Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Магістр
(назва освітнього ступеня)

на тему: Технології комп'ютерного аналізу та візуалізації біомедичних даних людини

Виконав(ла): студент(ка) 6 курсу, групи СМ-61
спеціальності
123 «Комп'ютерна інженерія»
(цифр і назва спеціальності)

Кіяк І.В. Кіяк І.В.
(підпис) (прізвище та ініціали)

Керівник _____ Луцик Н.С.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Гунько І.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Осухівська Г.М.
(підпис) (прізвище та ініціали)

Рецензент _____ Боран Т.О.
(підпис) (прізвище та ініціали)

Тернопіль
20___

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Комп'ютерних систем та мереж
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Осухівська Т.М.
(підпис) (прізвище та ініціали)
«01» 10 2020 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 123 "Комп'ютерна інженерія"
(шифр і назва спеціальності)

студенту Леськів Вітанію Михайловичу
(прізвище, ім'я, по батькові)

1. Тема роботи Вейвлет-аналіз біомедичних даних людини

Керівник роботи Р.h.D, доцент кафедри КС Пучак Надія Степанівна
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «12» вересня 2020 року № 417-ОД

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Медична інформаційна клієнт-серверна система у вигляді web-додатку

4. Зміст роботи (перелік питань, які потрібно розробити)
Аналіз існуючих методів та засобів аналізу і візуалізації біомедичних даних людини
Обґрунтування вибору методів для реалізації модуля інтелектуального аналізу медичної інформаційної клієнт-серверної системи
Охорона праці та безпека життєдіяльності
Висновки

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека в надзв. ситуаціях	Клепчик В.М., ст. викладач	[illegible]	22.12.20
Охорона праці	Осипенко П.М.		

7. Дата видачі завдання 01.10.2020р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз існуючих та найбільш популярних видів інформаційних систем в Україні	29.09.2020 - 10.10.2020	виконано
2.	Вибір технології для реалізації основних частин інформаційної системи	11.10.2020 - 24.10.2020	виконано
3.	Вибір методу опрацювання біометричних даних	26.10.2020 - 19.11.2020	виконано
4	Охорона праці в надзвичайних ситуаціях	21.11.2020 - 04.12.2020	виконано
5	Оформлення пояснювальної записки	04.12.2020 - 06.12.2020	виконано
6	Оформлення графічного матеріалу	07.12.2020 - 11.12.2020	виконано
7	Попередній захист кваліфікаційної роботи магістра		
8.	Захист дипломної роботи магістра	13.12.2020	виконано

Студент [підпис] Лесюк В. М.
(підпис) (прізвище та ініціали)

Керівник роботи [підпис] Лущик Н. С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Технології комп’ютеризованого аналізу та візуалізації біомедичних даних пацієнта // Дипломна робота магістра // Леськів Віталій Михайлович // ТНТУ, Комп’ютерна інженерія, група СІм-62 // Тернопіль, 2020 // С. –,__ рис. –, __ бібліогр. – __

Ключові слова: клієнт-сервер, медична інформаційна система, веб-додаток, біомедичні дані.

У даній магістерській кваліфікаційній роботі розробляється медична клієнт-серверна система з техногогією візуалізації біомедичних даних. Основними вимогами до проектування є цільова платформа Java EE, середовище Spring та Hibernate, клієнт Веб-браузер.

В ході виконання даної магістерської кваліфікаційної роботи виконано аналіз медичних клієнт-серверних систем та проведено огляд існуючих технічних рішень, та підходів до побудови таких систем.

Спроектовано архітектуру клієнт-серверної системи та базу даних, на основі результатів проектування реалізовано клієнт-серверну систему для вирішення поставленої задачі.

За результатами практичної реалізації проведено дослідження роботи розробленої клієнт-серверної системи та підраховано економічний ефект при впровадженні у виробництво.

ANNOTATION

Technology of computer analysis and visualization of patient biomedical data // Master’s qualification work // Leskiv Vitaliy Mikhaylovich // TNTU, Comptuer ingeniring, Sim-62 // Ternopil, 2020 // C. –,__ img. –, __ bibliograph. – __

Keywords: client-server, medical information system, web application, biomedical data.

In this Master's qualification work developed a medical information client-server system with biomedical data visualization techlonogy. Platform Java EE, framework Spring and Hibernate, the client Web-browser.

In this Master's qualification work developed a medical information client-server system with biomedical data visualization techlonogy. Platform Java EE, framework Spring and Hibernate, the client Web-browser.

In the course of this master's qualification work analyzed medical client-server systems, and a review of existing technical solutions and approaches to building such systems.

Designed the architecture of the client-server system and database, based on the design is implemented client-server system for solving the problem.

Based on the results of practical implementation a study was conducted work of the client-server system and estimated the economic impact of the introduction in production.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, СИМВОЛІВ ТА СПЕЦІАЛЬНИХ ТЕРМІНІВ

MIC – Медична інформаційна система.

БД – База даних.

HTML (англ, HyperText Markup Language) – Мова розмітки гіпертексту.

ODBC (англ. Open Database Connectivity Standard) – Стандарт відкритого підключення до бази даних.

ASP (англ. Active Server Pages) – Активні серверні сторінки.

JSP (англ. Java Server Pages) – Джава серверні сторінки.

IIS (англ. Internet Information Services) – Набір інформаційних сервісів для Інтернету.

CLR (англ. Common Language Runtime) – Загальне середовище виконання мов програмування, що входять до середовища .NET.

JVM (англ. Java Virtual Machine) – Віртуальна машина Java.

ОС (англ. OS – Operation system) – Операційна система.

RIA (англ. Rich Internet application) – Веб-додаток насичений функціональністю традиційних прикладних програм.

HTTP (англ. Hyper Text Transfer Protocol) – Протокол передачі гіпертексту.

AJAX (англ. Asynchronous JavaScript And XML) – Асинхронні запити, що працюють у фоні і не потребують перезавантаження веб-сторінки.

SOAP (англ. Simple Object Access Protocol) – Простий протокол доступу до об'єктів.

IDE (англ. Integrated Development Environment) – Інтегроване середовище проектування та розробки.

ORM (англ. object-relational mapping) – Відображення об’єктно-реляційних зв’язків.

SQL (англ. Structured query language) – Мова структурованих запитів, використовується для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення, видалення даних і керування реляційними БД.

HQL (англ. Hibernate Query Language) – Мова запитів Hibernate до бази даних, що є дуже схожою до SQL.

MVC (англ. Model-view-controller) – Найпопулярніший дизайн паттерн що дозволяє реалізувати шаблон модель-вигляд-контролер.

DAO (англ. Data Access Object) – Об'єкт доступу до даних.

JDK (англ. Java Development Kit) – Набір стандартних бібліотек, класів та компілятора, що дозволяють програмувати та виконувати програми на Java.

CSS (англ. Cascading Style Sheets) – Каскадна таблиця стилів, що використовується для створення веб-сторінок.

XML (англ. Extensible Markup Language) – Мова розмітки із ієрархічною структурою даних для обміну між різними додатками.

DOM (англ. Document Object Model) – Специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами (наприклад, документи XML, HTML).

JSON (англ. JavaScript Object Notation) – Легкий формат обміну даними між програмами, найбільш розповсюджений при написанні веб-програм, а саме при використанні технології AJAX

ВСТУП

Актуальність теми. Сучасні інформаційні технології все більше використовуються в галузі охорони здоров'я, наприклад єдина база даних історій хвороб, системи класифікації термінів, міжнародна класифікація хвороб (ICD-10). Це все так звані медичні інформаційні системи (MIC). Завдяки цьому медицина, в тому числі і нетрадиційна, набуває сьогодні абсолютно нових рис. У багатьох медичних дослідженнях просто не можливо обійтися без комп'ютера і спеціального програмного забезпечення до нього. Цей процес супроводжується суттєвими змінами в медичній теорії та практиці, пов'язаними з внесенням коректив до підготовки медичних працівників. Беручи до уваги стан медицини в нашій країні, де досі вся необхідна для пацієнта інформація зберігається в медичних книгах, що супроводжується безліччю проблем з централізацією доступу до його біомедичних даних, впровадження новітніх технологій у процес ведення обліку, визначення діагнозів і лікування в медичних закладах є досить актуальним питанням.

Метою роботи є розробка клієнт-серверної системи, що представляє собою медичну інформаційну систему, яка призначена для автоматизації бізнес процесів в медичному закладі для аналізу і визначення діагнозу на основі біомедичних даних пацієнта. Розроблена система буде відповідати усім існуючим вимогам до таких систем.

Об’єктом дослідження є бізнес процеси у медичному закладі, які застосовуються для ведення обліку пацієнтів, аналізу їхніх даних, а також візуалізації призначених їм діагнозів.

Предметом дослідження є медична інформаційна система, яка використовується в медичному закладі, як місце для централізованого доступну до даних пацієнта

Методи дослідження. У даній роботі за осовновний метод дослідження біомедичних даних пацієнта було взято статистичний метод. За домопогою нього на основі статистики і збору цих даних буде формуватися звітність і результат різних обстежень.

Наукова новизна одержаних результатів.

Результатом виконання роботи є нова медична інформаційна система, яка поєднує в собі можливість ведення обліку пацієнтів в медичному закладі, а також можливість відображення аналізу даних пацієнта. Відмінність досягнених результатів полягає в тому, що на українському ринку мало продуктів, які поєднюють відразу два таких модуля. Зазвичай, це просто система ведення обліку, яка оперує даними пацієнтів медичного закладу. В процесі роботи було удосконалено метод ведення обліку пацієнтів, покращено інтерактивність інтерфейсу для цього, а також дістав подальший розвиток метод відображення біомедичних даних пацієнта за допомогою звітності і результуючих таблиць.

Практичне значення одержаних результатів.

Результат виконання магістерської роботи – медична інформаційна система, яка може бути використана у медичних закладах України з ціллю автоматизації процесів ведення обліку пацієнтів і візуалізації їх біомедичних даних. Застосування звітів по зроблених для пацієнта тестів допоможе чітко і швидко повідомляти пацієнта про його стан здоров’я і наступні кроки в процесі лікування.

Публікації.

Результати дослідження апробовано на IX Міжнародній науково-технічній конференції молодих учених та студентів у зібрнику «Актуальні задачі сучасних технологій», на VIII Науково-технічній конференції у збірника «Інформаційні моделі, системи та технології»

Структура роботи. Робота складається з пояснювальної записки та графічної частини. Пояснювальна записка складається із вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Обсяг роботи: пояснювальна записка – 91 аркуш формату А4, графічна частина – 8 аркушів формату А1.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА ЗАСОБІВ АНАЛІЗУ І ВІЗУАЛІЗАЦІЇ БІОМЕДИЧНИХ ДАНИХ ПАЦІЄНТА

1.1. Загальна характеристика медичних інформаційних систем

Головними завданнями в роботі закладів у сфері охорони здоров'я є підвищення рівня якості, доступності, оперативного надання медичної допомоги, профілактики і вчасне виявлення захворювань. Як видно на практиці, використання інформаційних технологій і застосування медичних інформаційних систем в медичних установах дозволяє швидко збільшити якість медичного обслуговування.

Основними пріоритетами в діяльності закладів галузі охорони здоров'я (ЗГОЗ) є підвищення рівня доступності та якості надання медичної допомоги, профілактика та раннє виявлення захворювань. Як демонструє практичний навик застосування інформаційних технологій в різних медичних установах, впровадження медичних інформаційних систем (МІС) дозволяє збільшити якість і ефективність медичних пропозицій [4].

Національна політична концепція інформатизації охорони здоров'я розглядає повну демократизацію процесу створення та використання інформації, наявність інформаційних ресурсів та пропозицій, а також захист прав людини від проникнення інформації. Метою державних політиків у створенні та розвитку інформаційної інфраструктури охорони здоров'я є досягнення інформаційної інфраструктури статусу світового класу. У 2017 році було впроваджено низку заходів з розвитку сімейної медицини як основи профілактики захворювань

Клієнт-серверна система – це вид розподіленої системи, що складається з двох базових частин клієнта та сервера, де сервер опрацьовує запити від клієнта, при чому клієнт та сервер спілкуються між собою за допомогою певних засобів (протоколів), що відбуваються через комп'ютерну мережу [2]. Слід відрізняти технологію клієнт-сервер в широкому сенсі, що може бути застосовано до будь-якої комп'ютерної системи, від особливості клієнт-серверної архітектури, що

застосовується для побудови інформаційних систем та автоматизованих систем управління бізнес-процесами.

Відмінність медичних клієнт-серверних систем від інших існуючих аналогів клієнт серверних-систем полягає тільки в тому, що медичні системи строго орієнтовані для використання в медичних цілях, тобто предметною областю такої системи є медицина та все що може бути пов’язано із нею. Для таких потреб існує окремий вид інформаційної системи медична.

Медична інформаційна система - це інструмент, що використовується для визначення та планування всіх ресурсів, необхідних медичним установам для медичної діагностики, адміністративної економіки, економіки, сервісної роботи та обліку в процесі надання медичних послуг. Представлені на українському ринку МІС поділяються на системи та процеси управління документами, різницю між ними можна виразити як:

- якщо предметом інформаційної концепції користувальницької діяльності є поведінка (хроніка захворювань, особливий порядок пацієнта, щоденник, щоденник тощо), у цьому випадку це є стандартна концепція управління документами;

- якщо користуючись інформаційною системою, користувач керує процесом лікування і статусом процесу, то така МІС є процесною

Результати їх реалізації отримуються за типом інформаційних систем. Системи управління документами можуть зменшити кількість паперових документів, використовуючи їх копіювання, а також зменшити кількість помилок у їх структурі, досягти типізації документів та обов’язковості їх створенню тощо. На відміну від систем управління документами, технологічні системи призначені для мінімізації використання ресурсів, поліпшення охорони здоров’я, підвищення продуктивності праці медичних працівників.

Інша класифікація медичних інформаційних систем пов'язана з областю, що надається:

- Radiology Information System (RIS) – системами, що дозволяють організувати зберігання медичних зображень та доступ до них.

- Enterprise Resource Planning (ERP) системи, що дозволяють організувати управління економічними, трудовими та матеріальними ресурсами установи.

- Electronic Medical Records (EMR) – системи медичної документації.

У більшості випадків системи, призначені для комунальних закладів охорони здоров’я, є системами медичних записів (EMR), а системи, що використовуються в приватних закладах, є системами управління (ERP) різної складності. Це пов’язано з різницею в підході до організації робочого процесу в установах з різними формами власності. Оскільки приватна медицина орієнтована на оптимізацію та продуктивність праці, міська медицина стоїть на першому плані - на записах та медових записах.

У більшості випадків система, розроблена для муніципальних медичних установ, є системою медичної документації (EMR), тоді як система, що використовується в приватних закладах, є системою управління (ERP) різної складності. Це пов’язано з різними методами організації робочих процесів в установах різних форм власності. Якщо приватна медицина орієнтована на оптимізацію та ефективність виробництва, то муніципальна – насамперед на записи та медичний облік. Як правило, медична інформаційна система складається з наступних основних модулів:

- електронні медичні картки користувачів (пацієнтів);
- інституційні ресурси та їх розподіл;
- медичні дані;
- економічна та адміністративна інформація закладу;
- засоби спілкування між персоналом організації.

1.2. Основні завдання медичних інформаційних систем

Одне із завдань, яке мають на меті медичні інформаційні системи, - це централізація даних. Він фактично визнає той факт, що вся інформація, що вводиться в інформаційну систему IGP, доступна в будь-який час під час процесу введення. Це дозволяє перевірити інформацію для всього персоналу закладу.

Другою необхідною функцією медичної інформаційної системи є типизація даних, яка використовується для інформації та процесів. Іншими словами, список пацієнтів поєднується разом із медичними даними і оформляється на єдиній основі.

Третя функція медичних інформаційних систем забезпечити доступність інформації для її подальшої обробки. Загальна та проміжна інформація доступна для кожного подальшого процесу та оцінки. Це вважається одним з основних переваг для MIC, яка, однак, часто залишається поза увагою.

1.3. Аналіз існуючих та найбільш популярних медичних інформаційних систем в Україні

Проведено порівняльний аналіз існуючих в Україні медичних інформаційних систем, включаючи систему Doctor Elex, єдину модульну медичну інформаційну систему EMSIMED та медичну інформаційну систему MedITEX.

Комплексна медична інформаційна система «Doctor Elex», розроблена компанією «Елекс» (Львів), включає такі модулі: реєстр, фінанси, стаціонар, лабораторія та інші.

Перевагами цієї системи є:

- інтегрована електронна медична карта пацієнта та система медичних оглядів;
- віддалений доступ до конфіденційних даних;
- інформаційне забезпечення медичних оглядів;
- сумісність із сучасним медичним обладнанням.

До недоліків належать:

- надмірна функціональність, яка може бути зайвою для деяких користувачів. Наявність надмірної функціональності несе непотрібне навантаження на систему, що негативно позначається на її швидкості роботи;

- висока вартість надання послуг;

- призначення, в основному, для великих установ, що не забезпечує переваг при використанні у відносно невеликих закладах.

Єдина модульна медична інформаційна система "EMSIMED" оптимальна для медичних центрів, які планують розвиватись. Система розроблена компанією АЛТ-Україна (Київ).

Переваги:

- пристосування до законодавства та вимог МОЗ;

- велика кількість параметрів процесів та характеристик установи, масштабованість;

- застосування рішень для захисту даних пацієнта та персональних даних;

- можливість використовувати лише необхідні модулі без переплати.

Недоліки:

- необхідність постійної підтримки від розробників;

- відсутність можливості встановлення на декілька платформ;

- необхідність надання власних серверів для встановлення системи;

Третя медична інформаційна система MedITEX, її перевагами є:

- 100% дотримання норм та вимог Міністерства охорони здоров’я;

- гнучка адаптація до процесу та характеристик організації, масштабованість;

- впровадження глобальних рішень щодо захисту даних про здоров'я та персональних даних пацієнтів;

- можливість використовувати лише необхідні модулі, без додаткової доплати.

Після аналізу розглянутих систем було визначено методи усунення подібних недоліків. Щоб задовольнити потреби клієнта медичного закладу (пацієнта), не завжди необхідно охоплювати безліч функцій, оскільки в цьому може бути необхідність, тому при розробці МІС необхідно чітко визначити функції. потрібні для реалізації, так як це скоротить час і витрати на розробку. В результаті така система буде мати більш низьку вартість, оскільки не доведеться дуже багато

платити за функції, які не потрібні установі. В системі повинні бути модулі, які можуть працювати незалежно один від одного, це підвищить стабільність системи і дозволить установі вибирати тільки ту функціональність, яка необхідна для роботи. Система повинна бути оплачена у вигляді підписки на програмний продукт. Іншими словами, клієнт буде платити певну суму один раз в певний період. Це забезпечує безперервну програмну підтримку продукту. Концепція веб-додатка дозволить використовувати систему на будь-якому пристрої, незалежно від операційної системи. Це дозволить користувачеві отримати доступ до даних в будь-який час.

Важливою функцією МІС є інтелектуалізація, тобто взаємодія лікарів - спеціалістів з автоматизованою медичною інформаційною системою для прийняття оптимальних та ефективних медичних рішень. Інтелектуальні інформаційні системи, засновані на наявних знаннях у базі знань (БД) та фактах у базі даних (БД), пропонують свої рішення і діагнози. У процесі взаємодії з системою лікар може або повністю прийняти або відхилити запропоноване рішення, або на власний розсуд її адаптувати, тобто відповідальність за прийняття рішення завжди несе певна особа – лікар. Оскільки в екстреній медицині швидка діагностика і прогнозування загрози життю пацієнта в надзвичайних ситуаціях, а також вживання відповідних екстрених заходів для їх усунення – досить важливе питання, було розроблено алгоритми виявлення та програмне забезпечення, яке стане наступним кроком в інтелектуалізації МІС. Проблемна ситуація в організмі пацієнта на підставі його електронного медичного опитування, а також використання інтелектуальної підтримки рішення лікаря допоможе вийти із ситуації. Звичайно, такі ситуації мають різний характер при різних патологічних станах органів і організму в цілому і вимагають відповідного підходу до їх усунення. Такі досягнення важливі і потрібні, особливо в тих галузях медицини, де елементи суб'єктивізму дуже важливі і відповідальність за прийняття рішень висока, що характерно особливо для хірургії, особливо для екстреної хірургії черевної порожнини.

1.4. Висновки до розділу 1

В даному розділі було проведено загальну характеристику медичних інформаційних систем, які присутні на ринку України, зроблено їх аналіз, , проаналізовано їхні переваги та недоліки. З метою уникнення недоліків проаналізованих медичних іформаційних систем, було вирішено розробити власну систему, як окремий веб-додаток.

РОЗДІЛ 2
ОБГРУНТУВАННЯ СТРУКТУРНИХ ТА ФУНКЦІОНАЛЬНИХ РІШЕНЬ

У відповідності до поставленого завдання, розроблювана клієнт-серверна медична інформаційна система повинна відповідати основним вимогам приведених в вихідних даних до роботи. Спроектована система повинна бути не гіршою за існуючі аналоги і перевершити їх. Проаналізувавши існуючий рівень технологій, що застосовуються для побудови клієнт-серверних систем, та підходів до побудови окремо клієнтської та серверної частини, та існуючі рішення подібних задач, було обрано наступний варіант, що в повній мірі допоможе реалізувати поставлену задачу та забезпечить їй кращу ефективність і продуктивність за існуючі аналоги.

2.1. Архітектура системи

Для побудови гнучкої та ефективної клієнт-серверної системи, найкраще обрати трирівневу клієнт-серверну архітектуру із окремим сервером бази даних [11], яка включає такі основні частини, як:

- база даних;
- рівень доступу до даних;
- прикладний рівень;
- рівень відображення.

На рис.2.1 приведена структурна схема архітектури клієнт-серверної системи. Перевагою трирівневої (або ще її називають багаторівневої) клієнт-серверної архітектури над дворівневою полягає в тому, що рівень доступу до даних відокремлений від прикладного рівня і дозволяє будувати гнучку систему, яка може працювати із різними середовищами зберігання даних (базами даних) або навіть декількома рівнями доступу до даних.

Вибір такої архітектури, що зображена на рис.2.1., дозволить в повній мірі використати усі особливості і переваги клієнт-серверної системи.

