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу | Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту | Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту |

Таблиця 5.2 — Рівні комерційного потенціалу розробки

| Середньоарифметична сума балів СБ, розрахована на основі висновків експертів | Рівень комерційного потенціалу розробки |
|---|---|

Продовження таблиці 5.2

| | |
|---|---|
| 0-10 | Низький |
| 11-20 | Нижче середнього |
| 21-30 | Середній |
| 31-40 | Вище середнього |
| 41-48 | Високий |

У таблиці 5.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 5.3 — Результати оцінювання комерційного потенціалу розробки

| Критерії | Прізвище, ініціали, посада експерта | | |
|---|---|---|---|
| | 1 — Азаров | 2 — Богомолов | 3 — Савицька |
| | Бали, виставлені експертами: | | |
| 1 | 4 | 4 | 4 |
| 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 |
| 4 | 3 | 4 | 3 |
| 5 | 2 | 2 | 2 |
| 6 | 3 | 3 | 3 |
| 7 | 2 | 3 | 3 |
| 8 | 3 | 4 | 3 |
| 9 | 3 | 2 | 1 |
| 10 | 3 | 2 | 3 |
| 11 | 3 | 4 | 4 |
| 12 | 4 | 3 | 3 |
| Сума балів | $СБ_1$=35 | $СБ_2$=36 | $СБ_3$=34 |
| Середньоарифметична сума балів $\overline{СБ}$ | $\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{35+36+34}{3} = 35$ | | |

Розрахована нами на основі висновків експертів середньоарифметична сума балів склала 35 балів. Згідно таблиці 5.2 вважається, що рівень комерційного потенціалу проведених досліджень є вище середнього.

## 5.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

Основна заробітна плата кожного із дослідників $З_0$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою (5.1):

$$З_0 = \frac{М}{Т_р} * t(грн), \quad (5.1)$$

де М — місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), 28000 грн.;

$Т_Р$ — число робочих днів в місяці; приблизно $Т_Р \approx 21…23$ дні;

t — число робочих днів роботи дослідника, 90 днів.

Зведемо сумарні розрахунки до таблиці 5.4.

Таблиця 5.4 — Основна заробітна плата розробників

| Найменування посади | Місячний посадовий оклад, грн. | Оплата за робочий день, грн. | Число днів роботи | Витрати на заробітну плату грн. |
|---|---|---|---|---|
| Керівник | 15000,00 | 681,81 | 3 | 2045,40 |
| Розробник | 28000 | 1272,72 | 90 | 114545,45 |
| Всього | | | | 116590,85 |

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали участь в розробці нового технічного рішення розраховується як 10-12% від основної заробітної плати робітників за формулою (5.2).

На даному підприємстві додаткова заробітна плата начисляється в розмірі 11% від основної заробітної плати.

$$З_д = \frac{(З_о + З_р) * Н_{дод}}{100\%} \quad (5.2)$$

$$З_д = 0{,}11 \cdot 116590{,}85 = 12825(грн)$$

Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (5.2):

$$Н_{ЗП} = \frac{(З_о + З_д) * \beta}{100} \quad (5.3)$$

де $З_о$ — основна заробітна плата розробників, грн.;

$З_д$ — додаткова заробітна плата всіх розробників та робітників, грн.;

β — ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, %.

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді:

$$Н_{ЗП} = \frac{(116590{,}85 + 12825) \cdot 22}{100} = 28471{,}48(грн)$$

Програмне забезпечення для наукової роботи включає витрати на розробку та придбання спеціальних програмних засобів і програмного

забезпечення необхідного для проведення дослідження. Для нової розробки використовувались безкоштовні програмні засоби.

Амортизація обладнання, комп'ютерів та приміщень, які використовувались під час виконання даного етапу роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо за формулою (5.4).

$$A = \frac{Ц \cdot T}{Ткор \cdot 12} \quad [грн], \tag{5.4}$$

де Ц — балансова вартість даного виду обладнання (приміщень), грн.;

$Т_{кор}$ — час користування;

Т — термін використання обладнання (приміщень), цілі місяці.

