МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія проектування репозиторію обліку пацієнтів університетської клініки СумДУ»

Завідувач випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Берест О.Б.**

Студентка групи ІН.м-02 **Мучарова Є.О.**

СУМИ 2021

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Мучаровій Євгенії Олександрівні
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія проектування репозиторію університетської клініки СумДУ*
затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналітичний предметної області; 2) Огляд існуючих програмних рішень; 3) Постановка задачі; 4) Проектування майбутнього додатку; 5)Виконання проектного завдання.*

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз проблеми. Постановка задачі дослідження*		
2.	*Вибір програмного забезпечення для реалізації інформаційної системи-репозиторію*		
3.	*Проектування веб-додатку*		
4.	*Розробка веб-додатку*		
5.	*Оформлення пояснювальної записки до кваліфікаційної магістерської роботи*		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 60 стор., 12 рис., 1 таблиця, 1 додаток, 22 літературних джерела.

Об’єкт дослідження — технології проектування репозиторію для закладів охорони здоров’я.

Мета роботи — розробка та впровадження технології Web-системи лікарні, що дозволяє зберігати інформацію про пацієнтів та їхні прийоми у лікарів.

Методи дослідження — метод аналітичного огляду, метод порівняння та аналогій, метод моделювання, методи розробки;

Результати — проведено інформаційний огляд, обрані методи реалізації, виконано розробку інформаційної системи. Веб-додаток надає змогу вносити нових пацієнтів та дані про їх медичні процедури до бази даних, переглядати історії медичних процедур та знаходити дані про пацієнтів за їх прізвищем.

ІНФОРМАЦІЙНИЙ РЕПОЗИТОРІЙ, КЛІНІКА, ПРОЦЕДУРА, ВЕБ ДОДАТОК, БАЗА ДАНИХ

Зміст

ВСТУП

У сучасному світі питання автоматизації монотонних, рутинних процесів діяльності людини постає все більш гостро. Коріння автоматизації проникає в усі сфери людського життя, починаючи з побутових необхідностей, таких як закупівля продуктів харчування або їх приготування, і закінчуючи сферою охорони здоров’я.

Нині, у зв'язку з пандемією, виникла необхідність зменшення часу прийому в лікаря у поліклініці, більшість з якого йде на запис історії хвороби, результатів обстеження і т.д. Також, наразі історії хвороби знаходяться на руках у самих пацієнтів, що може створювати проблему розпізнавання чужого почерку, або взагалі несе загрозу втрати чи псування документу. Тому у галузі медицини та здоров’я кожного дня спостерігаються все більш стрімкі тенденції на використання засобів автоматизації, таких як електронні журнали пацієнтів, електронні черги, електронний запис до лікаря і інші системи, що спрощують перебування в лікарнях як для медичного персоналу, так і для хворих.

Саме з цією тематикою і пов’язана тема даної роботи.

Виходячи з усього вищесказаного, можна зробити висновок про високу актуальність даної розробки.

Об’єктом дослідження є методи і інструменти створення веб-додатків.

Предметом дослідження є процес розробки інформаційної системи-репозиторію обліку пацієнтів для університетської клініки Сумського державного університету, що розпочала надавати послуги населенню північно-східного регіону України з лютого 2017 року. База даних лікарні, що призначена для зберігання інформації про пацієнтів з можливістю внесення даних, вибірки і зміни даних, виведення інформації в необхідному форматі не тільки полегшить роботу медичного персоналу, але і збільшить її якість. У поліклініці забезпечено можливість проведення сучасних діагностичних методик обстеження хворих: працює біохімічна лабораторія, кабінети

функціональних методів дослідження та УЗД, результати яких доречно було б зберігати в інформаційній системі.

Мета роботи: теоретично обґрунтувати, розробити та апробувати технологію Web-системи для лікарні.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальна характеристика

1.1.1 Інформаційна система (ІС)

Інформаційний репозиторій – це сукупність взаємопов’язаної інформації, яка зберігається в мережі на кількох серверах. Він створює уніфікований ресурс для будь-кого, хто пов’язаний із системою, щоб отримати доступ, коли йому потрібна інформація. Репозиторій регулярно розміщує відповідні дані разом з метаінформацією, включаючи зображення, відео та текст. Користувачі, які хочуть отримати доступ до інформації, можуть запустити пошук, щоб знайти матеріали, що відповідають їхнім інтересам. Також вони можуть звузити пошук за типом, щоб знайти конкретні матеріали.

