

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти)

на тему СИСТЕМА АНАЛІЗУ, ОБРОБКИ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ

Виконав: студент 2 курсу, групи КНТ-512м
спеціальності

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні системи та мережі

ПОГОСОВ А. А.

(ПРИЗВИЩЕ та ініціали)

Керівник ТЯГУНОВА М.Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент КОЗИНА Г.Л.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти магістерський
Спеціальність 123 Комп'ютерна інженерія

(код і найменування)

Освітня програма (спеціалізація) «Комп'ютерні системи та мережі»
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Зав. кафедри Кудерметов Р.К.

“ ” 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ПОГОСОВА Артема Альбертовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Система аналізу, обробки та візуалізації даних

керівник проєкту (роботи) доцент, ТЯГУНОВА Марія Юріївна

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від “ 24 ” жовтня 2023 року № 400

2. Строк подання студентом проєкту (роботи) 10 грудня 2023 року

3. Вихідні дані до проєкту (роботи) Платформа розробки: Clojure, ClojureScript, MySQL, Reagent, Re-Frame та Docker.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз вимог до системи;

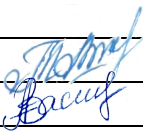
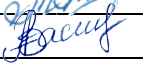
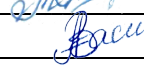
2. Проєктування системи;

3. Реалізація системи;

4. Дослідження розробленої системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ТЯГУНОВА М. Ю., доцент		
нормоконтроль	ПОЛЬСЬКА О.В., ст.викладач		

7. Дата видачі завдання 01.10.2023 р.

КАЛЕНДАРНИЙ ПЛАН

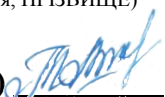
№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Порівняння з аналогічними системами	7.10.2023	
2	Аналіз вимог до системи	10.10.2023	
3	Розробка структури системи	18.10.2023	
4	Розробка алгоритму роботи системи	25.10.2023	
5	Реалізація системи	10.11.2023	
6	Дослідження розробленої системи	27.11.2023	
7	Оформлення отриманих результатів у ПЗ	05.12.2023	

Студент


(підпис)Артем ПОГОСОВ

(ім'я, ПРІЗВИЩЕ)

Керівник проєкту (роботи)


(підпис)Марія ТЯГУНОВА

(ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 65 сторінок, 21 рисунок, 3 таблиці, 20 джерел.

CLOJURE, MYSQL, REACT, АДАПТИВНІСТЬ, ОПТИМІЗАЦІЯ, ПОРІВНЯЛЬНИЙ АНАЛІЗ, РЕАКТИВНЕ ПРОГРАМУВАННЯ

Об'єкт дослідження – процес функціонування інформаційної системи аналізу, обробки та візуалізації даних.

Предмет дослідження – методи аналізу, обробки та візуалізації даних.

Мета роботи – створення електронної медичної системи аналізу, обробки та візуалізації різноманітної спеціалізованої інформації, на основі якої вчені зможуть робити висновки стосовно медичних показників тих чи інших пацієнтів.

У першому розділі проведено ґрунтовний порівняльний аналіз аналогічних систем та визначаються специфікації вимог до системи.

У другому розділі відбувається проектування системи на основі розроблених алгоритмів.

У третьому розділі детально описується програмна реалізація інформаційної системи з використанням Clojure, Docker, Reagent, Re-frame та база даних MySQL.

У четвертому розділі розроблена інформаційна система ретельно досліджується та проходить тестування. В результаті цього дослідження було встановлено, що розроблена система працює втричі швидше, ніж аналоги, при цьому забезпечуючи достатньо високий рівень надійності. Крім того, було проведено тестування адаптивного інтерфейсу користувача на різних пристроях.

ABSTRACT

MT: 65 pp., 21 fig., 3 tables, 20 sources.

CLOJURE, MYSQL, REACT, ADAPTIVE UI, OPTIMISATION,
COMPARATIVE ANALYSIS, REACTIVE PROGRAMMING

The object of research is the functioning process of an information system of data analysis, processing, and visualization.

The subject of research is the methods of data analysis, processing, and visualization.

The goal of the research is to create an electronic medical system for analyzing, processing, and visualizing various specialized information, which will enable researchers to draw conclusions regarding the medical indicators of patients.

The first chapter provides a thorough comparative analysis of similar systems and specifies the requirements for the system.

The second chapter involves the design of the system based on developed algorithms.

The third chapter provides a detailed description of the software implementation of the information system using Clojure, Docker, Reagent, Re-frame, and the MySQL database.

The fourth chapter thoroughly examines and tests the developed information system. As a result of this research, it was established that the developed system operates three times faster than its counterparts while maintaining a high level of reliability. Additionally, testing of the adaptive user interface on different devices was conducted.

ЗМІСТ

Скорочення та умовні позначки	7
Вступ.....	8
1 Аналіз вимог до системи	10
1.1 Аналіз предметної області.....	10
1.2 Порівняння аналогічних програмних продуктів.....	13
1.3 Вибір архітектури системи.....	20
1.4 Вибір СУБД	23
1.5 Вибір технологій розробки системи.....	26
2 Проєктування системи	29
2.1 Визначення та аналіз вимог до системи	29
2.2 Розробка специфікації вимог SRS	33
2.3 Моделювання системи.....	36
2.4 Розробка інтерфейсу користувача	41
3 Реалізація системи.....	43
3.1 Схема роботи проєктованої системи.....	43
3.2 Створення контейнеризації системи	44
3.3 Реалізація серверної частини системи	47
3.4 Реалізація клієнтської частини системи	52
4 Дослідження розробленої системи	55
4.1 Дослідження швидкодії та надійності системи.....	55
4.2 Дослідження адаптивності інтерфейсу користувача	59
Висновки	63
Перелік джерел посилання	64

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API	–	прикладний програмний інтерфейс
CRUD	–	"Create, Read, Update and Delete"
CSS	–	каскадні таблиці стилів, Cascading Style Sheets
HTML	–	розмітка гіпертекстових документів, HyperText Markup Language
JSON	–	текстовий формат обміну даними, JavaScript Object Notation
OC	–	операційна система
MVC	–	патерн проєктування, Model-view-controller
SRS	–	специфікація вимог до програмного забезпечення, Software Requirements Specification
UI	–	інтерфейс користувача, User Interface
UML	–	уніфікована мова моделювання, Unified Modeling Language
СУБД	–	система управління базами даних

ВСТУП

Останнім часом у світі активно набувають поширення системи аналізу, обробки та візуалізації даних. Особливо такі системи стали популярними у сфері електронної охорони здоров'я.

Головна мета подібних систем полягає у забезпеченні можливості працювати без паперу, створенні простору для інновацій в медицині та сприянні розвитку медичного IT-ринку. Провідну роль серед таких систем займають системи створені для медичних науковців.

Такі системи вміють:

- отримувати необроблену інформацію;
- аналізувати інформацію на основі визначених критеріїв;
- оброблювати інформацію шляхом систематизації та багаторівневої трансформації;
- візуалізувати оброблені дані у зручному для кінцевого користувача виді.

Як правило основний підхід подібних електронних медичних систем полягає у визначенні точної конвергенції живих організмів, на основі чого можна отримати цінну інформацію для подальшого аналізу та висновків.

Знаходження спільних рис та закономірностей, наприклад, допомагають експериментальним шляхом визначити як одні і ті самі віруси будуть взаємодіяти з однаковими за своєю структурою клітинами в різних організмах.

Ми живемо в час можливостей, які постійно збільшуються довкола нас. Буквально декілька десятків років тому годі й було уявити, що вчені зможуть визначати ризики ускладнень пацієнта або шанси на його одужання за допомогою електронної медичної системи, яка оперує мільйонами даних та візуалізує їх у наглядному виді. На сьогоднішній день в Україні досить слабо розвинений напрямок подібних систем. Тому створення подібної медичної системи може стати чудовим шансом аби зайняти нішу і внести певні інновації

на локальний ринок. Таким чином, обрана тема магістерської роботи є актуальною.

Мета роботи полягає у створенні електронної медичної системи аналізу, обробки та візуалізації різноманітної спеціалізованої інформації, на основі якої вчені зможуть робити висновки стосовно медичних показників тих чи інших пацієнтів.

Для вирішення поставленої мети необхідно вирішити наступні завдання:

- провести аналіз предметної області;
- створити базу із необробленої інформації;
- спроектувати інформаційну систему на основі реактивних методів програмування;
- реалізовувати спроектовану систему;
- проаналізувати отримані результати і, у разі необхідності, покращити параметри реалізованої системи.

Наукова новизна системи полягає у використанні методів функціонального та реактивного програмування з єдиним джерелом істини та чистими функціями для покращення користувацького досвіду при взаємодії з розробленою інформаційною системою аналізу, обробки та візуалізації даних.

1 АНАЛІЗ ВИМОГ ДО СИСТЕМИ

1.1 Аналіз предметної області

При створенні нової медичної системи завжди постає питання, яким чином будуть аналізуватися, оброблюватися та відображатися величезні обсяги даних про пацієнтів. Це можуть бути такі показники, як стать, вік, кількість відвіданих сеансів, стадія хвороби, кількість генів, які піддалися мутації та зразки крові.

Раніше всю подібну інформацію зберігали та систематизували у паперовому вигляді, як правило за допомогою різноманітних методів управління знаннями та ведення нотаток, що використовуються в дослідженнях та навчанні.

Одним з подібних методів нотування був метод Зетелькастен. Цей метод складається з великої кількості окремих розташованих нотаток з ідеями та іншою короткою інформацією, яка записується коли вони з'являються або отримуються. Нотатки завжди вказані в ієрархічному порядку, так що нові примітки можуть бути вставлені у конкретне розташування та містять метадані, що дозволяють записувачу поєднувати записи між собою. Наприклад, вони можуть містити теги, які описують важливі моменти в нотатці, і вони можуть посилалися на інші нотатки. Порядок, метадані, структура та формат приміток можуть відрізнятися залежно від використаного методу. До основних проблем подібного варіанту можна віднести погану візуалізацію великих об'ємів даних, повільний аналіз та довгий час прийняття рішення.

В еру цифрових технологій на допомогу науковцям приходять бази даних, швидкі алгоритми (сортування) обчислювання, групування даних та сучасні бібліотеки візуалізації. Якщо казати про бази даних, то їх можна визначити, як певний набір різноманітних даних, які зв'язані між собою спільною ознакою або характеристикою, та мають порядок, наприклад, за алфавітом.

База даних пройшла десятки років еволюції від плоскої файлової системи до реляційної та об'єктно-реляційної системи. Цей процес тривав кілька поколінь. Найбільшою перевагою бази даних є швидкість, з якою можна вводити та

використовувати необхідну інформацію. Завдяки спеціальним алгоритмам, що використовуються в базі даних, можна легко знайти потрібну інформацію всього за кілька секунд.

Розробка сучасної системи аналізу, обробки та візуалізації даних включає в себе кілька важливих внутрішніх завдань. Одним з найважливіших таких завдань є сортування великих об'ємів даних. Під сортуванням зазвичай розуміється перестановка елементів будь-якого масиву в певному порядку. Це завдання є одним з найважливіших, оскільки від ефективності сортування напряду залежить продуктивність системи.

Не менш значущою складовою розробки подібних систем є візуалізація вже оброблених даних для подальшого відображення для користувача. В основному для таких задач використовують спеціально розроблені бібліотеки для візуалізації складних графіків та таблиць на мільйони записів.

За допомогою сучасних технологій з'явилася можливість створення прецизійних медичних цифрових систем. Прецизійна медицина-це інноваційна парадигма в лікуванні, яка відрізняється від універсальної терапії, заснованої на індивідуальному підході до пацієнта. Прецизійна медицина відкидає підхід "проб і помилок" і виступає за індивідуальний вибір ліків і методів лікування для кожного пацієнта.

Першою країною, яка впровадила прецизійну медицину на національному рівні, були США в 2015 році за ініціативою президента Барака Обами. Концепція персоналізованого лікування сягає часів Гіппократа, але лише за останні роки набула популярності завдяки новим технологіям. Така модель медицини використовує діагностичне тестування генетичної, молекулярної та клітинної інформації для виявлення захворювань та вибору найкращого лікування. Геноміка сприяє створенню доказової бази для стратифікації схожих пацієнтів.

Прецизійна медицина може допомогти людині краще зрозуміти своє оточення, спосіб життя та спадковість. Ця інформація дозволяє точніше передбачити, які методи лікування є найбільш ефективними та безпечними, і, можливо, запобігти розвитку захворювання [1].

До інших переваг можна віднести:

- зміщенні акценту в медицині з лікування на профілактику;
- ранній діагностиці захворювань;
- точнішому виборі ліків та їх дозування;
- передбаченню та уникненню побічних наслідків;
- зменшенню часу, вартості і рівня невдач фармацевтичних клінічних випробувань.

Персоналізована медицина являє собою зміну парадигми від універсального підходу до методології, яка в свою чергу більш орієнтованої на пацієнта. Він спрямований на визначення ефективних підходів для пацієнтів на основі генетичних факторів, факторів навколишнього середовища та способу життя.

Завдяки персоналізованій медицині можна використовувати генетичні та молекулярні тести для визначення ризиків розвитку хвороб та виявлення їх на ранніх стадіях. Це дозволяє почати лікування раніше та зменшити ризик ускладнень.

Наявність геномних знань про людину дуже допомагає у розробці ліків. Також така медицина дозволяє уникнути непотрібного призначення лікарських засобів або доз, які можуть мати негативні побічні ефекти для пацієнта. Це зменшує ризик ускладнень та поліпшує якість життя пацієнта.

Можливість ідентифікувати пацієнтів, які найбільше виграють від процесу клінічних випробувань, підвищить безпеку пацієнтів від побічних ефектів, викликаних продуктом під час випробувань і дозволить проводити більш дрібні і швидкі випробування, що, в свою чергу, знизить загальну вартість необхідну для розробки і виведення препарату на ринок. Крім того, препарати, які вважаються неефективними для більшості населення, можуть бути схвалені для створення персоналізованого геному, який використовується для оцінки ефективності та потреби в цьому конкретному препараті або лікуванні, навіть якщо вони потрібні невеликому відсотку населення.

Лікарі часто використовують методологію "проб і помилок", поки не знайдуть найбільш ефективне лікування. Персоналізована медицина дозволяє прогнозувати ефективність лікування на основі геному конкретної людини.

Результати президційної медицини можуть покращити діагностику та прогноз захворювання шляхом виявлення біомаркерів та підтипів захворювання. Це може призвести до раннього виявлення та втручання, а також покращити результати лікування пацієнтів. Крім того, жовтневі препарати допомагають виявляти пацієнтів з високим ризиком розвитку певних захворювань, що дозволяє проводити профілактичні заходи [2].

Розуміючи унікальний молекулярний профіль пацієнта, медичний працівник може адаптувати лікування до конкретної людини. Багатоатомний аналіз може надати важливу інформацію про те які можуть бути потенційні цілі ліків, завчасно передбачити реакцію ліків та оперативно відстежувати хід лікування. Це може підвищити ефективність лікування та мінімізувати побічні ефекти.