Згідно пункту 137.3.3 Податкового кодексу амортизація нараховується на основні засоби вартістю понад 6500 грн. В нашому випадку для написання магістерської роботи використовувалися три персональних комп'ютери кожен вартістю 40000 грн, а також оренда приміщення, вартість оренди 15000 грн.

$$А = А_к + А_п = 5250$$

$$А_к = \frac{120000 \cdot 2}{5 \cdot 12} = 4000$$

$$А_п = \frac{15000 \cdot 2}{2 \cdot 12} = 1250$$

До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що використовуються з технологічною метою на проведення досліджень, та розраховуються за формулою (5.5).

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i} \tag{5.5}$$

де $W_{yt}$ — встановлена потужність обладнання, кВт;

$t_i$ — тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ — вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ — коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

$\eta_i$ — коефіцієнт корисної дії обладнання, $\eta_i < 1$.

Для написання магістерської роботи використовувалися три персональних комп'ютери для яких розрахуємо витрати на електроенергію.

$$В_e = \frac{1{,}5 \cdot 360 \cdot 7{,}6 \cdot 0{,}5}{0{,}8} = 2565 \text{ (грн)}$$

Витрати за доступ до Інтернет можна розрухувати за формулою (5.6):

$$В_{ді} = Ц_{ді} \cdot Т \text{ [грн]}, \quad (5.6)$$

де $Ц_{ді}$ — це ціна доступу за місяць;

Т — кількість місяців використання доступу до мережі.

$$В_{ді} = 420 \cdot 3 = 1260{,}00 \text{ (грн)}.$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР:

$$В = 116590{,}85 + 12825 + 28471{,}48 + 5250 + 2565 + 1260 = 166962{,}33 \text{(грн)}$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою (5.7):

$$ЗВ = \frac{В}{\eta}, \quad (5.7)$$

де η — коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії впровадження, то коефіцієнт $\eta = 0{,}9$. Звідси:

$$ЗВ = \frac{166962{,}33}{0{,}9} = 185513{,}7(\text{грн.}),$$

Витрати на виконання наукової роботи та впровадження її результатів становитиме 185513,7 грн.

5.3 Розрахунок економічної ефективності та обґрунтування економічної доцільності комерціалізації науково-технічної розробки.

Економічна ефективність дозволяє спрогнозувати чистий прибуток, який може бути отриманий від впровадження розробленої системи.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу.

Для розрахунку збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки використовується формула формулою (5.8):

$$\Delta П_i = \sum_1^n \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N\right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100}\right), \quad (5.8)$$

де $\Delta Ц_0$ — покращення основного оціночного показника від впровадження результатів розробки у даному році;

N — основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження нової розробки;

ΔN — покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

Цо — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів розробки;

n — кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт л = 0,8333;

ρ — коефіцієнт, який враховує рентабельність продукту. р = 0,3;

ϑ — ставка податку на прибуток. У 2023 році – 18%.

В результаті впровадження наукової розробки покращується якість певного продукту, що дозволяє підвищити ціну його реалізації на 1000 грн. Кількість користувачів збільшиться протягом першого року на 400, другого — 800, третього — 2400. Реалізація продукції до впровадження результатів наукової розробки складала 800 користувачів, а її ціна — 8000 грн.

Розрахуємо показник прибутку впродовж трьох років відносно базового.

$$\Delta П_1 = (8000 \cdot 1 + (8000 + 800) \cdot 400) \cdot 0.833 \cdot 0.3 \cdot \left(1 - \frac{18}{100}\right) = 722960{,}7 \text{ грн},$$

$$\Delta П_2 = (8000 \cdot 1 + (8000 + 800) \cdot 800) \cdot 0{,}833 \cdot 0{,}3 \cdot \left(1 - \frac{18}{100}\right) = 1444262 \text{ грн},$$

$$\Delta П_3 = (8000 \cdot 1 + (8000 + 800) \cdot 2400) \cdot 0.833 \cdot 0.3 \cdot \left(1 - \frac{18}{100}\right) = 4329507{,}5 \text{ грн},$$

Далі за формулою (5.9) розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} \quad (5.9)$$

де ΔПі — збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

Т — період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації розробки, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15. В Україні рівень інфляції за підсумком 2023 року склав 10.6%, прогнозований рівень на 2024 рік – 8.5% ;

t — період часу (в роках) від моменту початку впровадження розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = \frac{722960,7}{1+0.1} + \frac{1444262}{(1+0.085)^2} + \frac{4329507,5}{(1+0.085)^3} = 657237 + 1226835,99 + 3389606,48 = 5273679,47 \text{ (грн.)},$$