У сенсі підходу до управління даними інформаційний репозиторій є вторинним місцем зберігання даних. Коли дані більше не потрібні на первинному сервері або об’єкті, їх можна перемістити в репозиторій для архівних цілей. Це звільняє місце на системах, які зараз використовуються, не знищуючи дані, які можуть знадобитися пізніше. Процес може бути автоматизований, щоб полегшити безперебійне функціонування системи та забезпечити регулярну передачу даних, а не безсистемну [1].

Інформаційні системи – це взаємопов’язані компоненти, що працюють разом для збору, обробки, зберігання та поширення інформації для підтримки прийняття рішень, координації, контролю, аналізу та візуалізації в організації [2].

Інформаційна система, по суті, складається з п'яти компонентів: апаратного забезпечення, програмного забезпечення, даних, мережі та людей. Ці п’ять компонентів об’єднуються для виконання введення, процесу, виходу, зворотного зв’язку та керування [3].

- Програмне забезпечення: включає всі набори інструкцій з обробки інформації. Це не тільки набори операційних інструкцій, які називаються програмами, які спрямовують і керують апаратним забезпеченням комп'ютера,

а й набори інструкцій обробки інформації, необхідні людям, які називаються процедурами.

- Апаратне забезпечення: всі фізичні пристрої та матеріали, що використовуються при обробці інформації. Зокрема, це не тільки машини, такі як комп’ютери та інше обладнання, а й усі носії даних, тобто всі матеріальні об’єкти, на яких записуються дані, від аркушів паперу до магнітних дисків. Наприклад:
    - Комп'ютерні системи, які складаються з центральних процесорів, що містять мікропроцесори, і різноманітних взаємопов'язаних периферійних пристроїв (мікрокомп’ютерні системи, комп’ютерні системи середнього рівня та великі мейнфрейми.)
    - Комп’ютерні периферійні пристрої, які являють собою такі пристрої, як клавіатура або електронна миша для введення даних і команд, відеоекран або принтер для виведення інформації.
- Дані: факти, які використовуються системами для отримання корисної інформації. У сучасних інформаційних системах дані, як правило, зберігаються вигляді на диску або стрічці, поки комп’ютер не потребує їх.
- Комп’ютерна мережа: сукупність комп’ютерів та іншого обладнання, з’єднаних між собою каналами зв’язку, які дозволяють обмінюватися ресурсами та інформацією. Якщо принаймні один процес в одному пристрої може надсилати/отримувати дані до/від принаймні одного процесу, що знаходиться на віддаленому пристрої, тоді кажуть, що два пристрої знаходяться в мережі.
- Люди: кожна система потребує людей, щоб бути корисною. Часто елементом інформаційної системи, на який найчастіше не звертають уваги, є люди, ймовірно, той компонент, який найбільше впливає на успіх чи невдачу інформаційних систем.
    - Користувачі — це люди, які використовують інформаційну систему або інформацію, яку вона створює. Це можуть бути

бухгалтери, продавці, інженери, клерки, клієнти, адміністратори і т.д.

- Фахівці з ІС – це люди, які розробляють та експлуатують інформаційні системи. До них входять системні аналітики, програмісти, оператори комп’ютерів та інший управлінський технічний та канцелярський персонал ІС.

Інформаційні системи пронизують кожну сферу сучасного життя та є дуже різноманітними за своїм призначенням:

- економічна інформаційна система – інформаційна система, що охоплює майже всю діяльність суб'єктів господарювання, від комп'ютеризованих систем підтримки продажів, логістики, бухгалтерського обліку, звітності, управління ризиками до маркетингової діяльності в Інтернеті.
- інформаційна система управління освітою — це система, яка контролює виконання освітніх програм, які пропонує навчальний заклад, керує розподілом освітніх ресурсів, розробляє стратегію для впровадження робочих процесів для безперебійної роботи системи освіти.
- медична інформаційна система – інформаційна системи, призначена для автоматизації всіх основних процесів, пов'язаних з роботою медичних закладів загальної та вузької спеціалізації. Такі системи дозволяють налагодити електронний документообіг, вибудовувати роботу з пацієнтами, вести оперативний облік роботи адміністративного персоналу, контролювати всі організаційні та фінансові питання.

