Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ЛІКУВАННЯ

Виконав:
здобувач IV курсу
групи КН-41
спеціальності 122 «Комп'ютерні науки»
Киткайло Денис Андрійович
Науковий керівник:
к. т. н., доц. Сінчук А. М.

Рівне – 2024

Зміст

ВСТУП

Впровадження інформаційної системи для медичних закладів є важливим етапом у вдосконаленні управління та підвищенні ефективності надання медичних послуг. Така система може об'єднувати різні аспекти управління, зокрема, забезпечити зручний інструмент для взаємодії з пацієнтами. У сучасному світі, де конкуренція у сфері охорони здоров'я є вкрай високою, ефективне використання технологій стає ключовим чинником для досягнення успіху та задоволення вимог сучасного пацієнта. Однією з важливих стратегій у цьому контексті є розробка та впровадження інформаційної системи, спрямованої на оптимізацію усіх аспектів медичної діяльності.

Медичні заклади, незалежно від свого рівня та спеціалізації, стикаються з рядом викликів, таких як: високоякісне обслуговування пацієнтів, ефективне управління запасами медикаментів та впровадження сучасних маркетингових стратегій. У цьому контексті розробка інформаційної системи може виявитися ключовим інструментом, що допомагає вирішувати ці завдання та підвищувати конкурентоспроможність медичного закладу.

Метою даної роботи є дослідження технологій розробки інформаційної системи для оптимізації процесу лікування пацієнта у медичному закладі за допомогою сучасних інформаційних технологій.

Аналізуючи сучасний цифровий медичний простір, стає зрозумілим, як інноваційна стратегія може позитивно вплинути на різні аспекти функціонування медичного закладу: оптимізацію процесів обслуговування пацієнтів, раціональне використання фінансових ресурсів, тошо. Найбільш ефективною методологією проектування медичної інформаційної системи є об’єктно-орієнтований підхід для керування відповідною базою даних. Такий підхід забезпечить швидкий доступ до найбільш важливої інформації та дозволить зберігати великі обсяги даних для ретроспективного аналізу.

Об'єкт дослідження – технологічний процес розробки віртуальної платформи для діагностики хвороб людини.

Предметом дослідження є використання інноваційних технологій для розробки інформаційної системи у медичній сфері.

Завдання дослідження:

- використання глибокого навчання;
- аугментація даних;
- оптимізація моделі;
- валідація та оцінка результатів;
- технології передавання навчання (Transfer Learning);
- експерименти з архітектурою моделі.
- реалізація додатку для лікування з використанням розробленого алгоритму.
- тестування та оцінка ефективності розробленої інформаційної системи.

Апробація результатів дослідження.

Звітна науково-практична конференція професорсько-викладацького складу Рівненського державного гуманітарного університету, яка відбулася 16-17 травня 2024 року. Рівне.

Структура роботи. Дипломна робота складається зі вступу, трьох розділів, висновку та списку використаних джерел. Повний обсяг роботи складає 60 сторінки машинного тексту, в тому числі 40 рисунків.

РОЗДІЛ 1
ОСНОВИ МОДЕЛЮВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Різновид інформаційних систем

Інформаційні системи (рис 1.1) є невід'ємною частиною сучасних організацій, забезпечуючи автоматизацію процесів, збір, зберігання та аналіз даних. Це сприяє підвищенню ефективності управління та прийняття рішень. Інформаційна система - це комплексний механізм, який включає в себе комп'ютерне обладнання, програмне забезпечення, дані, процеси та людей, що співпрацюють для збору, зберігання, обробки, передачі та використання інформації з метою підтримки діяльностей та прийняття управлінських рішень [2]. Це не лише технічна система, але і соціальна, оскільки вона включає в себе взаємодію з користувачами.

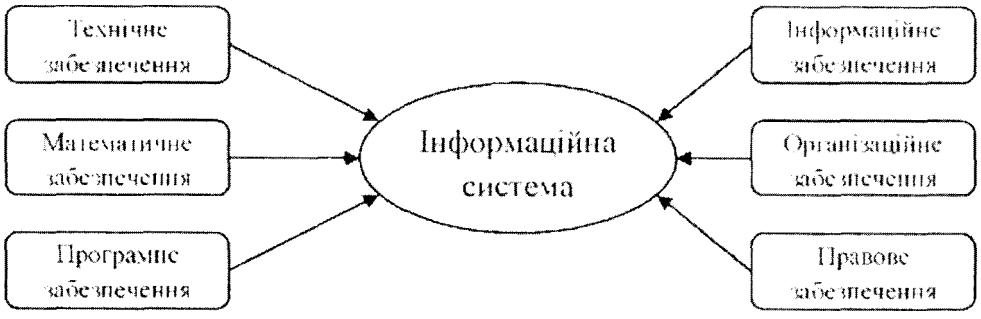

Рис 1.1. Інформаційна система

Інформаційні системи використовуються у різних галузях, таких як бізнес, освіта, медицина, наука та урядовість. Вони дозволяють ефективно управляти інформацією, підвищувати продуктивність та приймати обґрунтовані рішення.

Основні компоненти інформаційної системи включають комп'ютерне обладнання, яке забезпечує фізичну інфраструктуру системи, програмне забезпечення, що визначає функціональні можливості, дані, що є основним матеріалом для обробки, процеси, які визначають послідовність операцій обробки

інформації, і людей, які взаємодіють з системою для виконання завдань та прийняття рішень.

У медичних закладах ІС відіграють важливу роль у покращенні процесів лікування та моніторингу пацієнтів, забезпечуючи швидкий доступ до необхідної інформації та підтримку медичного персоналу в ухваленні обґрунтованих рішень.

1.2 Основні компоненти інформаційної системи та етапи її моделювання

Зважаючи на значення інформаційних систем у сучасному світі, розглянемо детальніше кожен з їх основних компонентів:

- Комп'ютерне обладнання: цей компонент охоплює усі фізичні пристрої, необхідні для роботи інформаційної системи [3]. Це включає в себе комп'ютери, сервери, мережеве обладнання, сховища даних, друкувальні пристрої та інші пристрої. Кожен з цих пристроїв має свої власні характеристики та функціональні можливості, які дозволяють їм виконувати свої завдання у складі інформаційної системи.

- Програмне забезпечення: цей компонент складається з різноманітних програм, які контролюють та керують роботою комп'ютерного обладнання. Основні складові програмного забезпечення включають операційні системи, які керують ресурсами комп'ютерів; системи управління базами даних, що забезпечують організацію та доступ до даних; програми для створення, редагування та обробки інформації; а також спеціалізовані програми для вирішення конкретних завдань або бізнес-процесів.

- Дані: це основний елемент будь-якої інформаційної системи. Дані представляють собою фактичну інформацію, яка обробляється та зберігається у системі. Вони можуть бути структурованими або неструктурованими, текстовими або числовими, графічними або аудіовізуальними. Керування та забезпечення безпеки даних є важливою складовою інформаційної системи.

- Процеси: цей компонент визначає послідовність операцій, які виконуються системою для обробки даних та виконання певних завдань. Процеси можуть

включати створення, зберігання, зчитування оновлення та видалення даних, а також взаємодію з користувачем та виконання автоматизованих бізнес-процесів.

- Користувачі: цей компонент включає всіх осіб, які взаємодіють з інформаційною системою. Це можуть бути кінцеві користувачі, які використовують програми та сервіси системи для виконання своїх завдань, а також адміністратори системи, які відповідають за налаштування, управління та підтримку системи.

Розуміння та правильне впровадження кожного з цих компонентів є важливим для успішного функціонування інформаційної системи і досягнення її цілей.

Моделювання інформаційних систем проходить через кілька ключових етапів (рис 1.3). Спочатку здійснюється аналіз вимог, де визначаються потреби користувачів та організаційні цілі [4]. Далі йде етап проектування, який включає розробку архітектури системи, створення структурних та поведінкових моделей [5].

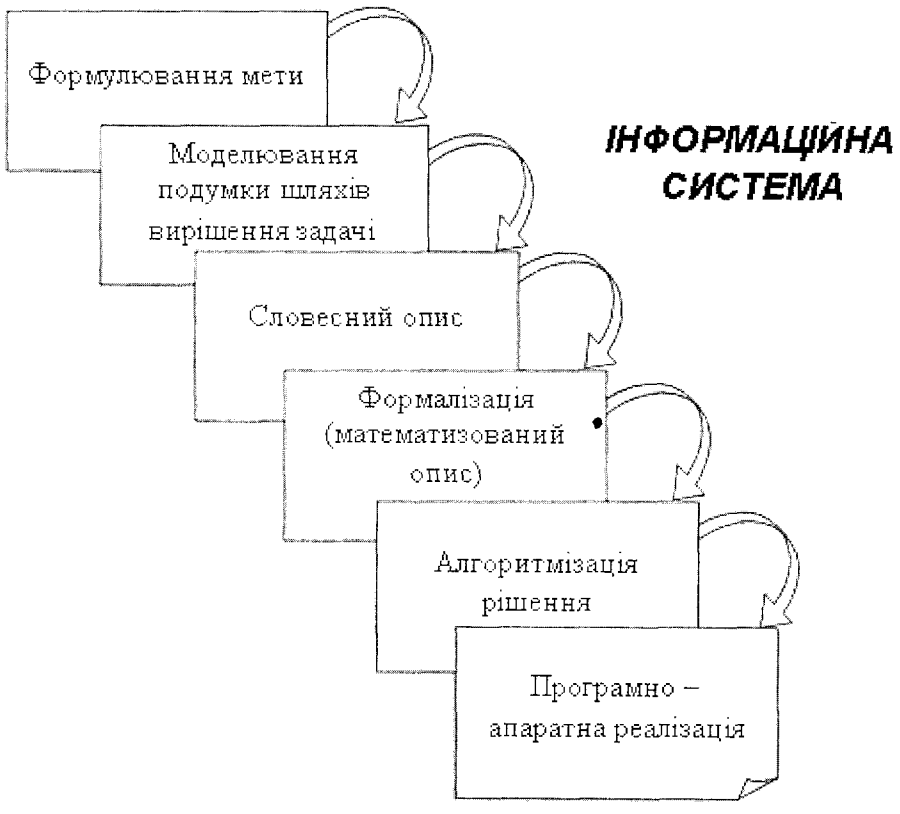

Рис 1.3. Етапи моделювання ІС

Потім настає етап розробки, під час якого пишеться код і налаштовується апаратне забезпечення. Наступним етапом є тестування, коли система перевіряється на наявність помилок та відповідність вимогам. Після цього система впроваджується в експлуатаційне середовище, і завершує цикл етап супроводу, що включає підтримку та оновлення системи в процесі її експлуатації. Моделювання інформаційних систем - це процес створення абстрактних представлень або моделей реальних інформаційних систем з метою розуміння їх функціонування, аналізу їх властивостей та вдосконалення.

Основні етапи моделювання інформаційних систем включають [6]:

1. Формулювання вимог: на цьому етапі визначаються потреби та очікування від інформаційної системи. Вимоги збираються від зацікавлених сторін - користувачів, клієнтів, керівництва та інших зацікавлених осіб.

2. Аналіз і проектування: після збору вимог проводиться аналіз, де вивчаються поточні процеси та системи, і розробляються концептуальні моделі майбутньої інформаційної системи. На цьому етапі визначаються функціональні вимоги та структура системи.

3. Розробка: на основі аналізу та проектування створюються конкретні компоненти інформаційної системи, такі як бази даних, програмне забезпечення, інтерфейси користувача тощо. Цей етап також включає в себе тестування розроблених компонентів для перевірки їхньої працездатності та відповідності вимогам.

4. Впровадження: після розробки і тестування інформаційна система впроваджується в дію. Це може включати перехід до нової системи або оновлення існуючої, навчання персоналу, підтримку користувачів та інші заходи для забезпечення успішного впровадження.

5. Експлуатація та підтримка: після впровадження інформаційна система використовується для підтримки бізнес-процесів або вирішення інших завдань. На цьому етапі здійснюється постійна підтримка системи, включаючи регулярне оновлення, виправлення помилок та вдосконалення.

6. Оцінка та оптимізація: цей етап включає в себе постійний аналіз та оцінку ефективності інформаційної системи з метою виявлення можливостей для покращення. На основі цього аналізу приймаються рішення щодо оптимізації та вдосконалення системи.

Ці етапи моделювання інформаційних систем допомагають забезпечити систематичний та структурований підхід до розробки та управління інформаційними системами, що дозволяє забезпечити їх ефективне функціонування та відповідність вимогам користувачів.

Розробка інформаційних систем базується на різних моделях та методологіях, які допомагають організувати процес розробки та забезпечують його ефективність та якість [8]. Ось кілька основних моделей та методологій:

- Каскадна модель (Waterfall): це класична модель розробки, в якій процес поділяється на послідовні етапи: аналіз, проектування, розробка, випробування та впровадження. Кожен етап починається тільки після завершення попереднього. Ця модель підходить для проектів з чітко визначеними вимогами та стабільними умовами.

- Прототипування (Prototyping): ця методологія передбачає створення прототипів програмного забезпечення для швидкого визначення та зрозуміння вимог користувачів. Прототипи дозволяють замовникам та розробникам активно спілкуватися та вдосконалювати систему під час її розробки.

- Ітеративний та інкрементальний розвиток (Iterative and Incremental Development): ця методологія передбачає розвиток системи через короткі ітерації або інкременти. Кожна ітерація включає аналіз, проектування, розробку та тестування. Після кожної ітерації додається новий функціонал, що дозволяє швидко реагувати на зміни в вимогах та забезпечує плавний процес розробки.

- Спрямованість на об'єкти (Object-Oriented): це підходить, який базується на концепції об'єктно-орієнтованого програмування. Розробка інформаційних систем організована навколо об'єктів, які мають властивості та методи. Цей підхід сприяє повторному використанню коду, полегшує розуміння та модифікацію системи.

- Scrum: це один з методів управління проектами, який базується на ітеративному та інкрементальному підході. Кожен ітераційний цикл, або спринт, зазвичай триває від одного до чотирьох тижнів і завершується готовим продуктом чи частиною функціоналу.

Ці моделі та методології можуть бути використані окремо або комбінуватися залежно від конкретного проекту та його потреб [7]. Обираючи найбільш підходящий метод, команда розробників може забезпечити успішне завершення проекту та відповідність його вимогам.

1.3 Важливість стандартизації та сумісності моделей інформаційних систем

Для успішного впровадження та функціонування інформаційних систем важливо дотримуватися стандартів та забезпечувати сумісність між різними системами та компонентами.