За формулою (5.10) необхідно розрахувати величину початкових інвестицій PV , які потенційний інвестор має вкласти для впровадження і комерціалізації розробки.

$$PV = k_{інв} \cdot ЗВ, \quad (5.10)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}$ =2...5, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 4 \cdot 185513{,}7 = 742054{,}8 \text{ грн},$$

Тоді абсолютний економічний ефект $Е_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації розробки становитиме:

$$Е_{абс} = ПП - PV, \quad (5.11)$$

де ПП — приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV — теперішня вартість початкових інвестицій, грн.

$$Е_{абс} = 5273679{,}47 - 742054{,}8 = 4531624{,}67 \text{ грн},$$

Оскільки величина економічного ефекту має велике додатне значення, це свідчить про потенційну зацікавленість інвесторів у впровадженні та комерціалізацію розробки.

## 5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Для остаточного прийняття рішення про впровадження розробки та виведення її на ринок необхідно розрахувати внутрішню економічну дохідність $Е_в$ або показник внутрішньої норми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$. Для цього користуються формулою (5.12):

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.12)$$

де $E_{абс}$ — абсолютний економічний ефект вкладених інвестицій, грн; PV – теперішня вартість початкових інвестицій, грн;

$T_ж$ — життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$E_в = \sqrt[3]{1 + \frac{4531624,67}{742054,8}} - 1 = \sqrt[3]{7,10} - 1 = 1,92 - 1 = 0,92 = 92\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою (5.13):

$$\tau = d + f, \quad (5.13)$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d= (0,14…0,2);

f — показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{min} = 0.18 + 0.07 = 0.25 ,$$

Так як $E_в > \tau_{min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Далі розраховується період окупності інвестицій, вкладених у реалізацію проекту за формулою (5.14).

$$T_{ок} = \frac{1}{E_в}, \quad (5.14)$$

де $E_в$ — внутрішня економічна дохідність вкладених інвестицій.

$$T_{ок} = \frac{1}{0.92} = 1{,}08 \text{ року} = 13 \text{ міс} = 1 \text{ рік та } 1 \text{ місяць},$$

Так як $T_{ок} \leq 3…5$-ти років, то це свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

## 5.5 Результати економічного аналізу

В даному розділі було проведено комерційний та технологічний аудит розробки із залученням трьох незалежних експертів. Оцінка комерційного потенціалу показала, що рівень комерційного потенціалу розробки є вище середнього.

Згідно із розрахунками всіх статей витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторської-технологічної роботи загальні витрати на розробку складають 185 513,70 грн.

Розрахована абсолютна ефективність вкладених інвестицій в сумі 5273679,47 грн свідчить про отримання прибутку інвестором від комерціалізації програмного продукту.

Відносна ефективність вкладних інвестицій в проведення наукових досліджень та впровадження їх результатів становить 92%, що вище за мінімальну бар'єрну ставку дисконтування, яка складає 25%. Це означає потенційну зацікавленість інвесторів у фінансуванні розробки.

Термін окупності вкладених у реалізацію проекту інвестицій становить 1,08 року, що також свідчить про доцільність фінансування нової розробки.

# ВИСНОВКИ

У ході виконання магістерської роботи було розроблено інформаційну web-систему обслуговування та діагностування пацієнтів медичного закладу з використанням дерева рішень, що спрощує діагностування пацієнтів онлайн.

У першому розділі було проведено аналіз предметної області, спрямованої на вдосконалення обробки та доступу до персональних медичних даних у медичних закладах, використовуючи інформаційні технології та web-системи

У другому розділі було дано огляд ключових вимог до системи, які будуть виконані на подальших етапах розробки проекту. Крім цього, надано детальний опис алгоритму роботи серверної частини веб-ресурсу, розкрито принципи функціонування адміністративної частини. У розділі також розглянуто структуру та принципи роботи клієнтської частини проекту, що визначають інтерфейс та взаємодію з користувачем.

У третьому розділі було розроблено web-додаток, спроектовано інтерфейс та функціонал системи, розроблено базу даних, у яку записуються дані пацієнтів і лікарів окремо, та базу даних діагнозів .