Мабуть, найважливішою проблемою виводу персоналізованої медицини на комерційний рівень є захист пацієнтів. Однією з найбільших проблем є страх та потенційні наслідки для пацієнтів, які схильні або не реагують на певні методи лікування після генетичного тестування. Це включає психологічний вплив на пацієнта результатів генетичних тестів. Інша проблема полягає в праві сімей на пряму незгоду, коли генетична схильність і ризик вважаються спадковими. Також необхідно враховувати вплив на певні етнічні групи та наявність спільних генів.

1.2 Порівняння аналогічних програмних продуктів

Наведемо приклади найбільш популярних систем, які ґрунтуються на методології президційної медицини.

В першу чергу розглянемо таку систему, як "iMatrix".

Знайти цю систему можна за посиланням <https://imatrix.com> (рис. 1.1).



Рисунок 1.1 – Головна сторінка системи “iMatrix”

Ця система надає можливість залучати до себе нових потенційних клієнтів, які в перспективі можуть стати повноцінними пацієнтами. Для цього на сайті є зручна форма реєстрації, після проходження якої клієнт має можливість стати або інвестором, або пацієнтом. Також система має зручну можливість безкоштовної консультації. Клієнту необхідно заповнити форму зворотного зв'язку і менеджер у назначений час зв'яжеться з людиною для консультації стосовно питань, які зацікавили.

Зручним аспектом сайту є те що на ньому розміщена вичерпна текстова та відео-інструкція, у якій покроково розповідається про найголовніші особливості системи. Насправді це дуже важливо, адже чим швидше клієнт розбереться в інформаційній системі, тим швидше він зможе прийняти рішення про необхідність стати пацієнтом саме цієї системи.

При консультації та лікуванні своїх пацієнтів “iMatrix” використовує методології презинційної медицини. Кожному пацієнту надається персоналізований підхід, завдяки якому застосовуються унікальні підходи в лікуванні для кожної людини. Є можливість здати аналізи та потім побачити детальну інформацію по результатам в особистому кабінеті.

Для медичних науковців є можливість завантажувати в систему “сирі” дані про пацієнтів та стан їх здоров’я на основі вже проведених аналізів. Як правило в подібних системах використовується CSV формат даних. Він є досить поширеним та зручним у використанні. Це може бути побудова складних графіків або величезних звичайних та вкладених таблиць з можливістю сортування та фільтрації даних в реальному часі.

Варто відмітити, що “iMatrix” для побудови графіків використовує Javascript бібліотеку highcharts, яка хоча і має певні переваги в стилізації та гнучкості взаємодії з графічним відображенням даних, проте страждає сповільненням швидкодії при великих об’ємах даних. Також треба відмітити що цей сайт використовує так зване серверне відображення сторінок, що може негативно впливати на швидкодію системи при повільному інтернеті і зайвий раз навантажує сервер.

Визначимо основні переваги та недоліки системи “iMatrix”.

Переваги:

- гнучка стилізація графіків;
- форма для безкоштовної консультації клієнтів;
- обширна інструкція по взаємодії з системою;
- можливість відображати складні графіки та таблиці.

Недоліки:

- бібліотека візуалізації графіків highcharts повільно оброблює великі дані;
- немає адаптивного інтерфейсу користувача;
- серверний рендеринг сторінок.

“Inotiv” – потужна альтернатива з великою кількістю користувачів.

Адреса головної сторінки – <https://www.inotivco.com>. (рис. 1.2)

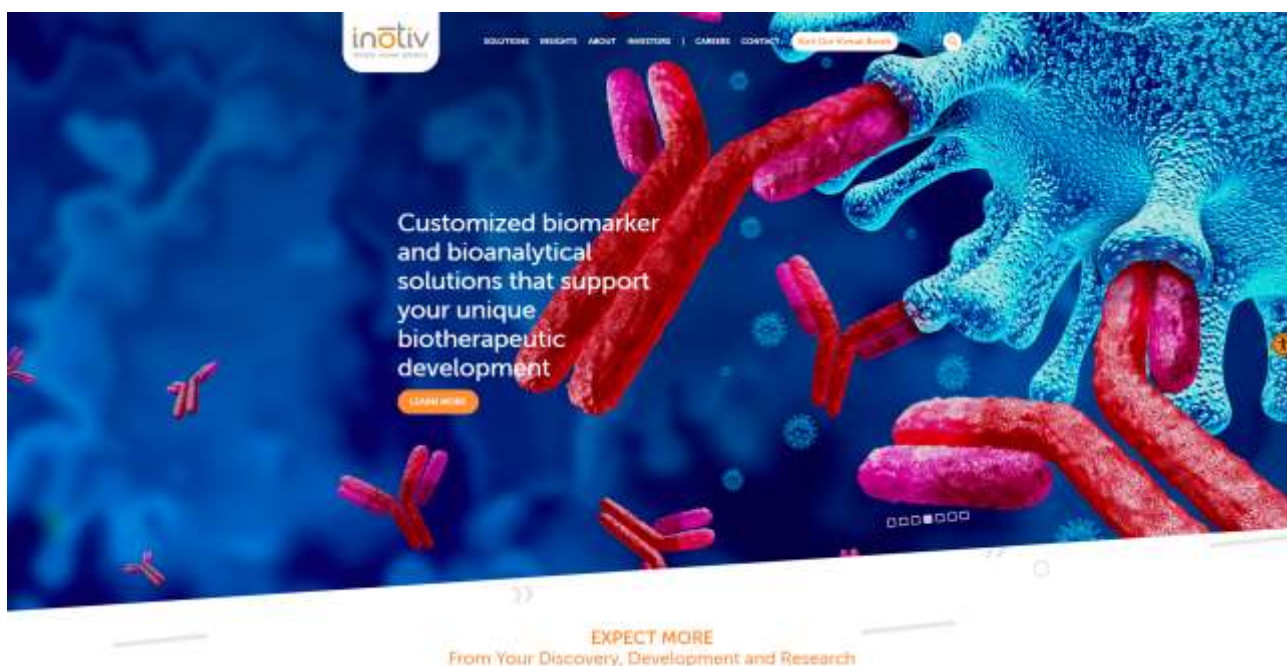


Рисунок 1.2 – Головна сторінка сайту “Inotiv”

“Inotiv” – система презиційної медицини, яка робить головний акцент на дослідженнях пов’язаних з різноманітними патологіями та мутаціями. При цьому важлива роль приділяється гістологічним дослідженням.

Система опирається на біоаналітичній інформації, яку отримує від пацієнтів. При цьому щоб потенційний пацієнт міг швидко розібратися в системі, існує спеціальний віртуальний тур по системі, який в свою чергу містить корисну інформацію у вигляді відео, тексту та голосових підказках.

Зручним є те, що відразу можна побачити відгуки клієнтів, які вже скористалися бажаними послугами системи. Також на головній сторінці знаходиться портфоліо “Inotiv”, що є гарним тоном, адже завдяки цьому можна з першого погляду зрозуміти, які послуги надає компанія та що цікавить клієнта. Не менш важливим є розміщення соціальних мереж та електронної пошти, бо це дозволяє швидко зв’язатися зі службою підтримки у разі виникнення проблеми або термінового запитання.

“Inotiv” використовує для візуалізації даних бібліотеку D3.js, яка являє собою чудову альтернативу відомій highcharts і при цьому не має недоліків при роботі з великими обсягами даних. Бібліотека D3.js забезпечує вибірку серед

десятків можливих графіків для побудови. Також варто зауважити що бібліотека об'єднала навколо себе багато користувачів, які як правило вже стикалась з певними проблемами і можуть підказати, як їх вирішити. З недоліків треба звернути увагу що D3.js все таки поступається кастомізацією у порівнянні з більш сучасними рішеннями і при цьому потребує більше часу для того щоб розібратися у документації.

Для побудови складних таблиць використовується бібліотека DataTables. Це досить зручне рішення, яке дозволяє досить швидко розробити потрібну таблицю, яка буде мати можливість сортування даних та виділення потрібних рядів для подальшої взаємодії з ними. Як відомо проблемою багатьох таблиць є адаптивний інтерфейс користувача, але DataTables забезпечує певну адаптивність в залежності від розміру дисплея. Єдиним суттєвим недоліком DataTables можна назвати неможливість за допомогою вбудованих інструментів зробити вкладену таблицю, яка іноді буває необхідна коли дані мають складну розгалужену структуру.

Визначимо основні переваги та недоліки системи “Inotiv”.

Переваги:

- акцент на дослідження мутацій;
- віртуальний тур по системі;
- відгуки реальних клієнтів;
- чудова швидкодія навіть при великих обсягах даних.

Недоліки:

- бібліотека D3.js поступається кастомізацією своїм аналогам;
- графіки не підтримують можливість генерації звітів по обраним точкам;
- серверний рендеринг сторінок.

Адреса головної сторінки “CareMe Health” – <https://www.careme.health>.

(рис. 1.3)



Рисунок 1.3 – Головна сторінка сайту “CareMe Health”

CareMe Health - це онлайн-платформа, яка надає широкий спектр медичних послуг і догляду за здоров'ям. Ця платформа дозволяє пацієнтам зручно спілкуватися зі своїми лікарями, отримувати консультації та лікування віддалено, а також вести електронний медичний запис.

CareMe Health також надає можливість відстежувати показники здоров'я, включаючи пульс, тиск, рівень цукру в крові та інші, за допомогою спеціальних пристроїв, які підключаються до платформи. Основна ідея CareMe Health полягає в полегшенні доступу до медичних послуг і покращенні комунікації.

CareMe Health пропонує індивідуальні рекомендації і програми здоров'я, враховуючи індивідуальні особливості та потреби кожного користувача, адже саме у цьому полягає суть прецизійного підходу в медицині. Клієнти можуть отримати персоналізовані поради щодо харчування, фітнесу, сну та інших аспектів здорового способу життя. Система дозволяє пацієнтам вести контроль за своїм здоров'ям, включаючи вимірювання пульсу, кров'яного тиску та рівня активності.

CareMe Health використовує для побудови графіків JavaScript-бібліотеку Chart.js. Це відкрита JavaScript-бібліотека для побудови графіків і діаграм на вебсторінках. Вона дозволяє легко та швидко створювати інтерактивні графіки,

що відображають дані в зрозумілій та привабливій формі. Chart.js надає простий та зрозумілий API, який дозволяє швидко створювати графіки без необхідності в глибоких знаннях JavaScript або графічного програмування.

Бібліотека підтримує побудову різних типів графіків, включаючи лінійні, стовпчасті, кругові, точкові та інші. Певно головною особливістю бібліотеки є можливість взаємодіяти з графіками, включаючи масштабування, вибір елементів, відображення підказок та інші функції.

У якості бібліотеки для побудови таблиць у системі використовується Vuetable. По суті це розширення для Vue.js, яке дозволяє легко створювати та керувати таблицями на вебсторінках. Vuetable дозволяє зручно відображати дані з різних джерел, таких як масиви, об'єкти або вебсервіси.

Незважаючи на багато переваг, Vuetable також має деякі недоліки. Бібліотека не має такого рівня функціональності, як у великих таблиць, таких як Excel або Google Sheets. Vuetable призначений для простих таблиць з базовими операціями. Бібліотека вміє сортувати дані, фільтрувати їх та має можливість обирати рядки для подальшої взаємодії з ними.

Варто зауважити що ця бібліотека вимагає встановлення багатьох залежностей, таких як Vue.js, Vuex, Vue Router тощо. Це може збільшити розмір проєкту та зробити його більш комплексним. Тому краще обирати бібліотеки які мають мінімум залежностей.

Визначимо основні переваги та недоліки системи “CareMe Health”.

Переваги:

- різнопланова взаємодія з графіками;
- електронний медичний запис;
- замір різних показників організму.

Недоліки:

- багато програмних залежностей від бібліотек;
- досить базові можливості відображення таблиць.

Таблиця 1.1 – Порівняльний аналіз обраних інформаційних систем

Ознака \ Назва	iMatrix	Inotiv	CareMe Health
Стилізація графіків	+	-	+
Вкладені таблиці	-	-	-
Клієнтський рендеринг	-	+	+
Адаптивний інтерфейс	+	+	-
Генерація звітів по графіку	-	-	+
Методи презиційної медицини	+	+	+
Гарна швидкодія при великих об'ємах даних	-	+	+
Підтримка генової теплової карти	+	-	-
Аналіз виживання	-	+	-
Експорт таблиць	-	-	-

1.3 Вибір архітектури системи

Архітектура програмного забезпечення - це організація та структура програмної системи, яка визначає взаємозв'язки між її компонентами, модулями та підсистемами. Вона визначає, які компоненти існують у системі, як вони взаємодіють між собою, які функції вони виконують та як вони організовані.

Приклади архітектурних шаблонів:

- модель - вид - контролер;
- представлення – абстракція – управління;

- клієнт - серверна архітектура;
- триярусна архітектура;
- сервісно - орієнтована архітектура.

При розробці програмного забезпечення можна створити клієнт, який може взаємодіяти з сервером шляхом відправки запитів. Таким чином, можна використовувати архітектурний стиль клієнт-сервер для проєктування та реалізації клієнта разом з сервером. Крім того, бажано мати компонентну архітектуру, яка дозволяє розбити дизайн та бізнес-логіку на незалежні компоненти зі спеціалізованими інтерфейсами [3].

Використання об'єктно-орієнтованого підходу до цих компонентів дозволить поліпшити можливість повторного використання, тестування і гнучкість. В популярному варіанті цей підхід передбачає наявність комп'ютерної мережі і розподіленої бази даних, яка включає корпоративну та індивідуальні бази даних. Архітектура клієнт-сервер є програмним забезпеченням з графічним інтерфейсом користувача. Тому у будь-якому випадку потрібно обмінюватися різноманітними даними з сервером бази даних, де розташована основна частина бізнес-логіки у формі процедур, або з виділеним файловим сервером.

Основні переваги клієнт-серверної архітектури:

- відсутність дублювання коду програми-сервера програмами-клієнтами;
- вимоги до комп'ютерів, на яких встановлено клієнт, знижуються;
- всі дані зберігаються на сервері, який, як правило, захищений набагато краще клієнтів.

До основних недоліків можна віднести:

- непрацездатність сервера може зробити непрацездатною всю обчислювальну мережу;
- підтримка роботи даної системи вимагає окремого фахівця - системного адміністратора.

Зображення клієнт-серверної архітектури (рис. 1.4).