Це включає використання загальноприйнятих протоколів, форматів даних та інтерфейсів. Дотримання стандартів дозволяє інтегрувати різні системи в єдину мережу, забезпечуючи їх довготривалу підтримку та масштабованість, а також зменшуючи витрати на виправлення або налагодження окремих модулів системи [9]. Стандартизація та сумісність в інформаційних системах мають величезне значення з кількох причин:

- Сприяння взаємодії інформаційних систем: стандартизація дозволяє різним системам і компонентам взаємодіяти між собою без проблем. Це особливо важливо у світі, де існує багато різних систем, які повинні спілкуватися між собою для обміну даними та виконання різних функцій.

- Підвищення інтероперабельності: стандартизація дозволяє різним системам працювати разом без проблем та забезпечує інтероперабельність між ними. Це дозволяє легко і ефективно обмінюватися даними та інформацією між різними системами та пристроями.

- Зменшення витрат на розробку та підтримку: стандартизація дозволяє розробникам використовувати загальноприйняті стандарти та специфікації для

розробки нових систем. Це зменшує витрати на розробку, тестування та підтримку систем, оскільки розробники можуть використовувати вже існуючі стандартизовані рішення.

- Підвищення безпеки та надійност : стандартизація може сприяти вдосконаленню заходів безпеки та забезпечити більшу надійність систем. Стандартизовані процедури та протоколи можуть допомогти уникнути помилок та забезпечити безпечний обмін даними.

- Сприяння інноваціям та розвитку: стандартизація може стимулювати інновації та розвиток технологій. Завдяки стандартизації різні розробники можуть працювати над різними аспектами системи, використовуючи загальні стандарти та специфікації, що сприяє виникненню нових ідей та технологій.

Отже, стандартизація та сумісність в інформаційних системах є ключовими аспектами, які сприяють ефективному функціонуванню та розвитку цих систем у сучасному світі.

1.4 Застосування інформаційних систем у медицині

Інформаційні системи в медицині відіграють критичну роль у покращенні якості надання медичних послуг, управління медичною інформацією та підвищенні ефективності медичного процесу [12]. Ось деякі з головних застосувань:

- Електронні медичні записи (EMP): системи управління електронними медичними записами дозволяють зберігати та обробляти дані про пацієнтів в електронному форматі. Це сприяє полегшенню доступу до медичної інформації, забезпеченню її конфіденційності та цілісності, а також покращує координацію медичного обслуговування.

- Системи управління лікарськими запасами: ці системи дозволяють вести облік та контроль за рухом лікарських засобів та медичних матеріалів у лікарнях та медичних установах. Вони допомагають уникнути дефіциту лікарських засобів, зменшують витрати та покращують управління запасами.

- Телемедицина: це використання інформаційних технологій для надання медичних консультацій та діагностики на віддаленій основі. Телемедицина дозволяє пацієнтам отримувати медичну допомогу там, де вони знаходяться, зменшує час та витрати на подорожі до лікарів, а також сприяє покращенню доступності медичної допомоги віддаленим та важкодоступним регіонам.

- Системи підтримки прийняття рішень: ці системи надають медичним працівникам інформаційну підтримку при процесі прийняття рішень, наприклад, щодо діагностики та лікування. Вони аналізують клінічні дані та рекомендують найбільш ефективні та безпечні процедури та ліки для конкретних пацієнтів.

- Медичні інформаційні системи для досліджень: ці системи допомагають організувати та вести медичні дослідження, збирати та аналізувати дані, спрощують процес планування та проведення клінічних випробувань, а також підвищують ефективність науково-дослідницької роботи.

Ці застосування інформаційних систем в медицині допомагають покращити доступність, якість та ефективність медичної допомоги, сприяють підвищенню рівня безпеки та задоволеності пацієнтів, а також сприяють розвитку медичної науки та практики [13].

Таким чином, інформаційні системи значно покращують якість медичних послуг, підвищують ефективність лікування та забезпечують безпеку пацієнтів. Вони сприяють більш ефективному управлінню медичними закладами, оптимізуючи всі аспекти їх діяльності, від управління пацієнтами до логістики медичних ресурсів та підтримки медичного персоналу в ухваленні рішень.

РОЗДІЛ 2
СИСТЕМНИЙ АНАЛІЗ ДІАГНОСТИКИ ХВОРОБ

Системний аналіз та розробка алгоритмів діагностики захворювань є критичним етапом у впровадженні інформаційних систем у сфері медицини. Цей процес охоплює глибоке вивчення методів діагностики, відбір оптимальних з них та створення алгоритмів, які допомагатимуть лікарям зробити обґрунтовані висновки.

Під час аналізу процесу діагностики враховуються різноманітність хвороб, їх симптоми, клінічні та лабораторні дані, які можуть бути використані для визначення діагнозу [14]. Також розглядається сучасне становище технологій, таких як медичні пристрої, програмне забезпечення та методи аналізу даних, які можуть допомогти у точній діагностиці.

Після аналізу визначаються головні критерії ефективності алгоритмів діагностики, такі як точність результатів, швидкість виконання та можливість застосування в реальному часі. На цій основі розробляються алгоритми, які враховують усі важливі аспекти та можуть бути успішно використані в медичній практиці.

Ключовим етапом є тестування та валідація розроблених алгоритмів на клінічних випадках. Це дозволяє перевірити їхню ефективність та точність у реальних умовах і виявити можливі недоліки перед фінальним впровадженням.

Отже, системний аналіз та розробка алгоритмів діагностики є складним та многобічним процесом, який потребує детального дослідження та експертної оцінки для створення ефективних інструментів медичної діагностики.

2.1. Основні принципи діагностики хвороб

Аналіз процесу діагностики хвороб є ключовим етапом у впровадженні інформаційних систем у медичній сфері [15]. Цей процес передбачає глибоке

дослідження та ретельний аналіз різноманітних аспектів діагностичного процесу, що включає в себе:

- Симптоми та клінічні дані: перший крок полягає у зборі та аналізі симптомів, які можуть свідчити про наявність певного захворювання. Лікарі досліджують характеристики скарг, фізичні ознаки та інші клінічні дані, які можуть бути використані для формулювання попереднього діагнозу.

- Медичні тести та обстеження: для підтвердження попереднього діагнозу часто потрібні спеціальні медичні тести та обстеження. Це може включати лабораторні аналізи, зображення (наприклад, рентгенівські або УЗД), а також інші процедури, що допомагають отримати додаткову інформацію про стан пацієнта.

- Медична історія пацієнта: важливо враховувати медичну історію пацієнта, включаючи раніше діагностовані захворювання, прийняті ліки, алергії та інші фактори, які можуть впливати на процес діагностики та лікування.

- Диференціальна діагностика: порівняння різних можливих діагнозів та виключення менш ймовірних захворювань на основі отриманих клінічних та лабораторних даних.

- Експертна оцінка: після збору та аналізу всієї необхідної інформації лікарі проводять експертну оцінку та формулюють кінцевий діагноз.

- Персоналізований підхід: Важливим аспектом є урахування індивідуальних особливостей кожного пацієнта та призначення оптимального плану діагностики, враховуючи його вік, стать, стан здоров'я та інші фактори.

- Стандартизація процесу: для забезпечення якості та надійності діагностики важливо використовувати стандартизовані методи та протоколи, що допомагають уніфікувати процес діагностики та знижувати ймовірність помилок.

Аналіз процесу діагностики хвороб дозволяє виявити основні проблеми та ризики, що можуть виникнути під час діагностики, та спрямувати зусилля на їх вирішення. Також він створює основу для подальшої розробки ефективних алгоритмів та інформаційних систем, які полегшують роботу медичного персоналу та покращують якість надання медичних послуг.

2.2. Розробка алгоритму діагностики хвороб

Автоматизована діагностика відкриває широкі можливості для підвищення якості медичного обслуговування та забезпечення більш ефективної та точної діагностики хвороб. Однією з ключових переваг є можливість алгоритмів машинного навчання та штучного інтелекту обробляти великі обсяги медичних даних та швидко видача результатів без зайвого запізнення. Це допомагає медичному персоналу оперативно реагувати на стан пацієнтів та вчасно призначати лікування.

Крім того, автоматизовані системи діагностики забезпечують високу точність та об'єктивність результатів, оскільки вони використовують об'єктивні критерії, що дозволяє уникнути людських помилок та забезпечити найкращі результати. Такі системи можуть виявляти симптоми хвороб на ранніх стадіях, коли їх важко виявити за допомогою традиційних методів діагностики, що сприяє початку лікування на етапі, коли хвороба ще не має виражених симптомів, збільшуючи шанси на успішне вилікування [11].

Додатково, автоматизовані системи діагностики допомагають зекономити час медичного персоналу. Частина діагностичного процесу виконується автоматично, що дозволяє лікарям зосередитися на інших аспектах лікування та догляду за пацієнтами. Це сприяє ефективнішому використанню робочого часу та забезпечує більш якісне обслуговування.

Інтеграція автоматизованих систем діагностики з електронними медичними записами є також важливою. Це дозволяє миттєво отримувати доступ до медичних даних пацієнтів та зберігати та обробляти їх з використанням передових технологій безпеки та конфіденційності. Розробка алгоритму діагностики хвороб є складним та важливим процесом, що передбачає створення системи, яка здатна аналізувати клінічні дані пацієнтів та надавати рекомендації щодо діагнозу та лікування [16]. Цей процес включає в себе кілька ключових етапів:

1. Збір та обробка даних: перший крок - це збір та обробка великого обсягу клінічних даних, таких як медичні записи, результати обстежень, лабораторні дані тощо. Ці дані можуть бути представлені у різних форматах та джерелах, тому їх потрібно структурувати та нормалізувати для подальшого аналізу.

2. Вибір моделей та методів аналізу даних: наступним етапом є вибір відповідних моделей машинного навчання та методів аналізу даних. Це може включати в себе класифікаційні алгоритми, нейронні мережі, методи кластеризації тощо, залежно від конкретних вимог та характеру даних.

3. Навчання моделі: після вибору моделей та методів проводиться навчання моделі на наявних даних. Цей процес полягає у встановленні відповідностей між вхідними даними та вихідними результатами, щоб модель могла навчитися розпізнавати паттерни та робити прогнози.

4. Тестування та валідація: після навчання моделі проводиться тестування на незалежних від навчального набора даних для перевірки її ефективності та точності. Також важливо валідувати модель на різних підмножинах даних, щоб переконатися в її стабільності та універсальності.

5. Оптимізація та підтримка: після успішного валідації моделі вона може бути оптимізована для підвищення її продуктивності або точності. Крім того, модель потребує постійного моніторингу та підтримки для виявлення та виправлення можливих проблем чи відхилень у роботі.

Отже, розробка алгоритму діагностики хвороб - це комплексний процес, що вимагає великої уваги до деталей, експертності у сфері медицини та обробки даних, а також використання передових технологій машинного навчання та штучного інтелекту.

2.3 Засоби побудови моделі інформаційної системи лікування

Програмування алгоритмів для інформаційної системи лікування - це складний і важливий процес, що передбачає розробку програмного коду, який

відповідає потребам медичної практики та забезпечує ефективну діагностику, лікування та моніторинг стану пацієнтів [17].

Розробка алгоритму діагностики хвороб вимагає чіткого розуміння кожного етапу процесу та відповідних інструментів, які можуть бути використані. Ось більш детальний опис із включенням конкретних інструментів:

1. Збір та підготовка даних

Етап збору та підготовки даних є критично важливим для створення якісного алгоритму діагностики хвороб. Він включає кілька підетапів, кожен з яких виконується з використанням певних інструментів.

Збір даних

- **Збір даних з медичних баз даних**: Отримання структурованих даних з електронних медичних записів (EMR), систем управління лікарнею (HMS) та інших джерел.
- **Збір даних з відкритих джерел**: Використання відкритих баз даних, таких як Kaggle, UCI Machine Learning Repository, або медичних досліджень.
- **Веб-скрапінг**: Збір даних з веб-сайтів, які містять медичну інформацію, якщо це дозволено умовами використання.

Інструменти:

- **SQL**: Для отримання даних з баз даних. Наприклад, використання запитів SQL для вибірки потрібних записів.
- **Python**: Для написання скриптів збору даних. Python має багато бібліотек, що спрощують цей процес.
- **Scrapy**, **BeautifulSoup**: Бібліотеки для веб-скрапінгу, які дозволяють автоматизувати процес збору даних з веб-сторінок.

Імпорт даних

- **Завантаження даних у середовище розробки**: Імпорт даних (рис2.1) у Python або інше середовище для подальшої обробки.

Інструменти:

• **Pandas**: Бібліотека Python для роботи з табличними даними. Вона дозволяє легко завантажувати дані з різних форматів (CSV, Excel, SQL) та проводити їхню первинну обробку.

```
import pandas as pd

# Приклад імпорту даних з CSV файлу
data = pd.read_csv('medical_data.csv')
```

Рис 2.1 Імпорт даних

Очистка даних

• **Виявлення та обробка пропущених значень**: Заповнення пропущених даних або видалення записів з пропущеними значеннями.

• **Видалення або корекція аномалій**: Виявлення та корекція помилкових або екстремальних значень, які можуть спотворити результати моделі.

Інструменти:

• **Pandas**: Для очистки даних (рис 2.2), виявлення та обробки пропущених значень.

```
# Виявлення пропущених значень
missing_values = data.isnull().sum()

# Заповнення пропущених значень середнім значенням колонки
data.fillna(data.mean(), inplace=True)

# Видалення записів з пропущеними значеннями
data.dropna(inplace=True)
```

Рис 2.2 Очистка даних

Нормалізація даних

• **Приведення числових даних до однакового масштабу**: Нормалізація або стандартизація числових ознак (рис 2.3), щоб уникнути домінування деяких ознак над іншими.

Інструменти:

• **Scikit-learn**: Для нормалізації даних.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
```

Рис 2.3 Нормалізація даних

Кодування категоріальних змінних

• **Перетворення текстових даних у числові формати**: Використання one-hot кодування або label encoding для перетворення категоріальних змінних у числовий формат (рис 2.4).

Інструменти:

• **Pandas**: Для кодування категоріальних змінних.

```
# One-hot кодування категоріальної змінної
data = pd.get_dummies(data, columns=['categorical_column'])

# Label encoding категоріальної змінної
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
data['categorical_column'] = label_encoder.fit_transform(data['categorical_colu
```

Рис 2.4 Кодування категоріальних змінних

На цьому етапі забезпечується збирання, очистка, нормалізація та кодування даних, що дозволяє підготувати їх до подальшого аналізу та навчання моделі. Використання відповідних інструментів допомагає автоматизувати процеси та підвищити ефективність роботи з даними.

Попередня обробка даних

Попередня обробка даних включає кілька ключових етапів, кожен з яких спрямований на підготовку даних для подальшого аналізу та моделювання. Цей

етап критично важливий для забезпечення високої якості даних, що вплине на точність та ефективність моделі.

Очищення даних

• **Виявлення та обробка пропущених значень**: Пропущені дані можуть негативно вплинути на модель, тому їх необхідно обробити.

• **Видалення дублікатів**: Видалення повторюваних записів для уникнення упередженості в даних.

• **Видалення або корекція аномалій**: Виявлення та обробка аномальних значень, які можуть бути помилками або крайніми значеннями.

Інструменти:

• **Pandas**: Для очищення даних (рис 2.5) , виявлення та обробки пропущених значень, видалення дублікатів таномалій.