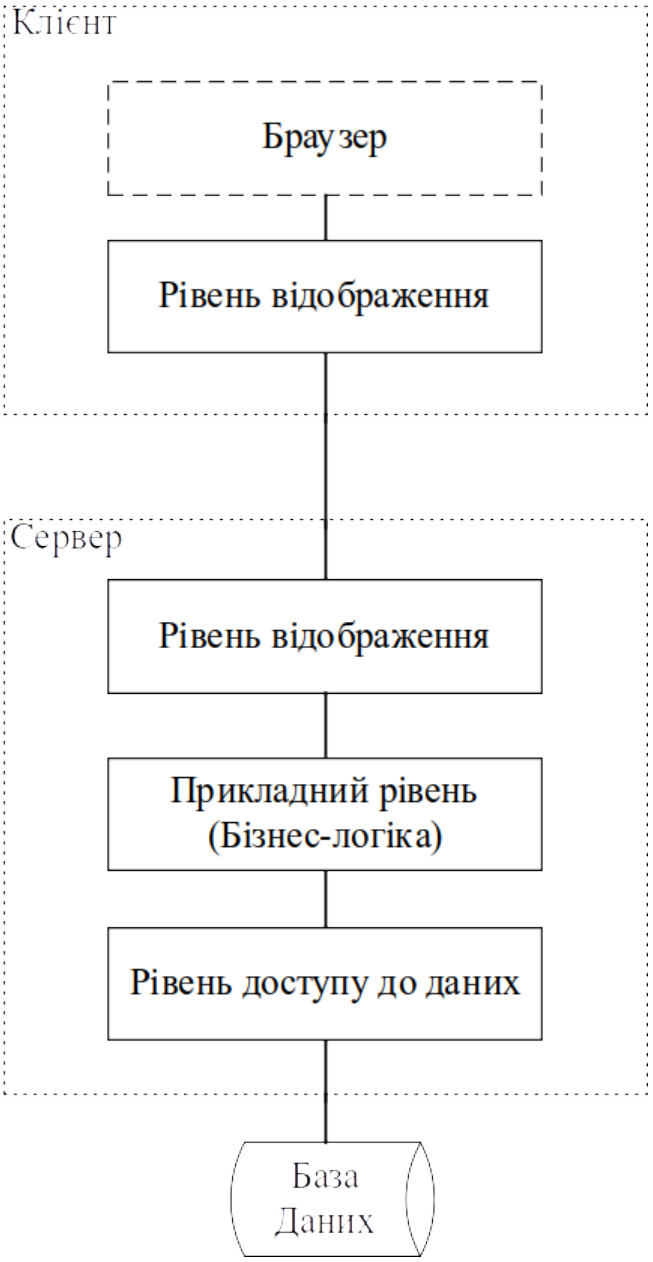


Рис. 2.1. Структурна схема архітектури клієнт-серверної системи.

2.2. Вибір технологій для реалізації основних частин архітектури системи

Відповідно до рис.2.1. розроблювана система буде побудована за приведеною на малюнку архітектурою. Архітектура системи включає в себе такі складові частини, як:

База даних. Може бути використаний будь-який із існуючих серверів баз даних (MySQL, Microsoft SQL Server, Oracle, PostgreSQL, dbm, Hyperware, Informix, InterBase, Sybase).

Рівень доступу до даних. Даний рівень буде реалізовувати усі необхідні команди для роботи із базою даних або іншими засобами збереження даних, та буде відповідати за збереження та отримання даних.

Прикладний рівень. На даному рівні буде зосереджена основна бізнес логіка розроблюваної системи. Тут будуть реалізовані основні алгоритми роботи серверної частини спроектованої системи.

Рівень відображення. Дана частина архітектури буде відповідати лише за представлення даних користувачу, отримання даних від користувача та первинний рівень валідації даних. Тут будуть реалізовані алгоритми з представлення та перетворення даних, а також буде реалізований інтерфейс користувача спроектованої системи.

Браузер. В роль браузера в даній архітектурі може бути будь-який сучасний браузер.

Для реалізації обраної архітектури клієнт-серверної системи обрано використання технологій Java EE (для реалізація серверної частини) та RIA веб-додаток (для реалізації клієнтської частини).

Основним IDE для написання програмного коду обрано Eclipse, для роботи і проектування бази даних MySQL Workbench.

Eclipse – це безкоштовне модульне, інтегроване середовище розробки програмного забезпечення. Розроблено та підтримується Фондом Eclipse. Написаний переважно на Java, його можна використовувати для розробки програм на Java та, використовуючи різні плагіни, інші мови програмування, включаючи Ada, C, C ++, COBOL, Fortran, Perl, PHP, Python, Ruby Rails framework), Scala, Clojure та Scheme. Середовище розробки включає, зокрема, Eclipse ADT (Ada Development Toolkit) для Ada, Eclipse CDT для C/C ++, Eclipse JDT для Java, Eclipse PDT для PHP.

Eclipse – це модульне середовище для розробки різних платформ з низкою функцій: – можливість розробки програмного забезпечення на багатьох мовах програмування (рідною для Java);

- багатоплатформна;

- модульний, призначений для подальшого розширення незалежними розробниками;

- системне ядро з відкритим кодом;

- Розроблено та підтримується Фондом Eclipse, до якого входять постачальники програмного забезпечення, такі як IBM, Oracle, Borland. MySQL Workbench – інструмент для візуального проектування баз даних, що інтегрує проектування, моделювання, створення й експлуатацію БД в єдине безшовне оточення для системи баз даних MySQL.

Можливості середовища:

- Дозволяє наочно представити модель бази даних в графічному вигляді.

- Наочний і функціональний механізм встановлення зв'язків між таблицями, в тому числі «багато до багатьох» із створенням таблиці зв'язків.

- Reverse Engineering – відновлення структури таблиць з вже існуючої на сервері БД (зв'язки відновлюються в InnoDB, при використанні MyISAM зв'язки необхідно встановлювати вручну).

- Зручний редактор SQL запитів, що дозволяє відразу ж відправляти їх серверові і отримати відповідь у вигляді таблиці.

Можливість редагування даних у таблиці в візуальному режимі.

В свою чергу кожен із рівнів системи буде реалізовуватись за допомогою спеціальних засобів та середовищ:

Реалізації бази даних. У ролі бази даних буде використовуватися безкоштовний сервер бази даних MySQL. MySQL – компактний багатонитевий сервер баз даних. Характеризується великою швидкістю, стійкістю і простотою використання. MySQL вважається гарним рішенням для малих і середніх застосувань. Сирцеві коди сервера компілюються на багатьох платформах [19]. Найповніше можливості сервера виявляються в UNIX-системах, де є підтримка багатонитковості, що підвищує продуктивність системи в цілому.

Можливості сервера MySQL:

- простота у встановленні та використанні;

- підтримується необмежена кількість користувачів, що одночасно працюють із БД;

- кількість рядків у таблицях може досягати 50 млн;

- висока швидкість виконання команд;

- наявність простої і ефективної системи безпеки;

- реалізація рівня доступу до даних.

Для реалізації рівня доступу до даних буде використано середовище Hibernate, що дозволить повністю позбутись залежності від сервера БД.

Hibernate це інструмент відображення між об'єктами та реляційними структурами ORM для платформи Java. Hibernate - це безкоштовне програмне забезпечення, що поширюється на умовах Загальної публічної ліцензії GNU Lesser. Hibernate забезпечує просту структуру для перегляду між об’єктно-орієнтованою моделлю даних та традиційною реляційною базою даних.

Мета Hibernate звільнити програміста від важливих типових завдань взаємодії програми з базою даних. Програміст може використовувати Hibernate як для розробки з нуля, так і для існуючої бази даних.

Hibernate дбає про асоціювання класів з таблицями баз даних (а типи даних мови програмування з типами даних SKL) і забезпечує засоби для автоматичної побудови запитів SKL і читання/запис даних і може значно скоротити час розробки, який зазвичай витрачається на введення типового SKL та код JDBC. Hibernate генерує виклики SKL і звільняє програміста для ручної обробки отриманого набору даних, перетворення об'єктів та забезпечення сумісності з різними базами даних.

Hibernate забезпечує прозору підтримку зберігання даних, тобто їх стабільність для об'єктів 'POJO', тобто для звичайних об'єктів Java; єдиною суворою вимогою до збереженого класу є конструктор за замовчуванням (для коректної поведінки в деяких додатках потрібно звернути особливу увагу на методи equal() та hashCode () [13].

Hibernate забезпечує використання SQL-подібної мови HQL, яка дозволяє виконувати SQL-подібні запити, записані поряд з об'єктами даних Hibernate. Запити критеріїв надаються як об'єктно-орієнтована альтернатива до HQL.

Реалізація прикладного рівня. Для реалізації даного рівня буде використано усі необхідні засоби об’єктно орієнтованої мови програмування Java.

Java це об'єктно-орієнтована мова програмування, випущений Sun Microsystems в 1995 році в якості основного компонента платформи Java. Oracle в даний час працює над мовою. Синтаксис мови багато в чому успадкований від C і C ++. В офіційній реалізації програми Java компілюються в байт-код, який інтерпретується віртуальною машиною для конкретної платформи.

У мові багато синтаксису запозичено у C і C ++. Зокрема, за основу взята об'єктна модель C ++, проте вона модифікована. Усунуто можливість виникнення певних конфліктних ситуацій через помилки програміста і спрощений процес розробки об'єктно-орієнтованих програм (4). Віртуальній машині призначається ряд дій, які програмісти повинні виконати на C/C ++. По-перше, Java була розроблена як переносних незалежний мову, тому у нього менше апаратних можливостей низького рівня. При необхідності java дозволяє викликати підпрограми, написані на інших мовах програмування (10).

Стандартні бібліотеки надають загальний спосіб доступу до залежних від платформи функцій, таких як обробка графіки, багатопоточність і робота в мережі. У деяких версіях для підвищення продуктивності JVM байт-код може бути скомпільований в машинний код до або під час виконання програми.

Основна перевага використання байт-коду - переносимість. Однак додаткова вартість інтерпретації означає, що інтерпретуються програми майже завжди будуть працювати повільніше, ніж програми, скомпільовані в машинний код, тому Java придбала репутацію «повільного» мови. Однак цей розрив значно скоротився після введення декількох методів оптимізації в сучасні реалізації JVM.

Один з таких методів англійська. JIT-компіляція, яка перетворює байт-код Java в машинний код при першому запуску програми, а потім кешируєт його. В результаті така програма запускається і виконується швидше, ніж простий інтерпретується код, але за рахунок додаткового часу компіляції під час виконання. Більш складні віртуальні машини також використовують динамічну перекомпіляцію, при якій віртуальна машина аналізує поведінку запущеної програми і вибірково перекомпілюються і оптимізує її частини. Динамічна перекомпіляція може забезпечити більш високий рівень оптимізації, ніж статична компіляція, тому що динамічний компілятор може виконувати оптимізацію на

основі знання середовище програмного продукту та завантажених класів. Крім того, він може виявляти так звані гарячі точки - частини програми, найчастіше внутрішні цикли, виконання яких займає найбільше часу. JIT-компіляція і динамічна перекомпіляція збільшують швидкість Java-додатків без втрати переносимості.

Реалізація рівня відображення. Для реалізації рівня відображення вибрано використання моделі MVC.

MVC це архітектурна модель, яка використовується при проектуванні та розробці програмного забезпечення. Ця модель ділить систему на три частини: модель даних, тип даних і елемент управління. Він використовується для відділення даних (моделі) від призначеного для користувача інтерфейсу (подання), щоб зміни в інтерфейсі мали мінімальний вплив на управління даними, а зміни в моделі даних можна було вносити без змінити призначений для користувача інтерфейс (15).

Метою моделі є гнучкий дизайн програмного забезпечення, який має полегшити подальші зміни або розширення програми, а також надати можливість повторного використання окремих компонентів програми. Крім того, використання цього шаблону у великих системах призводить до певного упорядкування їх структури і робить їх більш зрозумілими, одночасно знижуючи складність. Для реалізації цього паттерну буде використано HTML із використання JavaScript та середовища ExtJS (для клієнтської частини даного рівня) та середовище SpringMVC (для серверної частини даного рівня).

SpringMVC – це один із модулів універсального середовища Spring з відкритим вихідним кодом для Java-платформи. Також існує версія для платформи середовища .NET. Незважаючи на те, що Spring є широко поширеним в Java-співтоваристві головним чином як альтернатива і заміна моделі Enterprise JavaBeans. Spring надає велику свободу Java-розробникам в проектуванні, крім того, він надає добре документовані і легкі у використанні засоби вирішення проблем, що виникають при створенні додатків корпоративного масштабу[3].

Особливістю SpringMVC є:

- зрозумілий і прозорий поділ між шарами в MVC і запитах;
- стратегія інтерфейсів – кожен інтерфейс робить тільки свою частину роботи;
- інтерфейс завжди може бути замінений альтернативною реалізацією;

- інтерфейси тісно пов'язані з Servlet API;
- високий рівень абстракції для веб-додатків.

ExtJS – це бібліотека JavaScript для розробки веб-застосунків і користувацьких інтерфейсів, спочатку задумана як розширена версія Yahoo! UI Library, однак перетворена потім в окреме середовище. До версії 4.0 використовувала адаптери для доступу до бібліотек Yahoo! UI Library, jQuery або Prototype/script.aculo.us, починаючи з 4-ої версії адаптери відсутні. Підтримує технологію AJAX, анімацію, роботу з DOM, реалізацію таблиць, вкладок, обробку подій і всі інші нововведення Web 2.0.

Браузер. У ролі браузера, буде виступати будь-який сучасний браузер (Chrome, Firefox, Internet Explorer 9 і вище), що підтримують HTML версії 5 та JavaScript версії не нижче 1.6.

2.3. Вибір методу опрацювання біомедичних даних

Ознайомившись зі всіма доступними методами обробки медичних даних, було вирішено використотовувати статистичний метод, адже він є найбільш поширеним і простим у використанні, що дозволить досить швидко інтегрувати його в MIC і, відповідно, візуалізувати потрібні пацієнту дані.

Характерною особливістю будь-якого статистичного пакету є те, що він виробляє велику кількість інформації, що визначає результати статистичного аналізу. Майже всі статистичні пакети програм включають різноманітні інструменти для подання даних: графічне відображення, дво- та тривимірна графіка, а також часто різні інструменти ділової графіки. Характеристики основних пакетів аналізу та обробки даних представлені в табл. 2.1.

Пункти 1-4 таблиці показують переваги перед одним пакетом над іншим (рівень 1 найвищий).

Таким чином, пакет Statgraphics був розроблений для роботи в середовищі DOS, а потім був адаптований для операційної системи Windows і отримав нову назву Statgraphics Plus. За своїми характеристиками пакет займає проміжне місце між SPSS та Statistica.

Пакет SPSS призначений для "великих" машин і постійно перекладається для роботи в середовищі DOS, а потім у Windows. Пакет добре розроблений, підходить до можливостей професійних пакетів, а впровадження статистичних процедур відповідає практичній роботі.

Пакет Statistica розроблений спеціально для Windows. Він має найдосконаліший інтерфейс і багаті графічні можливості

Таблиця 2.1

Характеристики основних пакетів аналізу та обробки даних

Характеристика	Statgraphics Plus	SPSS	Statistica	Excel
Фірма	Manugistics	SPSS	StatSoft	Microsoft
Версія	3, 2	15 ,0	6,0	2007
Рік розробки	2003	2007	2007	2007
Рік 1 версії	1983	1975	1990	1996
Обсяг пакета, МБ	14,5	26,3	16,3	
Доступність	4	3	2	1
Русифікованість	_	-/+	-/+	+
Число процедур	>250	>250	>250	19
Простота освоєння	3	4	2	1
Література	+	—	+	+
Навчання на кафедрі	—	—	+	+
Зручність роботи	2	4	3	1
Візуалізація	2	3	1	4

Електронна таблиця Excel є найпоширенішою та найчастіше використовується при найпростішому статистичному аналізі даних. Важливою перевагою Excel є його українізація, а також її доступність, оскільки вона встановлюється автоматично під час встановлення MS Office. Тому пакет Excel найчастіше використовується для оформлення результатів роботи.

Слід зазначити, що всі ці пакети постійно оновлюються, і нові версії з’являються щороку.

При виборі пакету для аналізу даних є два аспекти:

- первинний вибір пакету аналізу;

- поточний вибір при переході на більш сучасний та потужний пакет. Кілька різних підходів у двох випадках.

Аналіз даних із використанням статистичного програмного пакету (разом із програмним пакетом) включає такі розділи:

- планування досліджень;
- підготовка даних для аналізу;
- вибір методу аналізу та реалізація;
- інтерпретація та предсталення результатів;

Враховуючи те, що пакет Excel є найпоширенішим і найдоступшіним з усіх предсатавлених аналогів, було вирішено використовувати саме його, як основний модуль опрацювання статистичних даних, оскільки, через його переваги час і простота інтеграції з розроблюваною МІС буде найменшою.

2.4. Висновки до розділу 2

В даному розділі було досліджено та визначено структурну схему архітектури клієнт-серверної системи. Було обрано технологію, засоби та середовища, що необхідні для реалізації розробленої архітектури, та виконано короткий опис по кожному із обраних засобів.

РОЗДІЛ 3 ПРОЕКТУВАННЯ МЕДИЧНОЇ ІНФОРМАЦІЙНОЇ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ

3.1. Схема роботи системи

Для реалізації поставленої задачі, проектування клієнт-серверної системи, було обрано трирівневу клієнт серверну систему, яка включає такі основні частини, як рівень відображення, прикладний рівень та рівень доступу до даних. Перевагою трирівневої (або ще її називають багаторівневої) клієнт-серверної архітектури над дворівневою полягає в тому, що в першій рівень доступу до даних відокремлений від прикладного рівня і дозволяє будувати гнучку систему, яка може працювати із різними середовищами зберігання даних (базами даних) або навіть декількома рівнями доступу до даних.

3.1.1. Схема роботи серверної частини. У відповідності до обраної архітектури клієнт-серверної системи, серверна частина розроблюваної системи повинна реалізовувати такі основні функціональні частини:

- рівень доступу до даних;
- прикладний рівень;
- рівень відображення.

На рис.3.1. приведена структурна схема роботи серверної частини. Робота серверної частини розпочинається після завантаження веб-серверу на якому буде розгорнуто спроектована система. Після того як система стартувала, відбувається початкове налаштування усіх модулів та компонентів необхідних для ефективної і коректної роботи системи. Коли все налаштовано відбувається запуск прослуховування на запити від клієнта. За цю частину відповідає модуль взаємодії із клієнтом.

Модуль взаємодії із клієнтом – відповідає за обмін даними між сервером та клієнтом. Після початку роботи системи, він приймає усі запити користувача, як синхронні так і асинхронні (AJAX) запити. Для цієї взаємодії використовується протокол HTTP. Усі запити отримані цим модулем передаються далі в систему, для подальшого опрацювання.

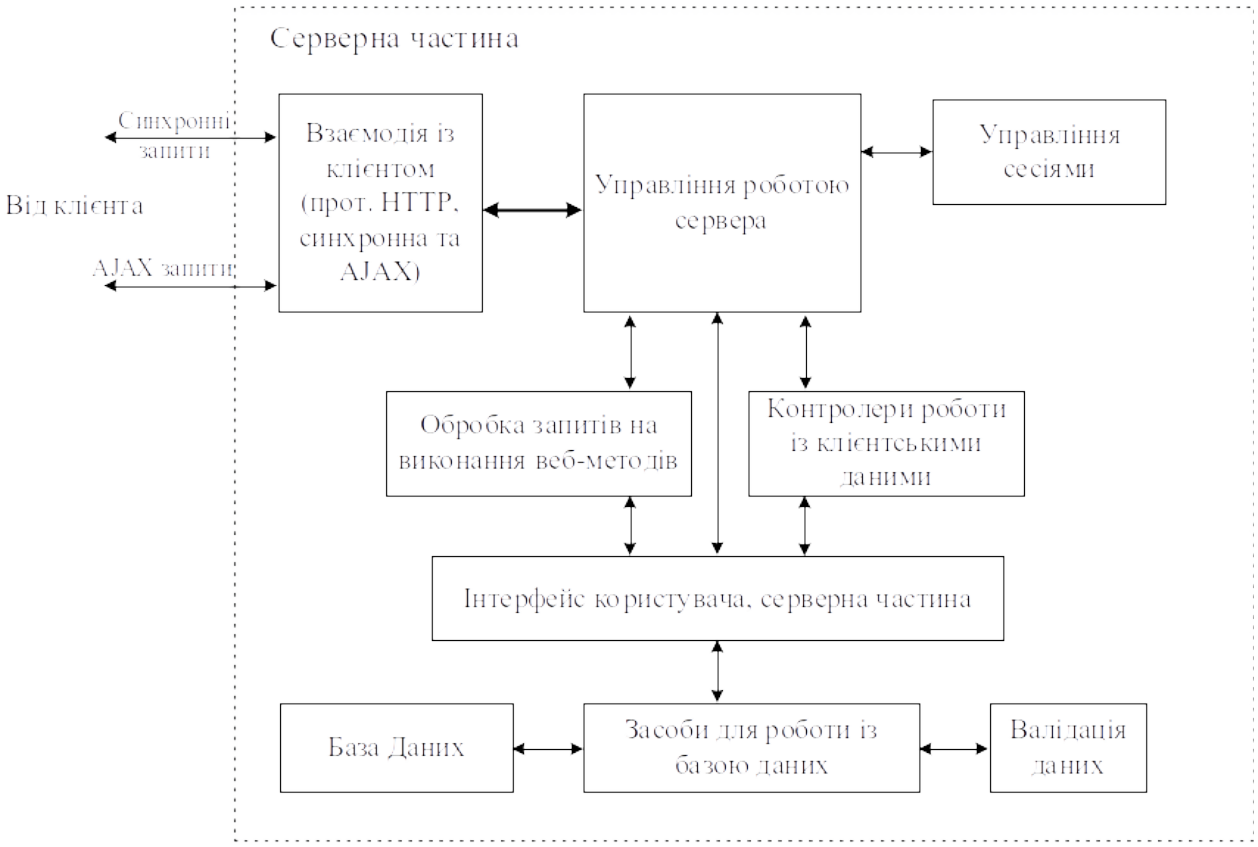


Рис. 3.1. Структурна схема роботи серверної частини.

Модуль управління роботою сервера – є наступним елементом системи, він приймає запити від попереднього модулю та визначає куди потрібно далі передати отриманий запит. Модуль управління роботою серверу і модуль взаємодії із клієнтом є ядром серверної частини, і основні функціональні можливості цього ядра забезпечуються веб-сервером на якому розгорнуто розроблювану систему.

Модуль обробки запитів на виконання веб-методів – призначений для опрацювання клієнтських запитів на виконання веб-методів. Веб-метод це запит до серверу на виконання певної дії та повернення результатів виконання цієї дії, результат що повертається конвертується до певного формату (JSON або XML). Даний модуль необхідний для того щоб виконувати такі функції, як: авторизація користувача, валідація даних, отримання додаткових даних, отримання інформації про користувача, створення/закриття сесії користувача та інші.

Модуль контролери роботи із клієнтськими даними – призначений для отримування, опрацювання, відсилання клієнтських даних. Саме даний модуль

забезпечує роботу користувацьких форм на клієнті. Основне завдання цього модуль це отримання клієнтського запиту, аналіз вхідних даних і вразі, якщо це запит на отримання інформації, то у відповідь контролер повертає усі необхідні дані для відображення запитуваної інформації. Якщо це запит на збереження, то контролер робить вибірку усіх даних що були йому надіслані від клієнта та передає їх іншому модулю для збереження.

Сукупність модуля взаємодії із користувачем, модуля виконання веб-методів та модуля контролери роботи із клієнтськими даними, реалізують рівень відображення на серверній частині.

Модуль управління сесіями – призначений для створення, видалення та роботи із сесіями користувача. Основне завдання даного модуля це забезпечення зберігання проміжних даних користувача та даних про самого користувача, та про його повноваження та можливості в контексті даної системи. Робота даного модулю розпочинається після авторизації клієнта в систему. Під час авторизації для кожного користувача створюється окрема сесія, в яку заносяться усі дані про користувача, дії які він виконує в продовж даної сесії та проміжні дані для їх тимчасового збереження і запобігання втраті інформації вразі якогось збою чи то на клієнті чи на сервері.

Модуль інтерфейс користувача – це є найголовніший модуль системи, саме в ньому міститься усі бізнес логіка системи. Даний модуль комунікує майже із усіма складовими частинами системи, забезпечує реалізацію усіх алгоритмів серверної частини, та деяких клієнтських.