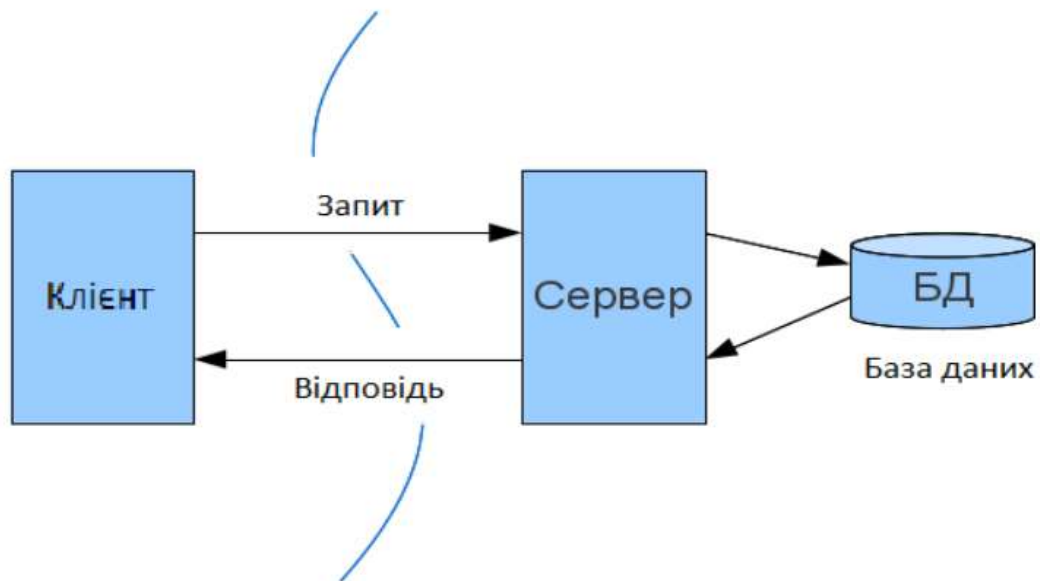


Рисунок 1.4 – Графічне зображення клієнт - серверної архітектури

Цікавим аспектом є те, що одна програма-сервер може обробляти запити від багатьох програм-клієнтів. Через це, зазвичай сервер розміщується на окремій обчислювальній машині, яка налаштована спеціальним чином, часто разом з іншими серверами. Тому, продуктивність цієї машини повинна бути достатньо високою.

Через особливу роль цієї машини в мережі, її також називають сервером, а машини, на яких працюють клієнтські програми, - клієнтами [4]. Серед усіх моделей розподілу даних найбільшою популярністю користується модель Model-View-Controller.

Ця модель має інтерфейс та керуючу бізнес-логіку, які поділяються на три окремі компоненти:

- модель надає дані і реагує на команди контролера, змінюючи свій стан;
- подання відповідає за відображення даних моделі користувачеві, реагуючи на зміни моделі;
- контролер інтерпретує дії користувача, сповіщаючи модель про необхідність змін.

1.4 Вибір СУБД

База даних (БД) може містити різноманітну інформацію, таку як дані про користувачів, створені конкурси, фотокартки та інше. Для зручного доступу та керування цією інформацією використовують систему управління базами даних (СУБД). СУБД - це набір мовних і програмних засобів, які дозволяють створювати, управляти та спільно використовувати БД кількома користувачами. Зазвичай СУБД поділяються за використовуваною моделлю даних, наприклад, реляційні СУБД базуються на реляційній моделі даних. Системи управління базами даних допомагають організувати інформацію, а також зв'язувати БД між собою та відстежувати зміни і події. Хоча всі СУБД виконують одну основну задачу, процес їх виконання може варіюватися. Кожна СУБД має свої функції та можливості, а також різний рівень документації та технічної підтримки [5].

Важливо враховувати, що при детальному порівнянні різних популярних баз даних, необхідно оцінити, наскільки зручною для розробника і масштабованою є конкретна СУБД, а також переконатися, що вона добре інтегрується з наявними продуктами. Також, при виборі необхідно враховувати вартість системи і підтримку, яку надає розробник.

Основними пріоритетними характеристиками СУБД є:

- відповідність типу моделі даних потребам моделювання в конкретній предметній області;
- апаратно - програмна платформа;
- наявність в даній СУБД засобів розробки додатків;
- масштаби розроблюваної системи;
- наявність системи з інструментарієм для адміністрування даними;
- підтримка транзакцій на вбудованому рівні;
- вартість СУБД.

Різні бази даних мають різні алгоритми та оптимізації, що впливають на швидкість та продуктивність системи.

Якщо вже використовуються інші програмні продукти або є певні технічні вимоги з точки зору сумісності, важливо вибрати базу даних, яка є сумісною з існуючими системами. Наприклад, деякі бази даних можуть мати певні обмеження щодо підтримки певних операційних систем або програмних мов.

Оскільки на сьогоднішній день є багато різних варіантів СУБД, важливо спочатку зосередитися на тих, які є найбільш актуальними та широко використовуються. До таких СУБД належать MySQL, MongoDB, MsAccess та PostgreSQL. Для того щоб зрозуміти яку конкретно СУБД треба обрати для побудування інформаційної системи, необхідно провести комплексне порівняння найбільш популярних СУБД. Важливо відмітити, що усі бази даних треба перевіряти з останніми версіями продукту. Адже з кожним роком і з кожною новою версією СУБД стають кращими. Зокрема нові версії привносять кращу оптимізацію та нові можливості.

Під різні системи можуть бути більш корисними ті чи інші СУБД. Конкретні особливості можуть варіюватися залежно від вибраної СУБД та її призначення. У таблиці 1.2 наведена порівняльна характеристика цих систем управління базами даних. Було враховано багато аспектів, за якими проводився порівняльний аналіз баз даних. В результаті отримано найактуальніші дані про сучасні СУБД [6].

Таблиця 1.2 – Порівняння обраних СУБД

Критерії	MySQL	MongoDB	MariaDB	PostgreSQL
Модель даних	Реляційна	Документо-орієнтована система	Реляційна	Об'єктно-реляційна СУБД
Обмеження за розміром БД	Максимальний розмір таблиць в MySQL 3.22 до 4 гігабайт	На 32-бітних машинах максимальний розмір однієї БД – 2 Гб.	Максимальний розмір таблиць приблизно до 4 Гб.	Немає обмежень на максимальний об'єм бази даних

Продовження таблиці 1.2

Критерії	MySQL	MongoDB	MariaDB	PostgreSQL
Апаратна платформа	Windows, Linux	Кросплатформова	Windows, Linux, MacOS	Кросплатформова
Мова запитів	SQL	NoSQL	SQL	SQL
Кількість користувачів	Необмежена кількість користувачів	До тисячі користувачів	Необмежена кількість користувачів	Близько тисячі користувачів
Вартість	Безкоштовно	Безкоштовно	Безкоштовно	Безкоштовно
Переваги	Розповсюджується безкоштовно. Прекрасно документована. Підтримує набір спеціальних інтерфейсів	Швидкість і простота у використанні Підтримує json. Дані будь-якої структури обробляються швидко і легко	Детальна документація. Гарна швидкість роботи. Підтримка плагінів. Підтримка шифрування	Є масштабованим і здатний обробляти терабайти даних. Підтримує формат json
Недоліки	Невеликий обсяг даних для збереження	SQL не використовується в якості мови запитів. Установк а займає багато часу	Стабільність нижче, ніж в MySQL. Платна підтримка	Конфігурація може збентежити невідповідного користувача

MySQL є досить швидкою і при цьому достатньо простою у використанні. Це була одна з перших розроблених систем управління базами даних, тому в даний час вона добре розвинена, і дозволяє користувачам управляти як структурованими, так і неструктурованими даними. MySQL легко може бути використана на більшості основних платформ, включаючи Linux. Чудово

справляється зі завданнями імпорту інформації з інших типів баз даних за допомогою власного інструментарію. Двигун БД може бути розміщений в ряді середовищ, в тому числі віртуальних, фізичних і хмарних. Найсвіжіша версія, MySQL 8.3.0, пропонує обробку великих обсягів даних, збільшення швидкодії та загальної стабільності. Для даної роботи була обрана саме MySQL через її звичний синтаксис запитів, кросплатформеність та через швидке та зручне виконання операцій CRUD, що є досить важливим для кожного розробника інформаційної системи.

1.5 Вибір технологій розробки системи

Будь-яка бізнес-орієнтована інформаційна система потребує зберігання досить великої кількості інформації, в даному випадку для більш чіткого та злагодженого зв'язку між пацієнтами та самою системою. Для реалізації цих задач було вирішено використовувати ClojureScript web-технології.

Для розробки клієнтської частини:

- HTML5;
- CSS3;
- Plotly.js;
- Reagent;
- ClojureScript;
- Re-Frame.

Для розробки серверної частини та бази даних:

- Clojure;
- Reitit;
- HoneySQL;
- MySQL.

Для обох частин - клієнтської і серверної - було використано Docker, який є програмним забезпеченням для автоматизації розгортання та управління додатками в середовищах, що підтримують контейнеризацію.

ClojureScript - це діалект мови програмування Clojure, який компілюється в JavaScript. Він дозволяє розробникам писати клієнтську вебпрограмування з використанням функціонального стилю програмування. ClojureScript спрямований на функціональне програмування, що дозволяє писати код, який є чистим, без побічних ефектів, і зосереджується на обробці даних. Також цей інструмент має багато засобів для розробки, таких як REPL (Read-Eval-Print Loop), що дозволяє експериментувати з кодом в реальному часі, і компілятор, який генерує ефективний JavaScript-код [7].

Reagent - це бібліотека для розробки інтерфейсу користувача в Clojure, яка базується на React, популярній JavaScript бібліотеці для створення вебінтерфейсів. Вона надає зручні функції та компоненти для створення інтерактивних та реактивних вебдодатків.

Основна ідея Reagent полягає в тому, що весь інтерфейс користувача описується за допомогою Clojure-коду, що робить розробку більш простою та зрозумілою. Reagent використовує спеціальні функції, які називаються "компонентами", для опису структури та поведінки вебінтерфейсу.

Plotly - це бібліотека для створення інтерактивних графіків та візуалізації даних. Вона підтримує багато типів графіків, таких як лінійні, кругові, стовпчасті, розсіювальні, контурні та багато інших. Plotly може бути використана в різних мовах програмування, включаючи Python та JavaScript.

Re-Frame - це бібліотека для розробки клієнтських вебдодатків на мові ClojureScript. Вона базується на архітектурному шаблоні Flux, який використовується в React-додатках, і надає зручний спосіб управління станом додатку. Re-Frame зберігає стан додатку в одному централізованому місці, відомому як "db". Це дозволяє зручно керувати станом та спрощує відлагодження та тестування. Re-Frame використовує систему подій та ефектів для управління змінами стану. Події викликаються для оновлення стану, а

ефекти виконуються для взаємодії з зовнішнім середовищем, таким як запити до сервера або маніпуляції з DOM.

Як відомо, сучасні інтерпретатори перед виконанням компілюють JavaScript в машинний код або близько до нього, оптимізують, а вже потім виконують. І навіть під час виконання намагаються оптимізувати. Тож JavaScript працює дуже швидко, як для мови програмування з одним потоком. Усі ці твердження розповсюджуються на ClojureScript також, бо на виході він компілюється у JavaScript.

Clojure - це діалект мови програмування Lisp, який працює на віртуальній машині Java (JVM). Він був розроблений Річом Хіккі та випущений в 2007 році. Clojure поєднує простоту та експресивність Lisp з потужністю та екосистемою Java.

Однією з головних особливостей Clojure є його функціональний характер. Він пропонує багато функцій першого класу, замикань, рекурсії та імутабельних даних. Імутабельність означає, що дані у Clojure не можуть бути змінені після їх створення, що робить код безпечнішим та більш простим у розумінні [8].

Clojure характеризується такими властивостями:

- імутабельність даних;
- персистентність;
- REPL (Read, Eval, Print, Loop);
- основана на Java Virtual Machine;
- потужна система макросів;
- гомоіконність;
- префіксний синтаксис;
- динамічна на строго типізована мова.

Для управління модулями використовується пакетний менеджер NPM (Node Package Manager). Цей пакетний менеджер є досить корисним застосунком, адже в ньому знаходиться багато модулів та бібліотек, які розробник через термінал може підключити до свого проєкту. При першому підключенні потрібного модулю, NPM завантажує для розробника останню

актуальну версію пакету. Також, можна виявити модулі з вже неактуальними версіями та оновити кожен з них окремо.

Docker – програмне забезпечення для автоматизації розгортання і управління додатками в середовищах з підтримкою контейнеризації. Дозволяє «упакувати» додаток з усім його оточенням і залежностями в контейнер, який може бути перенесений на будь-яку Linux-систему з підтримкою cgroups в ядрі, а також надає середовище з управління контейнерами.

В ході проведеного аналізу було інструментів для створення інформаційної системи було обрано технології з акцентом на функціональний та декларативний підхід. В першу чергу це Clojure для серверної частини та ClojureScript для клієнтської частини. Для зберігання даних було обрано базу даних MySQL, оскільки вона є безоплатною, працює на багатьох платформах, має гарну швидкодію та зрозумілий синтаксис запитів.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Визначення та аналіз вимог до системи

IEEE - (Institute of Electrical and Electronics Engineers - Інститут інженерів з електротехніки та електроніки) - дає визначення вимогам до системи як:

- умови необхідні для конкретного користувача з метою вирішення проблем або досягнення цілей;
- умови або можливості, якими повинна володіти система або її компоненти, щоб виконати контракт або відповідати стандартам, специфікаціям та іншим документам.

Основні вимоги до програмного забезпечення можна розділити на три рівні: бізнес-вимоги, вимоги користувачів і функціональні вимоги. Крім того, кожна система має свої нефункціональні вимоги. Ці типи вимог можна представити у вигляді показаному на рис. 2.1.

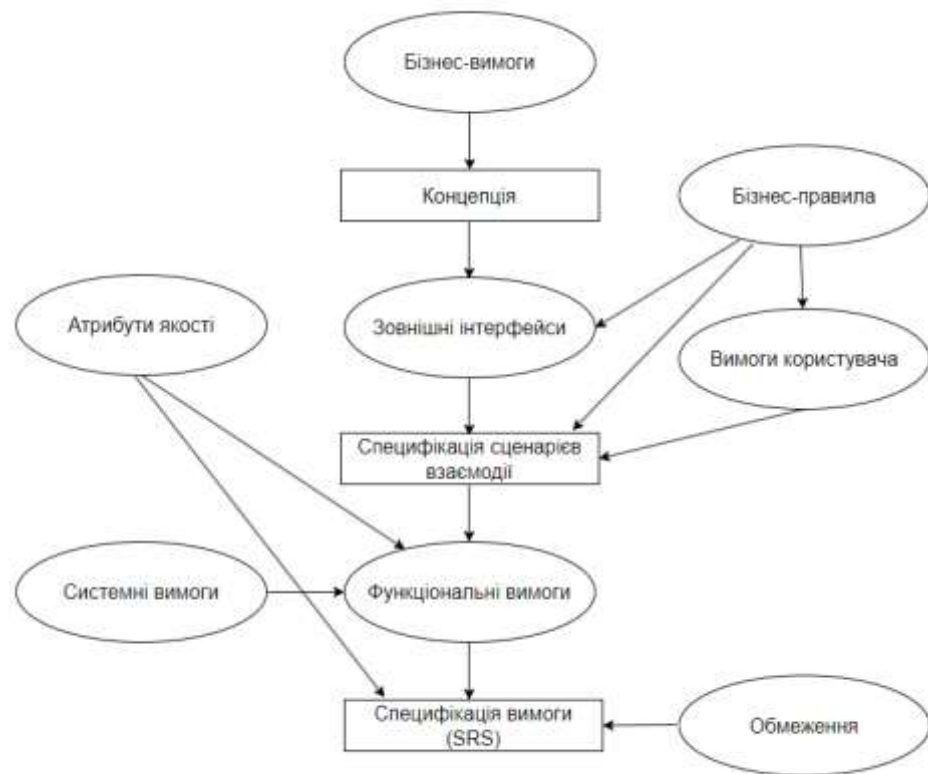


Рисунок 2.1 - Поєднання рівнів інформації для вимог

Зазвичай, в основі всіх бізнес-вимог лежить високорівнева мета компанії або клієнтів системи. Як правило ці вимоги формулюються фінансувальниками розробки інформаційної системи, клієнтами системи, менеджерами користувачів та відділом маркетингу. У цій документації обов'язково описуються цілі, які організація прагне досягти та проблеми, які потрібно вирішити, а також пояснюється, чому саме ця система є необхідною.