1.1.2 Веб-додатки

Веб-сайт — це сукупність веб-сторінок і пов’язаного з ними вмісту, ідентифікованих публічним доменом і опублікованих принаймні на одному веб-сервері. Усі загальнодоступні веб-сайти разом утворюють всесвітню мережу. Існують також деякі приватні веб-сайти, доступ до яких можна отримати лише через приватні мережі, наприклад внутрішні веб-сайти співробітників компанії.

Веб-сайти зазвичай присвячені конкретним темам або цілям, таким як новини, освіта, бізнес, розваги або соціальні мережі. Гіперпосилання між веб-сторінками спрямовують навігацію по сайту, зазвичай починаючи з домашньої сторінки.

Існує багато способів використання веб-сайту: особистий веб-сайт, корпоративний веб-сайт, державний веб-сайт, веб-сайт організації тощо. Веб-сайти можуть бути роботою окремих осіб, компаній чи інших організацій і, як правило, присвячені конкретним темам або цілям. Будь-який веб-сайт може містити гіперпосилання на будь-який інший веб-сайт, тому відмінність між різними веб-сайтами, які сприймається користувачем, може бути розмитою.

Статичний веб-сайт — це комбінація HTML-розмітки (текст, який ми бачимо, написаний на веб-сторінках) та CSS (каскадні таблиці стилів), які застосовуються для керування зовнішнім виглядом сайту. Статичний веб-сайт містить веб-сторінки з фіксованим вмістом. Кожна сторінка закодована в HTML і відображає однакову інформацію для кожного відвідувача. Ці веб-сторінки зазвичай надають інформацію (про якийсь продукт, компанію та її послуги) за допомогою тексту, зображення, відео та аудіо. На відміну від динамічних веб-сайтів, для створення статичного веб-сайту не потрібні знання веб-програмування та дизайну баз даних.

Динамічні веб-сайти – це веб-сайти, які часто змінюються або коригуються автоматично. Динамічні сторінки на сервері генеруються за допомогою комп'ютерного коду, що генерує HTML (CSS відповідає за зовнішній вигляд, а отже, і за статичні файли). Існує безліч програмних систем, таких як CGI, Java Servlet і Java Server Pages (JSP), Active Server Pages і ColdFusion (CFML), які можна використовувати для створення динамічних веб-систем і динамічних веб-сайтів. Поширені мови програмування, такі як Perl, PHP, Python, Ruby, забезпечують різноманітні фреймворки веб-додатків і системи веб-шаблонів, що полегшує та спрощує створення складних і динамічних веб-сайтів [4].

Веб-сайти можна розділити на дві категорії – статичні та інтерактивні. Інтерактивні сайти є частиною спільноти Web 2.0, що дозволяє вам взаємодіяти між власниками сайтів і відвідувачами або користувачами сайту. Статичні сайти надають або збирають інформацію, але не дозволяють безпосередньо взаємодіяти з аудиторією чи користувачами. Веб-розробка — це робота з розробки веб-сайтів для Інтернету (World Wide Web) або Інтранету (приватна мережа). [5] Веб-розробка може варіюватися від розробки простих статичних сторінок із простим текстом до складних Інтернет-програм (інтернет-додатків), електронної комерції та послуг соціальних мереж. Більш повний список завдань, який зазвичай включає веб-розробку, може включати веб-інженерію, веб-дизайн, розробку веб-контенту, комунікації з клієнтами, сценарії клієнт/сервер, налаштування безпеки веб-сервера та мережі, а також розробку електронної комерції.

Серед веб-експертів «веб-розробка» зазвичай відноситься до основних не-дизайнерських аспектів створення веб-сайту: написання розмітки та кодування. [6] Веб-розробник може використовувати систему керування вмістом (CMS), щоб зробити вміст простішим і доступнішим завдяки базовим технічним навичкам. Оскільки нові веб-програми знаходять нові вразливості безпеки навіть після їх тестування та запуску, виправлення безпеки часто оновлюються для широко використовуваних програм. Коли випущені виправлення безпеки та виявлені нові проблеми безпеки, веб-розробникам часто доводиться оновлювати свої програми.