Саме модуль інтерфейс користувача реалізує прикладний рівень, серверної частини.

Модуль засоби роботи із БД – призначений для роботи із базою даних, він забезпечує абстракцію роботи із БД. Тобто усі вищі рівні при звертанні до БД не потребують інформації, що за БД використовується в даній системі та як із нею працювати, все це відомо лише даному модулю. Основною перевагою використання такого модулю – це забезпечення незалежності від середовища зберігання даних. Із використання такого підходу, дані можна зберігати будь де і в будь якій формі.

Модуль засоби роботи із БД реалізує рівень доступу до даних, серверної частини.

Модуль валідація даних – призначений для виконання перевірки даних на відповідність до уже існуючих даних у БД, а також нових даних на коректність, валідність. Даний модуль є останнім етапом перевірки даних перед записом даних до БД.

3.1.2. Схема роботи клієнтської частини. У відповідності до обраної архітектури клієнт-серверної системи, клієнтська частина повинна реалізовувати лише одну функціональну частину, це рівень відображення.

На рис.3.2. приведено структурну схему роботи клієнтської частини. Головною особливістю даної частини є те що вона насправді знаходиться на сервері а робота з нею відбувається через веб-браузер. Робота клієнтської частини розпочинається після введення веб-адреси сервера (наприклад http://localhost:8080/), веб-браузер використовуючи протокол обміну HTTP, відсилає запит до сервера та у відповідь отримує початкову веб-сторінку із HTML розміткою та завантажує додаткові CSS та JavaScript файли, що підключені до веб-сторінки. Після завантаження усіх файлів розпочинається робота клієнта, тепер користувач може авторизуватися і виконувати доступні для нього дії.

Модуль взаємодії із сервером – призначений для виконання запитів до сервера на отримання, відсилання та валідації даних. Даний модуль підтримує два варіанти взаємодії: синхронна та асинхронна (AJAX). Основна відмінність між цими видами взаємодії полягає в тому, що при синхронній потрібно повністю оновлювати веб-сторінку, а при AJAX дані відсилаються та приймаються у фоновому потоці і відображаються автоматично після їх отримання. Використання асинхронної взаємодії дозволяє позбутися необхідності постійно оновлювати веб-сторінку і не завантажувати кожного разу усі файли із серверу.

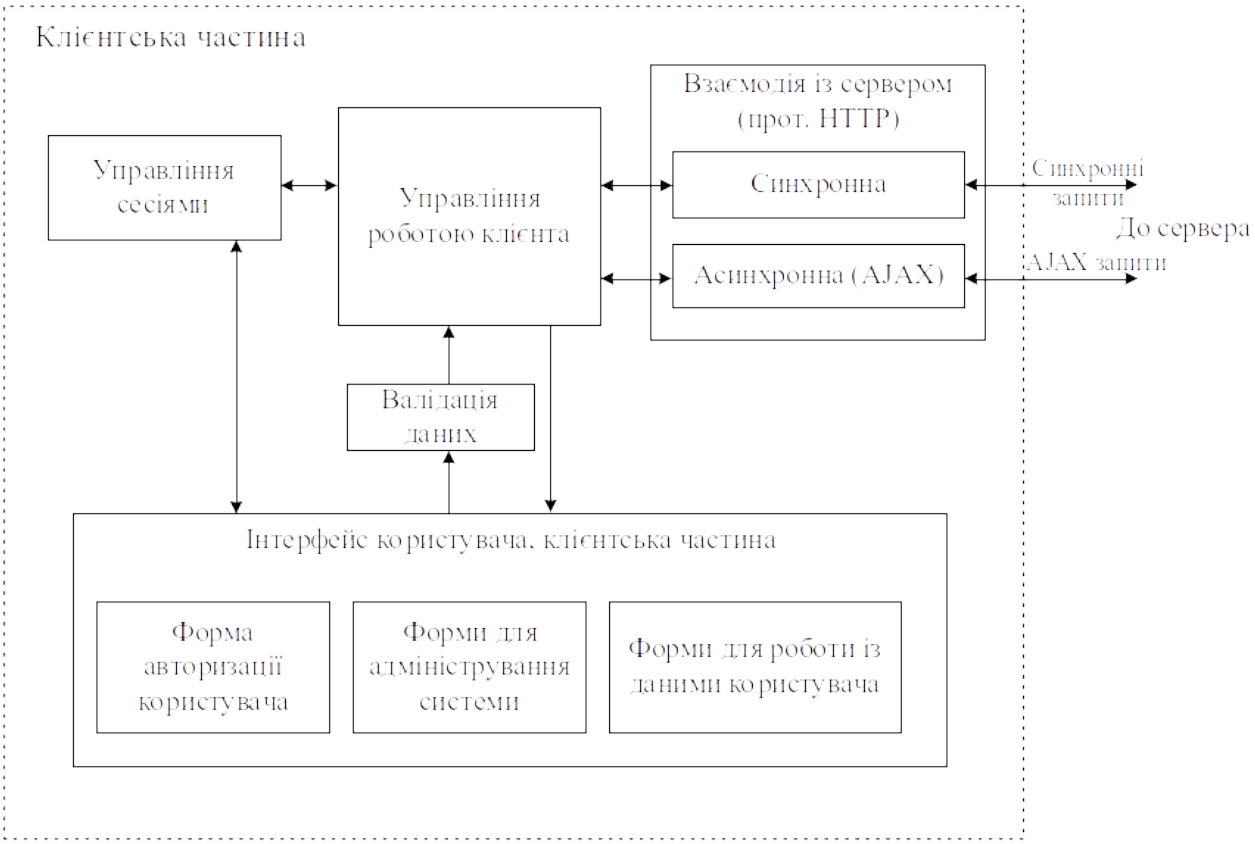


Рис. 3.2. Структурна схема роботи клієнтської частини.

Модуль управління роботою клієнта – призначений для керування клієнтською частиною. Реалізований за допомогою JavaScript, він організовує усю роботу та керує взаємодією із сервером.

Модуль інтерфейс користувача – призначений для інтерактивної взаємодії системи із користувачем. Він містить у собі усі засоби для відображення, отримання та редагування даних користувача. На рис.3.3. зображено структурну схему інтерфейсу користувача. Модуль інтерфейс користувача містить у собі набір форм: для авторизації користувача, для адміністрування системи (це форми для створення/редагування/видалення користувача, форми для роботи із ролями користувача, для зміни налаштувань системи), для роботи із даними користувача (це форми для вводу/редагування/видалення даних про пацієнта, лікаря, візит, обстеження та інші.).

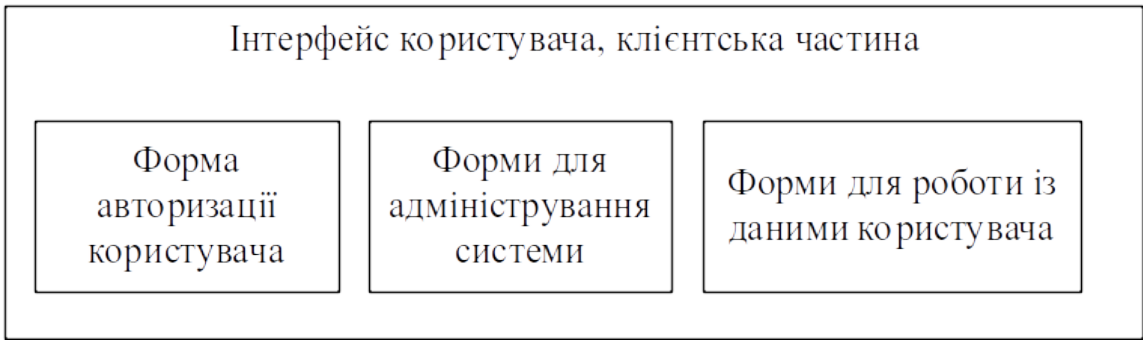


Рис. 3.3. Структурна схема модулю інтерфейс користувача.

Сукупність модуля управління роботою клієнта та модулю інтерфейс користувача реалізує рівень відображення клієнтської частини.

Модуль управління сесіями – призначений для управління сесією користувача на клієнтській частині. Він дозволяє зберігати на клієнтській частині даних про користувача та про сесію на серверній частині. Саме через цей модуль відбувається збереження проміжних даних користувача та їх відновлення вразі збою.

Модуль валідація даних – призначений для виконання валідації даних користувача. Даний модуль, для валідації використовує засоби як клієнтської частини, так і виконує звертання до серверної частини. Основне його призначення на клієнтській частині це візуальне відображення помилковості чи не коректності введених даних та забезпечення цілісності даних на інших рівнях системи.

3.2. Структура бази даних

У відповідності до обраної архітектури клієнт-серверної системи, ще однією частино даної системи є база даних. В даному проекті було вирішено використовувати сторонню реляційну базу даних MySQL.

Для коректної роботи розроблюваної системи необхідно в зручній формі виконувати збереження, читання та редагування даних. Для цього не достатньо просто обрати хороший сервер БД, необхідно правильно спроектувати структуру БД і встановити коректні зв’язки між таблицями, тобто використовувати нормалізацію БД.

Нормалізування схема бази даних - покрокова мережева інтеграція (виконуючи: таблиці) відповідно до алгоритму перевірки кількох посилань на основі застосовних залежностей [14].

Звичайна форма це властивість відношення реляційної моделі даних, що характеризує її з точки зору надмірності, що потенційно може призвести до логічно помилкових результатів вибірки або змін даних. Звичайна форма визначається як сукупність вимог, яким повинні відповідати відносини. Таким чином, схема реляційної бази даних переходить у першу, другу, третю і так далі звичайні форми. Якщо відношення задовольняє критеріям нормальної форми n та всіх попередніх нормальних форм, то це відношення вважається нормальною формою рівня n. Для даної системи в загально доцільно обрати третю нормальну форму, але при певних ситуація де потрібна вища швидкодія, обране правило може порушуватись.

Для роботи із користувачами системи, буде використовуватись таблиця AP_USERS (див рис.3.4.) вона мітить такі колонки:

- AP_USER_ID – Ключова колонка числового типу, забезпечує унікальність записів у табличці;

- AP_USER_LOGIN – Колонка текстового типу, зберігає коротке ім’я користувача та використовується при авторизації. Кожен запис в цій колонці повинен бути унікальним;

- AP_USER_PASSW – Колонка текстового типу, зберігає пароль користувача, попередньо перетворений в хеш код функцією md5(sha());

- AP_USER_PERSON – Колонка числового типу, є зовнішнім ключем та використовується для зв’язку із таблицею AP_PERSONS;

- AP_USER_ROLE – Колонка числового типу, є зовнішнім ключем та використовується для зв’язку із таблицею AP_ROLES;

- AP_USER_ACTV – Колонка типу біт, зберігає значення 1 якщо користувач активний та 0 якщо користувач неактивний;

- AP_USER_WEMAIL – Колонка текстового типу, зберігає робочу електронну адресу користувача. Кожен запис в цій колонці повинен бути унікальним.

Таблиця (Table)

AP_USERS	
Ім'я поля	Тип даних
AP_USER_ID	INT(255)
AP_USER_LOGIN	VARCHAR(10)
AP_USER_PASSW	VARCHAR(30)
AP_USER_PERSON	INT(255)
AP_USER_ROLE	INT(255)
AP_USER_ACTV	BIT
AP_USER_WEMAIL	VARCHAR(30)

Рис. 3.4. Структура таблиці AP_USERS.

Для зберігання додаткової інформації про користувачів буде використовуватись таблиця AP_PERSONS (див рис.3.5.) вона містить такі колонки:

- AP_PERSON_ID – Ключова колонка числового типу, забезпечує унікальність записів у табличці;

- AP_PERSON_L_NAME – Колонка текстового типу, зберігає прізвище користувача;

- AP_PERSON_F_NAME – Колонка текстового типу, зберігає ім’я користувача;

- AP_PERSON_M_NAME – Колонка текстового типу, зберігає ім’я по батькові користувача;

- AP_PERSON_INFORM – Колонка текстового типу, зберігає додаткову інформацію про користувача в форматі “[Тип інформації]:[Інформація]” елементи між собою розділяються комою.

Таблиця (Table)

AP_PERSONS	
Ім'я поля	Тип даних
AP_PERSON_ID	INT(255)
AP_PERSON_L_NAME	VARCHAR(30)
AP_PERSON_F_NAME	VARCHAR(30)
AP_PERSON_M_NAME	VARCHAR(30)
AP_PERSON_INFORM	VARCHAR(MAX)

Рис. 3.5. Структура таблиці AP_PERSONS.

Для зберігання ролей користувачів буде використовуватись таблиця AP_ROLES (рис.3.6.) вона містить такі колонки:

- AP_ROLE_ID – Ключова колонка числового типу, забезпечує унікальність записів у табличці;

- AP_ROLE_ROL_ID – Колонка текстового типу, зберігає текстовий код ролі користувача. Кожен запис в цій колонці повинен бути унікальним;

- AP_ROLE_SEC_OPT – Колонка текстового типу, зберігає через кому числовий номер опції, яка є доступною для даної ролі;

- AP_ROLE_DESCR – Колонка текстового типу, зберігає опис ролі користувача.

Таблиця (Table)

AP_ROLES	
Ім'я поля	Тип даних
AP_ROLE_ID	INT(255)
AP_ROLE_ROL_ID	VARCHAR(10)
AP_ROLE_SEC_OPT	VARCHAR(MAX)
AP_ROLE_DESCR	INT(MAX)

Рис. 3.6. Структура таблиці AP_ROLES.

Для зберігання доступних для користувача елементів меню буде використано таблиця AS_MENU_ITEMS (рис.3.7.) вона містить такі колонки:

- AS_MENU_ITEM_ID – Ключова колонка числового типу, забезпечує унікальність записів у табличці;

- AS_MENU_ITEM_FILE – Колонка текстового типу, зберігає назву фала із картинкою, для графічного відображення елемента меню.

- AS_MENU_ITEM_ACTI – Колонка текстового типу, зберігає назву методу, що відповідає за виклик обробника виконання подій для даного елемента меню.

- AS_MENU_ITEM_TITL – Колонка текстового типу, зберігає коротку назву елемента меню.

Таблиця (Table)

AS_MENU_ITEMS	
Ім'я поля	Тип даних
AS_MENU_ITEM_ID	INT(255)
AS_MENU_ITEM_FILE	VARCHAR(30)
AS_MENU_ITEM_ACTI	VARCHAR(30)
AS_MENU_ITEM_TITL	VARCHAR(255)

Рис. 3.7. Структура таблиці AS_MENU_ITEMS.

Для реалізації зв’язку таблиці AP_ROLES із AS_MENU_ITEMS за типом багато до багатьох потрібно використати додаткову таблицю AP_MENU_ITEMS_LINK (рис.3.8.) із такими колонками:

- AP_ROLE_ID – зовнішній ключ на таблицю AP_ROLES;

- AS_MENU_ITEM_ID – Зовнішній ключ на таблицю AS_MENU_ITEMS.

Таблиця (Table)

AP_MENU_ITEMS_LINK	
Ім'я поля	Тип даних
AP_ROLE_ID	INT(255)
AS_MENU_ITEM_ID	INT(255)

Рис. 3.8. Структура таблиці AP_MENU_ITEMS_LINK

Для зберігання інформації про пацієнтів буде використовуватися таблиця AP_PATIENTS (див рис.3.9.) вона містить такі колонки:

- AP_PATIENT_ID – Ключова колонка числового типу, забезпечує унікальність записів у табличці;

- AP_PATIENT_FNAME – Колонка текстового типу, зберігає ім’я пацієнта;

- AP_PATIENT_LNAME – Колонка текстового типу, зберігає прізвище пацієнта;

- AP_PATIENT_MNAME – Колонка текстового типу, зберігає ім’я по батькові пацієнта;

- AP_PATIENT_BDATA – Колонка текстового типу, зберігає дату народження пацієнта;

- AP_PATIENT_ADDR – Колонка текстового типу, зберігає адресу пацієнта;

- AP_PATIENT_PHONE – Колонка текстового типу, зберігає номер основного телефону;

- AP_PATIENT_PHONET – Колонка текстового типу, зберігає тип номеру основного телефону;

- AP_PATIENT_WPHONE – Колонка текстового типу, зберігає номер робочого телефону пацієнта;

- AP_PATIENT_WPHONET – Колонка текстового типу, зберігає тип номеру робочого теефону.

Таблиця (Table)

AP_PATIENTS	
Ім'я поля	Тип даних
AP_PATIENT_ID	INT(255)
AP_PATIENT_FNAME	VARCHAR(30)
AP_PATIENT_LNAME	VARCHAR(30)
AP_PATIENT_MNAME	VARCHAR(30)
AP_PATIENT_BDATA	DATA
AP_PATIENT_ADDR	VARCHAR(255)
AP_PATIENT_PHONE	VARCHAR(15)
AP_PATIENT_PHONET	VARCHAR(10)
AP_PATIENT_WPHONE	VARCHAR(15)
AP_PATIENT_WPHONET	VARCHAR(10)

Рис. 3.9. Структура таблиці AP_PATIENTS.

Детальна структурна схема бази даних із зображенням залежностей між таблицями приведена в графічній частині.

3.3. Програмна модель сервера

Програмна модель – це засоби які доступні програмісту для реалізації програм. Програмна модель розроблюваного серверу приведена на рис.3.10.

На найнижчому рівні програмної моделі знаходиться операційна система, в нашому випадку із використання технології Java EE та мови програмування Java, ОС може бути будь-якою (Windows, Unix/Linux, MAC OS/OS X та інші існуючі), це нам забезпечує наступний рівень програмної моделі JVM. JVM – є віртуальною машиною Java, яка перетворює скомпільований байт-код Java, у системні інструкції цільової ОС. Обравши таку технологію ми забезпечили платформну незалежність для сервера, хоча можливо дещо знизили швидкодію роботи сервера.

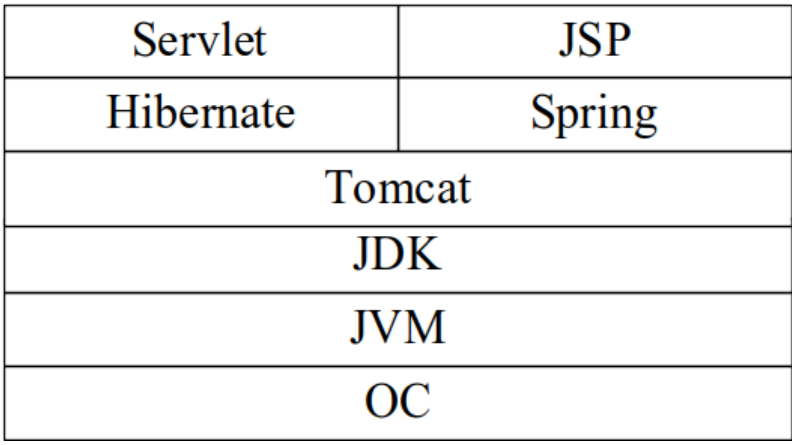


Рис. 3.10. Програмна модель сервера.

Наступним елементом програмної моделі є JDK, він забезпечує усіма необхідними засобами середовищем для написання програм на мові Java. Для даного проекту було обрано JDK версії 8.

Tomcat – є ще одним елементом програмної моделі. Tomcat дозволяє запускати веб-додатки, містить ряд програм для само конфігурування. Tomcat використовується в якості самостійного веб-сервера, як сервер контенту в поєднанні з веб-сервером Apache HTTP Server, а також в якості контейнера сервлетів в

серверах додатків JBoss і GlassFish. В нашому випадку він буде використовуватись як самостійний веб-сервер для запуску розроблюваної серверної частини [18].

Наступним рівнем програмної моделі є Hibernate та Spring, вони дають змогу працювати із базою даних та запитами від клієнта відповідно.

Найвищим рівне програмної моделі є Servlet та JSP. Даний рівень забезпечує безпосередню комунікацію із клієнтською частино. На даному рівні формуються веб-сторінки, передаються необхідні дані та отримуються відповідно.

3.4. Алгоритми роботи системи

Після того як було спроектовано архітектура та схема роботи клієнт-серверної системи, необхідно спроектувати основні алгоритми роботи системи.

Проаналізувавши завдання та вихідні умови до роботи було визначено такі основні алгоритми роботи клієнт-серверної системи «Стаціонар лікарні»:

- загальний алгоритм роботи системи;
- авторизація та створення сесії користувача;
- створення користувача;
- створення пацієнта;
- створення візиту;
- скерування на скерування для пацієнта.

3.4.1. Загальний алгоритм роботи системи. На рис.3.11. приведена схема алгоритму роботи системи. Після старту системи, сервер прослуховує мережу і приймає усю запити що направлені до нього. Після того як веб-браузер надішле перший запит до сервера на отримання головної сторінки, сервер у відповідь відправить веб-сторінку із HTML розміткою, далі веб-браузер відобразить сторінку із формою входу користувача до системи.

Після того як на екрані з’явиться форма входу в систему, користувач може авторизуватись в системі та розпочати свою роботу. Для авторизації потрібно ввести скорочене ім’я користувача та відповідний пароль. Після натиснення кнопки увійти, ім’я та пароль відсилаються на сервер для перевірки. Якщо користувача із такими даними немає то у відповідь сервер поверне помилку і на клієнті відобразиться

інформація про невірно введені дані користувача. Якщо ім’я та пароль були введені правильно, то сервер спробує знайти останню незавершену сесію користувача та спробує відновити її, вразі помилки чи при відсутності такої сесії, системою буде згенеровано нова сесія користувача та збережена до бази даних.