У четвертому розділі було проведено тестування користувацього інтерфейсу веб-застосунку, і зроблено висновок, що все працює коректно.

У п’ятому розділі було виконано економічні розрахунки, в результаті яких зроблено висновок, що потенційна зацікавленість інвесторів у фінансуванні розробки є досить високою.

Отже, у результаті виконання магістерської кваліфікаційної роботи було отримано працездатний web-додатку, з готовою робочою базою даних. Було розроблено алгоритм роботи системи, та принципи її функціонування.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Електронні медичні записи [Електронний ресурс]. Режим доступу: https://cabinet-health.mvs.gov.ua/.

2. Телемедицина [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Телемедицина.

3. Інтернет речей [Електронний ресурс]. Режим доступу: https://termin.in.ua/internet-rechey-iot/.

4. Медичні інформаційні системи [Електронний ресурс]. Режим доступу: https://emci.ua/.

5. Електронний рецепт [Електронний ресурс]. Режим доступу: https://bravo.moz.gov.ua/elektronnij-recept.

6. Академія НСЗУ [Електронний ресурс]. Режим доступу: https://academy.nszu.gov.ua/.

7. Epic Sys Corp. [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Epic_Systems.

8. Nextgen HelthCare [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/NextGen_Healthcare.

9. McKesson Corporation [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/McKesson_Corporation

10. Zocdoc [Електронний ресурс]. Режим доступу: https://www.zocdoc.com/.

11. HealthTap [Електронний ресурс]. Режим доступу: https://www.healthtap.com/.

12. Buoy Health [Електронний ресурс]. Режим доступу: https://www.buoyhealth.com/.

13. Your.MD [Електронний ресурс]. Режим доступу: https://faq.your.md/.

14. MongoDB [Електронний ресурс]. Режим доступу: https://www.mongodb.com/docs/drivers/node/current/fundamentals/connection/.

15. CMS [Електронний ресурс]. Режим доступу: https://justpro.com.ua/blog/shcho-take-cms-ta-ikh-vydy/.

16. Реєстрація [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Реєстрація.

17. Автентифікація [Електронний ресурс]. Режим доступу: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-authentication.

18. Авторизація [Електронний ресурс]. Режим доступу: https://introserv.com/ua/blog/u-chomu-rizniczya-mizh-autentifikaczieyu-ta-avtorizaczieyu/#авторизація.

19. Користувацький інтерфейс [Електронний ресурс]. Режим доступу: https://kr-labs.com.ua/blog/shho-take-frontend-i-backend/.

20. Introduction to C# [Електронний ресурс]. Режим доступу: https://www.w3schools.com/cs/cs_intro.php.

21. Introduction to C++ [Електронний ресурс]. Режим доступу: https://www.w3schools.com/cpp/cpp_intro.asp.

22. What is JavaScript [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript.

23. Visual Studio Code [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Visual_Studio_Code.

24. WebStorm [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/WebStorm.

# ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н.. Азаров О.Д..

“29” вересня 2023 р.

## ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

“Інформаційна web-система обслуговування та діагностування пацієнтів в медичному закладі”
08-54.МКР.042.00.000 ПЗ

Науковий керівник: проф, д.т.н. каф.ОТ

_____ Азаров О. Д.

Студент групи 2КІ-22м

_____ Степанов О. Д.

ВНТУ 2023

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність роботи полягає у інформаційному обслуговуванні пацієнтів та їх попередньому діагностуванні та сприяє підвищенню ефективності та швидкості діагностування, за рахунок адаптивного збору анамнезу пацієнта, отриманого із дерева рішень.

1.2 Наказ про затвердження теми МКР.

2 Мета МКР і призначення розробки

2.1 Мета роботи — удосконалення web-системи обслуговування та діагностування пацієнтів за рахунок проектування дерева рішень.

2.2 Призначення розробки — визначається необхідністю створення ефективної web-системи, що буде ефективно попередньо діагностувати та web-додатку, якому будуть записуватись до лікаря та попередньо діагностуватися.

3 Вихідні дані для виконання МКР

3.1 Проведення аналізу існуючих методів та принципів;

3.2 Розробка алгоритму системи та веб-застосунку;

3.4 Проведення верифікації та аналізу отриманих результатів;

3.5 Виконання розрахунків для доведення доцільності нової розробки з економічної точки зору.