У вимогах користувачів описуються конкретні цілі та завдання, які система повинна виконувати для користувачів. Для представлення цих вимог можна використовувати різні методи, такі як варіанти використання, сценарії або таблиці "подія - відгук". Цей документ визначає, які можливості системи будуть доступні клієнтам.

Функціональні вимоги, як правило, визначають функціональні можливості інформаційної системи. Іноді їх називають умовами поведінки, оскільки вони містять положення, які починаються з "повинен" або "повинна".

Функціональні вимоги документуються в специфікації вимог до ПЗ (software requirements specification, SRS), де описується так повно, як необхідно, очікувана поведінка системи [9].

Системні вимоги є високорівневими вимогами до конкретного продукту. При згадуванні системи мається на увазі програмне забезпечення або підсистеми програмного забезпечення та обладнання. Людина також є частиною системи, тому деякі функції системи можуть також впливати на людей. Бізнес-правила включають корпоративні політики, урядові регуляції, промислові стандарти та обчислювальні алгоритми.

Бізнес-правила не є вимогами до програмного забезпечення, оскільки вони знаходяться поза межами будь-якої системи програмного забезпечення. Однак, вони часто ставлять спеціальні обмеження, визначаючи, хто може диктувати, які функції повинна мати система, що підпорядковується відповідним правилам. Також важливо відмітити, що гарно спроектована система повинна бути надійною і доступною для користувачів весь час. Також добре якщо система здатна до гнучкого масштабування.

Нефункціональні вимоги визначають цілі і характеристики якості. Характеристики якості є додатковим описом функцій продукту, який виражається через опис його важливих характеристик для користувачів або розробників.

До таких характеристик належать:

- легкість і простота використання;
- легкість переміщення;
- цілісність;
- ефективність і стійкість до збоїв;
- зовнішні взаємодії між системою і зовнішнім світом;
- обмеження дизайну і реалізації.

Вимоги до інформаційної системи наведено у таблиці 2.1.

Таблиця 2.1 – Вимоги системи аналізу, обробки та візуалізації даних.

Тип вимоги	Опис вимоги
Бізнес-правило	Безперебійна робота інформаційної системи
Функціональні вимоги	- користувачі в системі: медичний науковець, адміністратор; - генерація різноманітних графіків по даним з бази; - валідація користувачів; - відображення таблиць з сортуванням та фільтрами; - наявність адаптивного інтерфейсу.
Системні вимоги	Будь який сучасний браузер з підтримкою CSS3, HTML5, JavaScript.
Вимоги користувачів	Медичний науковець: - доступ до системи за особистим логіном та паролем; - завантаження CSV таблиць в систему; - можливість бачити список пацієнтів; - можливість будувати генетичну теплову мапу; - можливість відслідковувати перевезення біологічних матеріалів;
Вимоги користувачів	Медичний науковець: – генерація звітів по наявним графікам; – експорт даних з графіків та таблиць у вигляді зображення чи CSV таблиці. Адміністратор: – вхід до панелі адміністратора; – зміна важливих параметрів системи; менеджмент користувачів у системі.
Нефункційні вимоги	Швидкий відгук сторінок інформаційної системи навіть при повільному інтернеті.
Обмеження	Не всі графіки підтримують можливість 3D відображення.

2.2 Розробка специфікації вимог SRS

Відповідно до методики складання специфікації вимог до програмного забезпечення Інституту Інженерів з Електротехніки та Радіоелектроніки (Стандарт IEEE 830 – 1998), правильно складена специфікація вимог повинна виконувати наступні завдання:

- чітко дати зрозуміти постачальникам програмного забезпечення, що саме очікує замовник;
- детально описувати замовникам програмного забезпечення, яке конкретно рішення вони хочуть отримати;
- розробити макет стандартної специфікації програмного забезпечення (SRS) для власних організацій;
- створювати додаткові допоміжні документи, такі як контрольний лист для перевірки якості SRS або довідник упорядника SRS;
- визначити формат і зміст конкретних специфікацій вимог до програмного забезпечення.

Система аналізу, обробки та візуалізації інформації на основі методології прецизійної медицини призначена для медичних науковців. За допомогою цієї системи науковці можуть будувати графіки та складні таблиці для отримання детальної інформації по стану здоров'я кожного пацієнта.

Головне завдання системи полягає у наданні медичним науковим співробітникам можливість отримувати добре візуалізовану медичну інформацію для подальшого аналізу та медичних висновків.

Специфікація вимог до програмного забезпечення.

Введення.

Призначення.

Специфікація вимог до ПЗ описує функціональні та нефункціональні вимоги до інформаційної системи.

Інформаційна система являє собою сайт, який надає можливість медичним науковцям будувати складні графіки та таблиці, генерувати звіти та отримувати корисну інформацію на основі якої можна зробити різноманітні висновки про той чи інший стан конкретного пацієнта.

Область дії.

Інформаційна система, що проєктується, отримала назву “Science For Medicine”. До основної мети програмного продукту належить забезпечити науковців та медичних аналітиків корисною медичною інформацією яка добре та зрозуміло візуалізована.

Визначення, акроніми та скорочення.

Software Requirements Specification (SRS) - специфікація вимог для програмної системи - це повний опис поведінки системи що розробляється.

Hyper Text Markup Language, version 5 (HTML5) - мова для впорядкування та подання вмісту всесвітньої павутини.

Cascading Style Sheets (CSS3) - каскадні таблиці стилів третього покоління.

Clojure - сучасний діалект класичного Lisp.

ClojureScript – реалізація Clojure з компіляцією в JavaScript.

СУБД - система управління базами даних.

MVC - патерн проєктування, Model-view-controller.

Публікації.

Аналогічні системи розглянуті за допомогою інтернет - ресурсів:

www.imatrix.com, www.inotivco.com, www.careme.health.

Короткий огляд.

В цій частині SRS зазначається повноцінний опис програмного продукту, який включає до себе контекст продукту, його функції, характеристики користувача, певні обмеження, а також допущення та залежності. Сама по собі структура специфікацій вимог складається з вступу та загального опису, до якого вже входить більш повний опис функцій та обмежень.

Повний опис та контекст продукту.

Інформаційна система аналізу, обробки та візуалізації даних є окремим незалежним продуктом, який не є частиною іншого продукту.

Програмний продукт є другим рівнем архітектури і повинен мати зв'язок з базою даних. Сервер повинен обмінюватися запитами та відповідями з базою даних.

Функції продукту.

Для того щоб користувач мав змогу використовувати усі можливості системи, необхідно буди зареєстрованим в системі та пройти аутентифікацію, при цьому адміністратор має більші повноваження.

До них відносяться:

- перегляд всіх користувачів та їх менеджмент;
- редагування важливих параметрів системи.

Характеристики користувача.

До загальних характеристик клієнта програмного продукту можна віднести мінімальні знання по роботі з комп'ютером або будь-яким іншим пристроєм з якого користувач виходить у мережу Інтернет. Обов'язковими є розуміння контексту розміщеної медичної інформації у системі.

До характеристик адміністратора відноситься розуміння правил по розміщенню контенту інформаційної-системи, впевнена робота з операційними системами Windows чи Linux, знання англійської мови та швидке орієнтування по основним розділам системи.

Обмеження.

Взаємодіяти з основним контентом системи можуть лише зареєстровані адміністратором медичні науковці. Також для того щоб мати гарний досвід користування інформаційною системою потрібен сучасний браузер та обов'язкове підключення до інтернету.

Не всі графіки підтримують можливість 3D відображення.

Специфічні вимоги.

Апаратні інтерфейси – наявність будь якого пристрою з можливістю виходу в Інтернет з актуального браузера.

Інтерфейси зв'язку.

Система повинна за допомогою Інтернет – з'єднання зберігати дані кожного зареєстрованого користувача, при цьому усі паролі повинні шифруватися за допомогою сучасних методів шифрування, наприклад за допомогою JWT.

Вимоги до користувачів:

- вхід до особистого кабінету;
- еенерація графіків;
- генерація таблиць;
- створення звітів у різних форматах;
- моніторинг відправлень з біологічними матеріалами;
- редагування свого профілю;
- перегляд всіх користувачів;
- менеджмент користувачів;
- редагування важливих параметрів системи.

2.3 Моделювання системи

Сучасні проекти все частіше використовують методи об'єктно-орієнтованого аналізу, дизайну та розробки. У цих проектах об'єкти відображають об'єкти реального світу в предметній області. Ці об'єкти мають свої екземпляри, які називаються класами. В класах містяться атрибути, що представляють дані, а також дії, які використовуються з цими атрибутами. Діаграма класів є графічним зображенням цих класів, які створюються під час об'єктно-орієнтованого аналізу.

Мовою моделювання об'єктно-орієнтованого підходу є Unified Modeling Language (UML). UML є графічною мовою для опису об'єктів і процесів в

розробці програмного забезпечення, моделюванні бізнес-процесів, системному проектуванні та візуалізації організаційних структур.

Діаграма об'єктів показує стан модельованої системи в певний момент часу, включаючи екземпляри класів (об'єкти) з їх поточними значеннями атрибутів і зв'язками між ними [10].

Діаграма прецедентів, у рамках UML, відображає відношення між акторами та прецедентами в системі. Це графічне зображення, що складається з акторів, прецедентів (варіантів використання) та їх зв'язків, а також відношень серед прецедентів та узагальнення між акторами. Діаграми прецедентів використовуються для відображення елементів моделі варіантів використання.

На рисунку 2.2 зображена діаграма прецедентів, на якій можна побачити взаємодію системи з медичним науковцем та адміністратором.

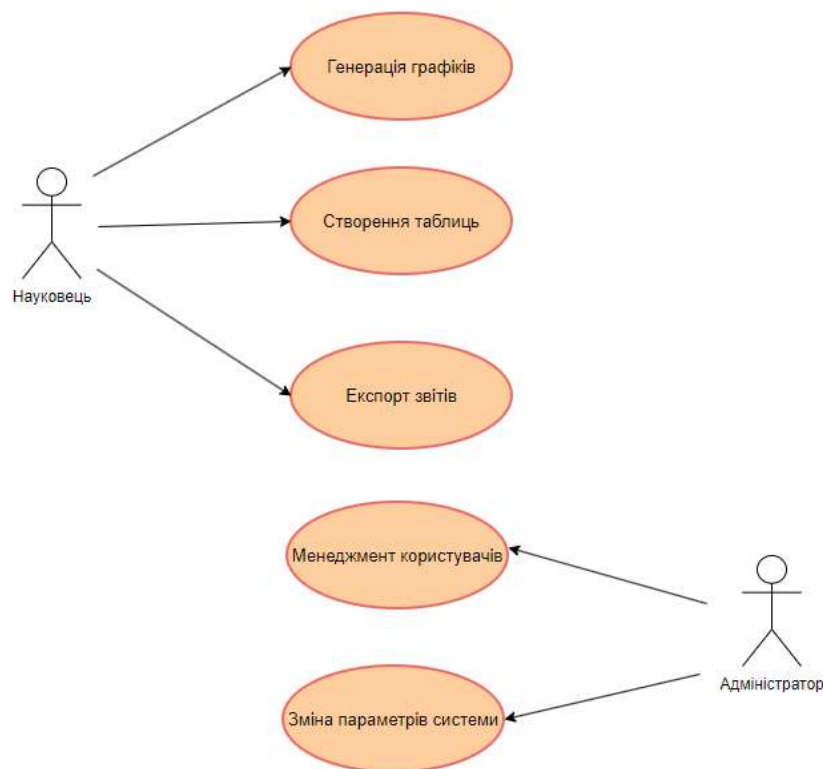


Рисунок 2.2 – Діаграма прецедентів

UML оперує таким терміном, як “діяльність”. Цей термін представляє сукупність деяких обчислень, що виконуються автоматично. В результаті

відбується очікуваний результат або дія. Тобто діаграма діяльності демонструє певну логіку і послідовність переходів з однієї дії в іншу. В такому випадку, відбувається зміна станів системи, або повернення до деякого значення.

Прецедент П1 – Реєстрація та вхід користувача.

Головна послідовність: адміністратор відкриває сайт інформаційної системи та переходить до адмін-панелі на сторінку реєстрації нового користувача. Далі адміністратор вводить допустимі дані, тобто логін та пароль. Після вдалого створення користувача відповідні дані записуються у базу даних.

Якщо користувач входить в особистий акаунт, то на час сесії створюється JSON Web Token. Якщо користувач виходить з особистого кабінету, то токен при цьому знищується (рис 2.3).

Альтернативна невдала послідовність: якщо користувач ввів дані, які недопустимі в системі, то вхід не відбудеться і система проінформує про це користувача (рис 2.4).

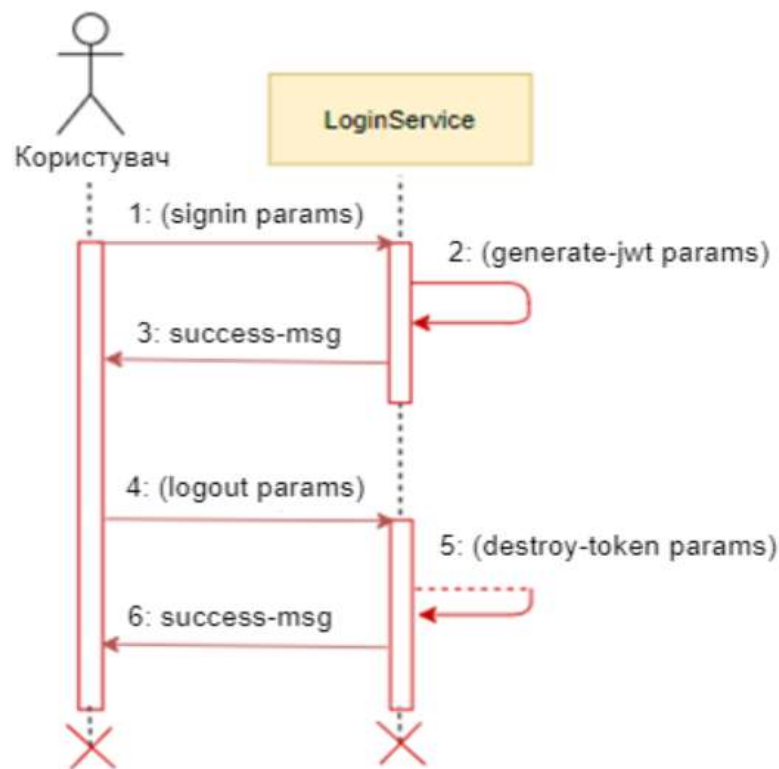


Рисунок 2.3 – Аутентифікація користувача

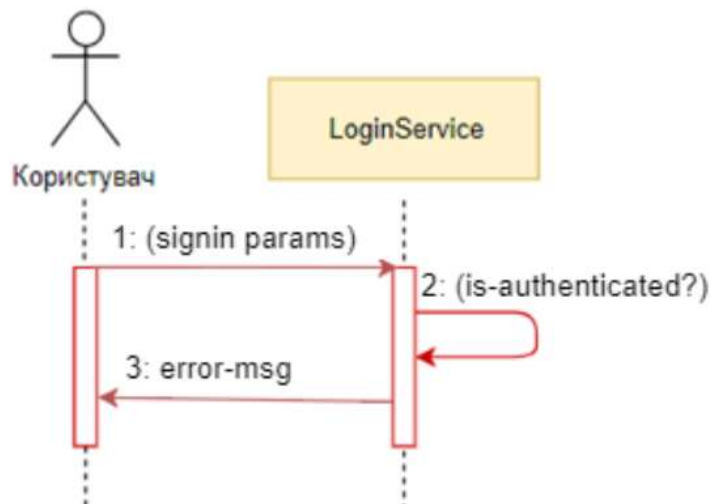


Рисунок 2.4 – Помилка при реєстрації

Прецедент П2 – Генерація графіку.