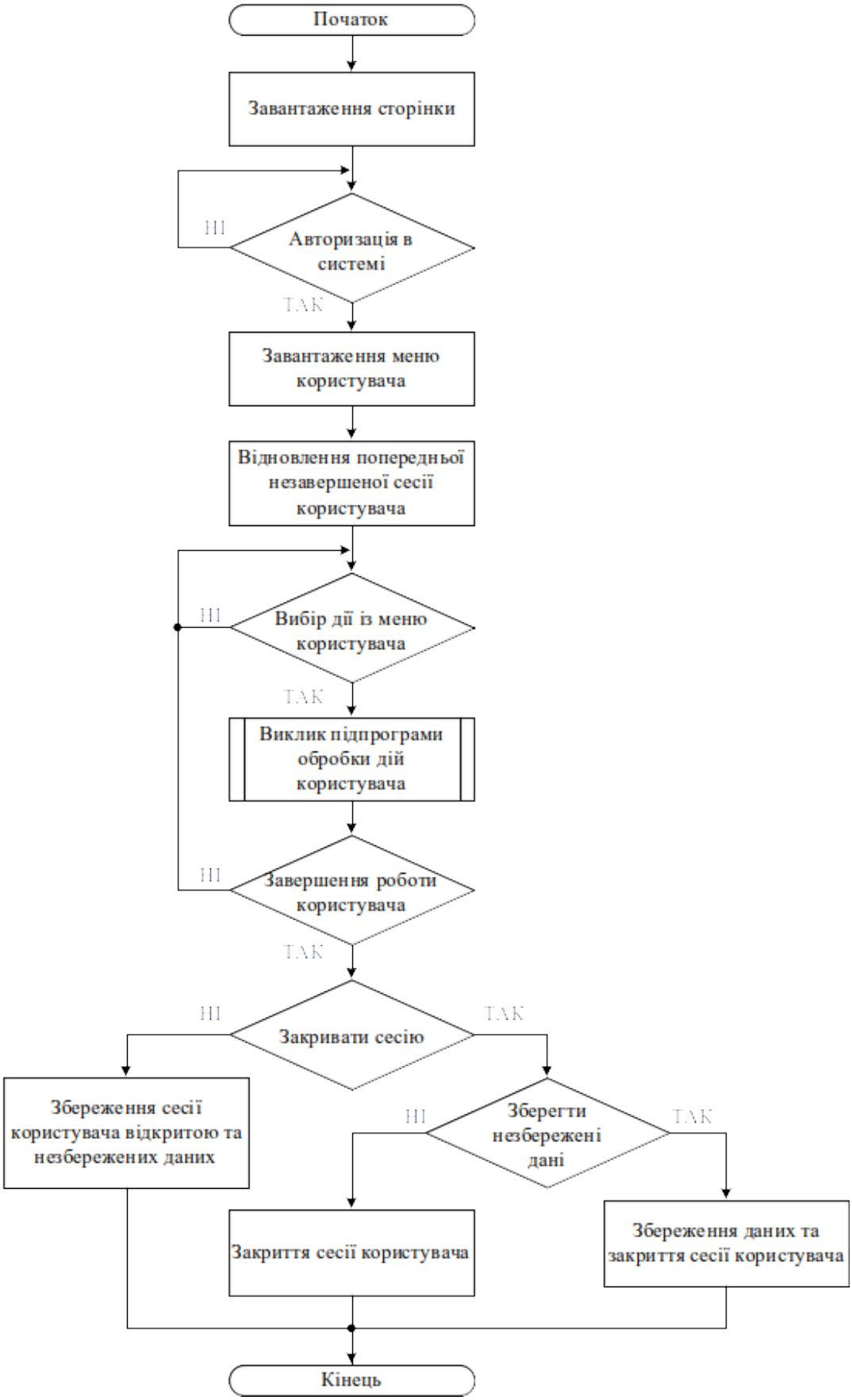


Рис. 3.11. Схема алгоритму роботи системи.

Після того як сесія буде знайдена або створена, із неї буде отримано роль користувача в системі і у відповідності із роллю користувача буде створено головне меню. Після того як сесія і меню створено, вони будуть надіслані клієнту, який їх потім відобразить, саме з цього моменту розпочнеться безпосередня взаємодія користувача із системою.

Маючи головне меню, користувач може обрати будь яку дію, яка є доступною йому в межах тієї ролі яку він виконує, наприклад адміністратор може створити нового користувача, головний менеджер має доступ до усіх існуючих дій і т. д..

Обравши певну дію викликається відповідний обробник подій, та виконує обрану діє, наприклад відображає форму створення користувачів. Після того як користувач завершить обрану дію, він може виконати наступну або завершити роботу взагалі.

Коли користувач виконує дію завершити роботу він має вибір закривати чи не закривати сесію. Якщо користувач бажає наступного разу розпочати роботу з тої дії, що він виконував, він може не закривати сесію і при наступному разі для нього буде відновлено попередній стан програми. Якщо користувач зробив все, що йому було необхідно, але для прикладу не виконав збереження результатів, то система запитає його чи хоче він зберегти зміни. Потім відбувається закриття сесії або спочатку збереження даних а потім уже закриття сесії і сторінки в браузері користувача.

3.4.2. Алгоритм авторизації та створення сесії користувача. Одним із головних алгоритмів роботи із системою є алгоритм авторизації, що забезпечує неможливість сторонніх людей чи зловмисників увійти в систему та виконувати несанкціоновані дії та псувати дані. На рис.3.12. приведено схема алгоритму авторизації користувача в системі.

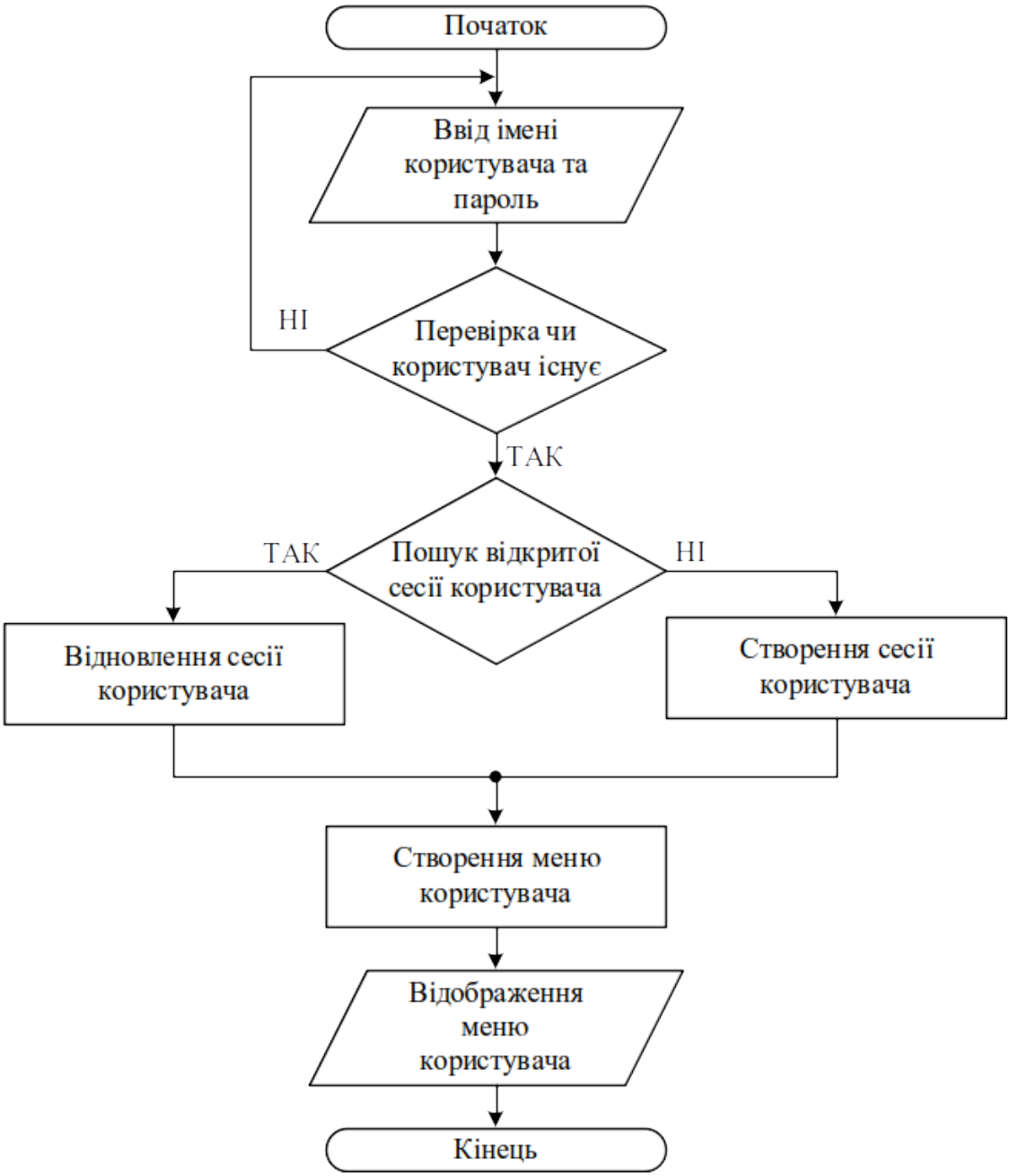


Рис. 3.12. Схема алгоритму авторизації та створення сесії користувача.

Після того як головна сторінка завантажилась, користувачу є доступним увійти в систему. Для цього необхідно ввести свої ім’я користувача та пароль і натиснути кнопку увійти. Далі система відсилає запит на сервер для перевірки чи такий користувач існує.

Якщо користувача із такими даними немає то у відповідь сервер поверне помилку і на екрані клієнта відобразиться інформація про невірно введені дані користувача. Якщо ім’я та пароль були введені правильно, то сервер спробує знайти останню незавершену сесію користувача та спробує відновити її, вразі помилки чи

при відсутності такої сесії, системою буде згенеровано нова сесія користувача та збережена до бази даних.

Після того як сесія буде знайдена або створена, із неї буде отримано роль користувача в системі і у відповідності із роллю користувача буде створено головне меню. Далі меню буде надіслано до клієнта де воно буде відображено. З цього моменту рахується, що клієнт авторизувався в системі.

3.4.3. Алгоритм створення нового користувача. Ще одним із головних завдань системи, це забезпечення можливості зручного та швидкого способу створення клієнтів, для цього розроблено алгоритм створення клієнтів, схема алгоритму створення нового користувача приведена на рис.3.13.

Після того як головне меню користувача було завантажено, він має доступ до усіх доступних для нього дій.

Обравши дію створити нового користувача, відобразиться форма із переліком існуючих користувачів, в під заголовком форми буде панель меню із такими діями, як: додати користувача, редагувати та видалити із системи.

Для того щоб створити нового користувача, необхідно натиснути кнопку додати нового користувача. У відповідь на цю дію відкриється форма із полями для вводу інформації про користувача, вона розділена на дів вкладки:

- Загальна інформація: коротке ім’я користувача, пароль, електронна адреса, роль в системі, статус (активований / не активований).

- Персональна інформація: повне ім’я користувача, розташування в лікарні, номер телефону, номер екстреного телефону, адреса проживання, та інша додаткова інформація.

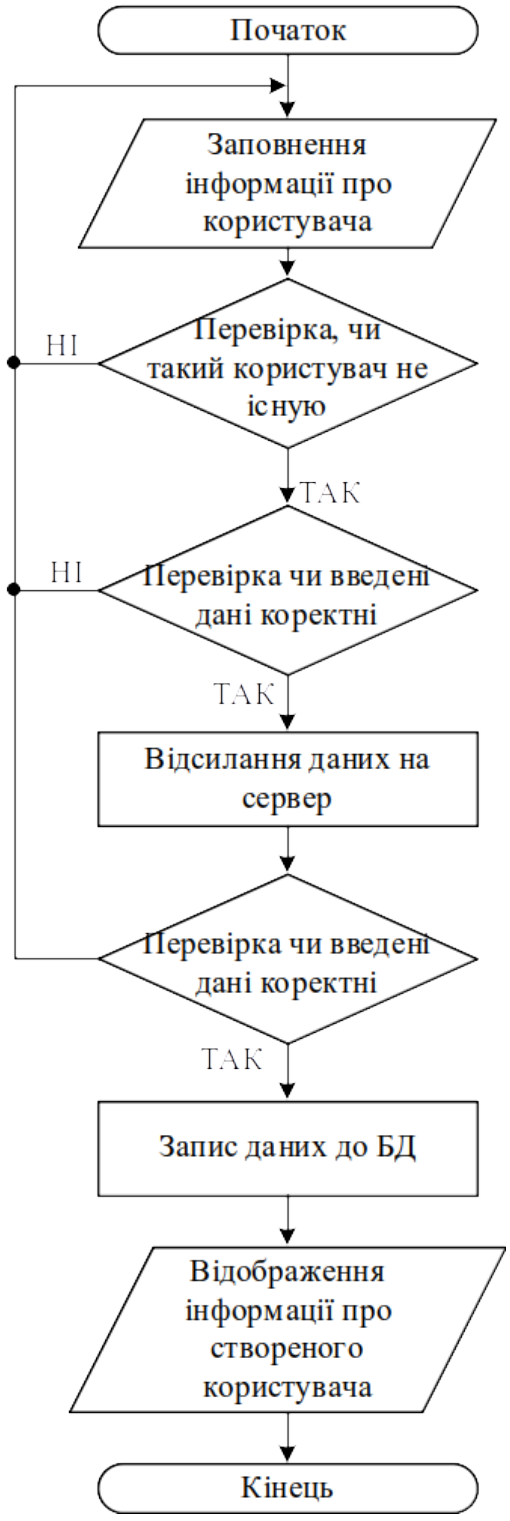


Рис. 3.13. Схема алгоритму створення нового користувача.

Після того як усі необхідні дані введено, потрібно натиснути на кнопку «зберегти». Спочатку після натиснення на кнопку «зберегти» система перевіряє чи нема такого користувача уже у системі, якщо користувач із такими скороченим ім’ям та електронною адресом уже існує то система проінформує про це і

запропонує змінити ці дані на унікальні або взагалі відмінити процес створення нового користувача.

Коли система підтвердить що такого користувача ще немає, відбувається перевірка введених даних на клієнті перед відсиланням на сервер. Після того як клієнт відіслав пакет даних на сервері вони знову перевіряються на наявність помилок. Якщо наприклад виникне ситуація, коли в системі декілька адміністраторів намагаються створити користувача із однаковим коротким ім’ям та електронним адресом, то той хто перший надішле запит на створення до сервера той пройде перевірку даних на сервері, а іншому буде виведено інформацію про те що такий користувач уже існує. Наявність декількох рівнів контролю даних забезпечує цілісність структур даних та виключає додаткове дублювання однакових даних.

Коли дані пройдуть усі перевірки вони будуть внесенні до бази даних, і повернуться назад на клієнт з інформацією що дані були успішно збережені.

3.4.4. Алгоритм створення нового пацієнта. Ще одним схожим алгоритмом роботи системи, до попередньо описаного є створення пацієнта. Головною відмінністю є те що про пацієнта вводиться в декілька разів більше інформації ніж про користувача. Окрім того кожен користувач може бути в свою чергу пацієнтом, в цьому випадку деякі дані підтягуються автоматично.

На рис.3.14. приведена схема алгоритму створення нового пацієнта. Початок створення пацієнта, майже такий самий, як і створення користувача. Необхідно обрати відповідну дію на панелі меню, у формі що відкрилась вказати усі необхідні дані про пацієнта, та натиснути кнопку «зберегти». Після цього на сервері створюється трансакція, яка буде слідкувати за цілісністю даних до завершення дії створення нового пацієнта.

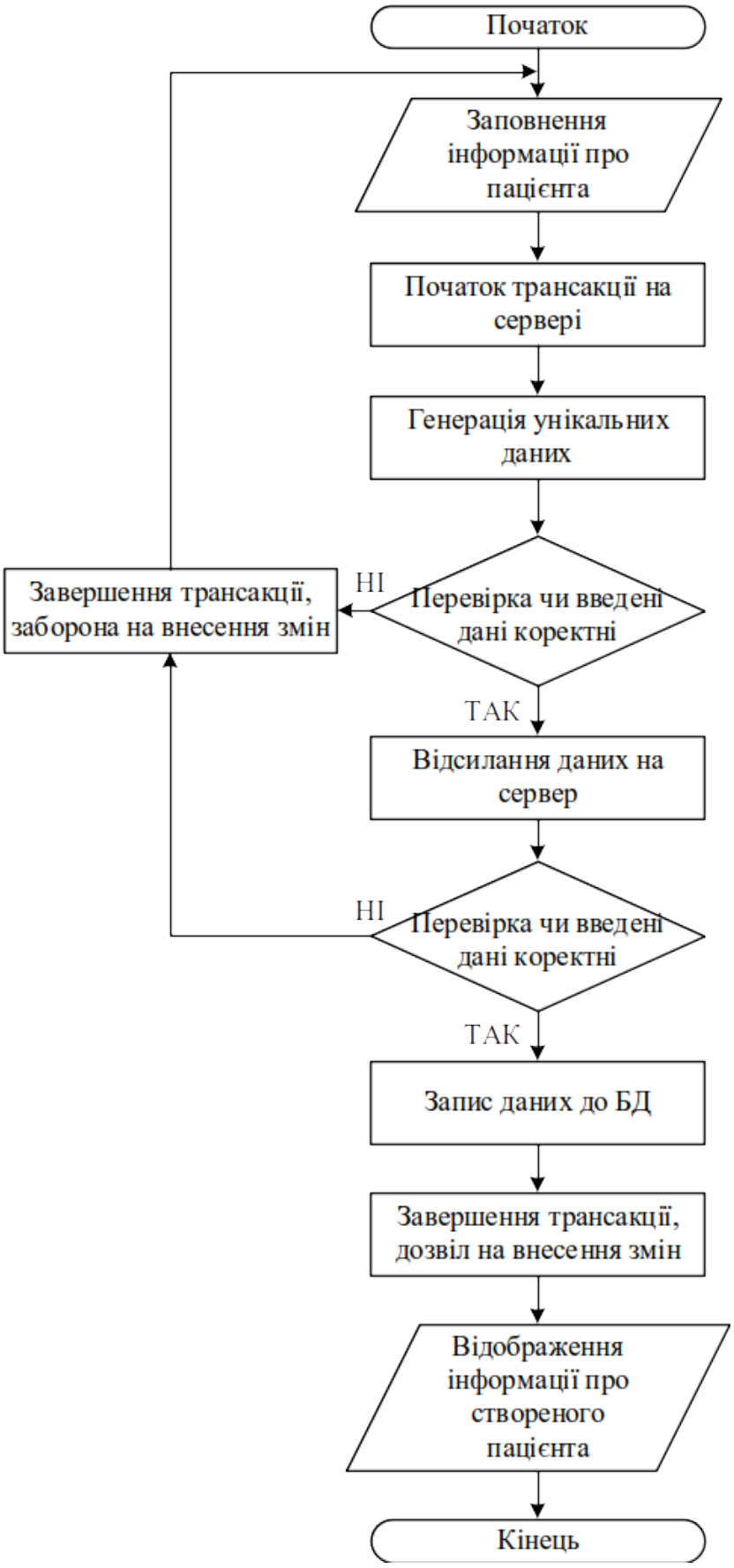


Рис. 3.14. Схема алгоритму створення нового пацієнта.

Після цього відбувається додаткова генерація унікальних ключових даних, для подальшої швидкої ідентифікації пацієнта, це є номер медичного запису, номер страхового медичного запису, та унікальний ідентифікатор що будується на основі паспортних даних (не є обов'язковим при створенні, але необхідний для подальшої ідентифікації).

Коли необхідні дані згенеровано відбувається перевірка введених даних на клієнтські частині, коли все вірно і немає дублювання унікальних даних, дані пакетом відсилаються на сервер, де вони приймаються і ще раз перевіряються. Коли виникають якісь помилки чи невідповідності даних, трансакція призупиняється і клієнт нотифікує користувача про помилку вводу даних чи збій в системі, а усі згенеровані додаткові дані видаляються а тригери баз даних повертаються до попереднього стану.

По завершенню усіх перевірок дані зберігаються до бази даних а трансакція завершується, підвереджуючи дозвіл для збереження введених даних. Як результат успішного збереження, клієнту відсилаються, щойно збережені дані для відображення та повідомлення про успішне завершення дії.

3.4.5. Алгоритм створення нового візиту. Наступним по важливості алгоритмом після створення нового пацієнта, є алгоритм створення нового візиту для пацієнта. На рис.3.15. приведена схема алгоритму створення нового візиту для пацієнта.

Після вибору дії, «створення візиту», в веб-сторінці відкривається форма додавання нового візиту. При відкриті форми відображається список існуючих пацієнтів, який можна фільтрувати та сортувати за різними критеріями. Щоб створити візит потрібно обрати існуючого пацієнта, після чого кнопка «додати візит» стає доступною, якщо це новий пацієнт то увесь час (в залежності від ролі користувача) доступна кнопка «створити пацієнта».

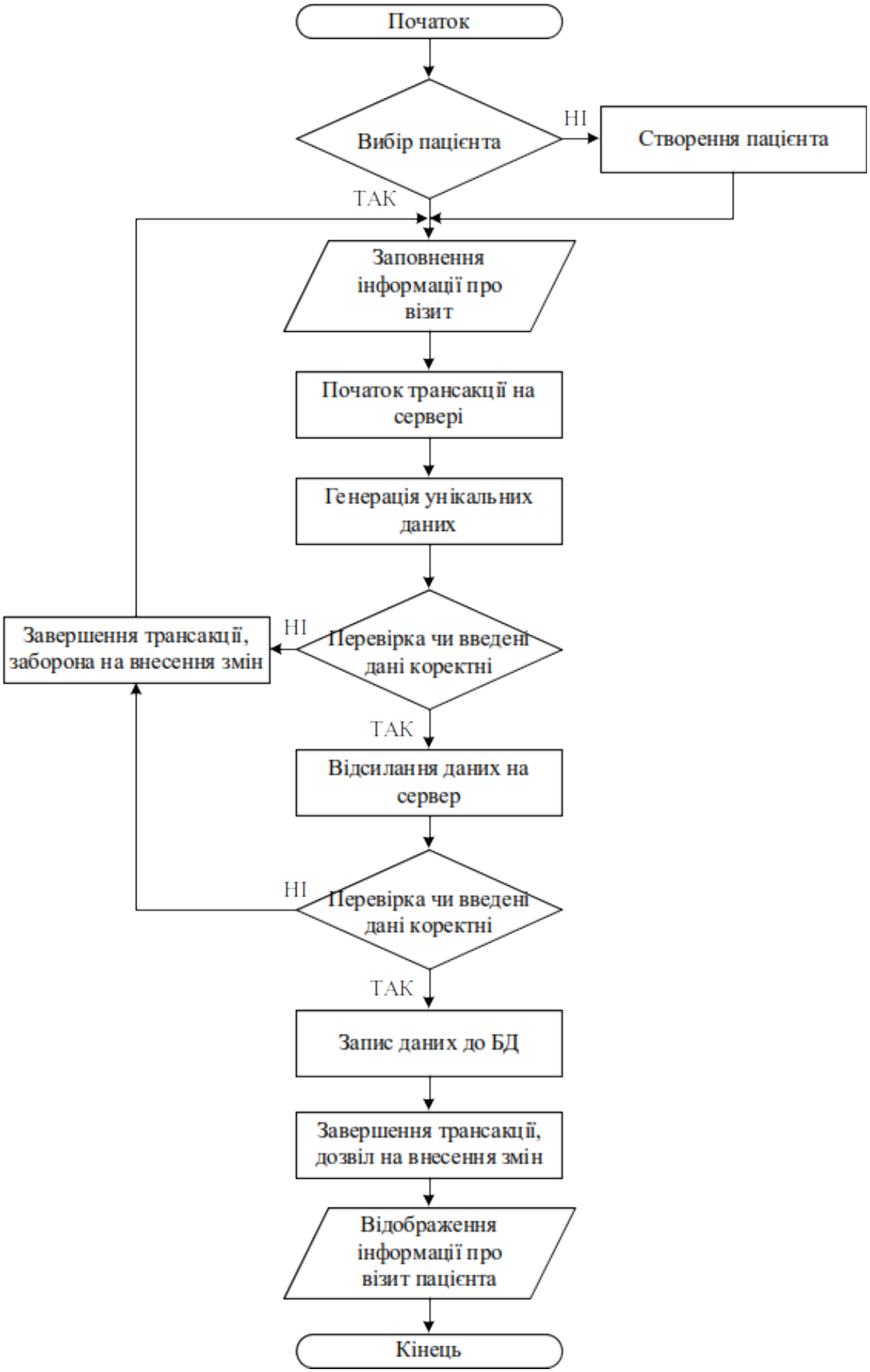


Рис. 3.15. Схема алгоритму створення нового візиту.

Після того як пацієнта було обрано або створено, відкривається нова форма, яка складається з двох частин:

- Верхньої. Тут відображається в дещо стислій формі головна інформаціє про пацієнта.

- Нижньої. Тут відображаються поля для заповнення інформації про візит.