4 Вимоги до виконання МКР

Головна вимога — що система має ефективно аналізувати анамнез з опитування за допомогою дерева рішень та видавати діагноз, для покращення попереднього діагностування та полегшення роботи лікаря.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи МКР

| № етапу | Назва етапу | Термін виконання | | Очікувані результати |
|---|---|---|---|---|
| | | початок | кінець | |
| 1 | Аналіз існуючих технологій, огляд аналогів системи. | 19.09.2023 | 28.09.2023 | Розділ 1 |
| 2 | Визначення архітектури web-системи та проектування дерева рішень | 5.10.2023 | 15.10.2023 | Розділ 2 |
| 3 | Розробка алгоритму та функціоналу веб-застосунку | 16.10.2023 | 23.10.2023 | Розділ 3 |
| 4 | Тестування системи | 23.10.2023 | 26.10.2023 | Розділ 4 |
| 4 | Підготовка економічної частини | 2.11.2023 | 12.11.2023 | Розділ 5 |
| 6 | Оформлення пояснювальної записки, графічного матеріалу і презентації | 4.12.2023 | 8.12.2023 | ПЗ, графічний матеріал і презентація |
| 7 | Підготовка і підпис супроводжуючих документів, нормоконтроль та тест на плагіат | 8.12.2023 | 11.12.2023 | Оформленні документи |

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі,

відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами.

## 7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

## 8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлюванні МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104–2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документи на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ–03.02.02 П.001.01:21