Головна послідовність: медичний науковець вже має акаунт і бажає сгенерувати графік за певним набором даних (рис 2.5).

Альтернативна невдала послідовність: якщо дані підібрані невірно або пусті, то графік покаже помилку.

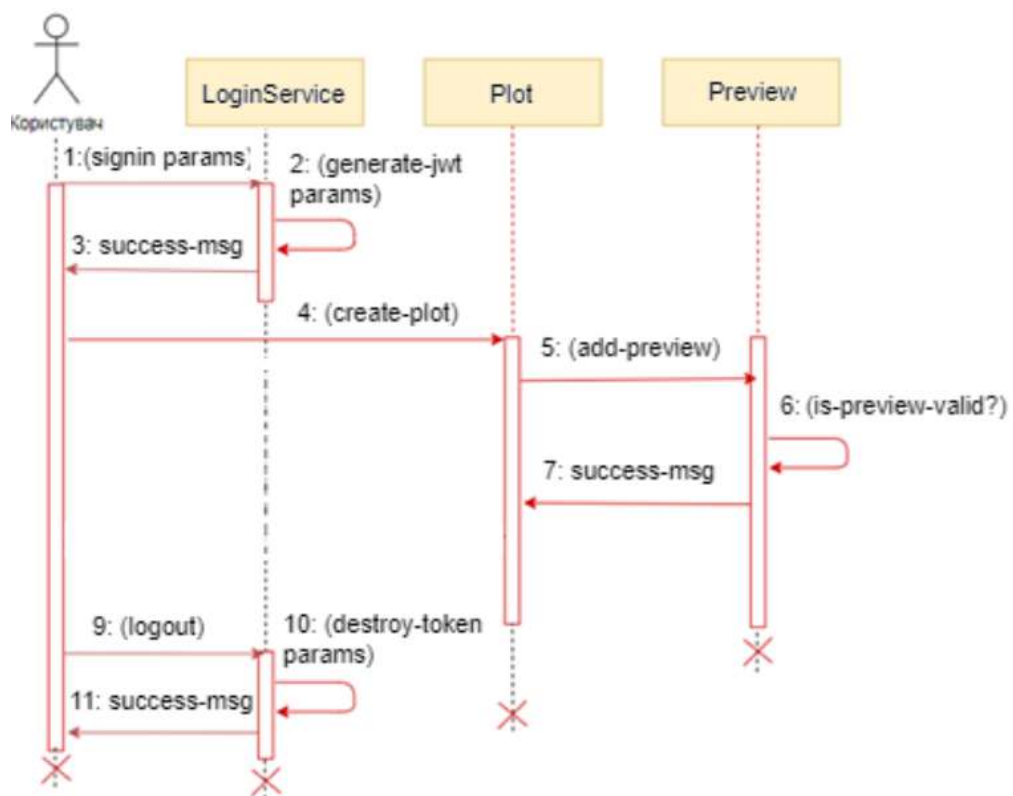


Рисунок 2.5 – Успішна генерація графіку та звіт по ньому

Прецедент ПЗ – Моніторинг відправлень з біологічними матеріалами.

Головна послідовність: користувач вже пройшов аутентифікацію та авторизацію в системі. Тепер він може переглянути таблицю зі статусом посилок з біологічними матеріалами від різних поштових провайдерів, а також подивитися на мапі де зараз знаходиться та чи інша посилка.

Відтворення списку посилок (рис. 2.6).

Альтернативна невдала послідовність: якщо API поштового сервісу не відповідає або повертає помилку, тоді користувача буде проінформовано про помилку. Таке повідомлення повинно приходити у реальному часі за допомогою сокетів які тримають зв'язок між клієнтською частиною та серверною. Таким чином можна отримати своєчасне інформування.

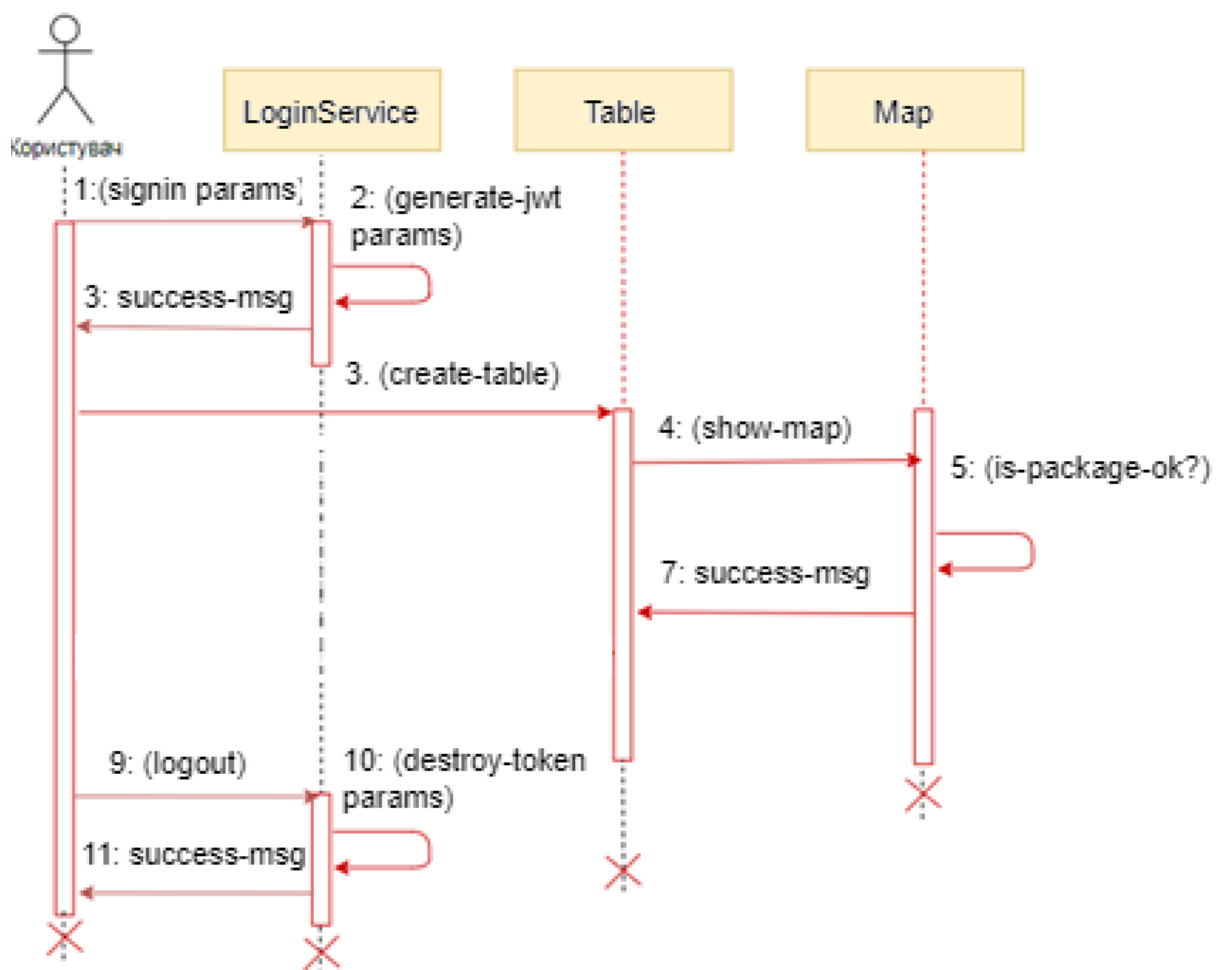


Рисунок 2.6 – Успішне створення таблиці

2.4 Розробка інтерфейсу користувача

Інтерфейс користувача є спеціальним засобом, який дозволяє користувачу взаємодіяти з програмним забезпеченням. Це набір інструментів, які відповідають за отримання та обробку інформації і повинні бути зручними для кінцевого користувача. У цій роботі був використаний інтерфейс WIMP (Window Image Menu Pointer), який використовує графічні вікна для взаємодії з програмою. Зовнішній вигляд графічних вікон може бути змінений. Цей тип інтерфейсу використовує фізичний пристрій для керування курсором і надає користувачеві необхідну інформацію у вигляді вікон [11].

Таким чином, для інформаційної системи були розроблені послідовні прототипи віконного дизайну, а на рисунку 2.7 можна побачити прототип головної сторінки.



Рисунок 2.7 – Прототип головної сторінки системи

На головній сторінці знаходяться верхнє меню сайту, в якому розташована назва проєкту, бокова меню-панель та кнопки “Довідка”, “Повідомлення” та

“Налаштування”. Трохи нижче розташовані пункти для перегляду важливої системної інформації яку користувач сам додав на свою голову сторінку. Варто зауважити, що також на головній сторінці можуть бути присутні віджети з графіками які користувач може самостійно налаштовувати та міняти за бажанням.

Бокова меню-панель може ховатися для того щоб не займати зайвий простір на головному екрані. Це є необхідністю для того щоб забезпечити адаптивний інтерфейс з правильним відображенням та всіх типах пристроях. При цьому розмір екрану вже не грає важливою ролі, адже CSS правила покривають всі можливі випадки.

У вкладці “Shipment Tracker” можна побачити таблицю як відображає список усіх посилок з біологічними матеріалами і статус кожної з посилок.

Прототип сторінки з табличним компонентом зображено на рисунку 2.8.

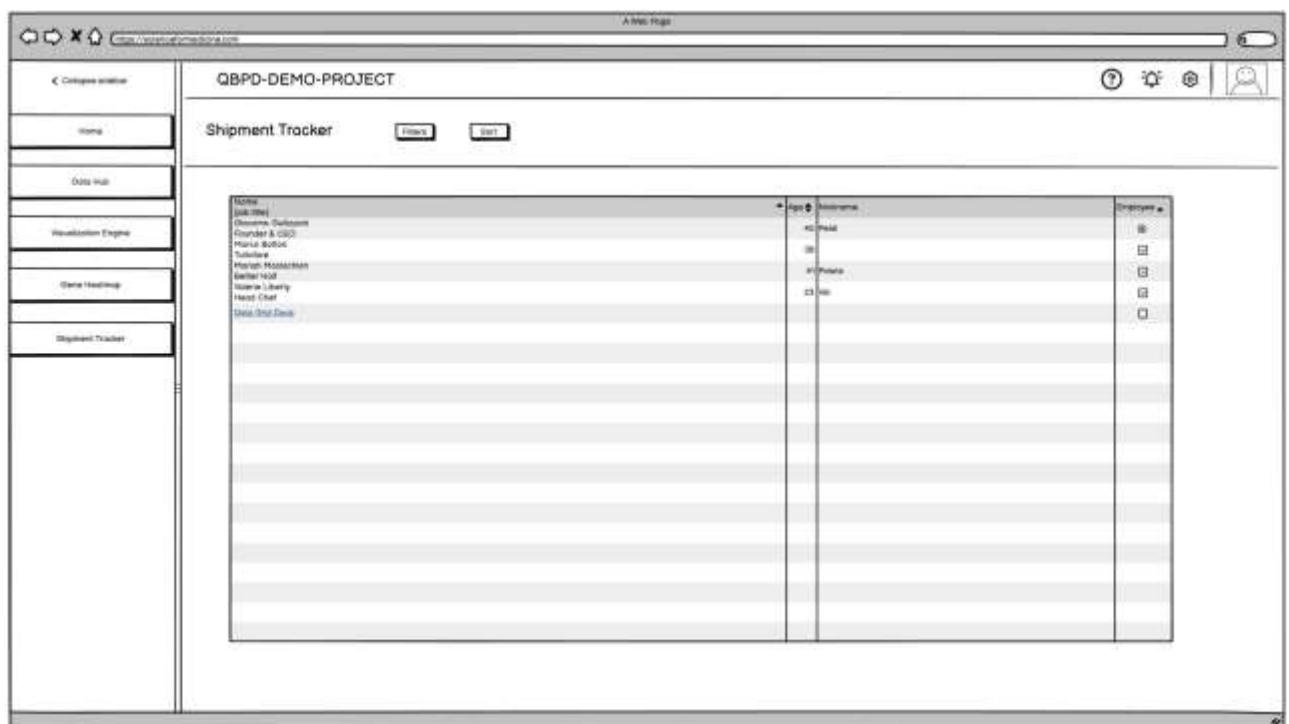


Рисунок 2.8 – Прототип сторінки з табличним компонентом

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Схема роботи проєктованої системи

Патерн програмування розуміють як шаблон або зразок, який можна використовувати багато разів. Його частини створюються для того, щоб можна було повторно використовувати код програми.

Model-View-Controller (MVC) - це схема, яка розділяє дані програми. Вона особливо корисна для користувацького інтерфейсу і керуючої логіки. MVC складається з трьох окремих компонентів: моделі, представлення і контролера, що дозволяє редагувати кожен компонент окремо [12].

Модель (Model) описує і надає дані, реагуючи на команди контролера. Це призводить до зміни стану моделі і відображення.

Представлення (View) відповідає за відображення даних моделі користувачеві, реагуючи на зміни моделі.

Контролер (Controller) розуміє дії користувача і сповіщає модель про необхідність оновлення. Варто зазначити, що реалізація MVC може відрізнятися, і не існує загальноприйнятого визначення щодо розташування бізнес-логіки. Це може бути реалізовано по-різному в залежності від фреймворку або бібліотеки. Також не вказано, де повинна знаходитися перевірка введених користувачем даних, але зазвичай вона розташовується в контролері або моделі, оскільки це впливає на безпеку даних.

Схема програмування MVC дозволяє розробникам створювати більш організовані та легкозрозумілі програми, оскільки розділення логіки на окремі компоненти спрощує тестування, підтримку і розширення коду. Вона також сприяє повторному використанню коду, оскільки модель, представлення і контролер можуть бути використані в різних проєктах [13].

Схематичне зображення MVC можна побачити на рисунку 3.1.