1.1.3 Заклади охорони здоров’я

Відповідно до Наказу «Про затвердження переліків закладів охорони здоров'я, лікарських посад, посад фармацевтів, посад фахівців з фармацевтичною освітою (асистентів фармацевтів), посад професіоналів у галузі охорони здоров'я, посад фахівців у галузі охорони здоров'я та посад професіоналів з вищою немедичною освітою у закладах охорони здоров'я» [7] лікарня – це лікувально-профілактичний заклад, призначений для надання стаціонарної медичної допомоги хворим. Лікарня, що надає стаціонарну

медичну допомогу хворим з лікарських спеціальностей одного профілю називається однопрофільною, а з декількох лікарських спеціальностей - багатопрофільною. Однопрофільні лікарняні заклади створюються для надання медичної допомоги населенню певної території, спеціалізовані - для надання спеціалізованої медичної допомоги населенню регіону. У своєму складі лікарня може мати поліклініку чи амбулаторію. Існує класифікація медичних закладів за типами [8]:

- лікарня — універсальний медичний заклад, що пропонує своїм пацієнтам широкий спектр медичних, хірургічних та психіатричних послуг;
- клінічна лікарня - заклад охорони здоров'я, у якому розміщуються наукові і навчальні структурні підрозділи вищих медичних навчальних закладів III - IV рівня акредитації, закладів післядипломної освіти, науково-дослідних інститутів, що поєднує допомогу людям із навчанням студентів-медиків та медсестер;
- медичний кабінет — заклад охорони здоров’я, що зазвичай спеціалізується в одній або кількох галузях медицини та пропонує амбулаторне лікування, не передбачає нічного перебування своїх пацієнтів;
- будинок престарілих — заклад, що забезпечує цілодобовий догляд за людьми похилого віку та інвалідами;
- центр психічного здоров'я та лікування наркоманії —загальна установа, що надає психіатричну та психологічну допомогу своїм пацієнтам, або спеціалізується на одній конкретній галузі психічного здоров’я.
- пологовий центр;
- хоспіс;
- санаторій та ін.

Якого би типу не був заклад охорони здоров’я, кожен з них працює з великим обсягом інформації, оскільки сучасні технології розширюють спектр діагностичних можливостей і розширюють можливості лікування. У результаті більшої кількості послуг та доступних методів лікування та

операцій потрібен більш висококваліфікований персонал та надійніші засоби збереження інформації, що циркулює в медичних закладах. Поєднання досліджень, інженерії та біотехнології створило широкий вибір нових методів лікування та інструментів, велика частина яких вимагає спеціальної підготовки та обладнання для використання. Також лікарям потрібно завжди стежити за даними своїх пацієнтів, а керівництву за своїми співробітниками [9]. Таким чином, все більше і більше організацій впроваджують електронні медичні записи (ЕМЗ) лікарні [10], що спираються на технологію реляційних баз даних (рис. 1.1-1.2), де структури даних - включаючи таблиці, індекси та представлення даних - залишаються відокремленими від структур фізичного зберігання, що дозволяє редагувати фізичне сховище даних, не впливаючи на їх логічну структуру.