# ДОДАТОК Б

Блок-схема алгоритму web-додатку

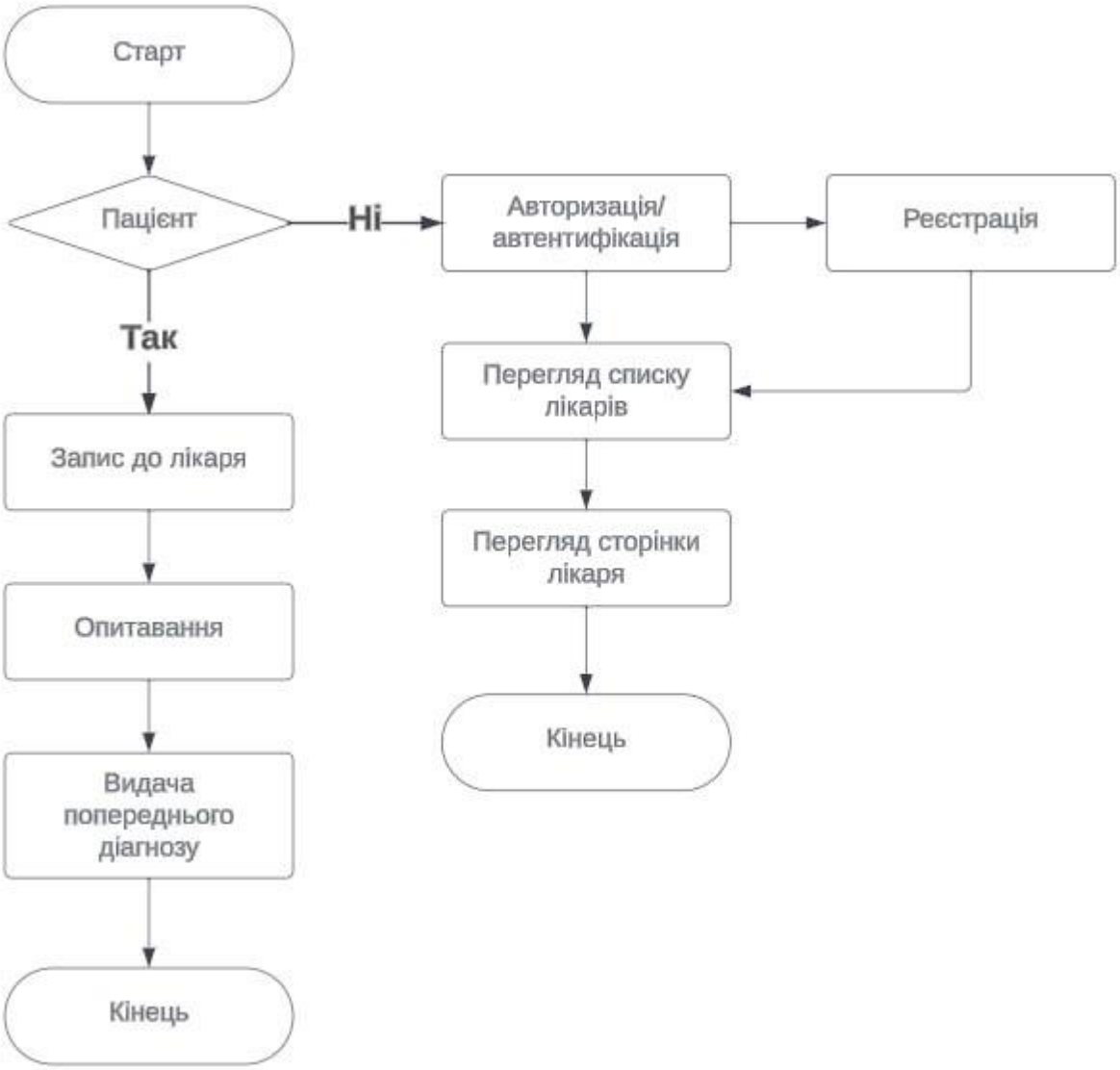



Рисунок Б.1 — Блок схема алгоритму

# ДОДАТОК В

Блок схема алгоритму роботи дерева рішень

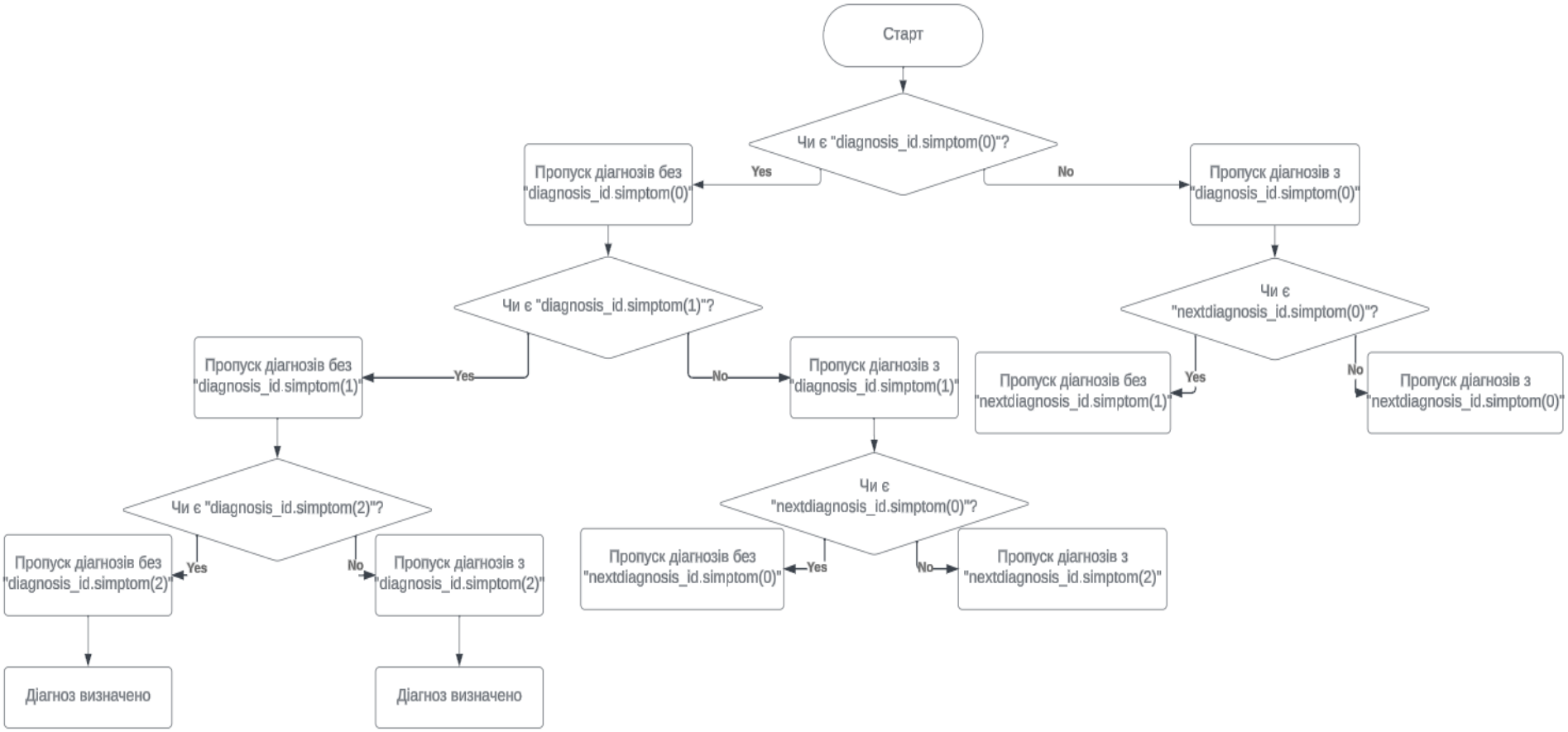



Рисунок В.1 — Блок-схема алгоритму

# ДОДАТОК Г

Лістинг для клієнтської частини

```
import React, { useState } from 'react';
import axios from 'axios';
import { useHistory } from 'react-router-dom';
const DoctorRegistrationPage = () => {
  const [formData, setFormData] = useState({
    firstName: '',
    lastName: '',
    phoneNumber: '',
    email: '',
    username: '',
    password: '',
    confirmPassword: '',
  });
  const history = useHistory();
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((prevData) => ({ ...prevData, [name]: value }));
  };
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('/api/doctors/register', formData);
      if (response.data.success) {
        history.push('/login');
      } else {
        console.error('Помилка при реєстрації лікаря:', response.data.message);
      }
```

```
    } catch (error) {
      console.error('Помилка при реєстрації лікаря:', error);
    }
  };
  return (
    <div>
      <h2>Реєстрація лікаря</h2>
      <form onSubmit={handleSubmit}>
        <label>
          Ім'я:
          <input type="text" name="firstName" value={formData.firstName}
onChange={handleChange} />
        </label>
        <label>
          Прізвище:
          <input type="text" name="lastName" value={formData.lastName}
onChange={handleChange} />
        </label>
        <label>
          Номер телефону:
          <input type="text" name="phoneNumber" value={formData.phoneNumber}
onChange={handleChange} />
        </label>
        <label>
          Пошта:
          <input type="text" name="email" value={formData.email}
onChange={handleChange} />
        </label>
        <label>
          Логін:
```

```
        <input type="text" name="username" value={formData.username}
onChange={handleChange} />
      </label>
      <label>
        Пароль:
        <input type="password" name="password" value={formData.password}