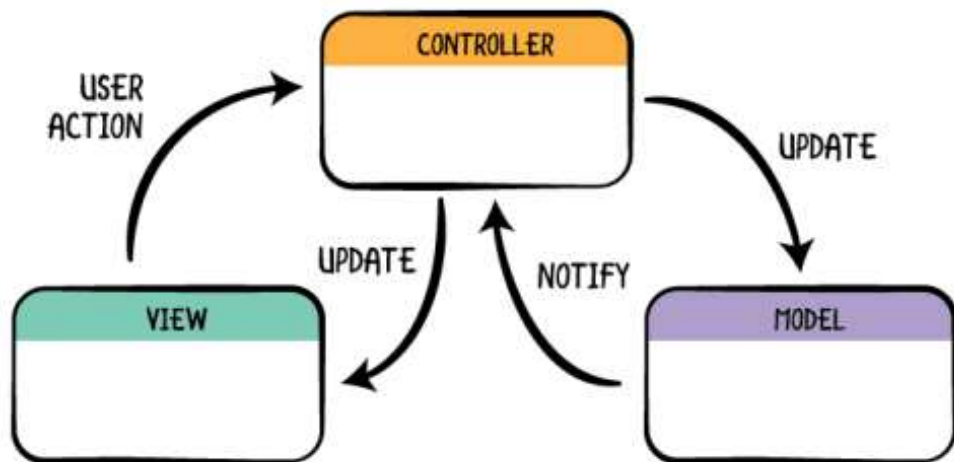


Рисунок 3.1 – Схема взаємодії компонентів MVC

Модель визначає та керує даними та методами їх обробки. Вона включає запити до бази даних та перевірку їх правильності. Модель не залежить від способу відображення даних та не взаємодіє безпосередньо з контролером. Таким чином, модель не має прямого контакту з користувачем, а лише надає доступ до даних та їх управління. Модель відповідає на запити, змінюючи свій стан. Представлення, у свою чергу, реагує на отримання необхідних даних з моделі та передає їх користувачеві, який вже бачить оновлені дані в інтерфейсі.

Важливо зазначити, що в MVC модель і представлення не взаємодіють безпосередньо між собою, а взаємодіють через контролер. Це дозволяє зберігати модульність і розділити відповідальності між компонентами програми.

3.2 Створення контейнеризації системи

Docker є програмним забезпеченням, яке дозволяє автоматично розгорнути та керувати додатками в середовищах, що підтримують віртуалізацію та контейнеризацію. Ця технологія є незамінною, коли проєкт готовий і потрібно передати його замовнику [14]. За допомогою Docker це можна зробити досить

просто. Замість довгих інструкцій для запуску додатку, де потрібно вказати, які скрипти запускати, можна просто підключити додаток до контрольованого середовища, яке ефективно керує серверними ресурсами. Docker створює власну операційну систему, на якій працюють його процеси, але ці процеси виконуються на фізичному сервері. Це називається контейнеризацією. Спочатку створюється docker-образ, а потім за допомогою директиви RUN можна виконувати будь-які команди в Linux. Крім того, є можливість копіювати файли в образ за допомогою директиви COPY. Приклад docker-файлу можна побачити на лістингу 3.1.

Лістинг 3.1 – Приклад docker-файлу

```
FROM=clojure:openjdk-16-lein-2.9.7
RUN ln -s /usr/local/openjdk-16/bin/* /usr/local/bin/
RUN apt-get update && \
    apt-get install -y \
    fonts-dejavu \
    libfreetype6 \
    clj-kondo \
    libfontconfig1 \
    fonts-dejavu-core \
    fontconfig-config --no-install-recommends
COPY ../rclone /usr/local/bin/rclone
WORKDIR /home/docker/backend2
EXPOSE 3020
RUN apt-get update && \
    apt-get install -y \
    fonts-dejavu \
    libfreetype6 \
    clj-kondo \
    libfontconfig1 \
    fonts-dejavu-core \
    fontconfig-config --no-install-recommends
COPY ../rclones /usr/local/bin/rclones
```

На лістингу 3.2 зображено створення так званих контейнерів, в яких будуть працювати окремі компоненти додатку.

Лістинг 3.2 – Створення docker контейнерів

```
version: '3.4'
services:

  backend:
    build:
      context: ./backend

    depends_on:
      - db-dev
    volumes:
      - ./client:/client
      - /client/node_modules

  healthcheck:
    test: ["CMD", "curl", "-f", "http://backend:3020"]
    start_period: 120s
    interval: 20s
    timeout: 5s
    retries: 10

  ports:
    - "3020:3020"

  frontend-dev:
    build:
      context: ./frontend

    environment:
      CLOJURESCRIPT_ENV: development
    ports:
      - "3440:3440"
    depends_on:
      - backend
    volumes:
      - /var/www/html/images:/var/www/html/images
      - ./frontend:/frontend
      - /frontend/node_modules

  db-dev:
    image: mysql:5.7
    environment:
      - MYSQL_PASSWORD=password
      - MYSQL_USER=postgres
      - MYSQL_DB=brandex-dev
    ports:
      - "3406:3406"
```

3.3 Реалізація серверної частини системи

Серверна частина була створена з використанням мови програмування Clojure. Clojure базується на функціональному підході до програмування, що дозволяє писати код, який є більш декларативним та зрозумілим. Це сприяє швидкості розробки та полегшує підтримку коду [15].

До основних переваг Clojure належать:

- маршрутизація запитів користувача з підтримкою проміжного шару;
- можливість створити та запустити елементарний сервер;
- розбір тіла запиту;
- візуалізація представлення.

Підключення основних модулів для створення серверу можна побачити на лістингу 3.3.

Лістинг 3.3 – Підключення основних модулів та створення серверу.

```
(ns path.core
  (:require
    [clojure.edn]
    [clojure.tools.logging :as log]
    [clojure.tools.nrepl.server :as nrepl] ; Compojure & ring
    [compojure.core :refer [context defroutes GET POST]]
    [compojure.route :as route] ; Path modules
    [ring.adapter.jetty :refer [run-jetty]]
    [qb.server.middleware.exception :refer [wrap-exception]]
    [qb.server.session-manager :as sm]))

(defn start-server [params]
  (if (:config-path params)
    (conf/update-config (:config-path params))
    (conf/update-config))
  (ss/setup-framework-hooks!)
  (reset! jetty-server
    (run-jetty #'app {:port [:framework- port] 3020}
      :join? false
      :async? true
      :max-threads 200
      :request-header-size 32768
      :response-header-size 32768)))
```

У цих модулях можна виділити наступні фрагменти:

- `ring.adapter.jetty` – підключення модулю для створення веб-серверу;
- `compojure.route :as route` – імпорт роутера, який включає в себе усі кінцеві точки;
- `qb.server.middleware.exception` – підключення самописного ловця помилок;
- `clojure.tools.logging` – підключення модулю `logging` для логування важливої інформації з `request`;
- `clojure.tools.nrepl.server` – підключення сокет-серверу для надання доступу до REPL;
- `qb.server.session-manager` – модуль для створення сесії користувача;
- `clojure.edn` – набір функцій який відповідає за перетворення JSON у рідний для Clojure формат даних EDN.

Файл конфігурації бази даних можна побачити на лістингу 3.4.

Лістинг 3.4 – конфігураційний файл бази даних.

```
{
  "development": {
    "username": "mysql",
    "password": "password",
    "database": "scienceformedicine-dev",
    "host": "db-dev",
    "dialect": "mysql"
  },
  "production": {
    "username": "mysql ",
    "password": " password ",
    "database": "scienceformedicine-prod",
    "host": "db-prod",
    "dialect": "mysql"
  }
}
```

Тут можна зауважити, що використовуються різні параметри для `development` та `production build`. У полях `"username"` та `"password"` вказуються дані, за якими БД зможе аутентифікувати розробника. Однак, цього недостатньо для початку роботи з базою даних MySQL. Для підключення до БД бібліотеки

Sequelize, необхідно передати раніше створені параметри до екземпляра класу Sequelize (лістинг 3.5).

Лістинг 3.5 – Підключення БД до Sequelize.

```
(def fs (require "fs"))
(def path (require "path"))
(def Sequelize (require "sequelize"))
(def basename (.basename path __filename))
(def env (or (.. js/process -env -NODE_ENV) "development"))
(def configPath
  (if (= env "production")
    (.join path __dirname ".." ".." "src/config/config.json")
    (.join path __dirname ".." "config/config.json")))
(def config (aget (require configPath) env))
(def db #js {})
(def sequelize nil)
```

Наступним кроком необхідно описати модель сутностей системи. Розглянемо одно з таких сутностей написану з використанням Sequelize на прикладі моделі користувача (лістинг 3.6).

Лістинг 3.6 – Опис моделі користувача

```
'use strict'
module.exports = (sequelize, DataTypes) => {
  def user = sequelize.define('User', {
    id: {
      allowNull: false,
      autoIncrement: true,
      primaryKey: true,
      type: DataTypes.INTEGER
    },
    firstName: {
      type: DataTypes.STRING,
      allowNull: false,
      validate: {
        notEmpty: true
      }
    },
    lastName: {
      type: DataTypes.STRING,
      allowNull: false,
      validate: {
        notEmpty: true
      }
    },
    nickName: {
```

```

        type: DataTypes.STRING,
        allowNull: false,
        unique: true,
        validate: {
            notEmpty: true
        }
    },
    email: {
        type: DataTypes.STRING,
        allowNull: false,})

```

По суті в даній структурі описуються поля користувача, які будуть наявні в базі, їх типи та початкова перевірка. Після усіх налаштування описаних вище, вже можна створювати подальший контролер який буде оброблювати певні дії стосовно користувача.

Для прикладу розглянемо як виглядає API для генерації графіку. Перед обробкою даних які прийшли на сервер необхідно обов'язково зробити додаткову перевірку усіх параметрів за допомогою бібліотеки `clojure.spec` [16].

Генерацію графіку користувачем можна побачити на лістингу 3.7.

Лістинг 3.7 – Генерація графіку користувачем

```

(defn generate-chart
  "Calls the appropriate function for generating chart
  widget"
  [chart-params]
  (log/info "Begin chart request")
  (let [{:keys [module module-gen chart-type chart-template
               chart-draw
               chart-gen chart-data chart-uuid]}
        (prepare-chart-data)
        opts {:uuid chart-uuid}
        widget-type (or chart-template chart-type)
        chart-output
        (case chart-type
          "pie-bar" (chart-gen chart-data opts)
          "line-plot" (chart-gen chart-data opts)
          "box-violin" (chart-gen chart-data opts)
          "heatmap" (chart-gen chart-data opts)
          "clinical-tables" (chart-gen chart-data opts)
          (as-> chart-data $
            (module-gen widget-type $)
            (update-in $ [:result :visuals]
              (fn [elements]
                (->> elements

```

```

        (map-indexed
          (fn [idx item]
            (assoc item :draw chart-draw))))))
      (:result $))))
(log/info "End chart request")

(defn- chart [{:session :session body :body} res raise]
  (let [params (get-in body [:params])
        spec (chart-specs/chart-spec params)
        params (if spec
                  (validation/validate spec params))

```

Варто пильно звернути увагу на те, що перед тим як намалювати графік, необхідно визначити тип віджету який буде цей графік розміщувати та тип графіку який надійшов з клієнтської частини.

Згідно з специфікацією вимог, користувач повинен мати можливість переглядати необхідну інформацію в табличному компоненті. Створення таблиці зображено на лістингу 3.8.

Лістинг 3.8 – Генерація табличного компонента користувачем

```

(ns path.tables.track-list
  (:require [path.tables.core.table-definitions :refer
    [define-table create-table]]))
(defmethod define-table :tracking_data
  [table-id {:keys [filters columns view] :as attrs}]
  (let [active-table? (= view "active")
        columns-attributes
          (or (= field-key "trackingNum")
              (and (= field-key "serviceType")
                    (not active-table?)))
        [field-key label (assoc opts
                                :type "custom"
                                :referTo field-key
                                :className "dt-link")]

        :else [field-key label opts]])
    columns)]
  (create-table
    table-id
    {:data-source          {:filters filters}
     :table-attributes      {:id-column      "id"
                           :selection-type "multiple"}
     :column-attributes     columns-attributes
     :datatables-attributes {:paging      true
                           :scrollY      "520px"
                           :scroller     false}}))

```



```

:icon "gear"}

["" {:feature :settings
:label "Settings"
:name :settings}]

["/user" {:breadcrumbs [:settings/user]
:feature :user-settings
:label "Profile information"
:name :settings/user}]

["/notifications" {:breadcrumbs [:settings/notifications]
:feature :notification-settings
:label "Notifications"

:name :settings/notifications}]
["/transfer-options" {:breadcrumbs [:settings/transfer-
options]
:feature :transfer-options-settings
:label "Transfer Options"
:name :settings/transfer-options}]]

["/not-available" {:name :not-available
:breadcrumbs []
:component not-available/page
:controllers []}]]))

```

Після створення `Reitit` та всіх необхідних компонентів, необхідно використовувати вибраний MVC патерн на основі `React`. Це вимагає описати `Dispatch Actions`, які реагують на зміни, внесені користувачем у зовнішнє середовище. Параметри з `Actions` передаються до проміжного обробника, а потім виконується запит до серверу (див. лістинг 3.10).

Лістинг 3.10 – Приклад обробників `Dispatch Actions`

```

(ns qb.ui.components.report-modal.events
  (:require [qb.data.api :as api]
            [qb.ui.app.interceptors :as interceptors]
            [qb.ui.components.chart.core :as core]
            [qb.ui.components.report-modal.utils :as rmu]
            [re-frame.core :as rf]))

(rf/reg-event-fx
  :report-modal/close

```

```

[interceptors/current-solution]
(fn [{db :db} [_]]
  (-> {:db db, :fx []}
    (assoc-in [:db :report-modal :report-modal/opened?])
    (update-in [:db :report-modal] dissoc :report)
    (update-in [:db :report-modal] dissoc :report))
(rf/reg-event-fx
 :report-modal/change-figure
 [interceptors/current-solution]
 (fn [{db :db} [_ [chart-id value] mode]]
   (let [path (if (= mode "chart")
                  [:db :charts (keyword chart-id) :report]
                  [:db :report-modal :report-modal/dashboard])]
     (-> {:db db, :fx []}
       (assoc-in path (str chart-id "--" value))))))
(rf/reg-event-fx
 :report-modal/change-figure
 [interceptors/current-solution]
 (fn [{db :db} [_ [chart-id value] mode]]
   (let [path (if (= mode "chart")
                  [:db :charts (keyword chart-id) :report]
                  [:db :charts (keyword chart-id) :report])
         ]
     :report-modal/set-element-param
     [interceptors/current-solution])

```

Також важливо відмітити що ці Dispatch Actions грають важливу роль у Re-Frame середовищі. Адже вони отримують інформацію після якоїсь дії із зовнішнього світу та оброблюють цю інформацію з подальшими змінами в загальному сховищі даних для того щоб усі компоненти на клієнтській частині змогли підписатися на оновлення та вчасно їх отримати [18].

Варто зауважити що Dispatch Actions відпрацьовують у тому випадку коли користувач виконав якусь дію. Це може бути натискання кнопки, заповнення форми або відкриття якое зображення.

Вартий уваги той факт, що глобальний стан забороняється модифікувати напряму. Аби React компоненти оновлювалися він глобального стану, необхідно використовувати Reducers. Інакше пряма модифікація стану може призвести до дивних помилок, які потім важко знайти та виправити.

На цей випадок існують так звані “чисті” функції. Вони не модифікують стан напряму, проте можуть повертати нове значення. Чим більше подібних функцій, тим менше шанс зустріти некоректну поведінку.

4 ДОСЛІДЖЕННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1 Дослідження швидкодії та надійності системи

Для вивчення загальної швидкодії системи було вибрано кілька методів вимірювання швидкості переходу між сторінками. Перший метод включає проведення тестів швидкодії та загальної якості системи за допомогою Google Audit системи. Для порівняння швидкості, інформаційна система, що була створена, була порівняна з сайтом "iMatrix".