Після введення даних про візит (також тут може бути змінена деяка інформація про пацієнта), відбувається створення трансакції, яка буде відбувати за цілісність даних під час даної операції. Потім відбувається запит на генерацію додаткових унікальних даних (такі як номер візиту, розрахунковий рахунок для сплати послуг) та перевірка на коректність і унікальність. Коли перевірку пройдено, дані відсилаються на сервер.

Сервер приймає пакет даних, знову виконує їх перевірку та вразу успіху переходить до їх збереження. Після цього викликається закриття трансакції що підтверджує дозвіл на внесення змін до бази даних.

В результаті успішного проходження усіх операцій, клієнту у відповідь надсилаються щойно збережені дані з інформацією про успішне збереження.

3.4.6. Алгоритм створення та редагування скерування для пацієнта. Ще одним із основних алгоритмів роботи системи є алгоритм створення та редагування скерування для пацієнта, схема алгоритму якого приведена на рис.3.16.

Під скеруванням в даній розроблюваній системі розуміється сукупність попереднього діагнозу лікаря, результатів аналізів, підтвердження або спростування і нового діагнозу лікаря, курс проходження лікування та перелік фізіотерапефтичних кабінетів, що були призначені для курсу лікування користувача (якщо це курс лікування на дому то обирається відповідний тип із спеціальною формою).

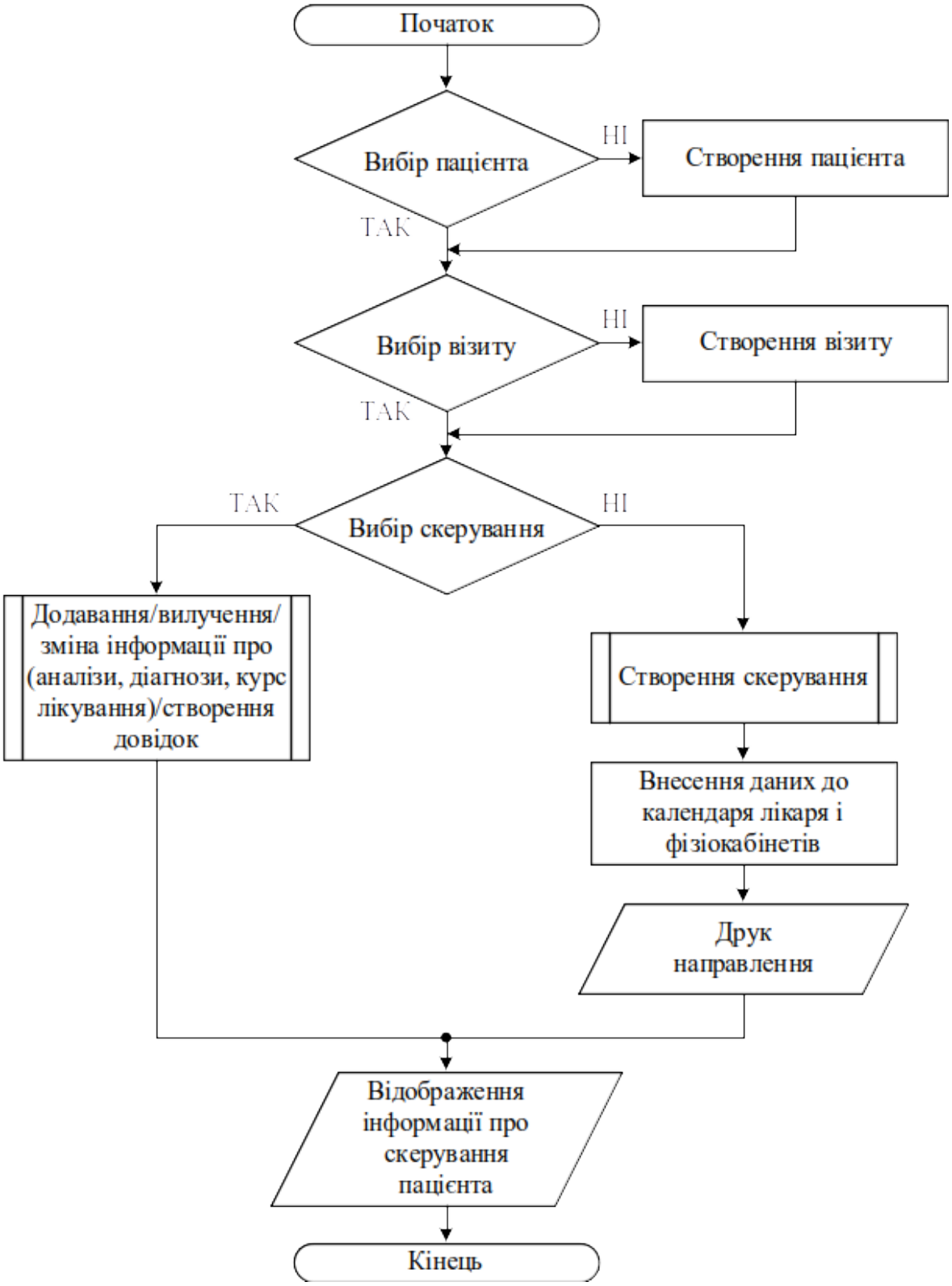


Рис. 3.16. Схема алгоритму створення та редагування скерування для пацієнта.

Після того як користувач обрав пункт меню «Скерування», він бачить форму із списком пацієнтів із вкладеними списками візитів, які можна фільтрувати та сортувати за різними критеріями. Також користувач може напряму ввести номер скерування та перейти до його редагування.

В даній дії, користувач також може швидко створити пацієнта та візит для нього, створені дані будуть містити н повний перелік інформації а тільки найголовніші, це: повне ім’я по можливості, контактні дані, контактна особа і хоча б один із елементів що є ідентифікаторами особи (паспорт, водійські права, студентський, ідентифікаційний код, номер страхівки), решта даних будуть введені пізніше відповідними особами.

Після того, як пацієнт та візит були обрані, відбувається створення скерування. До скерування вносяться початковий діагноз лікаря, перелік аналізів та обстежень що необхідно пройти пацієнту. Після того як усі необхідні дані було введено відбувається перевірка коректності введених даних та створення трансакції, далі дані передаються на сервер де вони знову проходять перевірку та в результаті успіху, зберігаються до бази даних і повертаються клієнту із повідомленням про успіх.

Ще одним можливим варіантом роботи цього алгоритму є ситуація коли скерування уже створено, а лікуючому лікарю прийшли результати аналізів, обстежень і т. д., йому в цьому випадку потрібно безпосередньо відкрити створене скерування і уже в ньому вносити зміні, принцип редагування працює за аналогією до створення.

В даному розділі було спроектовано схему роботи системи та приведено детальний опис основних функціональних частин системи. Була спроектована структура бази даних, описано таблиці що необхідні для роботи системи із зв’язками між ними. Також було складено програмну модель розроблюваної серверної частини та алгоритми роботи клієнт-серверної системи.

3.5. Реалізація серверної частини

Для реалізація серверної частини, обрано використання Java EE та середовищ Hibernate, Java Servlet та Spring MVC. Дані середовища забезпечують можливість створення веб-додатку, що може запускатися на сервері Tomcat і працює як звичайний веб-сервер, що написаний наприклад на php. Клієнт із таким сервером може взаємодіяти за допомогою звичайних GET, POST запитів за протоколом HTTP та обмінюватися даними за технологією AJAX та plain text.

Для того щоб під час реалізації серверної частини працювати із попередньо обраними середовищами, необхідно додати відповідні бібліотеки. Для того щоб додати нові бібліотеки, потрібно відредагувати один із головних конфігураційних файлів проекту, файл pom.xml. Даний файл використовується для компілювання проекту та контролю версій бібліотек. Використання такого файлу дозволяє компілювати проект в будь який момент без участі IDE Eclipse. В початковому стані він виглядає так, як зображено на рис. 3.17.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>LiVEZer.Medicine</groupId>
  <artifactId>LiVEZer.Medicine</artifactId>
  <version>0.1.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>LiVEZer.Medicine</name>
  <description>LiVEZer.Medicine - WebApp</description>
  <dependencies>
    <!-- Перелік необхідних бібліотек -->
          ...
  </dependencies>
  <build>
    ...
  </build>
</project>
```

Рис. 3.17. Початковий стан файлу pom.xml

Між тегами <dependencies></dependencies> слід додати усі необхідні бібліотеки із версіями. Приклад того, як саме потрібно додавати необхідні зв’язки на бібліотеку приведено на рис. 3.18.

```
<dependency>
   <groupId>Назва бібліотеки</groupId>
   <artifactId>Артефакт</artifactId>
   <version>Версія</version>
</dependency>

<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-webmvc</artifactId>
   <version>3.2.4.RELEASE</version>
</dependency>
```

Рис. 3.18. Приклад додавання посилань на бібліотеки

Для роботи розроблюваного проекту слід додати такі бібліотеки:

- org.springframework – середовище Spring

- org.hibernate, org.hibernate.common, org.hibernate.javax.persistence – середовище Hibernate.

Остаточний варіант файлу pom.xml приведено в додатку Б.

Ще одним головним конфігураційним файлом проекту є файл web.xml, вміст файли приведено в додатку Б. Даний файл відповідає за роботу розроблюваного сервера, в ньому визначається маршрутизація між запитами та класами, що їх оброблятимуть.

Для виконання основних функцій керування роботою серверу будуть використані такі класи:

- AppManager: даний клас відповідає за керування сервером, він містить у собі усі необхідні методи для початкового налаштування сервера, та забезпечення стабільної роботи системи.

- AppContextListener: даний клас призначений для початкової ініціалізації системи, він викликається в момент запуску сервера і викликає метод Configure(boolean async) із класу AppManager для початкового само налаштування.

- DBManager: даний клас відповідає за стабільний з’єднання із базою даних. При першому звертанні до цього класу відбувається створення сесії з’єднання із сервером бази даних та ініціалізація середовища Hibernate.

- SessionManager: даний клас призначений для роботи із сесіями користувача, в даному класі містяться функції створення сесії користувача, пошук останньої незавершеної сесії, відновлення сесії користувача.

- ServiceManager: даний клас призначений для забезпечення реалізації обробки запитів на виконання веб-методів від клієнта;

- DBTools: це допоміжний клас, призначений для виконання деяких функцій при роботі із базою даних, а саме: закриття сесії із БД, вилучення заборонених символів із запиту до БД (SQL injection), створення умов вибірки із БД.

Вміст файлів приведено в додатку В.

Реалізація доступу до даних реалізована за допомогою використання середовища Hibernate. Для спрощення роботи із даним середовищем, створено спеціальний клас CRUD (англ. Create Read Update Delete – Створення Читання Оновлення Видалення), що реалізує інтерфейс ICRUD [13]. Даний інтерфейс містить такі методи:

- boolean Save(T row) throws SQLDataException – метод для створення нового запису, тип даних T, повертає «істина» якщо все збережено, у разі помилки генерує виняткову ситуацію типу SQLDataException;

- boolean Delete(T row) throws SQLDataException – метод для видалення існуючого запису, тип даних T, повертає «істина» якщо запис видалений із БД, у разі помилки генерує виняткову ситуацію типу SQLDataException;

- T Read(pK id) throws SQLDataException – метод для читання одного запису із певної таблички в БД і повертає у відповідь прочитаний запис, тип даних T, у разі помилки генерує виняткову ситуацію типу SQLDataException;

- List<T> Read() throws SQLDataException – метод для читання усіх записів із відповідної таблички БД, в результаті повертає колекцію записів, тип даних T, у разі помилки генерує виняткову ситуацію типу SQLDataException;

- List<T> Read(Criterion criterion) throws SQLDataException – метод для читання записів із відповідної таблички БД у відповідності до вхідних критерій, повертає колекцію записів, тип даних T, у разі помилки генерує виняткову ситуацію типу SQLDataException;

- List<T> Read(Criterion criterion, Order order) throws SQLDataException – метод для читання записів із відповідної таблички БД у відповідності до вхідних критерій і сортування результату відповідно до вхідних умов, повертає відсортовану колекцію даних, тип даних T, у разі помилки генерує виняткову ситуацію типу SQLDataException.

Примітка**:** Тип даних T залежить від таблички з якої відбувається читання, а сама таблиці з якої відбуватиметься робота, вказується при створення об’єкту класу CRUD.

Повний лістинг класу CRUD та інтерфейсу ICRUD приведено в додатку Б.

Реалізація контролерів роботи із клієнтськими даними, використано засоби середовища Spring MVC. Для створення класу що буде виконувати роль контролера необхідно під час його створення додати @Controller перед описом класу. Приклад створення контролера приведено на рис. 3.19.

```
package LiVEZer.Medicine.WebApp.Controllers.Ajax;

import org.apache.log4j.Logger;
import org.springframework.stereotype.Controller;
import LiVEZer.Medicine.WebApp.DataProviders.AdminDataProvider;
import LiVEZer.Medicine.WebApp.Services.JSONResponse.*;

@Controller
@RequestMapping(value = "/admin/**")
public class AdminAjaxController extends BaseAjaxController
{
   private static final Logger logger = Logger.getLogger(AdminAjaxController.class);

   @RequestMapping(value = "/ajax.index")
   @ResponseBody
   public JSONResponse Index() throws IOException
   {
      JSONResponse response = new JSONResponse();
      response.setSuccess(false);
      response.setData((new AdminDataProvider()).getAllUsers());
      return response;
   }

}
```

Рис. 3.19. Приклад створення контролера

Для реалізації описаних раніше алгоритмів використано наступні класи:

- AdminAjaxController: контролер для роботи із користувачами, забезпечує створення, читання, редагування, видалення користувачів;

- PatientAjaxController: контролер для роботи із пацієнтами, забезпечує створення, читання, редагування, видалення пацієнтів;

- VisitAjaxController: контролер для роботи із візитами, призначений для створення, читання, редагування та видалення візитів пацієнта;

- OrderAjaxController: контролер для роботи із скеруваннями пацієнтів, призначений для створення, читання, редагування, видалення скерувань.

Усі можливості, що не реалізовані контролерами роботи із клієнтськими даними, реалізуються базовим обробником запитів на виконання веб-методів. Забезпечення виконання такої функціональності реалізовано, за допомогою використання Java Servlet. Для цього використовується клас BaseServlet, що розширює клас HttpServlet, приклад класу приведено на рис. 3.20.

```
package LiVEZer.Medicine.WebApp.Servlets;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import LiVEZer.Medicine.WebApp.Globals;
import LiVEZer.Medicine.WebApp.Services.ServiceManager;

public class BaseServlet extends HttpServlet
{
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException
   {
      doPost(request, response);
   }

   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException
   {
      String method = request.getParameter(Globals.Method);
      ServiceManager.doWithoutResponse(method, request, response);
   }
}
```

Рис. 3.20. Приклад класу BaseServlet

Даний клас не містить особливих функції він призначений лише для перехоплення запитів призначених для нього та перенаправляти ці запити до класу ServiceManager, котрий знає як опрацьовувати вхідні запити.

Основний код серверної частини приведено в додатку Б.

3.6. Реалізація клієнта

Клієнтська частина буде реалізована із використанням технології HTML, CSS та JavaScript із використання середовища ExtJs. На цьому рівні буде виконуватися формування веб-сторінок що потім будуть відображатися в браузері [21]. Особливістю побудови таких сторінок буде такою, що усі сторінки будуть формуватися динамічно в процесі роботи користувача через веб-браузер із сервером. Також сторінки будуть працювати за технологією AJAX, тобто при першому звертанні користувача до сервера, користувач отримає веб-сторінку, що завантажить необхідні JavaScript файли для реалізації AJAX роботи із сервером, і на далі увесь обмін інформацією буде відбуватися у фоні без необхідності перевантажувати сторінку. Що буде не тільки зручно, а й при такому використанні буде забезпечуватися цілісність введених даних користувачем та захист їх від втрати.

Головна сторінка матиме наступний вигляд, див рис. 3.21.

```
<%@ page language="java"  contentType="text/html; charset=utf-8" pageEncoding="utf-
8"%>
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8" />
<title>Test</title>
<link type="image/ico" rel="icon" href="favicon.ico" />
<!-- Link CSS files -->
<link type="text/css" rel="stylesheet" href="css/style.css" />
<!-- Link JavaScript files-->
<script type="text/javascript" src="js/LAB.min.js"></script>
</head>
<body>
   <div id="main"> <div class="dock"></div> </div>
   <script type="text/javascript">
   $LAB.script("js/jquery/jquery-1.10.2.min.js").wait().script("js/jquery/jquery-ui-
1.10.3.min.js") .wait().script("js/jquery/jquery.md5.js").wait()  .script("js/ext/ext-
all.js").wait().script("js/main.js");
   </script>
</body>
</html>
```

Рис. 3.21. – Опис головної сторінки index.jsp

Головна сторінка при завантаженні не містить жодних об’єктів чи елементів, вона є контейнером для подальшого їх відображення, див рис.3.22. Наприклад, одразу після завантаження сторінки, до сторінки автоматично підвантажується JavaScript файл main.js, у даному файлі містяться усі необхідні функції для подальшої роботи системи, для прикладу однієї із них є функція створення вікна авторизації, приклад наведено у рис. 3.23 – 3.24.

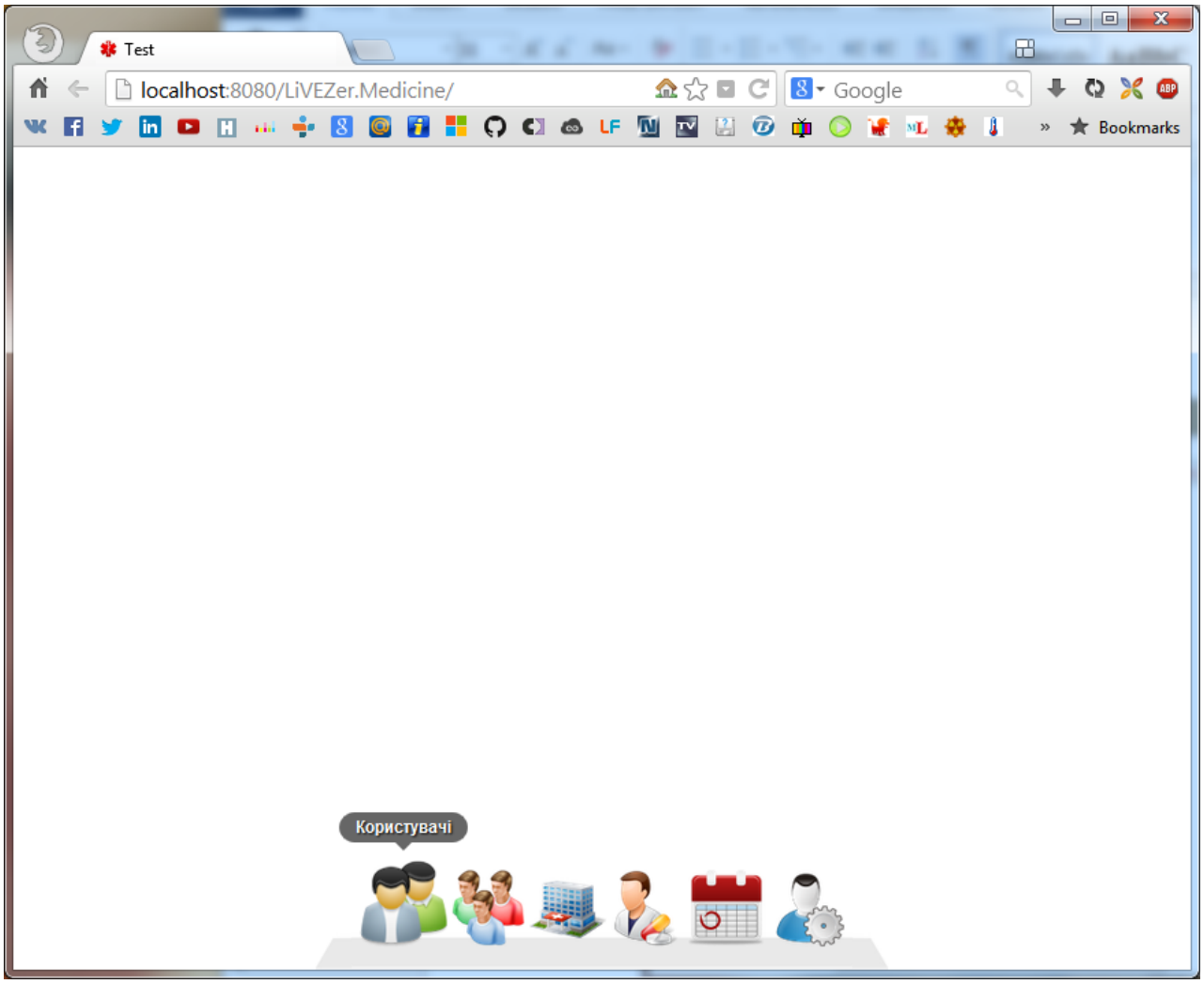


Рис. 3.22. Вигляд головної сторінки після авторизації.

```
function LogInProvider() {
    var isCreated = false;
    this.isCreated = function() { return isCreated; };
    this.createLogInForm = function() { createLogInForm(); };
    this.closeLogInForm = function() { closeLogInForm(); };
    this.logInToSystem = function() {
        $('#login-in-box-title').html("LogIn");
        var val = $('#passw').val();
        if ((val != "") && ($('#login').val() != "")) {
            $('#passw').val($.md5(val));
            var params = $('#login-in-form').serialize();
            $('#passw').val(val);
            performLogIn(params);
        } else { wrangCredentials(null); }
    };
    function createLogInForm() {
```

Рисунок 3.23. Метод створення вікна авторизації

```
if (!isCreated) {
        $('#main').append("<div id=\"login-in-box\"></div>");
        generateForm();
        setEventHandler();
        isCreated = true;
      }
    }
    function closeLogInForm() {
      if (isCreated) {
        $('#login-in-box').remove();
        removeSelection();
        isCreated = false;
      }
    }
    function generateForm() {
      $('#login-in-box').html("<div id=\"login-in-box-title\">Авторизація</div><form
id=\"login-in-form\" autocomplete=\"off\"><input type=\"text\" name=\"login\" id=\"login\"
placeholder=\"Ім'я користувача\" /><input type=\"password\" name=\"passw\" id=\"passw\"
placeholder=\"Пароль\" /><input type=\"submit\" value=\"Увійти\" /></form>");
    }
    function wrangCredentials(message) {
      if (message == null)
        message = "Невірні дані";
      $('#login-in-box-title').html(message);
      $("#login-in-box").effect("bounce", { times : 3 }, "slow");
    }
    function performLogIn(params) {
      var bOk = true;
      params = "method=logIn&" + params;
      $.ajax({ url : 'do', type : 'POST', dataType : 'json', async : false, data : params,
        success : function(data) { bOk = checkSession(data); },
        error : function() { bOk = false; }
      });
      if (bOk) {
        closeLogInForm();
        loadingVisualization.showMmainLoading();
        bOk = menuCreator.createMainMenu();
        loadingVisualization.closeMmainLoading();
      }
    }
    function checkSession(data) {
      if (data.success) {
        return true;
      } else {
        wrangCredentials(data.message);
        return false;
      }
    }
}
function setEventHandler() {
    $('#login-in-form').submit(function() {
      logInProvider.logInToSystem();
```

Рис. 3.24. Метод створення вікна авторизації

При створенні форми авторизації вона виглядає наступним чином, див рис.3.25.

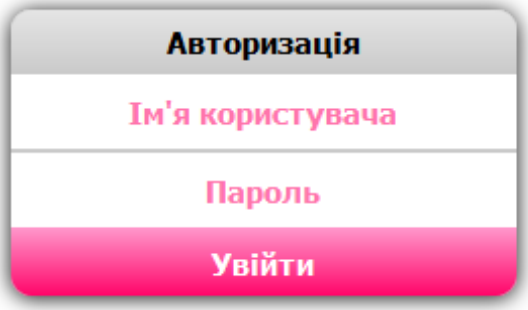


Рис. 3.25. Форма авторизації в системі.

Після того як користувач авторизувався в системі йому відображається головне меню користувача, для створення цього меню використовується створення головного меню користувача, рис 3.26.