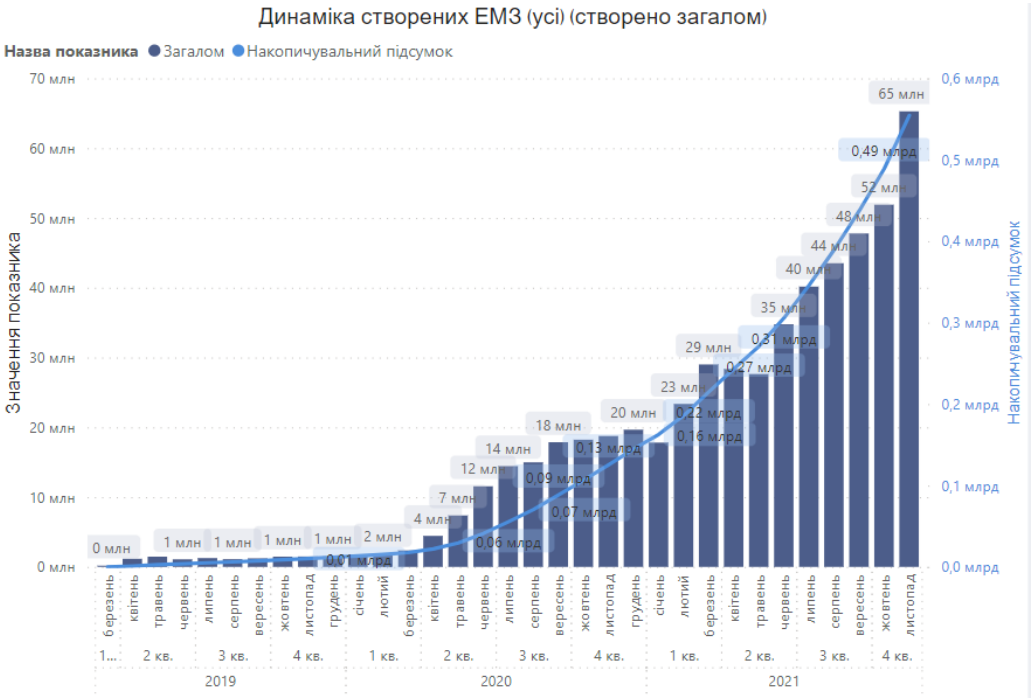

Рисунок 1.1 — Динаміка створення ЕМЗ за 2019-2021 роки

Рисунок 1.2 — Динаміка створення діагностичних звітів за 2019-2021 роки

Для лікарів такий формат медичної карти не просто зручний, він принципово змінює систему їхньої роботи:

- Кожен лікар має швидкий доступ до всіх важливих даних.
- Пацієнти можуть зберігати та керувати медичними даними незалежно від документальних зобов’язань лікарів.
- Можливість в автоматичному режимі, відбирати дані за деякою ознакою або набором ознак, впорядковувати їх відображення по різних стовпцях таблиці, наприклад, дати, прізвища, діагнозу.
- Можливість відновлення медичної картки з електронної історії хвороби у разі псування паперової.
- Електронний формат медичних записів зручний для формування звітності та аналізу діяльності клініки.

Аналізуючи переваги ІС в медичних закладах потрібно розглянути і недоліки:

- По статистиці багато лікарів вважають, що рукописні записи є більш деталізованими та містять більше відповідних деталей.
- Помилки введення, неправильний діагноз або маніпулювання введенням можуть призвести до неправильних медичних висновків.
- Захист даних та медична конфіденційність; недотримання цього протоколу може поставити під загрозу записи пацієнтів.

1.2 Опис методу дослідження

Університетська клініка СумДУ є лікарнею, що поєднує допомогу людям із навчанням студентів-медиків та медсестер. Клініка надає широкий спектр медичних послуг [11]:

- Лабораторна діагностика (клінічний аналіз крові, біохімічний аналіз крові, аналіз сечі, мікробіологічний аналіз, гістологічний аналіз, цитологічний аналіз).
- Ультразвукова діагностика (органи малого тазу у жінок, органи малого тазу у чоловіків, органи черевної порожнини та заочеревинного простору, УЗД молочної залози, УЗД щитоподібної залози).
- Рентгенографія (рентген легень, рентген носових пазух, мамографія, рентген шлунка, іригоскопія, рентген кісток, суглобів, а також дентальний рентген).
- Комп'ютерна томографія (органи черевної порожнини, головний мозок, органи грудної клітки, органи малого тазу, органи сечовидільної системи, опорно-рухову систему).
- Функціональна діагностика.

Наразі, в клініці СумДУ немає можливості подивитися результати всіх цих досліджень онлайн (рис. 1.3), тільки в паперовому вигляді.

Рис. 1.3 — Інтерфейс вкладки «Результати аналізів» в університетській клініці СумДУ

Для вирішення цієї проблеми ми проглянемо інформаційні системи, що надають можливість збереження і перегляду результатів дослідження, а також проаналізуємо їх переваги та недоліки.