```
import pandas as pd

# Завантаження даних
data = pd.read_csv('medical_data.csv')

# Виявлення пропущених значень
missing_values = data.isnull().sum()

# Заповнення пропущених значень середнім значенням колонки
data.fillna(data.mean(), inplace=True)

# Видалення записів з пропущеними значеннями
data.dropna(inplace=True)

# Видалення дублікатів
data.drop_duplicates(inplace=True)

# Виявлення аномальних значень
for column in data.select_dtypes(include=[np.number]).columns:
    data = data[(data[column] >= data[column].quantile(0.01)) & (data[column] <=
```

Рис 2.5 Очищення даних

Нормалізація даних

• **Приведення числових даних до однакового масштабу**: Нормалізація або стандартизація числових ознак для забезпечення їх порівнянності (рис 2.6).

Інструменти:

• **Scikit-learn**: Для нормалізації даних.

```
from sklearn.preprocessing import StandardScaler

# Вибір числових колонок
numerical_columns = data.select_dtypes(include=[np.number]).columns

# Нормалізація числових даних
scaler = StandardScaler()
data[numerical_columns] = scaler.fit_transform(data[numerical_columns])
```

Рис 2.6 Нормалізація даних

Кодування категоріальних змінних

• **Перетворення текстових даних у числові формати**: Використання one-hot кодування або label encoding для перетворення категоріальних змінних у числовий формат.

Інструменти:

• **Pandas**: Для кодування категоріальних змінних.

• **Scikit-learn**: Для кодування категоріальних змінних.

```
from sklearn.preprocessing import LabelEncoder

# Label encoding
label_encoder = LabelEncoder()
data['categorical_column'] = label_encoder.fit_transform(data['categorical_colum

# One-hot кодування категоріальних змінних
data = pd.get_dummies(data, columns=['categorical_column'])
```

Рис 2.7 Кодування категоріальних змінних

Попередня обробка даних є критичним етапом у розробці алгоритму діагностики хвороб. Вона включає очищення даних, нормалізацію числових ознак, кодування категоріальних змінних та обробку текстових даних. Використання відповідних інструментів, таких як Pandas, Scikit-learn, NLTK та SpaCy, допомагає автоматизувати процеси та забезпечити високу якість даних для подальшого аналізу та моделювання.

Вибір ознак (фічей)

Вибір ознак є одним із найважливіших етапів у розробці алгоритму діагностики хвороб, оскільки від нього залежить точність і ефективність моделі. На цьому етапі визначаються найбільш інформативні ознаки, які будуть використані для побудови моделі.

Аналіз кореляції

• **Виявлення взаємозв'язків між ознаками та цільовою змінною**: Оцінка кореляції (рис 2.8) між незалежними змінними (ознаками) та залежною змінною (цільовою ознакою) для визначення релевантних ознак.

Інструменти:

• **Pandas**: Для обчислення кореляцій.

• **Seaborn**, **Matplotlib**: Для візуалізації кореляційних матриць.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Завантаження даних
data = pd.read_csv('medical_data.csv')

# Обчислення кореляційної матриці
corr_matrix = data.corr()

# Візуалізація кореляційної матриці
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```

Рис 2.8 Аналіз

Відбір ознак

• **Використання методів відбору ознак**: Використання статистичних методів (рис 2.9) для вибору найбільш інформативних ознак.

Інструменти:

• **Scikit-learn**: Для реалізації методів відбору ознак, таких як SelectKBest, Recursive Feature Elimination (RFE) та інші.

```
from sklearn.feature_selection import SelectKBest, f_classif

# Відбір найбільш інформативних ознак
X = data.drop('target', axis=1)  # Незалежні змінні
y = data['target']  # Цільова змінна

# Використання SelectKBest для вибору топ-K ознак
kbest = SelectKBest(score_func=f_classif, k=10)
X_new = kbest.fit_transform(X, y)

# Отримання обраних ознак
selected_features = X.columns[kbest.get_support()]
print("Обрані ознаки:", selected_features)
```

Рис 2.9 Методи відбору

Інженерія ознак

• **Створення нових ознак на основі існуючих даних**: Інженерія ознак включає створення нових ознак (рис 2.10), які можуть покращити продуктивність моделі. Це можуть бути взаємодії між ознаками, поліноміальні ознаки або інші трансформації.

Інструменти:

• **Pandas**: Для створення нових ознак та трансформацій.

```
# Приклад створення нової ознаки: взаємодія між двома ознаками
data['new_feature'] = data['feature1'] * data['feature2']

# Приклад створення поліноміальних ознак
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, include_bias=False)
poly_features = poly.fit_transform(data[['feature1', 'feature2']])

# Додавання поліноміальних ознак до початкового DataFrame
poly_features_df = pd.DataFrame(poly_features, columns=poly.get_feature_names
data = pd.concat([data, poly_features_df], axis=1)
```

Рис 2.10 Створеня нових ознак

На етапі вибору ознак використовуються різні методи та інструменти для визначення найбільш інформативних ознак, які будуть використані для побудови моделі. Використання таких інструментів, як Pandas, Seaborn, Matplotlib, Scikit-learn, дозволяє ефективно аналізувати та вибирати ознаки, що покращує продуктивність моделі.

Розробка моделі

Розробка моделі є ключовим етапом у створенні алгоритму діагностики хвороб. На цьому етапі вибирається відповідний алгоритм машинного навчання, створюються та оптимізуються моделі для досягнення найкращих результатів.

Вибір алгоритму

• **Аналіз проблеми**: Визначення типу завдання машинного навчання (класифікація, регресія, кластеризація і т.д.).

• **Вибір алгоритму**: Вибір відповідного алгоритму (рис 2.11) на основі типу завдання та властивостей даних.

Інструменти:

• **Scikit-learn**: Широкий вибір класичних алгоритмів машинного навчання.

• **TensorFlow**, **Keras**, **PyTorch**: Для створення та тренування нейронних мереж.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

# Розділення даних на тренувальний і тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Вибір кількох базових моделей
models = {
    "Random Forest": RandomForestClassifier(),
    "Logistic Regression": LogisticRegression(),
    "SVM": SVC()
}
```

Рис 2.11 Створення алгоритму

Створення моделі

• **Ініціалізація моделі**: Створення об'єкта моделі (рис 2.12).

• **Навчання моделі**: Навчання моделі на тренувальних даних.

Інструменти:

• **Scikit-learn**: Для класичних моделей машинного навчання.

• **TensorFlow**, **Keras**, **PyTorch**: Для глибокого навчання.

```
# Навчання моделей
for name, model in models.items():
    model.fit(X_train, y_train)
    print(f"{name} model trained.")
```

Рис 2.12 Створеня моделі

Налаштування гіперпараметрів

• **Підбір гіперпараметрів**: Використання методів налаштування гіперпараметрів (рис 2.13) для оптимізації продуктивності моделі.

Інструменти:

• **GridSearchCV**, **RandomizedSearchCV** з **Scikit-learn**: Для налаштування гіперпараметрі

```
from sklearn.model_selection import GridSearchCV

# Приклад налаштування гіперпараметрів для Random Forest
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30]
}

grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

print("Найкращі гіперпараметри:", grid_search.best_params_)
```

Рис 2.13 Налаштування гіперпараметрів

Розробка моделі включає вибір відповідного алгоритму, створення та навчання моделі, налаштування гіперпараметрів та використання ансамблевих методів для покращення продуктивності. Використання таких інструментів, як Scikit-learn, TensorFlow, Keras та PyTorch, дозволяє ефективно розробляти та оптимізувати моделі машинного навчання для діагностики хвороб.

Валідація та тестування моделі

Валідація та тестування моделі є критично важливими етапами, що дозволяють оцінити продуктивність моделі та її здатність до узагальнення на нових даних. Ці етапи включають використання різних методів для оцінки моделі та забезпечення її надійності.

Розділення даних на тренувальний та тестовий набори

• **Розділення даних**: Визначення тренувального та тестового наборів (рис 2.14) для забезпечення об'єктивної оцінки моделі.

Інструменти:

• **Scikit-learn**: Для розділення даних.

```
from sklearn.model_selection import train_test_split

# Розділення даних
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

Рис 2.14 Розділення даних

Перехресна валідація

• **Виконання перехресної валідації**: Оцінка моделі на різних підмножинах даних для перевірки її стабільності та узагальненості (рис 2.15).

Інструменти:

• **Scikit-learn**: Для реалізації перехресної валідації.

```
from sklearn.model_selection import cross_val_score

# Виконання перехресної валідації
model = RandomForestClassifier()
cv_scores = cross_val_score(model, X_train, y_train, cv=5)

print("Перехресна валідація: середній бал =", cv_scores.mean())
```

Рис 2.15 Перехресна валідація

Оцінка моделі на тестовому наборі

• **Оцінка продуктивності на тестовому наборі**: Обчислення метрик для оцінки якості моделі на нових даних (рис 2.16).

Інструменти:

• **Scikit-learn**: Для обчислення метрик продуктивності.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Навчання моделі на тренувальних даних
model.fit(X_train, y_train)

# Прогнозування на тестових даних
y_pred = model.predict(X_test)

# Обчислення метрик
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Точність: {accuracy}")
print(f"Прецизійність: {precision}")
print(f"Повнота: {recall}")
print(f"F1-бал: {f1}")
```

Рис 2.16 Оцінка моделі

Валідація та тестування моделі є важливими етапами, що забезпечують оцінку продуктивності моделі та її здатності до узагальнення на нових даних. Використання таких інструментів, як Scikit-learn та Matplotlib, дозволяє ефективно оцінювати моделі за допомогою різних метрик і методів, таких як перехресна валідація, і тестування на незалежних наборах даних. Це забезпечує впевненість у тому, що модель буде працювати надійно в реальних умовах.

РОЗДІЛ 3
РОЗРОБКА ПРОДУКТУ ТА ТЕСТУВАННЯ

3.1 Інтеграція програмного забезпечення

Інтеграція програмного забезпечення програми моделювання інформаційної системи лікування є критичним етапом, що забезпечує узгоджену роботу всіх компонентів системи для ефективного управління медичними процесами. Цей процес включає кілька основних аспектів:

Визначення потреб та вимог:

Інтеграція починається з аналізу потреб медичного закладу та вимог до інформаційної системи. Це включає вивчення існуючих робочих процесів, визначення необхідних функцій, а також обговорення вимог з медичним персоналом для розуміння їхніх очікувань та вимог.

Розробка архітектури системи:

Після визначення вимог розробляється архітектура системи (рис 3.1), яка описує, як різні компоненти програмного забезпечення будуть взаємодіяти між собою. Це включає бази даних, інтерфейси користувача, сервери обробки даних та зовнішні інтеграції з іншими медичними системами.

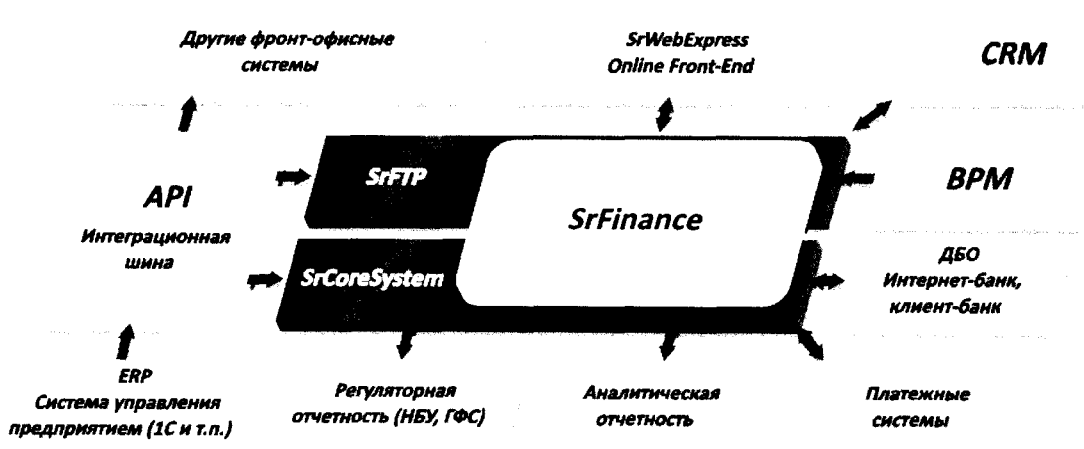

Рис 3.1 Приклад архітектури системи

Вибір технологій та інструментів:

Для успішної інтеграції необхідно вибрати відповідні технології та інструменти, які забезпечать надійну роботу системи. Це можуть бути платформи для управління базами даних, фреймворки для розробки веб-додатків, інструменти для безпеки даних та інші технології, що відповідають вимогам системи.

Вибір технологій та інструментів для програми моделювання інформаційної системи лікування є важливим етапом, що визначає успіх проекту. Оскільки система повинна бути надійною, масштабованою, безпечною та зручною у використанні, слід враховувати кілька ключових аспектів при виборі технологій:

Бази даних:

- Реляційні бази даних (наприклад, PostgreSQL, MySQL) підходять для зберігання структурованих медичних даних, таких як електронні медичні записи, результати лабораторних тестів та історія хвороби пацієнтів.

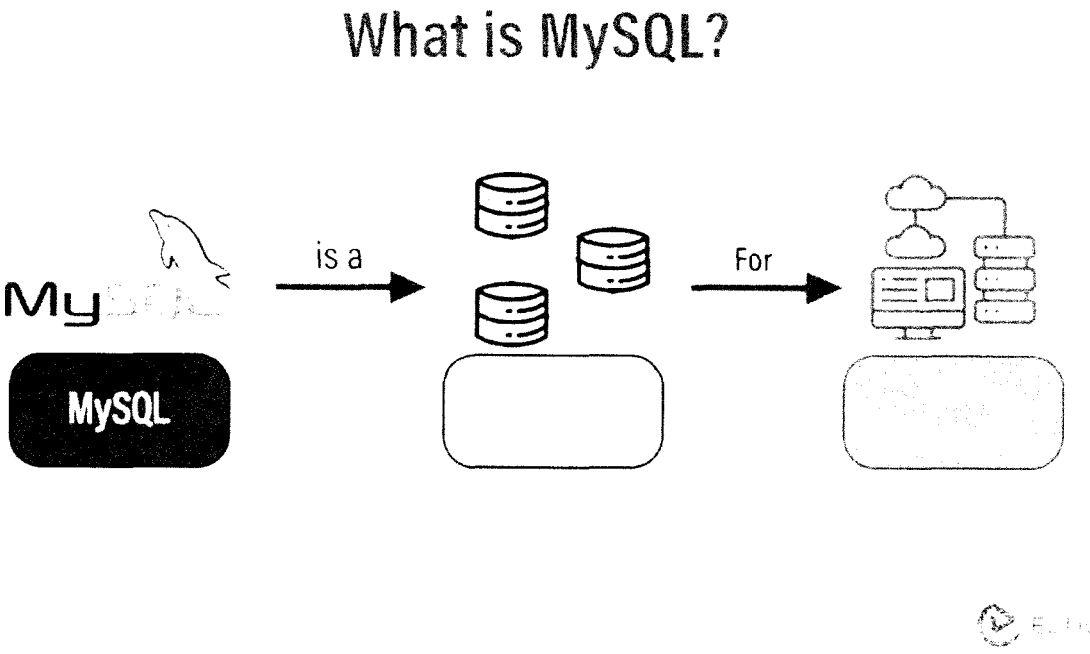

Рис 3.2 Приклад функціоналу MySQL

MySQL (рис 3.2) це потужна, широко використовувана система керування реляційними базами даних (RDBMS), яка базується на SQL (Structured Query Language). Вона була розроблена для ефективного управління великими обсягами даних та забезпечення високої продуктивності, надійності й масштабованості.