onChange={handleChange} />
      </label>
      <label>
        Підтвердження паролю:
        <input
          type="password"
          name="confirmPassword"
          value={formData.confirmPassword}
          onChange={handleChange}
        />
      </label>
      <button type="submit">Зареєструватися</button>
    </form>
  </div>
 );
};
export default DoctorRegistrationPage;

import React, { useState } from 'react';
import axios from 'axios';
const ReestraturaPage = () => {
 const [formData, setFormData] = useState({
  fullName: '',
  age: '',
```

```
    selectedDoctor: '',
    dateTime: '',
  });
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((prevData) => ({ ...prevData, [name]: value }));
  };
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('/api/reestratura', formData);
      console.log(response.data);
    } catch (error) {
      console.error('Помилка при записі до реєстратури:', error);
    }
  };
  return (
    <div>
      <h2>Запис до лікаря</h2>
      <form onSubmit={handleSubmit}>
        <label>
          ПІБ:
          <input type="text" name="fullName" value={formData.fullName}
onChange={handleChange} />
        </label>
        <label>
          Повних років:
          <input type="text" name="age" value={formData.age}
onChange={handleChange} />
        </label>
```

```
        <label>
          Лікар:
          <select name="selectedDoctor" value={formData.selectedDoctor}
onChange={handleChange}>
            <option value="doctor1"> Окуліст</option>
            <option value="doctor2"> Пульмонолог</option>
            <option value="doctor3"> Хірург</option>
            <option value="doctor4"> Терапевт</option>
            <option value="doctor5"> Дерматолог</option>
          </select>
        </label>
        <label>
          Дата та час прийому:
          <input type="text" name="dateTime" value={formData.dateTime}
onChange={handleChange} />
        </label>
        <button type="submit">Підтвердити запис</button>
      </form>
    </div>
  );
};
export default ReestraturaPage;
```