На рисунку 4.1 показані аналітичні дані щодо швидкодії, оптимізації, найкращих практик та SEO-оптимізації.

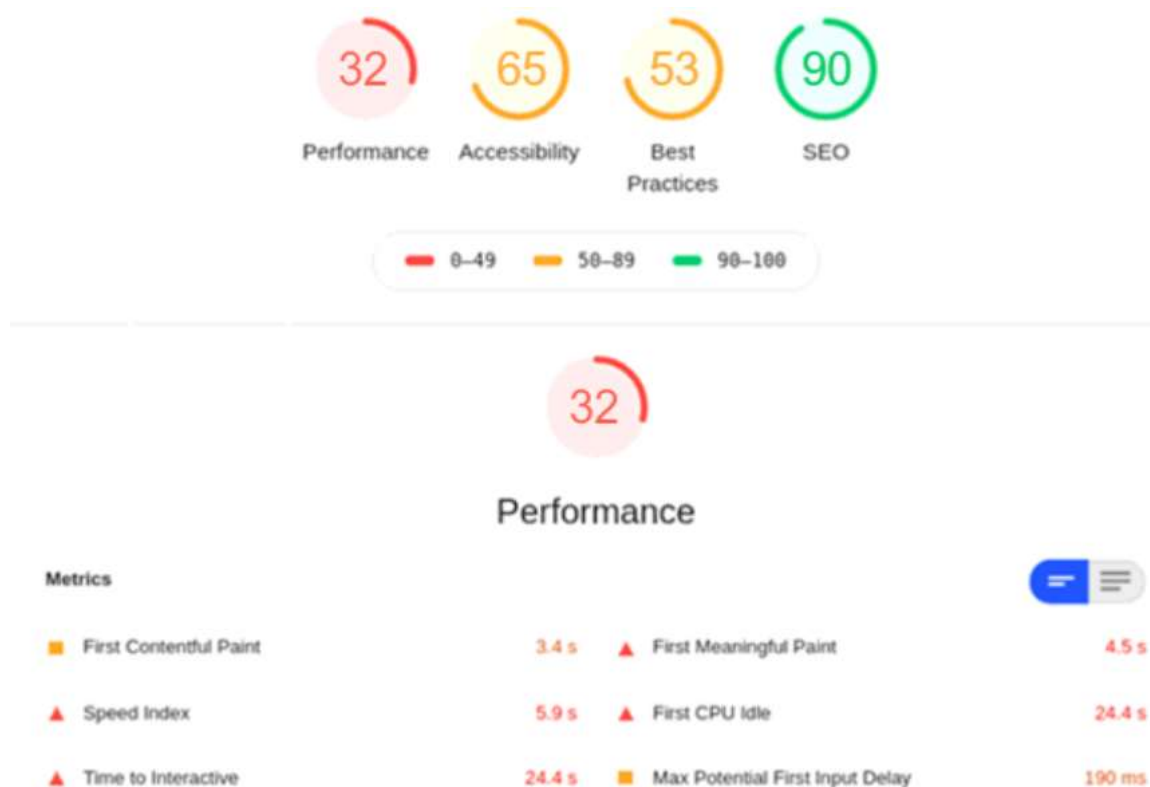


Рисунок 4.1 – Аналітичні дані сайту "iMatrix"

За результатами тесту сайту "iMatrix" за допомогою вбудованої в Google Chrome утіліти Google Audit було встановлено, що в середньому сторінка

завантажується близько 4.5 секунди. При цьому найменша оцінка була виставлена саме за продуктивність системи.

На рисунку 4.2 зображено аналітичні дані Google Audit для розробленої інформаційної системи.

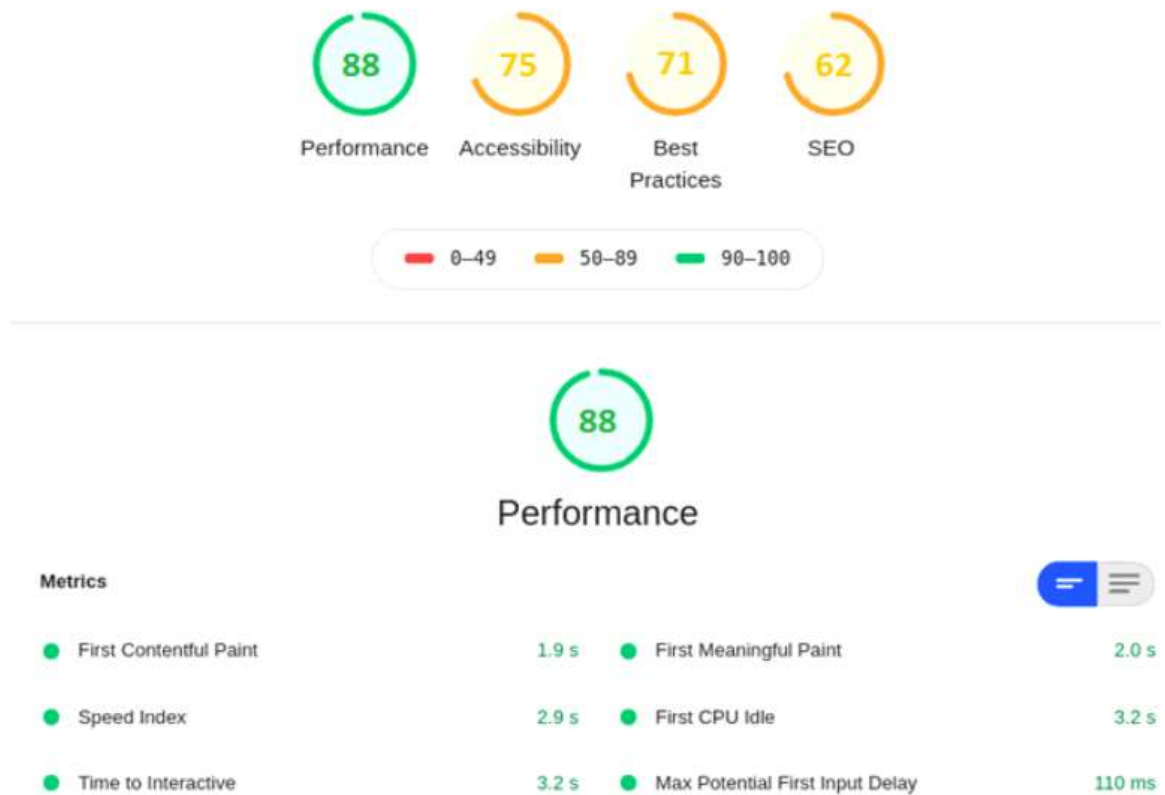


Рисунок 4.2 – Аналітичні дані розробленої системи

Після проведення тесту розробленої інформаційної системи було виявлено, що середній час завантаження всіх елементів інтерфейсу системи становить приблизно 2 секунди. Згідно з дослідженням, проведеним у 2018 році корпорацією Google та Kantar TNS, 53% користувачів Інтернету закривають сторінки інформаційних систем, якщо вони не завантажуються протягом 3 секунд. Тест був повторений 10 разів на різних сторінках для кожної з систем. На основі отриманих даних було побудовано графік, що демонструє залежність між часом відкриття сторінки та кількістю переходів на ці сторінки [19].

На рисунку 4.3 представлені результати дослідження швидкодії обох інформаційних систем.

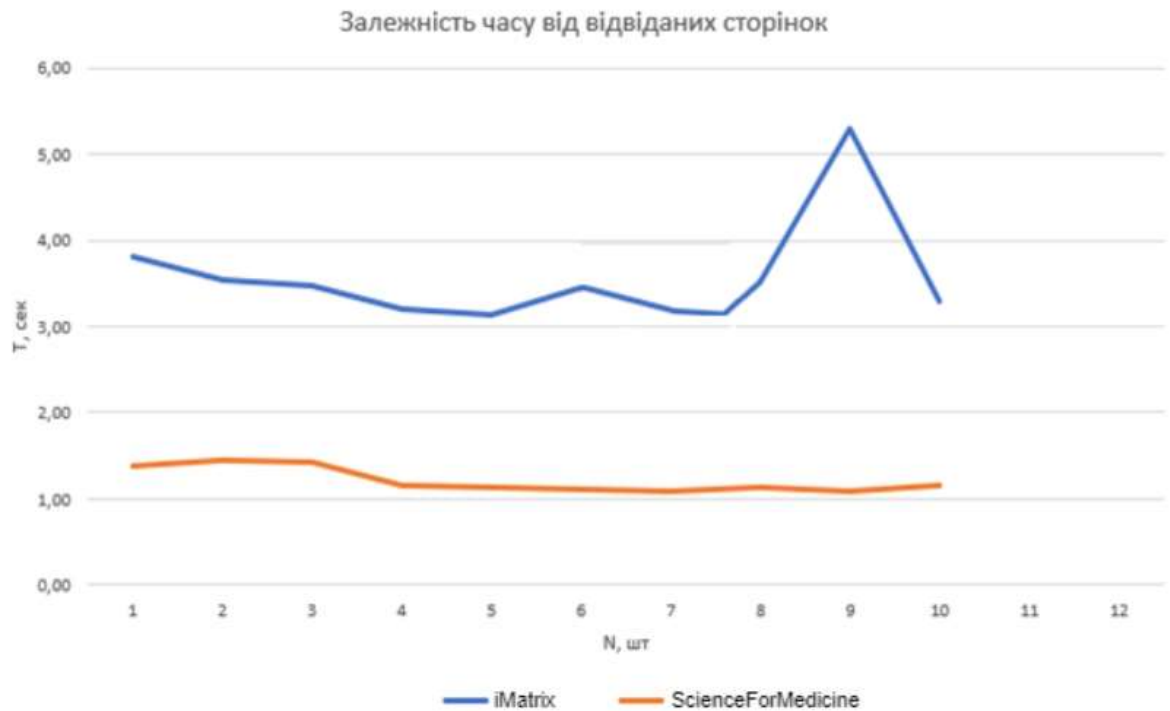


Рисунок 4.3 – Графік залежності витраченого часу від кількості відвіданих сторінок

Зробивши заміри середнього значення по часу для кожної системи отримаємо наступні значення: 3,21 секунди для “iMatrix” та 1,35 секунди для розробленої системи. Також було підраховано сумарне значення часу для обох сторінок. Для “iMatrix” це значення становить 32,12 секунд, проте як значення для розробленої системи становить лише 13,26 секунди.

Основна причина такої швидкості досягається завдяки використанню методів реактивного програмування, які активно пропагує та використовує бібліотека React.js. Ця бібліотека побудована на концепції компонентного підходу в розробці додатків, де компоненти можуть бути вкладеними або незалежними один від одного. Основна ідея полягає в тому, що компоненти реагують на зміни джерела істини.

У цій системі джерелом істини є стан самого компонента для невеликих локальних змін та глобальний стан всього додатку, наприклад, для збереження списку користувачів. Важливо зазначити, що миттєве оновлення інформації досягається за допомогою віртуального DOM-дерева, що дозволяє уникнути повного оновлення структури дерева. Замість цього, віртуальне DOM-дерево

порівнюється з оригінальним деревом, і якщо є різниці, то система оновлює лише змінену частину. Для користувача цей процес відбувається миттєво. Якщо ж порівняння двох дерев не виявляє різниць, то нічого не відбувається [20].

Схематичне зображення віртуального DOM-дерева порівняно зі звичайним DOM-деревом браузера показано на рис. 4.4.

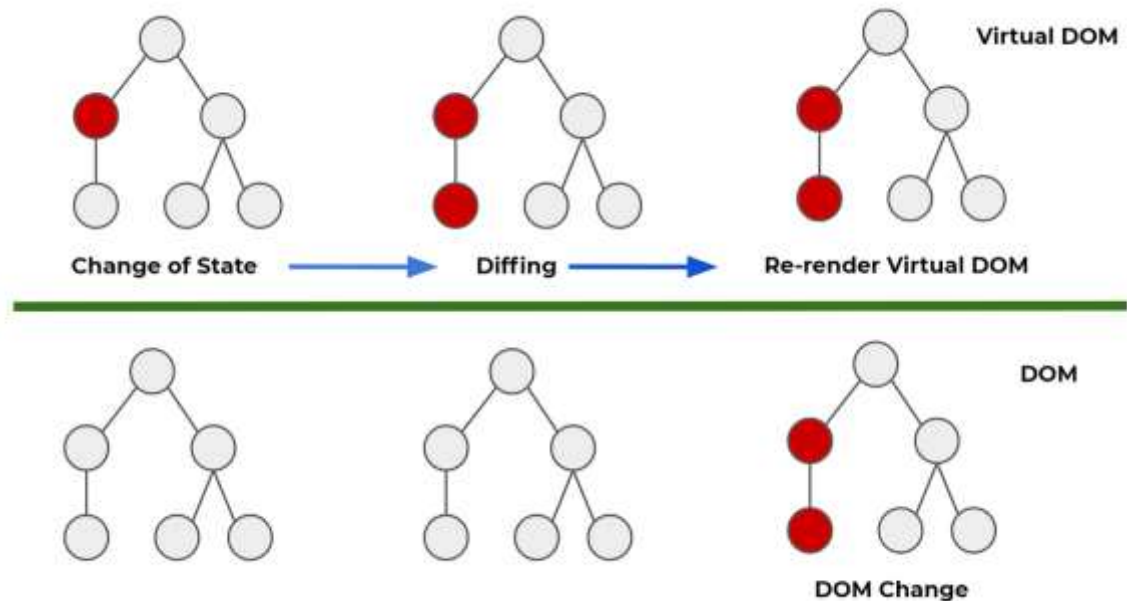


Рисунок 4.4 – Схематичне відображення віртуального та звичайного DOM-дерева

Отже, після проведення дослідження, щодо залежності часу, витраченого на відвідування сторінок, від кількості одночасно відкритих сторінок в інформаційних системах "iMatrix" та "ScienceForMedicine", можна зробити висновок, що розроблена інформаційна система в середньому працює втричі швидше, ніж її аналог. Така швидкодія забезпечується клієнтський рендерингом та розгалуженим віртуальним DOM деревом яке дуже швидко змінює в певний час після зміни лише потрібний елемент зі своєї структури. При цьому користувач бачить зміни миттєво.

4.2 Дослідження адаптивності інтерфейсу користувача

Для тестування наявності адаптивного інтерфейсу було обрано браузер Google Chrome, оскільки в ньому є зручні інструменти розробника, які дозволяють емулювати роботу системи на різних діагоналях з будь-якою роздільною здатністю. Також, в налаштуваннях цього браузера можна вибрати один з популярних пристроїв, що використовуються в наш час.

Початково, була протестована головна сторінка сайту (рис. 4.5).