MySQL використовується для зберігання та управління даними в численних веб-додатках, таких як веб-сайти, онлайн-магазини, соціальні мережі та інші. Вона відома своєю швидкістю та простотою використання, що робить її популярною серед розробників і адміністраторів баз даних.

Основні характеристики MySQL включають:

- Висока продуктивність: MySQL оптимізована для швидкого виконання запитів, забезпечуючи ефективне обслуговування великої кількості одночасних користувачів.

- Масштабованість: Вона підтримує роботу з великими обсягами даних, дозволяючи легко масштабувати систему відповідно до зростаючих потреб бізнесу.

- Надійність і відмовостійкість: MySQL забезпечує високу надійність завдяки підтримці резервного копіювання, відновлення після збоїв, транзакцій з використанням ACID (атомарність, узгодженість, ізольованість, стійкість) та реплікації даних.

- Безпека: Вона включає механізми автентифікації користувачів, шифрування даних та управління правами доступу, що допомагає захищати дані від несанкціонованого доступу.

- Підтримка транзакцій: MySQL підтримує транзакції, що дозволяє забезпечити цілісність даних та виконання операцій у рамках транзакційного контролю.

- Підтримка кількох сховищ: MySQL дозволяє вибирати між різними типами сховищ даних, такими як InnoDB (для транзакційних баз даних) та MyISAM (для швидких запитів без транзакційної підтримки).

- Гнучкість і розширюваність: Вона дозволяє використовувати розширення і плагіни, що дозволяє додавати нові функціональні можливості відповідно до вимог проекту.

MySQL є відкритим програмним забезпеченням з відкритим вихідним кодом, що означає, що вона безкоштовна для використання та має активну спільноту розробників, які постійно працюють над її поліпшенням. Водночас, MySQL має

комерційні версії з додатковими функціями та підтримкою від Oracle Corporation, яка зараз володіє цією системою У підсумку, MySQL є потужним інструментом для управління реляційними базами даних, який пропонує високу продуктивність, надійність, безпеку і гнучкість, що робить її ідеальним вибором для багатьох видів додатків і сервісів.

- NoSQL(рис 3.3) бази даних можуть бути корисними для зберігання великих обсягів неструктурованих даних, таких як медичні зображення, аудіозаписи консультацій та інші мультимедійні дані.NoSQL (Not Only SQL) - це категорія баз даних, які відрізняються від традиційних реляційних баз даних. Вони призначені для зберігання та обробки великих обсягів даних, які можуть бути нереляційними, такими як документи, графи, ключ-значення або колонки. Вони забезпечують гнучкість, масштабованість та високу продуктивність, але можуть мати обмежену підтримку транзакцій та складність запитів порівняно з реляційними базами даних .

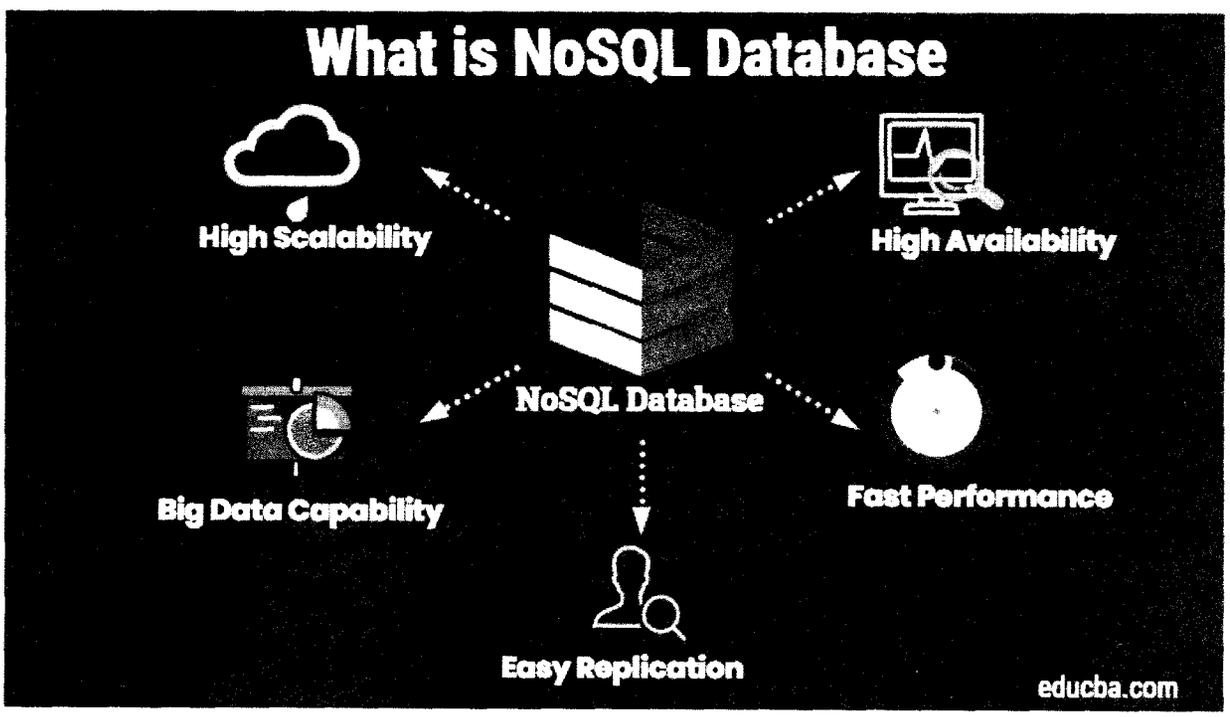

Рис 3.3 Приклад функціоналу NoSQL

Серверна частина:

- Node.js: Забезпечує високу продуктивність та масштабованість завдяки асинхронній обробці запитів. Підходить для розробки реальних додатків та API.

Node.js (рис 3.4) - це вільна та відкрита платформа, побудована на JavaScript runtime для створення швидких та масштабованих веб-застосунків. Вона дозволяє виконувати JavaScript на стороні сервера, що раніше було характерно для браузерного середовища. Node.js заснована на подійному зворотному виклику (event-driven), неблокуючому вводі/виводі (non-blocking I/O), що робить її ідеальним вибором для розробки веб-застосунків, які вимагають великої кількості одночасних з'єднань.

Вона має широкий екосистему модулів, яка дозволяє розробникам швидко розширювати функціональність застосунків. Node.js також широко використовується для розробки мережевих програм і інструментів командного рядка.

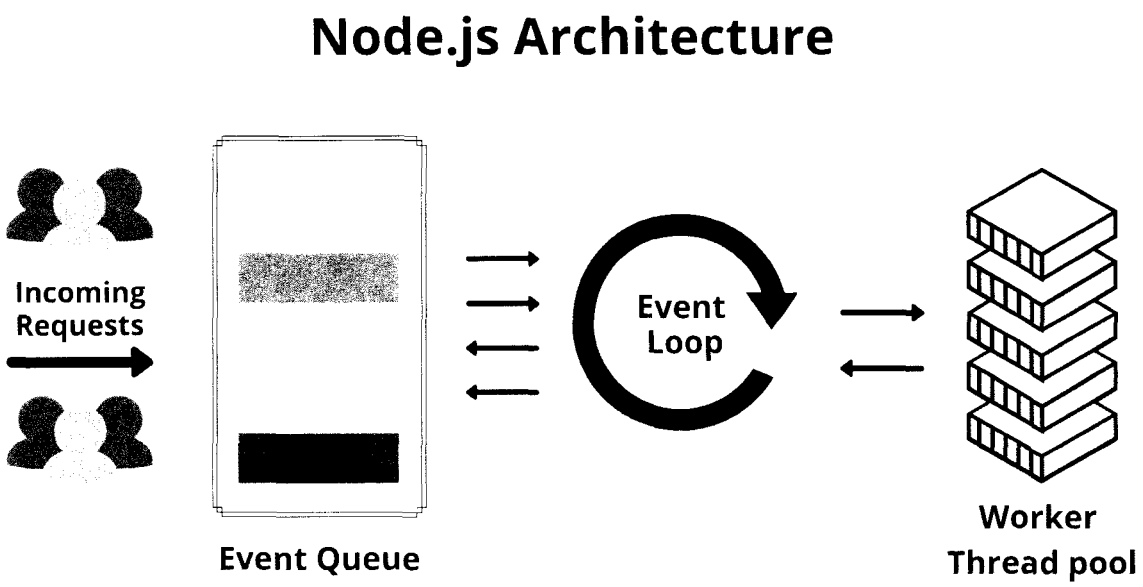

Рис 3.4 Архітектура Node JS

- JJava Spring Boot - це фреймворк для розробки Java-додатків. Він надає зручний спосіб створення мікросервісів і веб-додатків на основі принципів

інверсії управління та контейнеризації [10]. Spring Boot дозволяє швидко створювати додатки, використовуючи готові модулі та автоматизовані налаштування. Він забезпечує широкі можливості для роботи з базами даних, обробки HTTP-запитів, забезпечує підтримку безпеки і багато іншого. Все це робить Spring Boot популярним вибором для створення різноманітних веб-застосунків і мікросервісів.

Клієнтська частина:

- React.js: Відомий своєю швидкістю та можливістю створення інтерактивних інтерфейсів користувача. Підходить для динамічних веб-додатків. React.js - це бібліотека JavaScript для розробки інтерфейсів користувача. Вона дозволяє створювати веб-додатки з високим ступенем інтерактивності та динамічного змісту.

Однією з ключових особливостей React є використання компонентного підходу до побудови UI, що дозволяє розділити веб-сторінку на невеликі, незалежні компоненти, які легко управляти. React також забезпечує високу швидкодію завдяки використанню віртуального DOM та механізму ефективного оновлення сторінки.

Вона широко використовується для розробки веб-додатків, включаючи односторінкові додатки, соціальні мережі, панелі адміністрування та багато іншого.

Основні особливості React (рис 3.5) включають у себе:

1. Компонентна модель: React базується на ідеї розділення інтерфейсу користувача на невеликі, незалежні компоненти, які можуть бути повторно використані. Це сприяє структуруванню коду та полегшує підтримку додатків.

2. Віртуальний DOM: React використовує віртуальний DOM для оптимізації швидкодії відображення змін на сторінці. Це дозволяє React оновлювати тільки ті частини сторінки, які змінилися, замість перерисовки всього документа.

3. Односторінкові додатки: React підтримує створення односторінкових додатків (SPA), що дозволяє створювати динамічні веб-додатки без перезавантаження сторінки.

4. JSX: React використовує JSX (розширений синтаксис JavaScript), що дозволяє вбудовувати HTML-подібні структури прямо в код JavaScript. Це полегшує роботу зі структурою документа та забезпечує більшу читабельність коду.

5. Велика спільнота: React має велику та активну спільноту розробників, що активно підтримується компанією Facebook. Це забезпечує доступ до багатьох корисних розширень, бібліотек та інструментів для розробки.

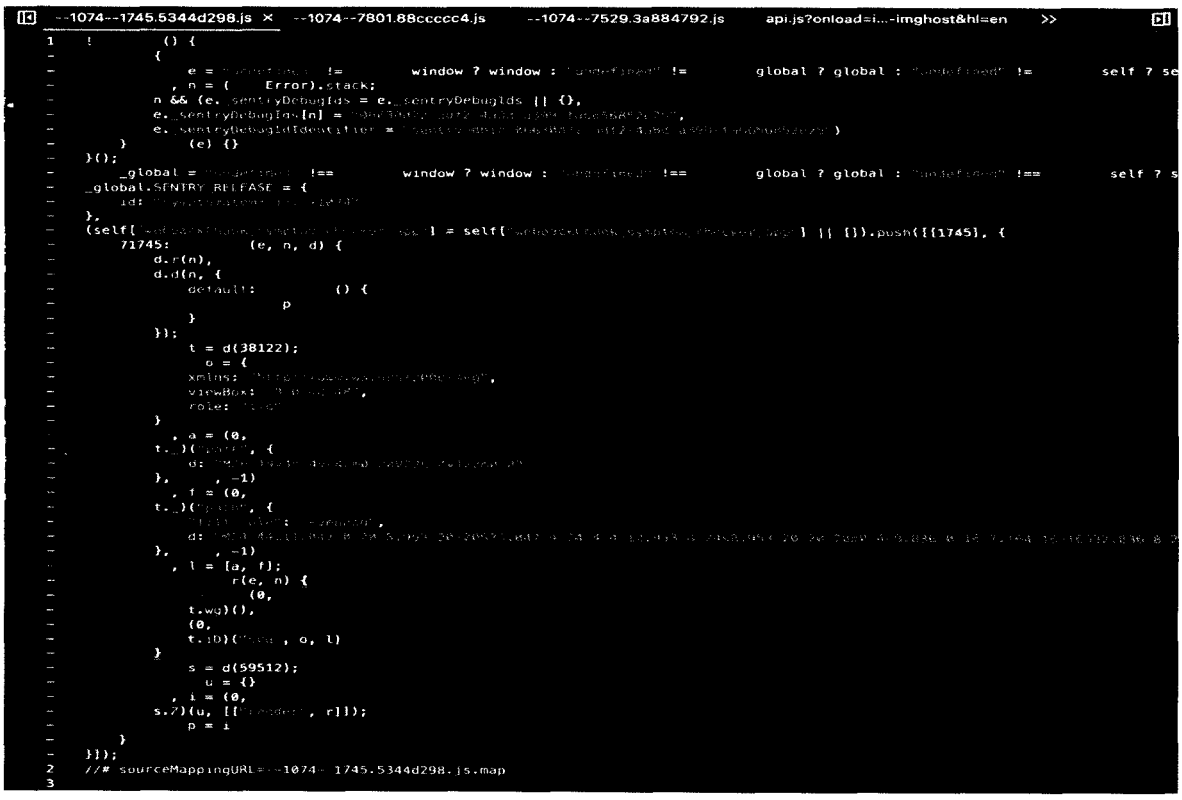

Рис 3.5 Використання React

- Angular (рис 3.6) Забезпечує потужну архітектуру для великих і складних додатків, зручний для організації структури проекту. Angular - це фреймворк для розробки веб-додатків, який створено в компанії Google.