```
function MenuCreatoClass()  {
  this.createMainMenu  = function() {
    var bRes = false;
    $.ajax({ url : 'do', type : 'POST', dataType : 'json', async : false,
      data : { method : "getMainMenu", data : userSession },
      success : function(data) {
        buildMainMenu(data);
        bRes = true;
      },
      error : function() { bRes = false; }
    });
    return bRes;
  };
  function buildMainMenu(data)  {
    if (data.success)  {
      var htmlText = "<ul>";
      $.each(data.items, function(index, value) {
        htmlText += addMenuItem(value);
      });
      htmlText += "</ul>";
      $('.dock').html(htmlText);
    } else { showErrorCode(data); }
  }
  function addMenuItem(e)  {
    if (e.selected)  {
      return "<li id=\"" + e.itemId + "\"><a id=\"executed\"
onmouseup=\"selectTarget(this)\"  onclick=\"" e.action + "\"><em><span>" + e.title +
"</span></em><img src=\"" + e.file + "\" /></a><span id=\"indic\"></span></li>"
    } else {
      return "<li id=\"" + e.itemId + "\"><a onmouseup=\"selectTarget(this)\"  onclick=\"" +
e.action + "\"><em><span>"  + e.title + "</span></em><img src=\"" + e.file + "\"
```

Рис. 3.26. Функція створення меню користувача

При створенні головного меню воно виглядає так як зображено на рис.3.27.

Рис. 3.27. Головне меню користувача, для роль із необмеженим доступом.

Розглядаючи рис.3.27. з ліва на право, кожен елемент зображення відповідає за певну дію:

- «Користувачі» – відповідає за роботу із користувачами;
- «Пацієнти» – відповідає за роботу із пацієнтами;
- «Візити» – відповідає за роботу із візитами;
- «Скерування» – відповідає за роботу із скеруваннями;
- «Календар» – відповідає за роботу із календарем;
- «Налаштування» – відповідає за роботу із персональними налаштуваннями користувача.

Повний текст кодів клієнтської частини приведено в додатку В.

3.7. Реалізація клієнт-серверної взаємодії

Для реалізації клієнт серверної взаємодії необхідно створити засоби як на серверній так і клієнтській стороні.

Реалізація клієнт-серверної взаємодії на серверній частині системи, виконана із використанням контролерів та Java Servlet, опис їх реалізації було приведено вище.

Реалізація клієнт-серверної взаємодії на клієнтській частині системи, виконана із використання середовища JQuery. Дане середовище містить у собі набір методів для реалізації AJAX технології. Приклад використання такого методу приведено на рис. 3.28.