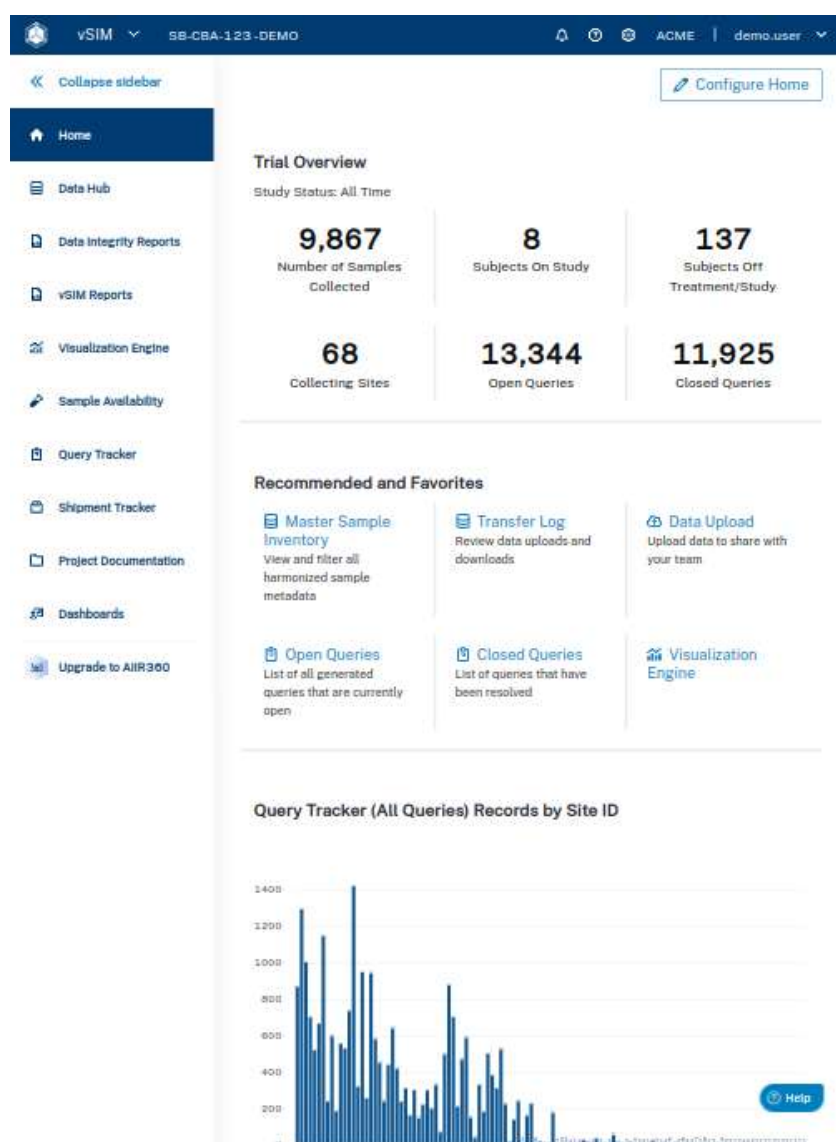


Рисунок 4.5 – Адаптивна головна сторінка сайту на невеликому планшеті

Варто відзначити, що користувач з мобільного пристрою не має жодних обмежень у функціональності сайту і має доступ до всіх можливостей, як і користувач комп'ютера. Завдяки використанню CSS3 Media правил, інтерфейс сайту автоматично адаптується до діагоналі пристрою, що дозволяє зручно генерувати графіки або проходити керувати власним акаунтом всього за один або два дотики по екрану.

Сторінку зі згенерованим графіком можна побачити на рисунку 4.6.

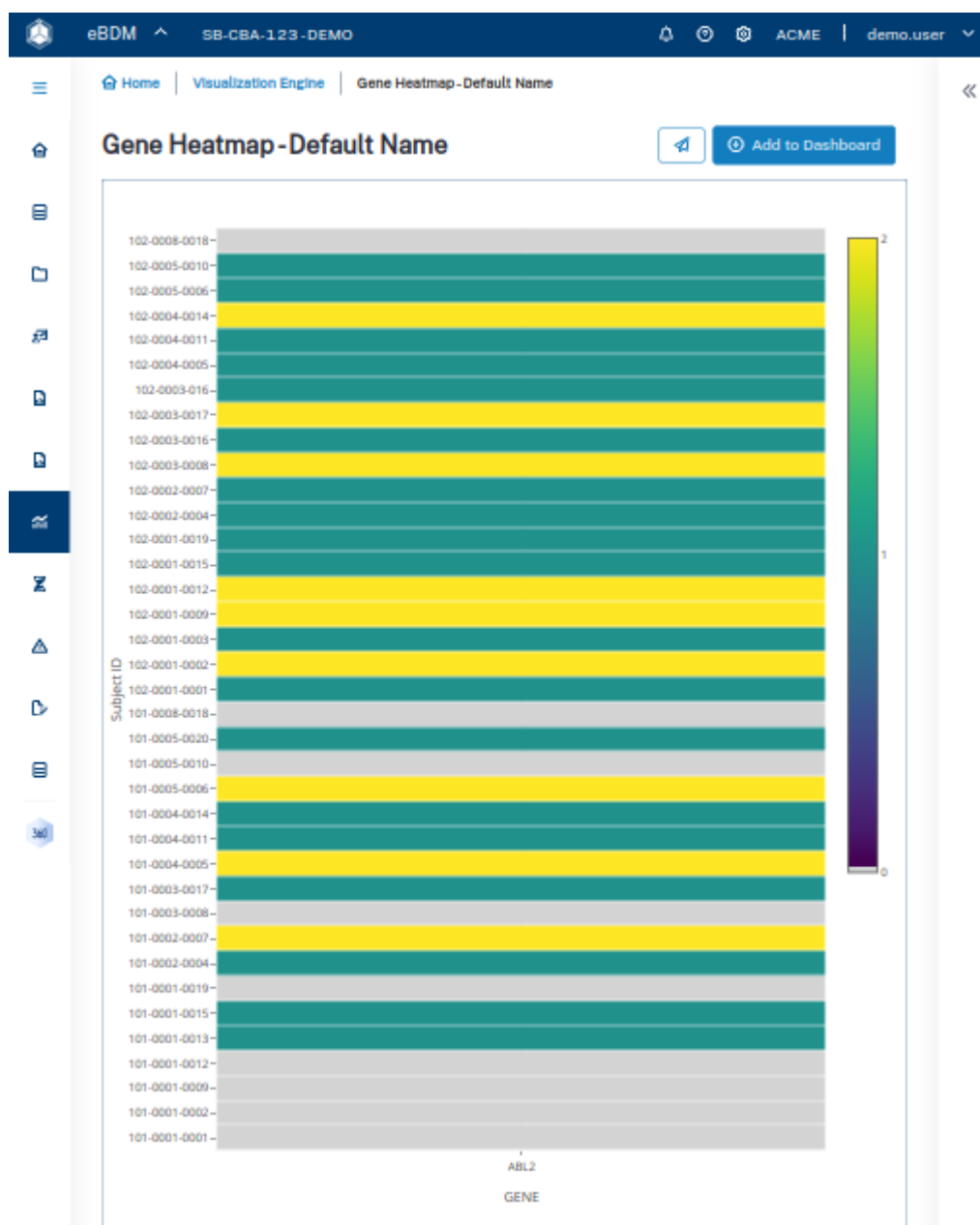


Рисунок 4.6 – Адаптивна сторінка з графіком генової теплової мапи

При переході на сторінку генерації графіку з планшетного пристрою можна скористатися боковими меню-панелями в яких є всі налаштування і які при бажанні можуть сховатися для того щоб не займати корисним простір.

Сторінка реєстрації на великому екрані з роздільною здатністю 2560x1440 зображена на рисунку 4.7.



Рисунок 4.7 – Сторінка з графіком генової теплової мапи на великому екрані

Як можна побачити, сторінка з компонентом графіка гарно виглядає як на великому екрані стаціонарного комп'ютера, так і на маленькому екрані мобільного пристрою. До речі, якщо користувач обере дані які неможливо обробити, то система обов'язково сповістить про це.

Як виглядає головна сторінка на великому екрані можна побачити на рисунку 4.8.

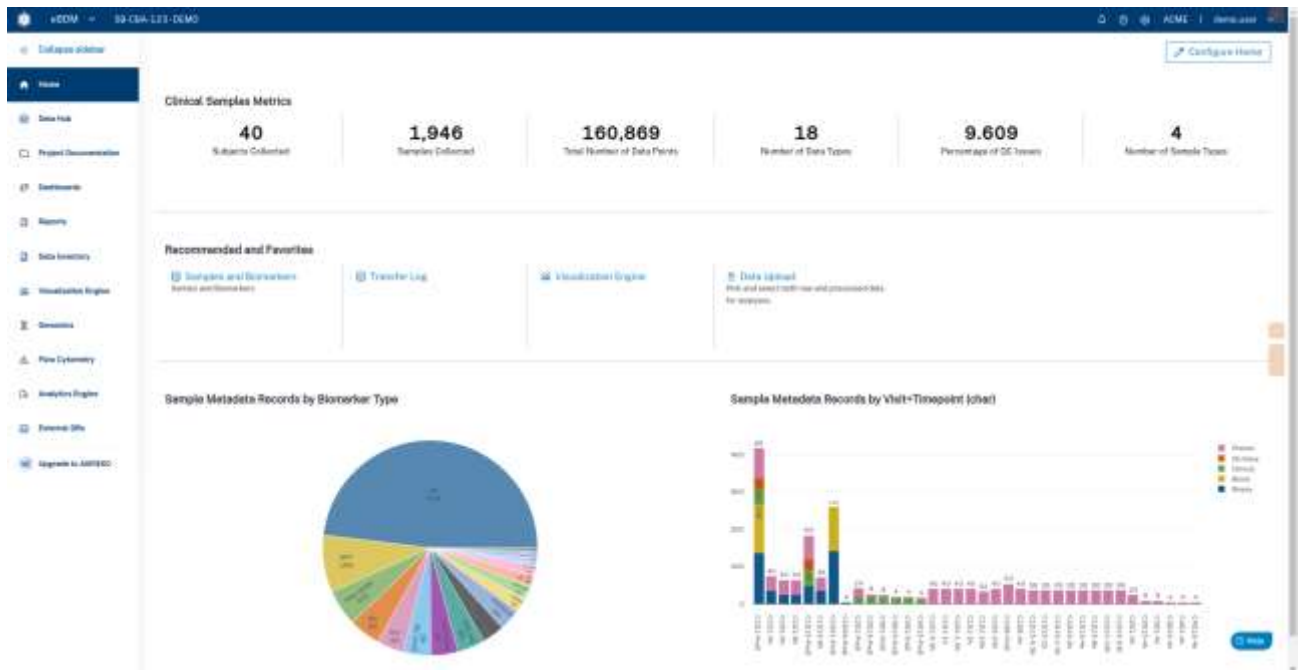


Рисунок 4.8 – Головна сторінка системи на великому екрані

Загалом, якщо опиратися на отриманий результат, то можна зробити висновок що користувачі завжди хочуть отримати зручний і зрозумілий інтерфейс незалежно від того, з якого пристрою вони використовують сайт або додаток. Адаптивний дизайн дозволяє автоматично пристосовувати контент до розміру екрану, забезпечуючи зручне і навіть візуально привабливе відображення.

Адаптивний дизайн впливає на показники SEO (пошукової оптимізації), оскільки пошукові системи, такі як Google, віддають перевагу сайтам з адаптивним дизайном. Це означає, що сайти з адаптивним інтерфейсом мають більші шанси на високе розташування у пошукових результатах, що збільшує їх видимість для користувачів.

Також важливо зауважити, що замість створення окремих версій сайту або додатку для кожного типу пристрою, адаптивний дизайн дозволяє створити одну універсальну версію, яка пристосовується до різних пристроїв. Це зберігає час і зусилля при розробці, оновленні та підтримці.

ВИСНОВКИ

Отже, в результаті виконаної дипломної роботи було розроблено інформаційну систему аналізу, обробки та візуалізації даних для медичних науковців, які за допомогою подібних систем можуть зробити методи прецизійної медицини більш наглядними та зрозумілими.

Варто відмітити, що з великою ймовірністю подібна система є першою у Україні. Вона зможе забезпечити потреби медичних науковців і більш наглядному аналізі даних, а пацієнти зможуть отримати більш точні прогнози на лікування.

Таким чином, під час виконання магістерської роботи були отримані наступні результати:

- проведено аналіз предметної області, в ході якого порівняли інформаційні системи, розглянули можливі технології для виконання завдання і обґрунтували вибір бази даних та актуальність роботи;

- проведено проектування інформаційної системи аналізу, обробки та візуалізації даних, під час якого було висунуто ряд певних вимог до системи, які в першу чергу стосуються властивостей, поведінки та особливих атрибутів, яким повинна відповідати розроблена система;

- проведено моделювання діаграми варіантів використання та послідовності взаємодії у системі, розроблено специфікацію вимог до програмного забезпечення (SRS);

- реалізовано програмне забезпечення клієнт-серверної архітектури на основі наступного технічного стеку: Clojure, Docker, Reagent, Re-frame та база даних MySQL;

- проведено дослідження реалізованої інформаційної системи з метою визначення її швидкодії та надійності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

- 1 Персоналізована медицина (Індивідуальна, персоніфікована медицина)
URL: <https://www.pharmencyclopedia.com.ua/article/8240/personalizovana-individualizovana-personifikovana-medicina>.
- 2 The Promise of Precision Medicine. URL: <https://www.nih.gov/about-nih/what-we-do/nih-turning-discovery-into-health/promise-precision-medicine>.
- 3 Роберт С. М. Чиста архітектура. Харків : Фабула, 2019. 368с.
- 4 Casimir Saternos. Client-Server Web Apps with JavaScript and Java 1st Edition. O'Reilly : California. 2014. 260p.
- 5 Балик. Н.Р., Мандзюк В.І. Бази даних MySQL. Тернопіль : Навчальна книга - Богдан, 2010. 160с.
- 6 Типи баз даних: особливості, відмінності та приклади. URL: <https://dou.ua/lenta/articles/types-of-databases>.
- 7 Розплутаний ClojureScript. URL: <https://lambdabooks.github.io/clojurescript-unraveled>
- 8 Chas Emerick, Brian Carper, Christophe Grand. Clojure Programming Practical Listp for the Java World. O'Reilly : California. 2012. 630 pages.
- 9 Специфікація вимог до програмного забезпечення (SRS) URL: <https://visuresolutions.com/uk/software-requirement-specification-srs-tips-template>
- 10 Martin Fowler. UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd Edition Print2print : Addlestone. 2016. 208 pages.
- 11 WIMP. URL: <https://www.computerhope.com/jargon/w/wimp.htm>
- 12 Adam Freeman. Pro ASP.NET MVC 5 (Expert's Voice in ASP.Net) 5th ed. Edition. Apress : New York. 2013. 832 pages.
- 13 MVC. URL: <https://brander.ua/technologies/mvc>
- 14 Bernd Oggl, Michael Kofler. Docker: Practical Guide for Developers and Devops Teams. Rheinwerk Computing : Boston. 2023. 492 pages.

- 15 The Clojure Programming Language. URL: <https://clojure.org>
- 16 Clojure Spec Guide. URL: <https://clojure.org/guides/spec>
- 17 Reitit Introduction. URL: <https://cljdoc.org/d/metosin/reitit/0.7.0alpha7/doc/introduction>
- 18 Chas Emerick, Brian Carper, Christophe Grand. Clojure Programming Practical Lisp for the Java World O'Reilly : California. 2012. 630 pages.
- 19 Qualitative Research. URL: <https://tns-global.ge>
- 20 Alex Banks, Eve Porcello. Learning React: Modern Patterns for Developing React Apps 2nd Edition. O'Reilly : California. 2020. 300 pages