Цей фреймворк використовується для створення клієнтських додатків у веб-середовищі. В основі Angular лежить TypeScript - мова програмування з використанням сучасних стандартів JavaScript.

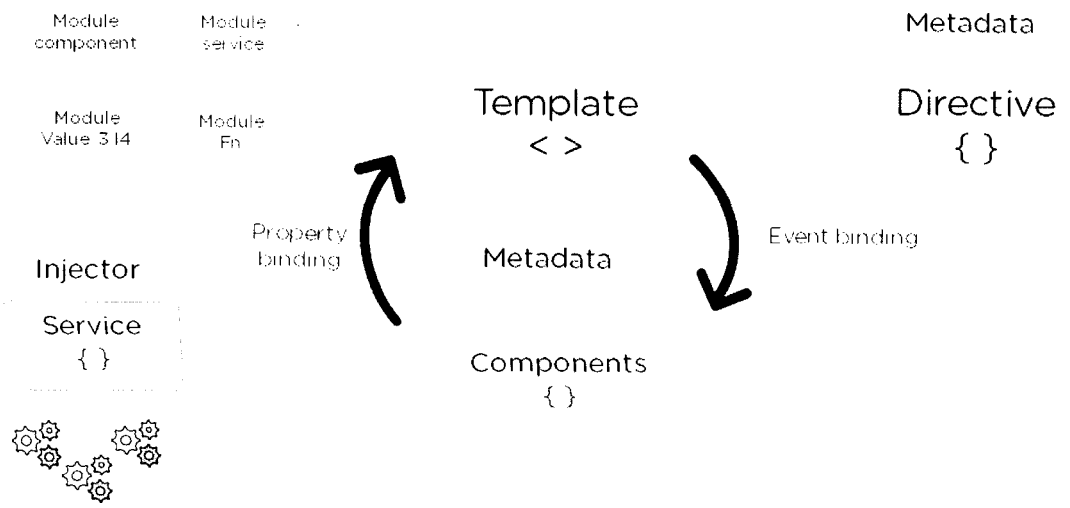

Рис 3.6 Архітектура Angular

Основні характеристики Angular включають у себе:

1. Компонентну архітектуру: Angular розділяє інтерфейс користувача на невеликі, незалежні компоненти, що спрощує розробку та підтримку веб-додатків.

2. Модульність: Додатки в Angular організовані за допомогою модулів, які дозволяють структурувати код та забезпечувати розділення функціональності.

3. Маршрутизація: Angular надає зручні інструменти для створення односторінкових додатків з допомогою маршрутизації.

4. Двостороннє зв'язування даних: Angular забезпечує механізм двостороннього зв'язування даних між моделлю та представленням, що дозволяє автоматично оновлювати дані на сторінці при їх зміні.

5. Служби та залежності: Angular надає механізми для роботи зі службами та залежностями, які дозволяють організовувати загальні функції та логіку додатка.

Angular широко використовується для розробки різноманітних веб-додатків, включаючи корпоративні системи, соціальні мережі, адміністративні панелі та інші. Він має велику спільноту розробників і активно розвивається командою Angular.

Безпека:

- JWT (рис 3.7) - Використовуються для управління автентифікацією та авторизацією користувачів, забезпечують захищений доступ до медичних даних. JWT (JSON Web Token) - це компактний, самостійний та безпечний спосіб представлення претендентства для обміну даними між сторонами. Він використовується для автентифікації та авторизації користувачів в веб-додатках.

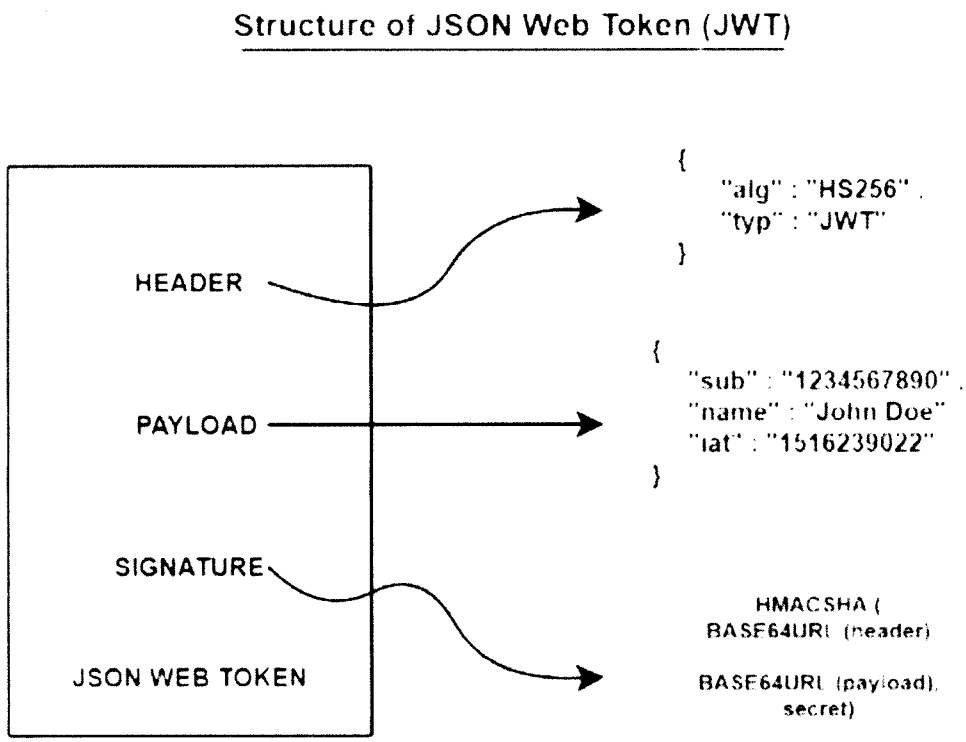

Рис 3.7 Структура JWT

Основні особливості JWT:

- Компактність: JWT складається з трьох частин: заголовка, тіла та підпису. Він займає мінімальний обсяг даних та може передаватися в заголовку запиту або в тілі відповіді.

- Самостійність: JWT містить всю необхідну інформацію для перевірки претендента, включаючи інформацію про термін дії токену та ролі користувача

- Безпека: JWT підписується за допомогою секретного ключа, що забезпечує його цілісність та невідмінність. Це унеможливлює зміну токену сторонніми особами.

- Розширюваність: JWT підтримує розширення через додаткові поля в тілі токену, що робить його гнучким для різних потреб автентифікації та авторизації.

JWT (рис 3.9) широко використовується для забезпечення безпеки веб-додатків та API, а також для створення синхронізації між різними системами. Він дозволяє створювати безпечні та ефективні механізми автентифікації та авторизації без необхідності зберігання сесійного стану на сервері.

- SSL/TLS: Забезпечують шифрування даних при передачі через мережу, захищаючи від несанкціонованого доступу. SSL (Secure Sockets Layer) і TLS (Transport Layer Security) є протоколами шифрування, які забезпечують безпеку зв'язку в Інтернеті. Вони використовуються для захисту конфіденційності, цілісності та автентифікації даних, які передаються між клієнтом і сервером.

Рис 3.8 SL/TLS

Основні характеристики SSL/TLS:

- Шифрування даних: SSL/TLS (рис 3.8) використовують симетричне та асиметричне шифрування для захисту даних в транзиті. Це дозволяє уникнути перехоплення та читання інформації третіми особами.

- Аутентифікація: SSL/TLS дозволяє сторонам перевіряти свою взаємну автентичність, щоб підтвердити, що вони спілкуються з правильним сервером або клієнтом.

- Захист від перехоплення ізміни даних: SSL/TLS забезпечує цілісність даних, запобігаючи їхній модифікації під час передачі.

- Сумісність з різними протоколами: SSL/TLS може використовуватися з різними протоколами вищого рівня, такими як HTTP, SMTP, FTP тощо, для захисту комунікацій.

SSL/TLS використовується для захисту різних видів веб-сайтів, електронної пошти, онлайн-платежів, мобільних додатків та інших онлайн-сервісів. Він є ключовим компонентом інфраструктури безпеки Інтернету, забезпечуючи захист користувачів і їхніх даних в онлайн-середовищі.

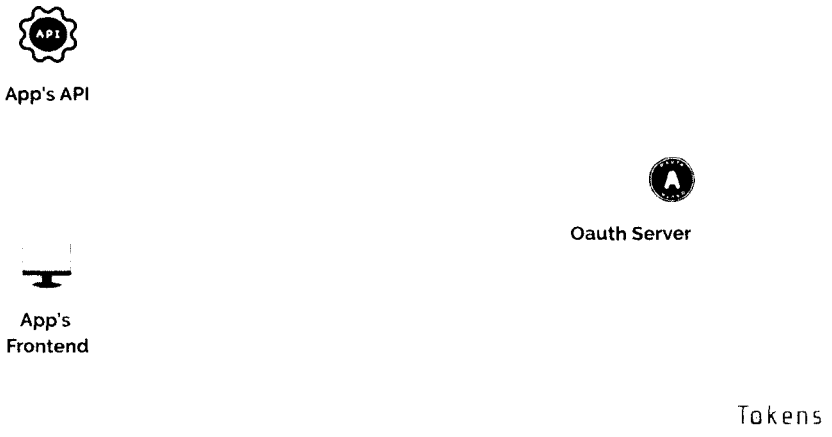

Рис 3.9 Приклад роботи JWT

Інтеграція:

- RESTful API: Підходить для інтеграції з іншими системами, забезпечує легкий доступ до функцій системи через HTTP-запити [17]. RESTful API (Representational State Transfer) - це архітектурний стиль для веб-сервісів, що дозволяє взаємодіяти з клієнтами через стандартні HTTP-методи (GET, POST, PUT, DELETE). Він базується на принципах роботи з ресурсами, кожен з яких має унікальний ідентифікатор URI. RESTful API забезпечує передачу даних у форматі,

зрозумілому клієнту, частіше за все це JSON або XML. Важливою особливістю є безстандартність, тобто відсутність збереження стану між запитами.

Це дозволяє створювати легкі, масштабовані та підтримувані веб-сервіси, що використовуються різними типами клієнтів. REST є стандартом для розробки веб-додатків, що базуються на взаємодії між клієнтами та серверами. Основні принципи REST включають:

1. Клієнт-серверна архітектура: Взаємодія між клієнтом і сервером розділена, що дозволяє їм розвиватися незалежно один від одного.

2. Без стану (stateless): Кожен запит містить всю необхідну інформацію для сервера для зрозуміння запиту. Сервер не зберігає жодної інформації про стан клієнта між запитами.

3. Кешування: Клієнти можуть зберігати копії відповідей сервера для майбутніх запитів, що дозволяє зменшити навантаження на сервер та покращити швидкість відповідей.

4. Єдинообразність інтерфейсу (uniform interface): Всі ресурси в системі доступні через єдиний інтерфейс, що спрощує розробку та розуміння системи.

5. Шарова система (layered system): Система може бути розгорнута на різних рівнях, з кожним рівнем мають взаємодіяти лише ті, що перебувають на сусідніх рівнях.

6. Код на запит (code on demand) (не є обов'язковим): Сервер може передавати клієнту виконуваний код для покращення функціональності клієнтського додатка.

REST API (рис 3.10) дотримується цих принципів, використовуючи HTTP протокол для комунікації між клієнтом та сервером. Він використовує HTTP методи, такі як GET, POST, PUT та DELETE, для виконання операцій над ресурсами на сервері. Такий підхід дозволяє створювати ефективні, масштабовані та легко зрозумілі веб-служби.

Рис 3.10 API

- GraphQL (рис 3.11) Забезпечує гнучкий доступ до даних, дозволяючи клієнтам запитувати саме ті дані, які їм потрібні. GraphQL - це мова запитів для вашого API та інструмент для виконання цих запитів з сервера на клієнт. Вона дозволяє клієнтам запитувати тільки ту інформацію, яка їм потрібна, та отримувати всі дані в одному запиті.

Однією з основних переваг GraphQL є те, що вона дозволяє описати структуру даних, яку клієнт хоче отримати, що зменшує кількість запитів до сервера та сприяє ефективнішій роботі додатків. Також GraphQL дає можливість реалізувати взаємодію з даними навіть для складних схем та дозволяє гнучко налаштовувати запити.

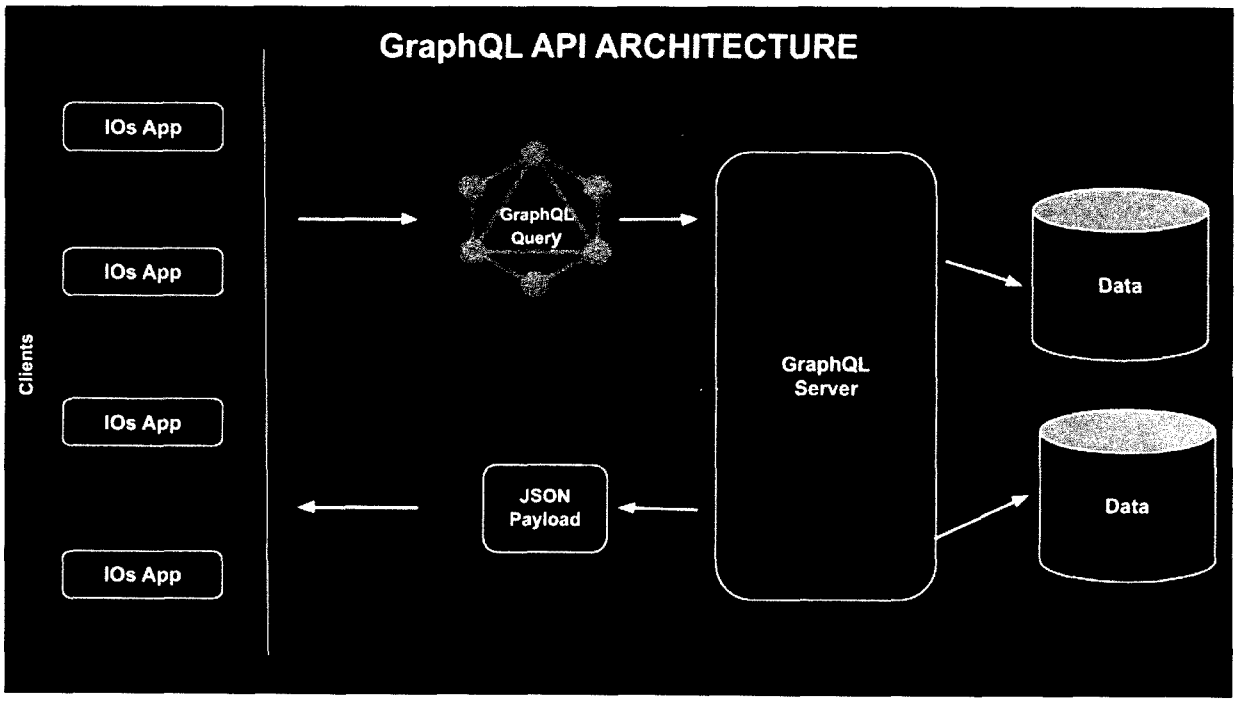

Рис 3.11 Архітектура GraphQL

Контейнеризація та оркестрація:

- Docker (рис 3.12) Забезпечує контейнеризацію додатків, що дозволяє легко розгортати та масштабувати систему. Docker - це платформа для розробки, доставки та запуску застосунків в контейнерах. Вона дозволяє упаковувати програмне забезпечення та всі його залежності у стандартизовані контейнери, які можна запускати в будь-якому середовищі, де встановлено Docker.