# ДОДАТОК Д

Лістинг для серверної частини

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const app = express();
mongoose.connect('mongodb://localhost:27017/hospital-app', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
const regdocSchema = new mongoose.Schema({
  firstName: String,
  lastName: String,
  phoneNumber: String,
  email: String,
  username: String,
  password: String,
});
const RegDoc = mongoose.model('RegDoc', regdocSchema);
app.use(bodyParser.json());
app.post('/api/doctors/register', async (req, res) => {
  try {
    const newDoctor = new RegDoc(req.body);
    await newDoctor.save();
    res.json({ success: true });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
});
const reestraturaSchema = new mongoose.Schema({
```

```
  fullName: String,
  age: String,
  selectedDoctor: String,
  dateTime: String,
});
const Reestratura = mongoose.model('Reestratura', reestraturaSchema);
app.use(bodyParser.json());
app.post('/api/reestratura', async (req, res) => {
  try {
    const newReestraturaEntry = new Reestratura(req.body);
    await newReestraturaEntry.save();
    res.json({ success: true });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
});
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

# ДОДАТОК Е

Лістинг дерева рішень

```
const makeDecisionTree = (diagnosesDatabase) => {
  const diagnosisKeys = Object.keys(diagnosesDatabase);
  if (diagnosisKeys.length === 0) {
    return null;
  }
  // Вибрати перший діагноз як кореневий вузол
  const rootDiagnosis = diagnosisKeys[0];
  const rootSymptoms = diagnosesDatabase[rootDiagnosis].symptoms;
  return {
    diagnosis: rootDiagnosis,
    symptoms: rootSymptoms,
    question: `Чи є у вас ${rootSymptoms[0]}?`,
    yes_node: makeSubtree(diagnosesDatabase, rootDiagnosis, rootSymptoms[0],
true),
    no_node: makeSubtree(diagnosesDatabase, rootDiagnosis, rootSymptoms[0],
false),
  };
};
const makeSubtree = (diagnosesDatabase, currentDiagnosis, currentSymptom,
isAnswerYes) => {
  const nextDiagnosis = getNextDiagnosis(diagnosesDatabase, currentDiagnosis,
currentSymptom, isAnswerYes);
  if (nextDiagnosis) {
    const nextSymptoms = diagnosesDatabase[nextDiagnosis].symptoms;
    return {
      diagnosis: nextDiagnosis,
      symptoms: nextSymptoms,
      question: `Чи є у вас ${nextSymptoms[0]}?`,
```

```
      yes_node: makeSubtree(diagnosesDatabase, nextDiagnosis, nextSymptoms[0],
true),
      no_node: makeSubtree(diagnosesDatabase, nextDiagnosis, nextSymptoms[0],
false),
    };
  }
  return null;
};
const getNextDiagnosis = (diagnosesDatabase, currentDiagnosis, currentSymptom,
isAnswerYes) => {
  const currentDiagnosisSymptoms =
diagnosesDatabase[currentDiagnosis].symptoms;
  const currentIndex = currentDiagnosisSymptoms.indexOf(currentSymptom);
  if (isAnswerYes) {
    if (currentIndex < currentDiagnosisSymptoms.length - 1) {
      return currentDiagnosis;
    }
  } else {
    const nextDiagnosisIndex =
Object.keys(diagnosesDatabase).indexOf(currentDiagnosis) + 1;
    if (nextDiagnosisIndex < Object.keys(diagnosesDatabase).length) {
      return Object.keys(diagnosesDatabase)[nextDiagnosisIndex];
    }
  }
  return null;
};
```

# ДОДАТОК Ж

Протокол перевірки навчальної (кваліфікаційної) роботи

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інформаційна web-система обслуговування та діагностування пацієнтів у медичному закладі з використанням дерева рішень

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

**Показники звіту подібності Unicheck**

Оригінальність 85,3% Схожість 14,7%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Степанов О. Д.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Азаров О. Д.
(підпис) (прізвище, ініціали)