```
$.ajax({ url : 'do',
        type : 'POST',
        dataType : 'json',
        async : true,
        data : { data: 'Data'},
        success : function(data) { /**/ },
        error : function() { /**/ }
    });
```

Рис. 3.28. Приклад використання методу $.ajax()

Описи та призначення параметрів функції:

- url – веб-адреса до контролеру чи Java Servlet;

- dataType – тип даних що передаються та приймаються, може бути JSON або XML;

- async – вказує як виконувати запит синхронно чи асинхронно;

- data – дані що відсилаються разом із запитом;

- success – метод, який відбудеться, якщо запит пройшов успішно;

- error – метод, який відбудеться, якщо запит пройшов із помилками.

3.8. Демонстрація роботи додатку

Одним з основних завдань роботи додатку було створення пацієнта, як окремої сутності для розроблюваної предметної області. Це дозволить заносити всі необхідні дані про особу, яка перебуває на лікуванні в медичному закладі і вести весь необхідний медичний облік.

Після того, як користувач перейде на вкладку «Пацієнти» і спробує створити нового, система відкриє форму по створенні пацієнта, рис. 3.29

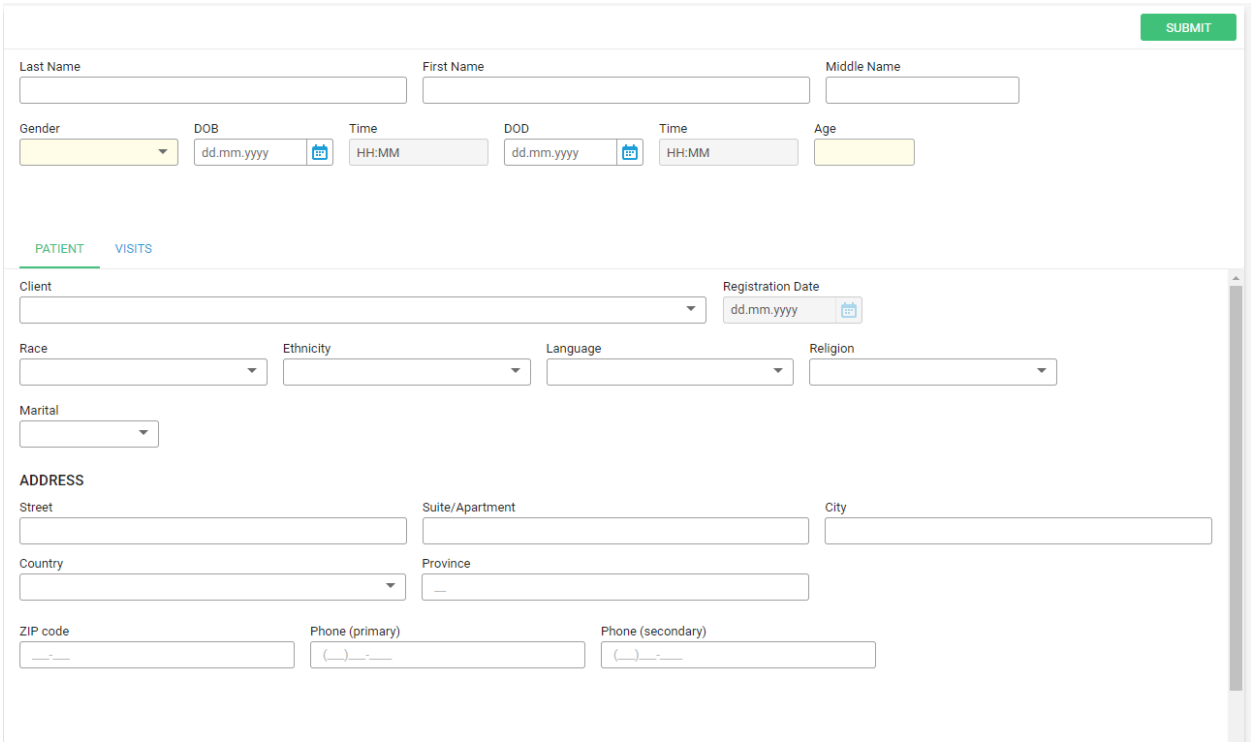


Рис. 2.29. Форма створення пацієнта

На даній формі розміщені усі необхідні поля, які будуть ідентифікувати особу пацієнта. Деякі з полів підсвічені жовтим кольором, це означає, що вони є обов'язковими до заповнення.

Після того, як пацієнт був створений, йому необхідно створити «Замовлення» і вказати лікуючу особу. «Замовлення» потрібне для того, щоб в нього помістити усі необхідні пацієнту тести під час його візиту. Результат створення «Замовлення» на рис. 2.30.

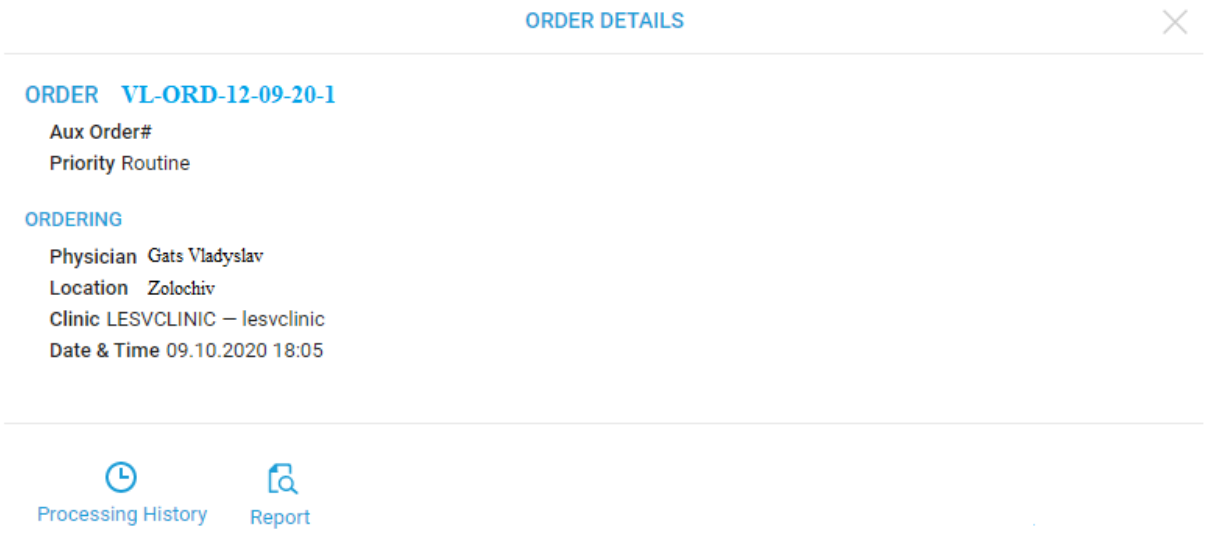


Рис. 2.30 Результат створення замовлення

Як видно з рисунку, жирним шрифтом відображається номер замовлення, який формується наступним чином: система бере перші літери з імені і іпрзвища пацієнта, після чого додається приставка «ORD» (від англ. слова Order – замовлення). Далі додаються дата та час створення і порядковий номер замовлення в контексті візиту пацієнта, в даному випадку – 1, тобто, перше замовлення. На цій формі користувач має можливість переглядати, які дії були виконані на цьому замовленні (кнопка «Processing History») і переглядати результати тестів, які були виконані в замовленні.

Після того, як користувач відкриє створене замовлення, він переміститься на керуючу форму, яка дозволяє обробляти замовлення, рис. 2.31.

Головне завдання даної форми – це додавання тестів, які в подальшому будуть виконуватися для пацієнта, наприклад аналіз крові, тест на цукровий діабет і так

далі. Також може бути створений комплексний тест, який містить в собі декілька інших. Це зроблено для випадку, коли потрібне комплексне обстеження для ідентифікації проблеми і визначення діазнозу.

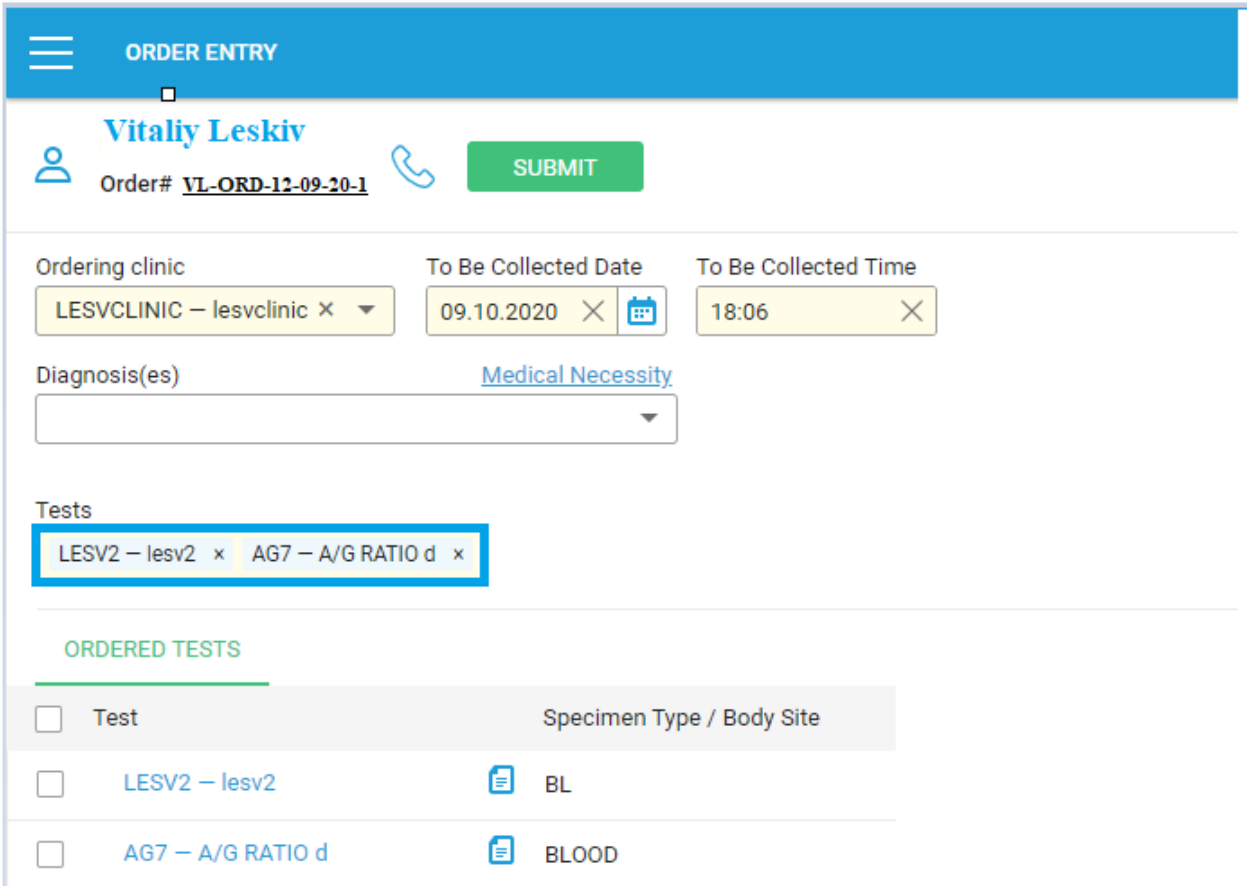


Рис. 2.31. Форма керування замовленням

Важливим завдання була реалізація можливісті візуалізації даних, які були отримані під час вибраного тесту і їх узагальнення. В контексті додатку для цього завдання був створений модуль звітності, який побудований на компоненті Active Reports. Наприклад, так буде виглядати звіт тесту на Covid-19 через розроблюваний додаток, рис. 2.32.

Оформлення звітності лежить на плечах лікуючого лікаря, саме він проводить усі необіхдні тести, і заносить їх результати в додаток, після чого, за домогою модуля Excel, дані обробляються і потім формується їх показ у формі таблиці, як показано вище. Шаблон звітності можна налаштовувати відповідно до того, що саме потрібно бачити в кінцевому результаті. Тобто, можна демострувати будь-яке поле, яке є в інформації про пацієнта, чи про замовлення. Результуючі таблиці можна

форматувати будь яким чином, наприклад, так відображається тест на аналіз крові засобами додатку, рис. 2.33.

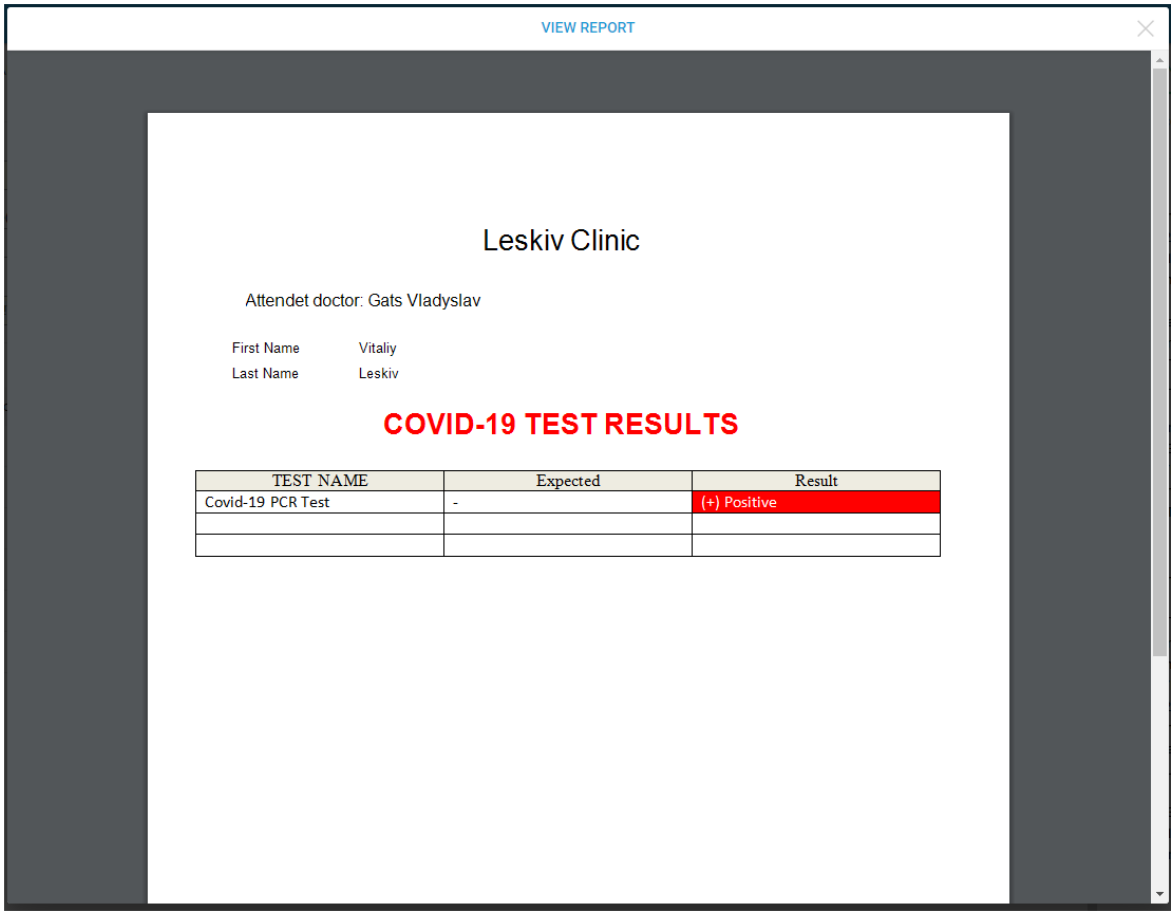


Рис. 2.32 Перегляд звіту про тест на Covid-19

В кінцевому результаті, звіт може бути збережений в форматі PDF, чи відправлений на друк, рис. 2.33.

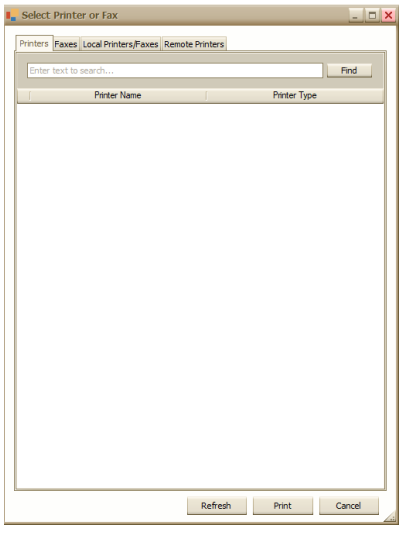


Рисунок 2.33. Друк звіту

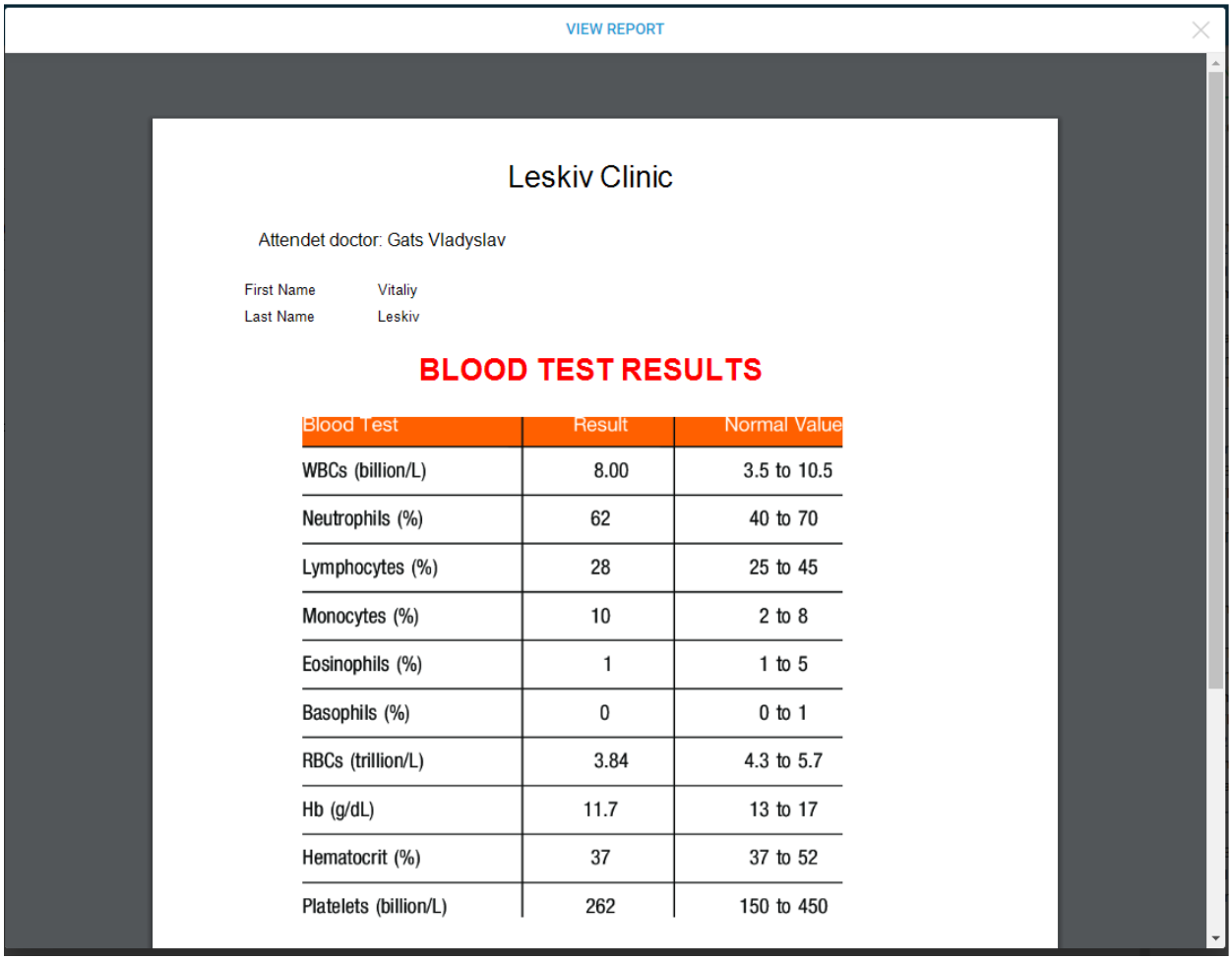


Рис. 2.33. Перегляд звіту про аналіз крові

3.9. Висновки до розділу 3

В даному розділі було спроектовано схему роботи системи та приведено детальний опис основних функціональних частин системи. Була спроектована структура бази даних, описано таблиці що необхідні для роботи системи із зв’язками між ними. Було складено програмну модель розроблюваної серверної частини та алгоритми роботи клієнт-серверної системи. Детально описано роботу представленої медичної інформаційної системи із прикладами коду, що необхідні для реалізації основних частин розроблюваної клієнт-серверної системи. Результатом виконання даного розділу є практична реалізація спроектованої системи. Основні частини коду реалізованої клієнт-серверної системи приведені у додатках.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Безпека в надзвичайних ситуаціях

Важливу роль в збереженні здоров’я та життя людей в сучасному та майбутньому закликана грати інформаційна діяльність держави в галузі прогнозування небезпек серед мешкання. Компетентність людей в світі небезпек і способи захисту від них - необхідна умова досягнення безпеки життєдіяльності людини на всіх етапах його життя.

Небезпека – це явища, процеси, об’єкти, які здатні за певних умов завдавати шкоди здоров’ю людини як відразу, так і в майбутньому, тобто викликати небажані наслідки. в загальному їх класифікують:

– за сферою (джерелом) походження: природна, техногенна, соціальна та ін.;

– за часом прояву: імпульсні, кумулятивні;

– за локалізацією: пов’язані з космосом, атмо-, гідро-, літосферою;

– за наслідками: захворювання, травми, смертельні випадки, аварії, пожежі;

– за збитками: соціальні, екологічні, технічні та ін.;

– за сферою прояву: побутова, виробнича, спортивна тощо;

– за структурою: прості, складні, похідні;

– за характером дії на людину: активні та пасивні (останні активізуються за рахунок енергії, носієм якої є сама людина, що наражається на гострі, ріжучі нерухомі елементи, ями, ухили, нерівності поверхні тощо).

Безпека – це збалансований стан людини, соціуму, держави, природних, антропогенних систем.

Надзвичайна ситуація – порушення нормальних умов життя і діяльності людей на об'єкті або території, спричинене аварією, катастрофою, стихійним лихом, епідемією, епізоотією, епіфітотією, великою пожежею, застосуванням за собі в ураження, що призвели або можуть призвести до людських і матеріальних втрат.

Ліквідація наслідків надзвичайних ситуацій – це комплекс заходів щодо негайного реагування на надзвичайну ситуацію, проведення у зоні НС усіх видів

рятувальних та невідкладних робіт, а також організація та проведення першочергового життєзабезпечення постраждалого населення і рятувальників.

Основою життєдіяльності є чинники і параметри навколишнього середовища (сонця, повітря, води, ґрунту, біосфери) та штучного середовища життя (житлові та виробничі будівлі, споруди, транспортні та повітряні комунікації, системи забезпечення енергоресурсами, продуктами харчування) і багато іншого, що створено руками людини для забезпечення життя.

Природні небезпеки обумовлені кліматичними і природними явищами. Вони виникають при зміні погодних умов та природної освітленості в біосфері, а також внаслідок стихійних явищ, що відбуваються в біосфері (повені, землетруси тощо).

Негативний вплив на людину і середовище існування, на жаль, не обмежується природними небезпеками. Людина, вирішуючи завдання досягнення комфортного і матеріального забезпечення, безперервно впливає на середовище існування своєю діяльністю і продуктами діяльності (технічними засобами, викидами різних виробництв тощо), генеруючи в середовищі техногенні та антропогенні небезпеки**.**

Техногенні небезпеки створюють елементи техносфери: машини, споруди, речовини і т. п., а антропогенні виникають в результаті помилкових або несанкціонованих дій людини або груп людей.

В даний час перелік техногенних реально діючих небезпек значний і включає більше 100 видів. До поширеним, мають досить високий рівень небезпеки, належать виробничі небезпеки: запиленість і загазованість повітря, шум, вібрації, електромагнітні поля, іонізуючі випромінювання, підвищені або знижені параметри атмосферного повітря (температури, вологості, рухливості повітря, тиску), недостатнє і неправильне освітлення, монотонність діяльності, важка фізична праця та ін, а до травмуючим (травмонебезпечним) відносяться: електричний струм, падаючі предмети, висота, рухомі машини і механізми, частини конструкцій, що руйнуються та ін.

Шкідливий фактор – негативний вплив на людину, яке призводить до погіршення самопочуття або захворювання.

Травмує (травмонебезпечний) фактор – негативний вплив на людину, яке призводить до травми або летального результату.

Різновид небезпек, що загрожують особистості, безперервно збільшується. У виробничих, міських, побутових умовах на людину впливає одночасно декілька негативних факторів.

Потенційна небезпека представляє загрозу загального характеру, непов'язану з простором і часом впливу.

Наявність потенційних небезпек знаходить своє відображення у затвердженні, що життєдіяльність людини є потенційно небезпечною. Воно визначає, що всі дії людини й усі компоненти середовища існування, насамперед технічні засоби і технології, крім позитивних властивостей і результатів мають здатність генерувати травмуючі і шкідливі фактори. При цьому будь-яка нова позитивна дія людини або його результат неминуче призводять до виникнення нових негативних факторів.

Реальна небезпека завжди пов'язана з конкретною загрозою впливу на об'єкт захисту (людини), вона координована в просторі і в часі.

Сьогоднішня ситуація в Україні щодо небезпечних природних явищ, аварій і катастроф характеризується як дуже складна. Тенденція зростання кількості надзвичайних ситуацій, важкість їх наслідків змушують розглядати їх як серйозну загрозу безпеці окремої людини, суспільству та навколишньому середовищу, а також стабільності розвитку економіки країни. До роботи в районі надзвичайної ситуації необхідно залучати значну кількість людських, матеріальних і технічних ресурсів. Запобігання надзвичайним ситуаціям, ліквідація їх наслідків, максимальне зниження масштабів втрат та збитків перетворилося на загальнодержавну проблему і є одним з найважливіших завдань органів виконавчої влади і управління всіх рівнів.

15 липня 1998 р. Постановою Кабінету Міністрів України № 1099 «Про порядок класифікації надзвичайних ситуацій» затверджено «Положення про класифікацію надзвичайних ситуацій». Згідно з цим Положенням залежно від територіального поширення, обсягів заподіяних або очікуваних економічних збитків, кількості людей, які загинули, розрізняють чотири рівні надзвичайних ситуацій.

Надзвичайна ситуація загальнодержавного рівня — це надзвичайна ситуація, яка розвивається на території двох та більше областей (Автономної Республіки Крим, міст Києва та Севастополя) або загрожує транскордонним перенесенням, а також у разі, коли для її ліквідації необхідні матеріали і технічні ресурси в обсягах, що перевищують власні можливості окремої області (Автономної Республіки Крим, міст Києва та Севастополя), але не менше одного відсотка обсягу видатків відповідного бюджету.

Надзвичайна ситуація регіонального рівня — це надзвичайна ситуація, яка розвивається на території двох або більше адміністративних районів (міст обласного значення) Автономної Республіки Крим, областей, міст Києва та Севастополя або загрожує перенесенням на територію суміжної області України, а також у разі, коли для її ліквідації необхідні матеріальні і технічні ресурси в обсягах, що перевищують власні можливості окремого району, але не менше одного відсотка обсягу видатків відповідного бюджету.

Надзвичайна ситуація місцевого рівня — це надзвичайна ситуація, яка виходить за межі потенційно небезпечного об'єкта, загрожує поширенням самої ситуації або її вторинних наслідків на довкілля, сусідні населені пункти, інженерні споруди, а також у разі, коли для її ліквідації необхідні матеріальні і технічні ресурси в обсягах, що перевищують власні можливості потенційно небезпечного об'єкта, але не менше одного відсотка обсягу видатків відповідного бюджету. До місцевого рівня також належать всі надзвичайні ситуації, які виникають на об'єктах житлово-комунальної сфери та інших, що не входять до затверджених переліків потенційно небезпечних об'єктів.

Надзвичайна ситуація об'єктового рівня — це надзвичайна ситуація, яка не підпадає під зазначені вище визначення, тобто така, що розгортається на території об'єкта або на самому об'єкті і наслідки якої не виходять за межі об'єкта або його санітарно-захисної зони.

Для організації ефективної роботи із запобігання надзвичайним ситуаціям, ліквідації їхніх наслідків, зниження масштабів втрат та збитків дуже важливо знати причини їх виникнення та володіти теорією виникнення катастроф.

Запобігання виникненню надзвичайних ситуацій — це підготовка та реалізація комплексу правових, соціально-економічних, політичних, організаційно-технічних, санітарно-гігієнічних та інших заходів, спрямованих на регулювання безпеки, проведення оцінки рівнів ризику, завчасне реагування на загрозу виникнення надзвичайної ситуації на основі даних моніторингу (спостережень), експертизи, досліджень та прогнозів щодо можливого перебігу подій з метою недопущення їх переростання у надзвичайну ситуацію або пом'якшення її можливих наслідків. Зазначені функції запобігання надзвичайним ситуаціям техногенного та природного характеру в нашій країні виконує Єдина державна система запобігання надзвичайним ситуаціям техногенного і природного характеру і реагування на них, затверджена Постановою Кабінету Міністрів України від 3 серпня 1998 р. № 1198.

Єдина державна система запобігання надзвичайним ситуаціям техногенного і природного характеру і реагування на них (ЄДСЗР) включає в себе центральні та місцеві органи виконавчої влади, виконавчі органи рад, державні підприємства, установи та організації з відповідними силами і засобами, які здійснюють нагляд за забезпеченням техногенної та природної безпеки, організують проведення роботи із запобігання надзвичайним ситуаціям техногенного та природного походження і реагування у разі їх виникнення з метою захисту населення і довкілля, зменшення матеріальних втрат.

Основною метою створення ЄДСЗР є забезпечення реалізації державної політики у сфері запобігання і реагування на надзвичайні ситуації, забезпечення цивільного захисту населення.

До складу сил і засобів ЄДСЗР входять відповідні сили і засоби функціональних і територіальних підсистем, а також недержавні (добровільні) рятувальні формування, які залучаються для виконання відповідних робіт. Військові і спеціальні цивільні аварійно-рятувальні (пошуково-рятувальні) формування, з яких складаються зазначені сили і засоби, укомплектовуються з урахуванням необхідності проведення роботи в автономному режимі протягом не менше трьох діб і перебувають у стані постійної готовності. СПГ згідно із законодавством можуть залучатися для термінового реагування у разі виникнення надзвичайної ситуації з повідомленням про це відповідних центральних

та місцевих органів виконавчої влади, виконавчих органів рад, керівників державних підприємств, установ та організацій.

У виняткових випадках, коли стихійне лихо, епідемія, епізоотія, аварія чи катастрофа ставить під загрозу життя і здоров'я населення і потребує термінового проведення великих обсягів аварійно-рятувальних і відновлювальних робіт, Президент України може залучати до виконання цих робіт у порядку, визначеному Законом України «Про надзвичайний стан», спеціально підготовлені сили і засоби Міноборони.

Запобігання виникненню надзвичайних ситуацій природного і техногенного характеру та ефективна ліквідація їх наслідків є одним із головних пріоритетів у діяльності місцевих органів виконавчої влади.

Як свідчить досвід ліквідації наслідків надзвичайних ситуацій, затрати на проведення заходів і робіт з попередження надзвичайних ситуацій в декілька разів менші ніж затрати на проведення робіт з ліквідації їх наслідків. А тому зусилля місцевих органів виконавчої влади повинні бути спрямовані на забезпечення сталого і безаварійного функціонування об’єктів економіки, систем життєзабезпечення та потенційно-небезпечних об’єктів, а також надійного захисту населення і працюючого персоналу від негативного впливу надзвичайних ситуацій.

4.2. Підвищення стійкості роботи об’єктів господарської діяльності у воєнний час

Заходи з підвищення стійкості планують з урахуванням місцевих умов, важливості об'єкта, його географічного положення, економічної доцільності проведення заходів. На мирний час планують головним чином трудомісткі заходи, які потребують значних матеріальних витрат і часу, а на період загрози нападу противника - такі заходи, що не потребують багато часу чи проведення яких не є доцільним у мирний час.

У період загрози нападу противника проводять ті заходи з підвищення стійкості роботи об'єкта, які недоцільно здійснювати у мирний час. До таких заходів належать:

- проведення згідно з особовим розпорядженням евакуаційних засобів;

- приведення в готовність системи сповіщення, захисних споруд та пунктів керування;

- видача робітникам і службовцям засобів індивідуального захисту;

- будівництво швидко будованих захисних споруд;

- підготовка об'єкта до швидкої та безаварійної зупинки виробництва згідно з сигналом "Повітряна тривога";

- проведення заходів з підвищення стійкості інженерно-технічного комплексу (підсилення будівель та споруд, встановлення зонтів, навісів, захисних козирків над цінним обладнанням, запасів паливно-мастильних матеріалів, сильнодіючих отруйних речовин та вибухонебезпечної сировини, обваловка складів і т ін.);

- здійснення переведення об'єкта на режим роботи воєнного часу (двозмінна праця) та перехід на випуск запланованої на воєнний час продукції;

- введення до дії графіка цілодобового чергування керуючого складу;

- підсилення охорони об'єкта і встановлення суворого пропускного режиму;

- здійснення світломаскування об'єкта.

На період загрози нападу противника згідно зі спеціальним розпорядженням на всіх об'єктах у темний час доби здійснюють світломаскування за режимом "часткове затемнення", при ньому обмежується зовнішнє освітлення до допустимої норми, затемнюють світлові пройми, вікна.

За сигналом "Повітряна тривога" в темний час здійснюють світломаскування за режимом "повного затемнення". При цьому живлення електроенергією усіх об'єктів і жилих районів припиняється за винятком тих об'єктів, на яких не можна зупиняти виробничий процес, а також вузлів зв'язку, станцій переливання крові, операційних.

Для організованого й своєчасного проведення заходів з підвищення стійкості роботи ОГД завчасно складають плани-графіки заходів з підвищення стійкості. Питання підвищення стійкості відображають також у плані ЦЗ об'єкта. У плані-графіку наводять перелік заходів на шкалі часу вказують початок і закінчення виконання кожного заходу. Для начальника ЦЗ і штабу ЦЗ цей документ є керівним

під час вирішення одного з найважливіших завдань - підвищення стійкості роботи об'єкта.

Під час раптового нападу, коли термін на організацію та виконання заходів ЦЗ гранично обмежений, здійснюють виконання тільки першочергових завдань, які направлені передусім на захист робітників, службовців та членів їх сімей, на безаварійну зупинку виробництва та прийняття екстрених заходів, що дозволяють, якоюсь мірою, зменшити ступінь ураження в надзвичайних ситуаціях. Під час виконання заходів цивільного захисту особливе значення має надійність і оперативність керування цивільним захистом об'єкта як одна з основних ланок успішного вирішення завдань з підвищення стійкості роботи об'єкта господарської діяльності.

Отже, розробка й планування заходів, що є економічно обґрунтованими, щодо стійкості роботи об'єкта залежать від всебічного вивчення умов, які мають скластися під час надзвичайних ситуацій. Вивчення ступеня їх впливу на виробничу діяльність підприємства будь-якої форми приналежності й власності дозволяє значно скоротити витрати на строки підвищення стійкості роботи в надзвичайних ситуаціях, а це, в свою чергу, підвищує життєздатність як об'єкта, так і всього господарства в цілому.

4.3. Висновки до розділу 4

Проаналізовано вимоги з охорони праці та безпеки в надзвичайних ситуаціях, що дало змогу визначити шляхи мінімізації негативного впливу комп’ютерної техніки на користувачів програмного засобу виявлення.

Досліджено питання безпеки в надзвичайних ситуацій та запобігання їх винекнення.

Розглянуто найбільш небезпечні надзвичайні ситуації на території України, до яких належать великі повені, катастрофічні затоплення, землетруси та зсувні процеси, лісові та польові пожежі, великі снігопади та ожеледі, урагани, смерчі та шквальні вітри тощо.

Досліджено питання підвищення стійкості роботи об’єктів господарської діяльності у воєнний час.

ВИСНОВКИ

Під час виконання магістерської кваліфікаційної роботи було розроблено медичну інформаційну клієнт-серверну систему. Розроблена система призначена для автоматизації бізнес процесів у лікарні і відображення біомедичних даних пацієнта.

В ході виконання роботи:

1. Проведено аналітичний огляд технологій та структурних рішень по вирішенню задачі. В роботі проаналізовані можливі способи реалізації серверної частини системи, клієнтської частини системи, клієнт-серверної взаємодії і міжсистемної взаємодії в медичних інформаційних системах та обрано найкращий.

2. На основі отриманих результатів аналізу технологій та структурних рішень створена архітектура системи та здійснено обґрунтування обраних технологій.

3. Спроектовано архітектуру клієнт-серверної системи та схему роботи клієнт-серверної системи.

4. Розроблено структуру бази даних для медичної інформаційної системи.

5. Створена програмна модель серверної частини та розроблено основні алгоритми роботи клієнт-серверної системи. Приведено опис реалізації серверної та клієнтської частини системи.

6. По результатах розробленої клієнт-серверної системи виконано дослідження її роботи. В результаті було продемоснстровано візуалізацію біомедичних даних пацієнта у вигляді звітів про тест на Covid-19 і аналіз крові.

7. Проаналізовано вимоги з охорони праці та безпеки в надзвичайних ситуаціях, що дало змогу визначити шляхи мінімізації негативного впливу комп’ютерної техніки на користувачів програмного засобу виявлення.

Магістерська кваліфікаційна робота виконана повністю у відповідності із завдання та вихідним даними до роботи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Кудрявцева С.П., Колос В.В., «Міжнародна інформація» Навчальний посібник – К.: Видавничий Дім «Слово», 2005. - 400с.

2. Самсонов В.В., Єрохін А.Л. «Методи та засоби Інтернет-технологій» Х. : СМІТ, 2008. - 263 с.

3. Chris Giametta “Pro Flex on Spring”, APRESS, 2009. – p. 445c.

4. Роберт Дж. Оберг “Технология COM + Основы и программирование = Understanding and Programming COM+: A Practical Guide to Windows 2000 First Edition”. – М.: Вильямс, 2000. – С. 480.

5. Макгрегор Дж., Сайкс Д. “Тестирование объектно-ориентированного программного обеспечения”. – К: Диасофт, 2002. – С. 432.

6. Татарчук М. І. “Корпоративні інформаційні системи: Навч. посібник”. – К., 2005. – С. 245.

7. Мухамедзянов Н. “Java. Server applications”. – М: СОЛОН, 2003. – С.267.

8. Орфалі Роберт, Ден Харкі “JAVA and CORBA in client server applications” John Wiley & Sons, Inc.: March, 1998 С. 1072, ISBN 0-471-24578-X

9. Фленов М. Є. “Web-сервер глазами хакера: Проблемы безопасности Web-серверов; Ошибки в сценариях на PHP, Perl, ASP; SQL-инъекции”, 2005. – С. 365.

10. Крістіан Бауер, Гевін Кінг і Гері Грегорі «Java Persistence with Hibernate, Second Edition» MEAP Began: March 2013, С. 600, ISBN: 9781617290459

11. Гарнаев А. Web-програмування на Java і JavaScript. - СПб.: БХВ Санкт-Петербург, 2002.

12. Аткинсон Л. MySQL. Бібліотека професіонала, 2002.

13. Спейнауер С., Куэрсиа В. Довідник Web-майстра. - К: "BHV", 1997. - 368 с.

14. Мак-Федрис П. Використання JavaScript. Спеціальне видання. - Діалектика, 2002.

15. Будилов В. JavaScript, XML и объектная модель документа. - СПб.: НиТ, 2001.

16. Морозов Ю.В., Пастернак І.І. / Класифікація засобів модульної взаємодії між клієнтом і сервером // Вісник «Комп’ютерні системи та мережі».- Львів: НУ «Львівська політехніка», 2011.- №717.- С. 108-113.

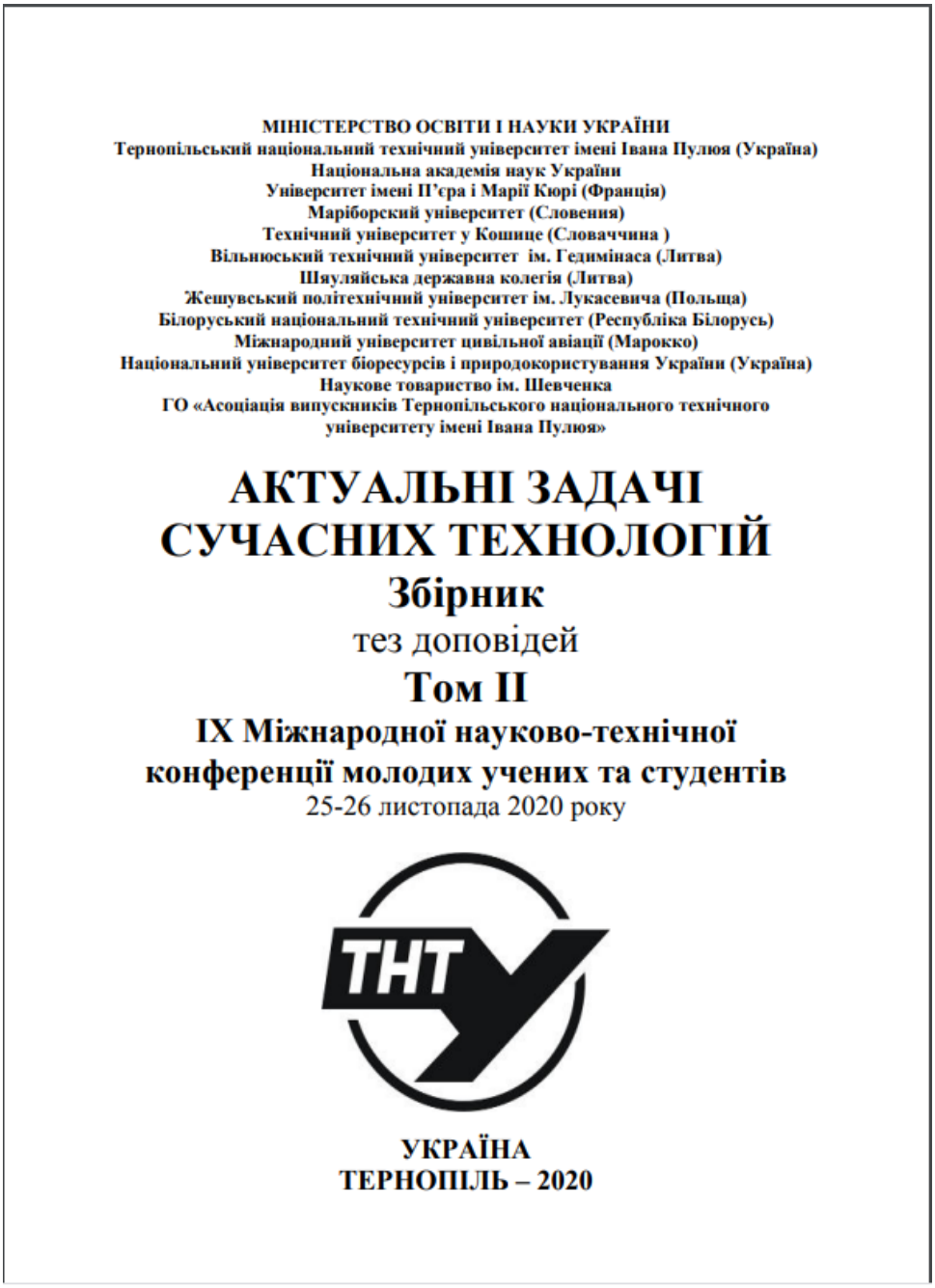
17. Морозов Ю.В., Пастернак І.І. / Модель об’єктної клієнт-серверної взаємодії // Вісник «Комп’ютерні науки та інформаційні технології».- Львів: НУ «Львівська політехніка», 2011.- №719.- С. 164-167.

18. Морозов Ю.В., Пастернак І.І. / Мережні інтерфейси рівня клієнт-сервер // Вісник «Інформаційні системи та мережі».- Львів: НУ «Львівська політехніка», 2012.- №743.- С. 121-131.

ДОДАТКИ

Додаток А

Тези конференцій



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Національна академія наук України
Університет імені П'єра і Марії Кюрі (Франція)
Маріборский університет (Словения)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Шяуляйська державна колегія (Литва)
Жешувський політехнічний університет ім. Лукасевича (Польща)
Білоруський національний технічний університет (Республіка Білорусь)
Міжнародний університет цивільної авіації (Марокко)
Національний університет біоресурсів і природокористування України (Україна)
Наукове товариство ім. Шевченка
ГО «Асоціація випускників Тернопільського національного технічного університету імені Івана Пулюя»

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник

тез доповідей

Том II

IX Міжнародної науково-технічної конференції молодих учених та студентів

25-26 листопада 2020 року

УКРАЇНА
ТЕРНОПІЛЬ – 2020

Матеріали IX Міжнародної науково-технічної конференції молодих учених та студентів.
Актуальні задачі сучасних технологій – Тернопіль 25-26 листопада 2020.

УДК 004.4
Д.Р. Яценко, В.М. Леськів, Н.С. Луцик докт. філос.
Тернопільський національний технічний університет імені Івана Пулюя, Україна

МЕТОДИ ЗАХИСТУ ЦЕНТРАЛЬНИХ ПРОЦЕСОРІВ КОМП'ЮТЕРІВ ВІД АТАК

D.R. Yatsenko, V.M. Leskiv, N.S. Lutsyk Ph.D.
METHODS OF COMPUTER CENTRAL PROCESSORS PROTECTION AGAINST ATTACKS

Одним з основних складових комп'ютера є процесор, від якого залежить програмне керування всіма складовими системи, що впливає на ефективну роботу комп'ютера. При роботі процесор оперує важливими даними, витік котрих є неприпустимим [1]. Тому безпека процесора є першочерговою задачею захисту. Особливо це актуально для персональних робочих станцій, де можливий витік особистої інформації. Для організації безпеки даних використовуються системи захисту. Вони бувають апаратними, або ж як у випадку з вразливостями Meltdown/Spectre, програмними [2].

Для захисту системи від вразливостей Meltdown/Spectre, виробники BIOS та операційних систем створили програмні версії систем захисту. Варто зазначити, що апаратна система захисту присутня лише в комп'ютерних системах на базі процесорів 2020 лінійного року. Основна проблема програмних систем захисту, це вплив на швидкодію процесора і системи в цілому [3, 4].

Дослідження впливу існуючих програмних засобів захисту центральних процесорів на швидкодію дасть відповідь наскільки критичний вплив цих засобів. Дослідження буде проводитися в ігрових застосунках (для отримання середніх значень та 1%/0.1% fps) та бенчмарках, а саме - Cinebanch r20 (кількість балів) та X 265 (час перекодування відеофайлу) із використанням різних версій BIOS та Windows [5].

Це дозволить визначити які версії програмних засобів захисту типу BIOS та Windows використовувати, щоб зменшити негативний вплив програмних систем захисту на робочу станцію.

Література

1. Tanenbaum A.S., Austin T. Structured Computer Organization. Pearson, 2013. 775p.
2. База даних загальновідомих вразливостей інформаційної безпеки [Електронний ресурс] – Режим доступу до ресурсу: https://cve.mitre.org/.
3. Kocher P., Horn J., Fogh A., Genkin D., Gruss D., Haas W., Hamburg M., Lipp M., Mangard S., Prescher T., Schwarz M., Yarom Y. Spectre Attacks: Exploiting Speculative Execution. San Francisco, California, 2019. 19p.
4. Lipp M., Schwarz M., Gruss D., Prescher T.,Haas W., Fogh A., Horn J., Mangard S.,Kocher P., Genkin D.,, Yarom Y., Hamburg M. Meltdown: Reading Kernel Memory from User Space. San Francisco, California, 2019. 18p.
5. Засоби тестування процесору та системи [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/computer-stress-test-software/.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

VIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

9–10 грудня 2020 року

ТЕРНОПІЛЬ
2020

УДК 004.4
В.М. Леськів, Н.С. Луцик докт. філос.
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ТЕХНОЛОГІЇ КОМП'ЮТЕРИЗОВАНОГО АНАЛІЗУ ТА ВІЗУАЛІЗАЦІЇ БІОМЕДИЧНИХ ДАНИХ ПАЦІЄНТА

UDC 004.4
V.M. Leskiv, N.S. Lutsyk Ph.D.

TECHNOLOGIES FOR COMPUTER ANALYSIS AND VISUALIZATION OF PATIENT BIOMEDICAL DATA

Актуальність теми

Питання збереження, догляду і покращення здоров'я людини завжди було, є і буде актуальним серед всіх медичних закладів не тільки України, а й усього світу, особливо, у наш час, коли кількість хворих від різних вірусів є неймовірно великою. Тому процес структуризації і відображення біомедичних даних пацієнта буде завжди актуальним в медичній галузі.

Переваги реалізації медичної інформаційної системи, як Веб додатку

Оскільки, весь світ проходить процес діджиталізації, що являє собою розміщення всієї необхідної особистої інформації в мережі Інтернет, важливо, щоб кожна людина мала до неї постійний, безпечний і, головне, зручний доступ. Розробка системи, як окремого Веб-додатку дозволяє дотримуватися усіх необхідних критерій, адже для того, щоб отримати необхідні дані, не потрібно стаціонарного комп'ютера з окремо встановленою прикладною програмою, достатньо лише смартфона з доступом до мережі Інтернет.

Особливості використання RIA веб-додатків

Особливістю використання RIA веб-додатків є те що вони використовують доступні ресурси браузера клієнта, щоб взаємодіяти із користувачем, що дозволяє досягти наступних переваг:

- Зручний інтерфейс
- Інтерактивність
- Збалансованість клієнт-сервера
- Асинхронна комунікація

Враховуючи всі ці аспекти, використання RIA є досить обумовленим в процесі побудови медичної інформаційної системи, оскільки дозволяє вирішити ряд важливих питань, які потрібні для коректної роботи системи і забезпечити зручний і швидкий процес отримання результатів, навіть, з зовнішніх машин, наприклад, калькулятора.

Стандарт HL7, як основний елемент обміну, управління та інтеграції електронної медичної інформації

Використання стандарту HL7 дозволяє досягнути усіх необхідних для нормального функціонування медичної системи результатів. Оскільки, його ступінь стандартизації є досить великим – це дозволяє поєднувати його вже з існуючими місцевими системи стандартів та його місцевими варіаціями. Застосування HL7 стандарту дозволяє обмінюватись необхідною інформацією між системами, які функціонують на самому широкому спектрі технічних засобів і мов програмування.

Головна мета застосування даного стандарту – це забезпечення єдиного стандарту сумісності, щоб покращити постачання медичного піклування, оптимізацію робочого процесу, зменшити невизначеність і підвищити передачу знань між усіма зацікавленими сторонами, у тому числі медичних працівників, державних установ, спільноти постачальників, пацієнтів

Додаток Б

Лістинг конфігураційних фалів серверу

Файл *pom.xml*

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>LiVEZer.Medicine</groupId>
  <artifactId>LiVEZer.Medicine</artifactId>
  <version>0.1.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>LiVEZer.Medicine</name>
  <description>LiVEZer.Medicine - WebApp</description>
  <dependencies>
    <!-- Spring MVC 3.2.4.RELEASE -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>3.2.4.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-orm</artifactId>
      <version>3.2.4.RELEASE</version>
    </dependency>
    <!-- Hibernate 4.2.6.Final -->
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>4.2.6.Final</version>
    </dependency>
    <dependency>
      <groupId>org.hibernate.common</groupId>
      <artifactId>hibernate-commons-annotations</artifactId>
      <version>4.0.4.Final</version>
    </dependency>
    <dependency>
      <groupId>org.hibernate.javax.persistence</groupId>
      <artifactId>hibernate-jpa-2.0-api</artifactId>
      <version>1.0.1.Final</version>
    </dependency>
    <dependency>
      <groupId>com.microsoft.sqlserver</groupId>
      <artifactId>sqljdbc4</artifactId>
      <version>3.0</version>
    </dependency>
    <!-- Commons Lang -->
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-lang3</artifactId>
      <version>3.1</version>
    </dependency>
    <dependency>
      <groupId>commons-codec</groupId>
```

```
      <artifactId>commons-codec</artifactId>
      <version>20041127.091804</version>
    </dependency>
    <!-- Apache Log4j -->
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.17</version>
    </dependency>
    <!-- Jackson CodeHouse -->
    <dependency>
      <groupId>org.codehaus.jackson</groupId>
      <artifactId>jackson-core-asl</artifactId>
      <version>1.9.13</version>
    </dependency>
    <dependency>
      <groupId>org.codehaus.jackson</groupId>
      <artifactId>jackson-mapper-asl</artifactId>
      <version>1.9.13</version>
    </dependency>
    <!-- json-lib -->
    <dependency>
      <groupId>net.sf.json-lib</groupId>
      <artifactId>json-lib</artifactId>
      <classifier>jdk15</classifier>
      <version>2.4</version>
    </dependency>
    <!-- java-object-merger -->
    <dependency>
      <groupId>net.sf.brunneng.jom</groupId>
      <artifactId>java-object-merger</artifactId>
      <version>0.8.2</version>
    </dependency>
  </dependencies>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <resources>
      <resource>
        <directory>src</directory>
        <excludes>
          <exclude>**/*.java</exclude>
        </excludes>
      </resource>
    </resources>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.1</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.3</version>
        <configuration>
          <warSourceDirectory>WebApp</warSourceDirectory>
          <failOnMissingWebXml>false</failOnMissingWebXml>
```

```
            </configuration>
          </plugin>
        </plugins>
    </build>
</project>
```