Завдяки цьому контейнеризація дозволяє розробникам легко переносити застосунки між різними середовищами, забезпечуючи консистентність робочого процесу від розробки до впровадження. Docker також дозволяє автоматизувати процеси розгортання та керування інфраструктурою, що спрощує роботу розробників та системних адміністраторів.

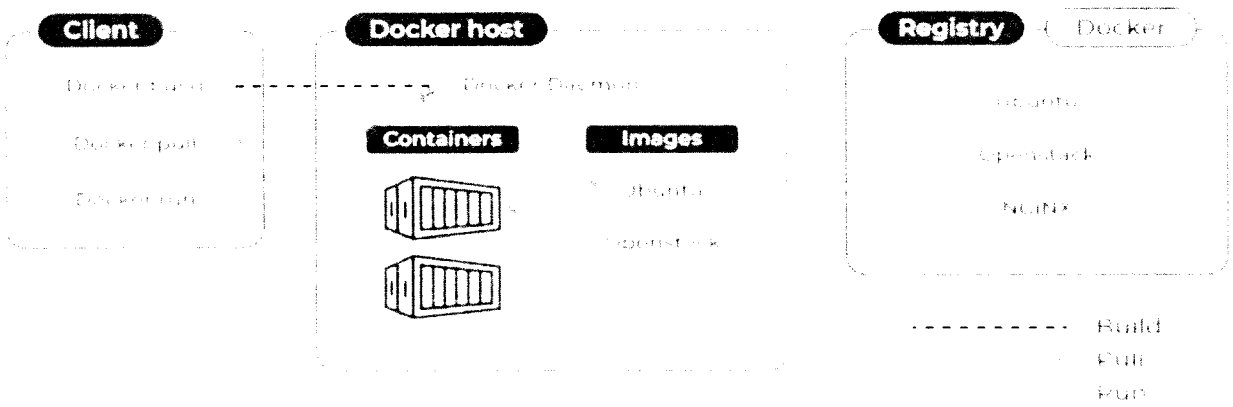

Рис 3.12 Архітектура Docker

- Kubernetes (рис 3.13) Використовується для оркестрації контейнерів, забезпечуючи автоматизоване управління, масштабування та підтримку додатків. Kubernetes - це відкрите програмне забезпечення для автоматизації розгортання, масштабування та управління контейнеризованими додатками. Воно дозволяє розробникам легко керувати розгортанням та управляти контейнерами у будь-якому середовищі, будь то власний центр обробки даних, хмарний сервіс або гібридне середовище. Kubernetes надає ряд можливостей, таких як оркестрація контейнерів, автоматизація розгортання, розподіл навантаження, масштабування за потребою та самовідновлення в разі відмови. Він дозволяє ефективно керувати ресурсами, забезпечуючи високу доступність та надійність додатків.

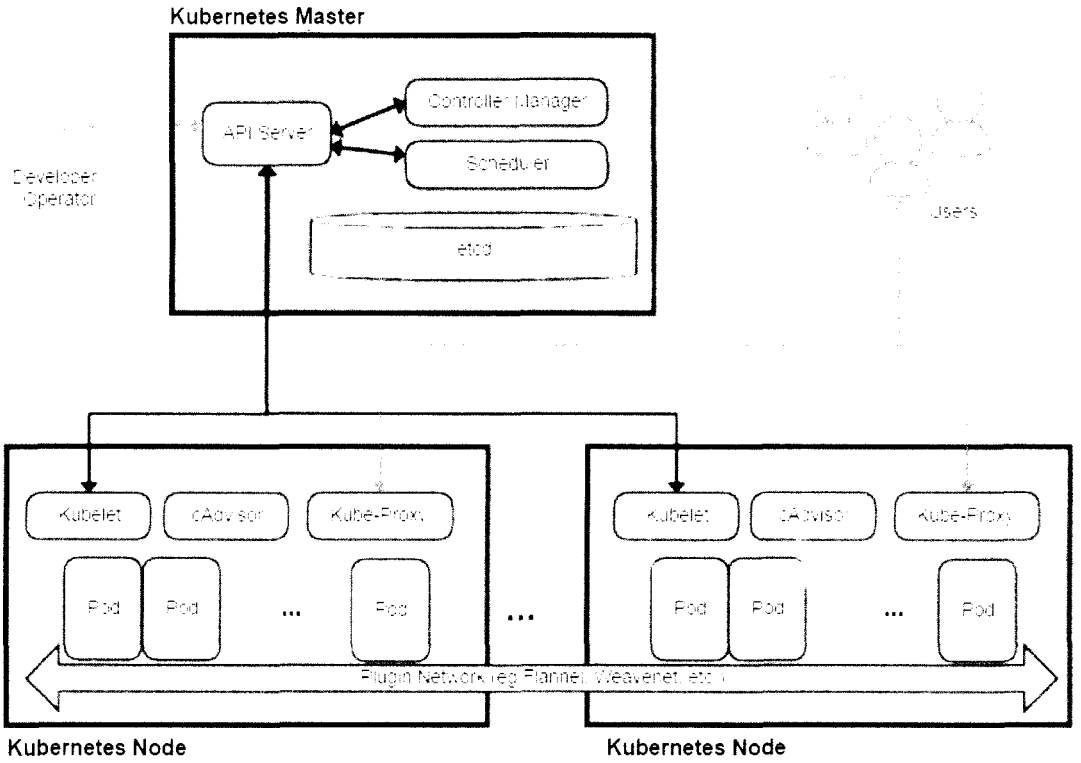

Рис 3.13 Архітектура Kubernetes

Kubernetes також підтримує розгортання мікросервісної архітектури, що дозволяє розробникам будувати складні додатки з набору невеликих, незалежних компонентів. Це сприяє швидкому розгортанню, масштабуванню та оновленню додатків, що відповідає потребам сучасних розробників.

Хмарні сервіси:

Microsoft Azure (рис 3.14) : Пропонують широкий спектр сервісів для зберігання даних, обробки та аналітики, забезпечують високу доступність та надійність. Microsoft Azure – це потужна хмарна платформа, створена компанією Microsoft, яка надає широкий спектр сервісів для обчислень, зберігання даних, аналітики та мережевих функцій [13]. Вона дозволяє компаніям створювати, розгортати та керувати додатками через глобальну мережу дата-центрів Microsoft.

Однією з основних функцій Azure є можливість створення та запуску віртуальних машин з різними операційними системами, такими як Windows і Linux, що забезпечує гнучкість і можливість масштабування. Крім того,

платформа пропонує служби для створення та розгортання веб-додатків і API, що забезпечують високу доступність і автоматичне масштабування.

Azure також надає можливість оркестрації контейнерів за допомогою Kubernetes, спрощуючи розгортання та управління контейнеризованими додатками. Безсерверні обчислювальні служби, такі як Azure Functions, дозволяють виконувати невеликі частини коду у відповідь на події, автоматично масштабуючись під навантаження.

Зберігання даних в Azure представлене масштабованими сховищами об'єктів для зберігання великих обсягів неструктурованих даних, такими як документи і відео. Також платформа пропонує керовані реляційні бази даних для високої продуктивності та глобально розподілені бази даних, що забезпечують масштабування без втрати продуктивності.

Azure включає аналітичні служби, що дозволяють об'єднувати великі обсяги даних для швидкого і складного аналізу. Середовище для обробки великих даних інтегрує Apache Spark з іншими сервісами Azure для аналізу та машинного навчання, а масштабоване сховище для великих даних дозволяє зберігати дані в їхньому вихідному форматі.

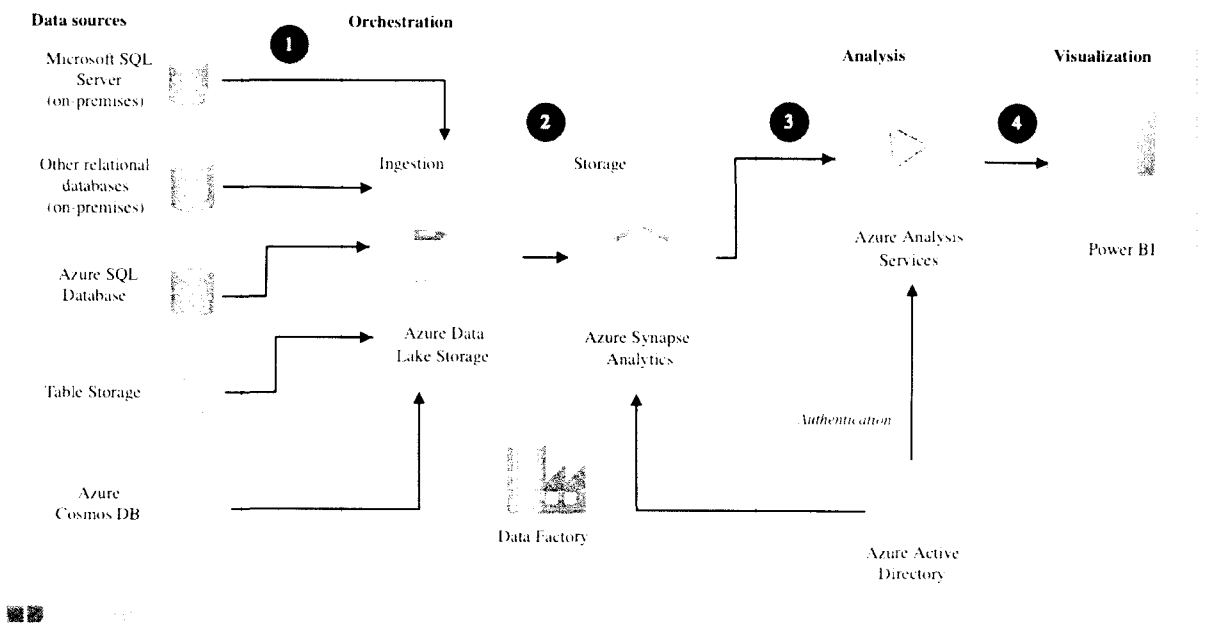

Рис 3.14 Azure

Мережеві функції Azure включають створення ізольованих мереж в хмарі та їх з'єднання з локальними мережами, глобальну мережу доставки контенту для швидкого розповсюдження веб-контенту, а також хмарну службу доменних імен. Для підтримки життєвого циклу розробки програмного забезпечення, Azure пропонує набір сервісів для управління вихідним кодом, CI/CD та управління проектами. Автоматизація збірки, тестування та розгортання додатків здійснюється за допомогою Azure Pipelines.

Azure також включає сервіси штучного інтелекту та машинного навчання, що дозволяють розробляти, навчати та розгортати моделі машинного навчання, а також додавати можливості штучного інтелекту до додатків, такі як розпізнавання мови, зображень та тексту.

Використання Microsoft Azure забезпечує масштабованість, надійність, безпеку та інтеграцію з іншими продуктами Microsoft і сторонніми рішеннями. Azure постійно оновлюється, додаючи нові функції та сервіси, що дозволяє організаціям залишатися на передовій технологій.

Правильний вибір технологій та інструментів для програми моделювання інформаційної системи лікування є вирішальним для забезпечення її надійності, ефективності та масштабованості. Це дозволить створити потужний інструмент для оптимізації процесів лікування, забезпечення високої якості медичних послуг та підвищення задоволеності пацієнтів.

Інтерфейси для обміну даними:

Інтеграція потребує створення інтерфейсів для обміну даними між різними компонентами системи. Це можуть бути API (Application Programming Interface), які забезпечують доступ до функцій системи та обмін даними між різними модулями. Інтерфейси повинні бути розроблені таким чином, щоб забезпечити сумісність та безпеку передачі даних.

Синхронізація даних:

Важливою частиною інтеграції є забезпечення синхронізації даних між різними системами. Це включає механізми для оновлення даних у режимі реального часу, обробку конфліктів даних та забезпечення цілісності даних.

Наприклад, зміни в електронних медичних записах пацієнтів повинні негайно відображатися у всіх пов'язаних системах.

Тестування інтеграції:

Після розробки та налаштування всіх компонентів системи необхідно провести тестування інтеграції. Це включає перевірку взаємодії між модулями, тестування обміну даними та оцінку продуктивності системи. Мета тестування - виявити та усунути можливі проблеми, забезпечивши надійну роботу системи.

Моніторинг та вдосконалення:

Після впровадження системи необхідно постійно моніторити її роботу та збирати зворотний зв'язок від користувачів. Це допоможе виявити можливі проблеми та вдосконалити систему відповідно до потреб користувачів. Регулярне оновлення програмного забезпечення та впровадження нових функцій також є важливими аспектами підтримки системи.

Інтеграція програмного забезпечення для моделювання інформаційної системи лікування є складним, але необхідним процесом, який забезпечує злагоджену роботу всіх компонентів системи. Це дозволяє забезпечити ефективне управління медичними даними, покращити якість медичних послуг та оптимізувати роботу медичного персоналу.

3.2 Розробка проекту

Для створення проекту Django пропонує широкий набір інструментів. Основним способом запуску нового проекту є утиліта `django-admin`. Під час виконання команди `startproject <project_name>` утиліта створює базовий каркас системи, що містить усі необхідні налаштування для подальшої розробки. Файлова структура після виконання виглядає як на рисунку 3.15

Рис 3.15 Базова структура системи

Файли 'asgi.py` та `wsgi.py` містять логіку, необхідну для налаштування комунікації між Python-кодом і вебсервером. Файл `urls.py` містить базові налаштування для розпізнавання URL-адрес. Файл `settings.py` включає всі налаштування системи, такі як перелік встановлених модулів, секретні ключі, налаштування підключень до бази даних, налаштування системи шаблонів, налаштування валідації даних, налаштування системи авторизації тощо. Файл `manage.py` є основною точкою входу для роботи з системою під час подальшої розробки та виступає інтерфейсом для взаємодії з усіма внутрішніми інструментами.

Оскільки Django має модульну структуру, необхідно створити новий модуль. За це відповідає команда `python manage.py startapp medical`. Після її виконання у файловій системі буде створено модуль `medical`, який матиме структуру, як показано на рисунку 3.16.

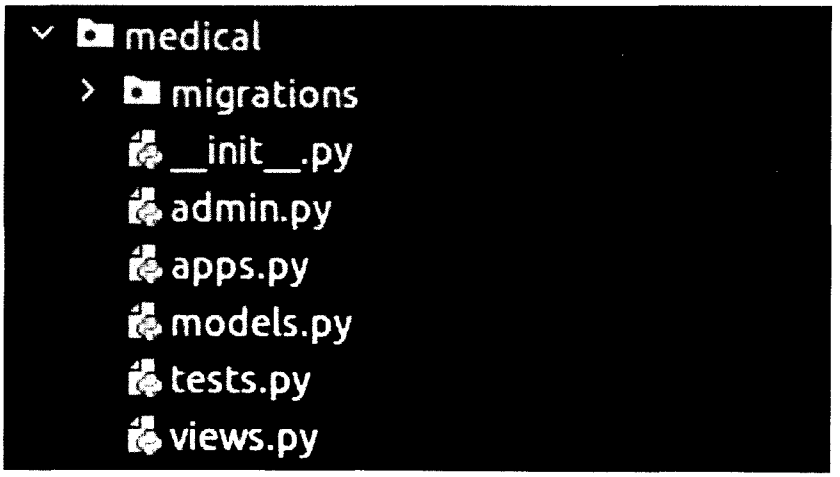

Рис 3.16 Структура нового модуля

Файл models.py описує у собі всі моделі, які представляють дані в базі даних. Тут міститься уся логіка взаємодії з даними. Файл views.py містить у собі контролери. Файл tests.py містить у собі автоматичні тести для данного модулю. Файл apps.py містить додаткові налаштування для модулю. Файл admin.py містить у собі налаштування для адміністративної панелі, які пов'язані з цим модулем. Папка migrations містить у собі файли з Python кодом, кожен з яких відображає послідовні зміни у моделях, а також дії , які потрібно виконати з базою даних для внесення цих змін.

Наступним кроком є створення моделей. Код моделей наведено у додатку А. Для роботи з базами даних у Django є вбудована система Object-Relational Mapping (ORM), яка дозволяє працювати з даними не за допомогою запитів, а виключно на рівні Python коду, описувати моделі за допомогою класів, а також автоматично створювати та застосовувати міграції. Для початку потрібно визначити моделі, які будуть містити інформацію про користувачів системи, про пацієнтів та працівників, а також всю інформацію, потрібну для реєстрації і авторизації користувачів у системі.

Django має вбудовану систему створення і аутентифікації користувачів, яка включає в себе автоматичне додавання метаданих (час створення акаунту, права доступу, додавання в групу користувачів з певними дозволами і інші), захищене хешування паролів і багато іншого. Система аутунтифікації фреймворку працює лише з однією моделлю. Тому інформація про усіх користувачів системи розділена на 3 моделі: User, Employee, Patient.

Інформація про користувачів системи представлена у моделі User, яка наслідується від базового класу фреймворку AbstractUser. Тут зберігаються усі дані, які потрібні для авторизації користувача та всі додаткові метадані фреймворку. Крім цього, модель також зберігає інформацію про тип користувача. У системі наявні 4 типи користувачів: пацієнти, лікарі, інший персонал, адміністратори. Для правильної роботи системи аутентифікації використовується UserManager, який відповідальниий за створення нових користувачів, а також за усю взаємодію з даними про них.

Для пацієнтів додаткова інформація зберігається у моделі Patient. У цій моделі міститься ім'я людини, дата народження, контактниий номер телефону. Для персоналу (всі користувачі крім пацієнтів) додаткова інформація міститься у моделі Employee. Тут зберігається ім'я, посада людини, контактний номер телефону, посилання на зображення, додатковий опис, а також ціна прийому, якщо даний працівник надає послуги пацієнтам.

Далі було створено моделі, які залежні від моделей Patient і Employee. В першу чергу, це EmployeeSchedule, Appointment і MedicalRecord. Тут містяться дані відповідно про розклад роботи працівників, відвідування лікарів і історію хвороб. Далі було створено моделі LaboratoryTest, TestResult, Procedure, ProcedureRecord, Supply, Invoice. На цьому створення моделей завершено. Далі потрібно створити міграції для бази, і застосувати їх для внесення змін.

Далі потрібно створити логіку взаємодії системи з моделями. Для цього потрібно створити контролери, в яких буде міститися потрібний для цього код. Django надає 2 піходи до створення контролерів: функціональний або класовий. У даній системі всі контролери побудовані на класах, оскільки це зменшує перевикористання коду. Кожен клас контролера має базовий інтерфейс, який дозволяє шаблонно створювати методи для обробки CRUD запитів. Контролери мають або стандартну відповідь HttpResponse, або JSON результат, або HTML документ. У даній системі представлення (View) будуються за допомогою статичних HTML шаблонів, які заповнені даними. Контролери при обробці запитів отримують певні дані з моделей, відставляють їх у шаблон та повертають HTML документ. Візуальне пояснення цього процесу можна знайти у графічних матеріалах на схемі послідовностей . Також у графічних матеріалах можна знайти UML діаграму класів, яка містить усі класи для моделей і відображень.

Для реалізацій представлення даних використовується вбудована у фреймворк система HTML шаблонів. Ідея полягає у тому, що при запиті на певниий ресурс клієнт отримує статичну сторінку з усіма потрібними даними. Для отримання якихось інших даних потрібно зробити окремий запит на інший ресурс, що відобразить нову сторінку. При обробці таких подій, як, наприклад,

обробка форми, відбувається запит з усі введеними даними, який обробляється контролером, після чого повертається нова сторінка, яка відображається для клієнта. Це є основною відмінністю від SPA (single page application) або MPA (multi page application), оскільки у цих підходах використовуються асинхронні запити для отримання нових даних, і більш компклесна система відображення. Для створення даної медичної системи функціоналу статичних шаблонів буде достатньо для реалізації усього потрібного функціоналу.

На даному моменті проект має готову MVC структуру, яка дозволяє користувачам працювати з системою. Для полегшення розгортання системи, потрібно загорнути програмну складову у контейнери Docker. Це значно спростить процес запуску програми, оновлення потрібних залежностей, моніторингу стану системи. Як уже описувалося, раніше потрібно 2 контейнери — для коду і для бази даних. Було обрано наступні версії Docker Image: python:3.9-slim для коду, postgres:15.3. Для запуску контейнерів було використано docker-compose, оскільки він дозволяє налаштувати мережу для контейнерів, правила запуску і залежності контейнерів.

3.3 Тестування програмного забезпечення

Тестування програмного забезпечення - це процес перевірки функціональності, якості та відповідності вимогам програмного продукту перед його випуском. Це важлива частина розробки програмного забезпечення, що дозволяє виявити помилки, проблеми та недоліки, що можуть вплинути на роботу програми.

Тестування програмного забезпечення включає в себе різноманітні види тестів, такі як модульні тести, функціональні тести, інтеграційні тести, системні тести, тести відмовостійкості тощо. Кожен вид тестування спрямований на перевірку певних аспектів програми.

Мета тестування програмного забезпечення - забезпечити, що програмний продукт працює коректно, відповідає вимогам і очікуванням користувачів, а також має надійність, безпеку та високу якість. Тестування допомагає знизити

ризики випуску програмного забезпечення та підвищити задоволення користувачів його роботою.

Рис 3.17 Інформаційна сторінка

На початку тестуванні програмного забезпечення (рис 3.17) користувач потрапляє на початковий екран тестувального програмного забезпечення - це перше, що бачить користувач при запуску програми. Він грає важливу роль у першому враженні користувача та впливає на спосіб, яким він сприймає програму. На початковому екрані може бути присутнім логотип або назва програми, яка допомагає ідентифікувати програму.

Опис функцій може надати користувачеві загальне уявлення про те, що можна зробити з програмою. Кнопки або посилання на важливі опції дозволяють користувачам швидко переходити до необхідних функцій, таких як реєстрація або вхід. Елементи навігації допомагають користувачам легко орієнтуватися в програмі, особливо якщо вона має складну структуру. Інформація про версію або оновлення може бути корисною для користувачів, які хочуть бути в курсі останніх оновлень.

Також можуть бути доступні посилання на довідкову інформацію або підтримку, щоб користувачі могли отримати додаткову допомогу або інформацію.

(Рис 3.18 - 3.22) де вказана коротка інстукція та присутння подальша навігація по додатку. Після ознайомлення з інструкцією розпочинається детальне визначення статті віку та певних фізичних особливостей користувача

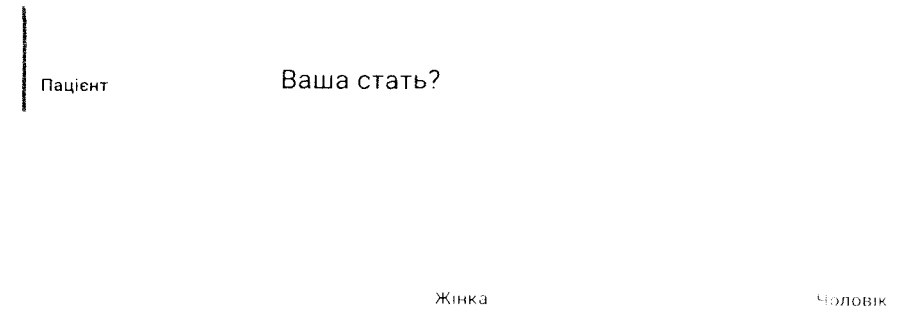

Рис 3.18 Вибір статі

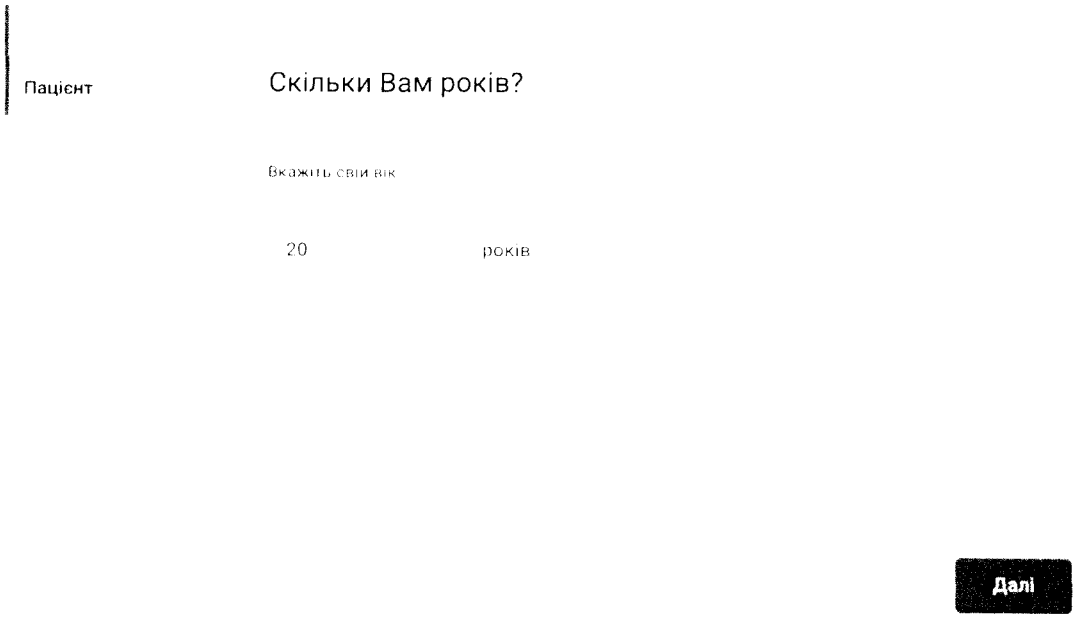

Рис 3.19 Ввід персональних даних

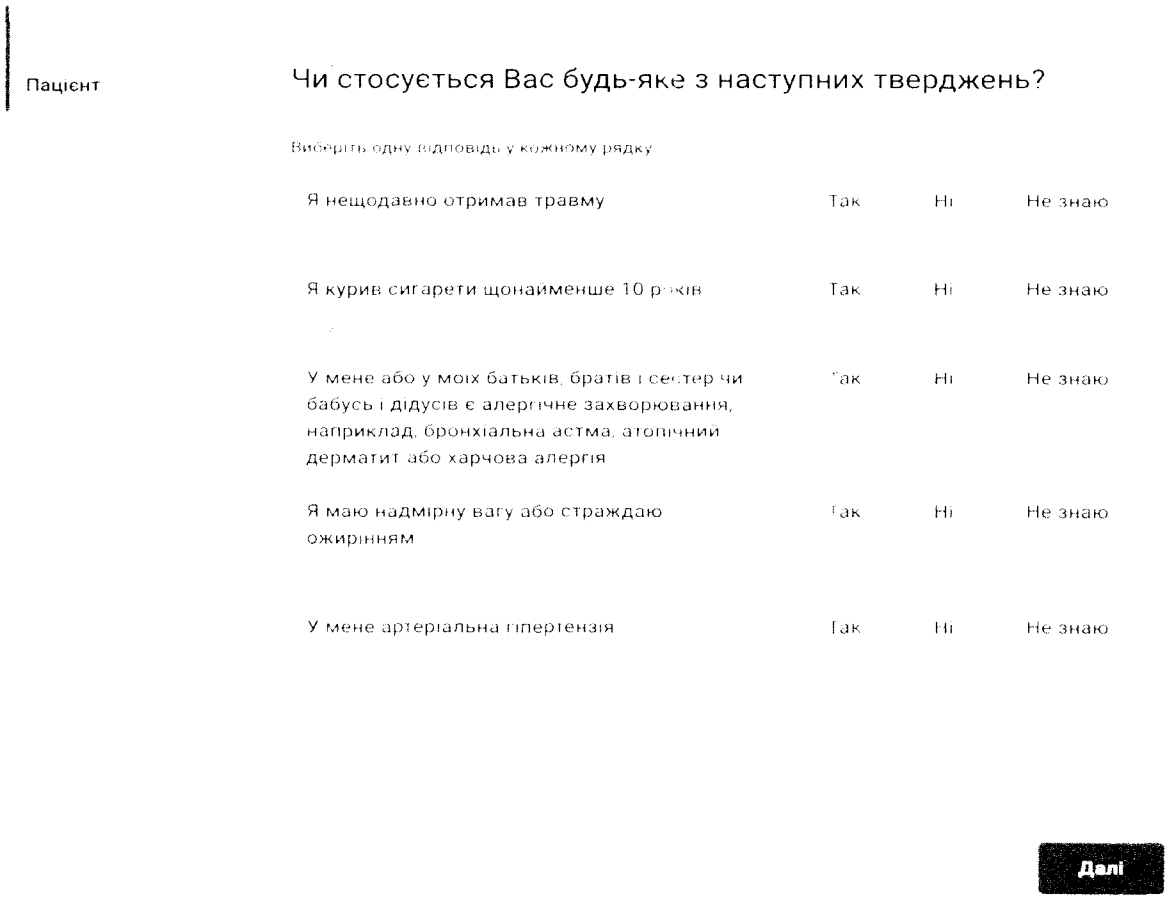

Рис 3.20 Перевірка додаткової інформації

Після переходу користувача на початковий екран програмне забезпечення може надати можливість ввести свої симптоми. Це може бути реалізовано через спеціальну форму або інтерфейс, де користувач може ввести інформацію про свої симптоми, такі як біль, дискомфорт, чи інші відчуття. Користувач може бути попрошений надати детальні описи симптомів, їх тривалість та інтенсивність.