Файл *web.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">
    <display-name>LiVEZer.Medicine</display-name>
    <!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/root-context.xml</param-value>
    </context-param>
    <!-- Log4j Configuration -->
    <context-param>
        <param-name>log4jConfiguration</param-name>
        <param-value>/WEB-INF/log4j.xml</param-value>
    </context-param>
    <!-- Creates the Spring Container shared by all Servlets and Filters -->
    <listener>
        <listener-class>LiVEZer.Medicine.WebApp.AppContextListener</listener-class>
    </listener>
    <!-- Processes application requests -->
    <servlet>
        <servlet-name>LiVEZer.Medicine-Servlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring/servlets/LiVEZer.Medicine-Servlet.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>LiVEZer.Medicine-Servlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
    <!-- The definition of custom servlets -->
    <servlet>
        <servlet-name>BaseServlet</servlet-name>
        <servlet-class>LiVEZer.Medicine.WebApp.Servlets.BaseServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>BaseServlet</servlet-name>
        <url-pattern>/do</url-pattern>
    </servlet-mapping>
</web-app>
```

Файл *LiVEZer.Medicine-Servlet.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
```

```
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
      http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context.xsd">
   <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->
   <!-- Enables the Spring MVC @Controller programming model -->
   <annotation-driven />
   <context:annotation-config />
   <context:property-placeholder location="/WEB-INF/main.properties" />
   <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources
      in the ${webappRoot}/resources directory -->
   <resources mapping="/css/**" location="/WEB-INF/css/" />
   <resources mapping="/js/**" location="/WEB-INF/js/" />
   <resources mapping="/img/**" location="/WEB-INF/img/" />
   <resources mapping="favicon.ico" location="/WEB-INF/favicon.ico" />
   <!-- Resolves views selected for rendering by @Controllers to .jsp
      resources in the /WEB-INF/views directory -->
   <beans:bean id="jspViewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <beans:property name="prefix" value="/WEB-INF/views/" />
      <beans:property name="suffix" value=".jsp" />
   </beans:bean>
   <context:component-scan base-package="LiVEZer.Medicine.WebApp.Controllers" />
</beans:beans>
```

Файл *hibernate.cfg.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
                                "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory name="LiVEZer.Medicine.Session">
      <property name="hibernate.connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>
      <property name="hibernate.connection.url">jdbc:sqlserver://HOT-
DEVICE:1189;instanceName=DBLOCALE;databaseName=medicineTest;integratedSecurity=false;</property>
      <property name="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</property>
      <property name="hibernate.connection.pool_size">7</property>
      <property name="hibernate.show_sql">true</property>
      <property name="hibernate.format_sql">true</property>
      <property name="hibernate.hbm2ddl.auto">update</property>
      <property name="hibernate.use_sql_comments">true</property>
      <property name="hibernate.session_factory_name">LiVEZer.Medicine.Session</property>
      <!-- Credentials -->
      <property name="hibernate.connection.username">LOCAL_USER</property>
      <property name="hibernate.connection.password">1111</property>
   </session-factory>
</hibernate-configuration>
```

Файл *log4j.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J 1.2//EN" "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
   <!-- Appenders -->
   <appender name="consoleAppender" class="org.apache.log4j.ConsoleAppender">
      <param name="Target" value="System.out" />
      <layout class="org.apache.log4j.PatternLayout">
```

```
            <param name="ConversionPattern" value="%-5p: %c - %m%n" />
        </layout>
    </appender>
    <appender name="fileAppender" class="org.apache.log4j.RollingFileAppender">
        <param name="File" value="${catalina.base}/logs/LIVZer.Medicine.log" />
        <param name="Append" value="true" />
        <param name="MaxBackupIndex" value="20" />
        <param name="MaxFileSize" value="20MB" />
        <param name="DatePattern" value="'.'yyyy-MM-dd" />
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d - %t %-5p %c{2} - %m%n" />
        </layout>
    </appender>
    <appender name="mainAppender" class="org.apache.log4j.AsyncAppender">
        <appender-ref ref="consoleAppender" />
        <appender-ref ref="fileAppender" />
    </appender>
    <logger name="LiVEZer.Medicine.WebApp">
        <level value="debug" />
    </logger>
    <logger name="org.springframework.core">
        <level value="warn" />
    </logger>
    <logger name="org.springframework.beans">
        <level value="warn" />
    </logger>
    <logger name="org.springframework.context">
        <level value="warn" />
    </logger>
    <logger name="org.springframework.web">
        <level value="warn" />
    </logger>
    <logger name="org.hibernate">
        <level value="info" />
    </logger>
    <root>
        <priority value="warn" />
        <appender-ref ref="mainAppender" />
    </root>
</log4j:configuration>
```

Додаток В

Лістинг головних класів серверної частини.

Файл *AppManager.java*

```
package LiVEZer.Medicine.WebApp;

import javax.servlet.ServletContext;

import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;

import LiVEZer.Medicine.WebApp.Services.ServiceManager;

public final class AppManager
{
   private static AppManager instance;

   public static AppManager GetInstance()
   {
      if (instance == null)
      {
         synchronized (AppManager.class)
         {
            instance = new AppManager();
         }
      }
      return instance;
   }

   private AppManager()
   {
   }

   private ServletContext context;
   private static final Logger logger = LogManager.getLogger(AppManager.class);

   public ServletContext getContext()
   {
      if (context == null)
      {
         throw new IllegalArgumentException("Context isn't set");
      }
      return context;
   }

   public void setContext(ServletContext context)
   {
      this.context = context;
   }

   public void Configure(boolean async)
   {
      configureLog4j();
```

```
        if (async)
        {
            Runnable run = new Runnable()
            {
                @Override
                public void run()
                {
                    configure();
                }
            };
            new Thread(run).start();
        }
        else
        {
            configure();
        }
    }

    private void configure()
    {
        synchronized (instance)
        {
            try
            {
                ServiceManager.Initialize();
                DBManager.RefreshConnection();
                SessionManager.InitializeSessions();
            }
            catch (Exception e)
            {
                logger.error("Can't Configure Application", e);
            }
        }
    }

    private void configureLog4j()
    {
        String log4jFile = context.getRealPath("/")
                + context.getInitParameter("log4jConfiguration");
        DOMConfigurator.configure(log4jFile);
        logger.info("Configuration Loaded: " + log4jFile);
    }
}
```

Файл *DBManager.java*

```
package LiVEZer.Medicine.WebApp;

import java.lang.annotation.Annotation;
import java.util.ArrayList;

import javax.persistence.Entity;

import org.apache.log4j.Logger;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;
import org.hibernate.service.ServiceRegistryBuilder;
```

```
import org.springframework.beans.factory.config.BeanDefinition;
import org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider;
import org.springframework.core.type.filter.AnnotationTypeFilter;

public final class DBManager
{
   private static final Logger logger = Logger.getLogger(DBManager.class);

   private static SessionFactory SESSION_FACTORY;

   private static ServiceRegistry SERVICE_REGISTRY;

   private static Configuration addAnnotatedClasses(Configuration config) throws Exception
   {
      logger.info("DBManager.addAnnotatedClasses()");

      ArrayList<Class<?>> classes = new ArrayList<Class<?>>();
      ClassPathScanningCandidateComponentProvider scanner =
            new ClassPathScanningCandidateComponentProvider(false);
      scanner.addIncludeFilter(new AnnotationTypeFilter(
            (Class<? extends Annotation>) Entity.class));
      for (BeanDefinition bd : scanner
            .findCandidateComponents("LiVEZer.Medicine.WebApp.DAO.Models"))
      {
         String name = bd.getBeanClassName();
         try
         {
            classes.add(Class.forName(name));
         }
         catch (Exception e)
         {
            logger.error("Can't add Entities", e);
            throw new Exception("Can't add Entities", e);
         }
      }

      for (Class<?> c : classes)
      {
         config.addAnnotatedClass(c);

         logger.debug(String.format("Entity {\"%s\"} added", c.getName()));
      }
      return config;
   }

   private static SessionFactory buildSessionFactory()
   {
      logger.info("DBManager.buildSessionFactory()");

      try
      {
         Configuration config = new Configuration();
         config.configure("config/hibernate.cfg.xml");
         SERVICE_REGISTRY = new ServiceRegistryBuilder().applySettings(
               addAnnotatedClasses(config).getProperties())
               .buildServiceRegistry();
         SESSION_FACTORY = config.buildSessionFactory(SERVICE_REGISTRY);

         logger.info("DBManager.buildSessionFactory() - Session Factory Created");
```

```
            return SESSION_FACTORY;
        }
        catch (Exception e)
        {
            logger.error("Can't create session!!!", e);
            throw new ExceptionInInitializerError(e);
        }
    }

    public static SessionFactory getSessionFactory()
    {
        logger.info("DBManager.getSessionFactory()");
        synchronized (DBManager.class)
        {
            return (SESSION_FACTORY != null) ? SESSION_FACTORY
                    : (SESSION_FACTORY = buildSessionFactory());
        }
    }

    public static void RefreshConnection()
    {
        logger.info("DBManager.RefreshConnection() - Begin Invoke");

        if (SESSION_FACTORY != null)
        {
            if (!SESSION_FACTORY.isClosed())
            {
                SESSION_FACTORY.close();
            }
            SESSION_FACTORY = null;
        }

        getSessionFactory();

        logger.info("DBManager.RefreshConnection() - End Invoke");
    }

    public static boolean CloseSession(Session session)
    {
        boolean closed = false;
        try
        {
            session.close();
            closed = true;
        }
        catch (Exception e)
        {
            logger.error("Error when close session", e);
        }

        logger.info("Session closed!");
        return closed;
    }

}
```

Файл *ServiceManager.java*

```
package LiVEZer.Medicine.WebApp.Services;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.lang3.StringUtils;
import org.apache.log4j.Logger;
import org.springframework.beans.factory.config.BeanDefinition;
import org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider;
import org.springframework.core.type.filter.AssignableTypeFilter;

import LiVEZer.Medicine.WebApp.Services.JSONResponse.Error;
import LiVEZer.Medicine.WebApp.Services.JSONResponse.JSONResponse;
import LiVEZer.Medicine.WebApp.Services.Methods.*;

public final class ServiceManager
{
   private static final Logger logger = Logger.getLogger(ServiceManager.class);

   private static ServiceManager instance;

   private Map<String, Class<?>> methodMap;

   private ServiceManager()
   {
      InitializeMethodsMap();
   }

   private final static ServiceManager GetInstance()
   {
      if (instance == null)
      {
         synchronized (ServiceManager.class)
         {
            instance = new ServiceManager();
         }
      }

      return instance;
   }

   public static void Initialize()
   {
      GetInstance();
   }

   public static JSONResponse doWithResponse(String method, HttpServletRequest request,
         HttpServletResponse response)
   {
      logger.info(String.format("Begin execute method {\"%s\"}", method));

      JSONResponse jsonResponse;
      if (StringUtils.isNotBlank(method) && GetInstance().methodMap.containsKey(method))
      {
         try
```

```
        {
            IServiceMethod imetod = (IServiceMethod) GetInstance().methodMap.get(method)
                    .newInstance();
            jsonResponse = imetod.doMethod(request, response);
        }
        catch (Exception ex)
        {
            Error error = new Error();
            error.setSuccess(false);
            error.setCode(1);
            error.setMessage(String.format("Can't execute \"%s\" method", method));
            jsonResponse = error;
            logger.error(String.format("Can't execute \"%s\" method", method), ex);
        }
    }
    else
    {
        Error error = new Error();
        error.setSuccess(false);
        error.setCode(0);
        error.setMessage(String.format("Method %s doesn't exist in system!",
                method == null ? "\"null\"" : method));
        jsonResponse = error;
    }

    logger.debug("Response: " + jsonResponse.toString());
    return jsonResponse;
}

public static void doWithoutResponse(String method, HttpServletRequest request,
        HttpServletResponse response) throws IOException
{
    JSONResponse jsonResponse = doWithResponse(method, request, response);
    Write(response, jsonResponse);
}

private static void Write(HttpServletResponse response, JSONResponse json)
        throws IOException
{
    response.setContentType("application/json");
    response.setCharacterEncoding("UTF-8");
    response.getWriter().write(json.toString());
}

private void InitializeMethodsMap()
{
    logger.info("ServiceManager.InitializeMethodsMap()");

    methodMap = new HashMap<String, Class<?>>();
    String packageName = "LiVEZer.Medicine.WebApp.Services.Methods";
    ClassPathScanningCandidateComponentProvider scanner =
            new ClassPathScanningCandidateComponentProvider(false);
    scanner.addIncludeFilter(new AssignableTypeFilter(IServiceMethod.class));
    for (BeanDefinition bd : scanner.findCandidateComponents(packageName))
    {
        String name = bd.getBeanClassName();
        try
        {
            IServiceMethod method = (IServiceMethod) Class.forName(name).newInstance();
```

```
                methodMap.put(method.getMethodName(), method.getClass());

                logger.debug(String.format("Method {\"%s\"} added", method.getMethodName()));
            }
            catch (Exception e)
            {
                logger.error("Can't add: " + name, e);
            }
        }
    }
}
```