Програма також включає функцію вибору симптомів зі списку передбачених варіантів, щоб полегшити користувачеві введення інформації. Цей етап є важливим для забезпечення програмі достатньої інформації для подальшого аналізу та надання рекомендацій або діагнозу.

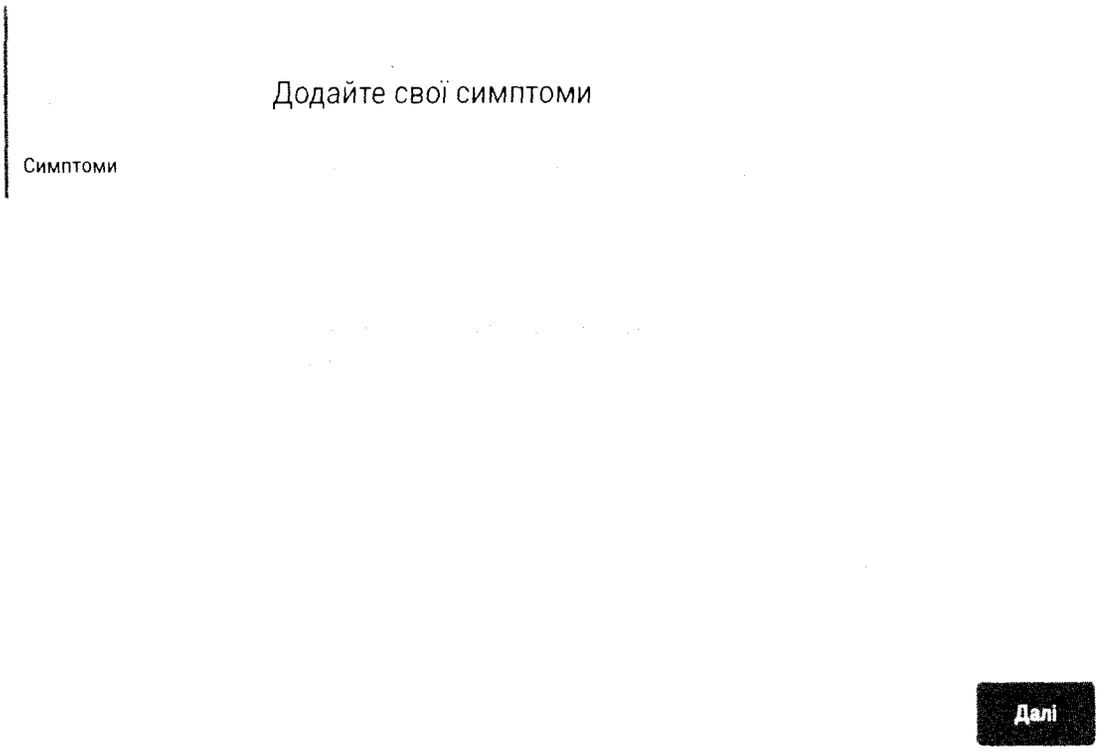

Рис 3.21 Вибір симптом

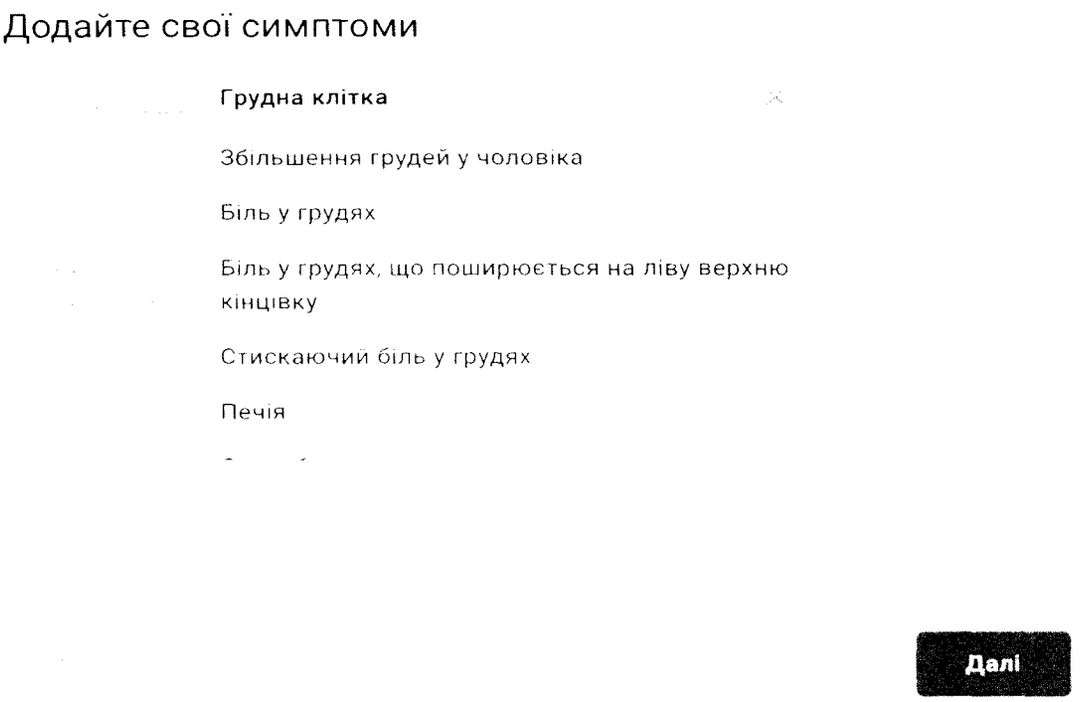

Рис 3.22 інтерфейс додавання симптом

Після введення своїх симптомів може розпочатися детальне опитування, спрямоване на отримання додаткової інформації про стан користувача. Це опитування може включати різноманітні питання щодо симптомів, які можуть допомогти уточнити діагноз або рекомендації щодо подальшого кроку. Наприклад, опитування може запитувати про:

1. Тривалість симптомів: Користувач може бути попрошений вказати, скільки часу він відчуває ці симптоми.

2. Інтенсивність симптомів: Питання може бути про те, як сильно вони відчуваються, наприклад, на шкалі від 1 до 10.

3. Супутні симптоми: Користувач може бути запитаний про будь-які інші симптоми, які він спостерігає разом із основними симптомами.

4. Фактори, які впливають на симптоми: Питання може бути про фактори, які збільшують або зменшують інтенсивність симптомів, такі як харчування, активність або час доби.

5. Медична історія та фактори ризику: Користувач може бути попрошений надати інформацію про свою медичну історію, хвороби, які він має або мав раніше, та будь-які інші фактори ризику.

Ці питання можуть допомогти уточнити діагноз або надати користувачеві більш точні рекомендації щодо подальшого кроку, наприклад, чи потрібно звернутися до лікаря чи виконати додаткові медичні тести (рис 3.23)

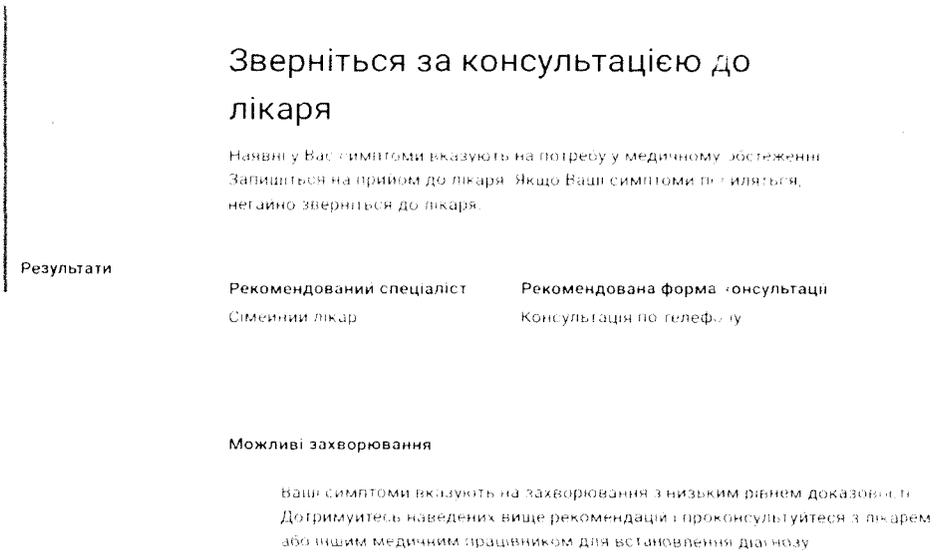

Рис 3.23 Результати обстеження

Оцінка зручності та ефективності використання інтерфейсу системи лікування є важливим кроком у забезпеченні якісної медичної допомоги.

- Легкість навігації: Інтерфейс має просту та зрозумілу структуру, щоб користувачі могли швидко знаходити потрібну інформацію та функції.

- Простота використання: Інтерфейс є інтуїтивно зрозумілим для користувачів, що дозволяє їм легко виконувати рутинні завдання без зайвих зусиль.

- Функціональність: Інтерфейс включає всі необхідні функції для реалізації різних медичних процедур та операцій, включаючи запис даних, перегляд історії пацієнтів, назначення лікування тощо.

- Адаптивність: Інтерфейс є адаптивним до потреб різних категорій користувачів, таких як лікарі. медичні сестри та адміністративний персонал. забезпечуючи їм лише необхідну інформацію та функції.

ВИСНОВОК

Моделювання інформаційних систем для лікування є ключовим етапом у вдосконаленні медичної сфери. Це не лише технологічний крок уперед, але й стратегічне вирішення для покращення результатів лікування та забезпечення більш ефективного взаємодії між медичним персоналом та пацієнтами.

Автоматизована діагностика, яка базується на алгоритмах машинного навчання та штучного інтелекту, може суттєво полегшити та прискорити процес встановлення діагнозів. Здатність алгоритмів обробляти великі обсяги клінічних даних і видача результатів без зайвого запізнення дозволяє медичному персоналу оперативно реагувати на стан пацієнтів та призначати необхідне лікування.

На етапі теоретичного аналізу було обґрунтовано вибір технологічного стеку та супроводжуючого інструментарію.

Створено інформаційну систему лікування для медичних установ та реалізовано веб-додаток з використанням сучасних веб-технологій. Запропоновано алгоритм для ефективного управління даними пацієнтів та медичного персоналу.

Спроєктовано та створено архітектуру додатку. Всі розроблені модулі пройшли етапи тестування. Програмний код було покрито модульними тестами, а також проведено функціональне тестування бази даних, що дозволило впевнитися у відповідності роботи системи.

Окрім того, автоматизовані системи діагностики можуть допомогти зберегти час медичного персоналу, оскільки частина діагностичного процесу виконується автоматично. Це дозволяє медичному персоналу зосередитися на інших важливих аспектах лікування та догляду за пацієнтами, підвищуючи ефективність використання робочого часу та забезпечуючи більш якісне обслуговування.

Нарешті, інтеграція автоматизованих систем діагностики з електронними медичними записами є ключовим аспектом удосконалення медичного обслуговування. Це дозволяє миттєво отримувати доступ до медичних даних пацієнтів, зберігати та обробляти ці дані з використанням передових технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Саймон Хайкин Нейронные сети: полный курс = Neural Networks: A Comprehensive Foundation. — 2-е изд. — М.: «Вильямс», 2006. — С. 1104. — ISBN 0-13-273350-1

2. Redmon, J., Divvala, S., Girshick, R., Farhadi, A. "You Only Look Once: Unified,Real-Time Object Detection." 2016 IEEE Conference on Computer Vision and PatternRecognition (CVPR), 2016. doi: 10.1109/CVPR.2016.91

3. E. Bush, JavaScript Applications with Node.js, React, React Native and MongoDB. 2018. 392 с.

4. J. Lebensold, React Native Cookbook: Bringing the Web to Native Platforms. 2018. 176 с.

5. What is data labeling? URL: [illegible] (дата звернення: 12.10.2023)

6. What are Recurrent Neural Networks (RNN)? URL: [illegible] (дата звернення: 22.05.2024)

7. Illustrated Guide to LSTM's and GRU's: A step by step explanation URL: [illegible] (дата звернення: 22.05.2024)

8. Generative adversarial networks (GANs) : a deep dive into the architecture and training process URL: [illegible] (дата звернення: 22.05.2024)

9. VGG16 – Convolutional Network for Classification and Detection URL: [illegible] (дата звернення: 22.05.2024)

10. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84-90.

11. Добровська Л. М. Теорія та практика нейронних мереж : навч. посіб. / Л. М. Добровська, І. А. Добровська. – К.: НТУУ «КПІ» Вид-во «Політехніка», 2015. – 396 с.

12. Hardesty L (14 April 2017). [illegible] MIT News Office. Retrieved 2 June 2022.

13. Artificial neural networks Evergreen. URL: [illegible] (дата звернення: 12.10.2023).

14. Ultralytics YOLOv8 Docs: URL: [illegible] (дата звернення: 12.10.2023)

15. Introducing Instance Segmentation in YOLOv5 v7.0 URL: [illegible] (дата звернення: 22.05.2024)

16. Block, Ryan (March 6, 2008). "Live from Apple's iPhone SDK press conference". www.Engadget.com. Engadget – AOL. Retrieved June 11, 2017.

17. Krill, Paul (March 7, 2008). "Sun: we'll put Java on the iPhone". www.InfoWorld.com. InfoWorld – International Data Group. Retrieved June 19, 2017.

18. Paul, Ryan (September 15, 2009). "MonoTouch drops .NET into Apple's walled app garden". ArsTechnica.com. Ars Technica – Condé Nast. Retrieved June 19, 2017.

19. YOLO-Based Pose Estimation: Bridging Speed and Accuracy URL: [illegible] (дата звернення: 22.05.2024)