1.2.1Аналіз існуючих систем

USU (універсальна система обліку)

Універсальна Система Обліку застосовна в будь-якій сфері діяльності та на будь-якому підприємстві, встановлюється на будь-який електронний пристрій (планшет, ноутбук, комп'ютер). Медична програма від компанії УСО має оптимальний набір звітності та широкий функціонал для повноцінної роботи у будь-якому напрямку. Кожен логін ведення електронної історії хвороби захищається індивідуальним паролем. Ведення електронної історії хвороби передбачає окремі права доступу. Медичне програмне забезпечення дозволить автоматизувати формування та заповнення будь-яких бланків та документів (рис. 1.4). Медична система дозволяє одночасно вести облік фінансів ЛПЗ та реєстр відвідувань пацієнтів з проведеними лікувальними заходами в одній базі [12].

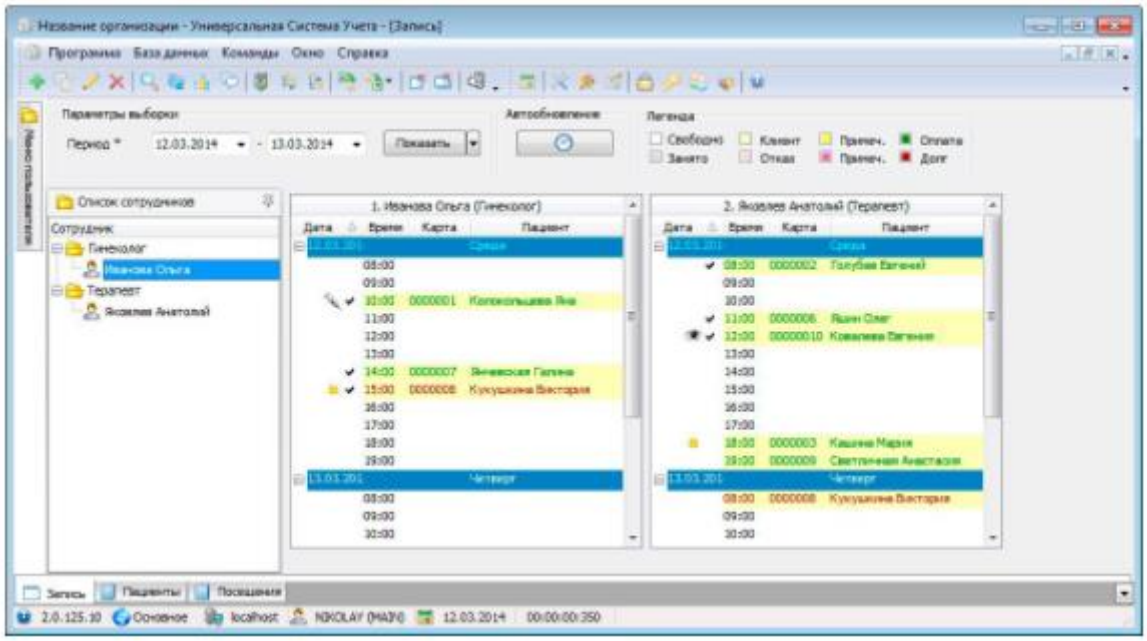

Рис. 1.4 — Інтерфейс USU

Основні можливості програми, що пропонуються на сайті:

- Підтримується будь-яка мова;

- Історія хвороби;
- Протоколи лікування;
- Результати аналізів, УЗД, знімків;
- Запис пацієнтів;
- Облік медикаментів;
- Система нотифікацій;
- Аналіз роботи співробітників.

З огляду на запропонований функціонал програма має переваги, але проаналізувавши детальніше можна виділити так недоліки:

- Висока вартість продукту;
- Складний функціонал, неповна інструкція по експлуатації;
- Додаткові витрати на консультації по користуванню програмою;
- Немає оперативного обліку та кваліфікованої технічної підтримки.

HELSI

Це сучасна, зручна та надійна електронна медична система, створена для пацієнтів, лікарів, державних та приватних медичних закладів. Медична інформаційна система допомагає автоматизувати роботу закладу охорони здоров'я, лікаря, лабораторії, стаціонару, ведення електронних медичних карток. Перевагами HELSI для пацієнтів є можливість швидкого запису на прийом on-line себе та членів своєї родини, доступ до своєї електронної медичної картки (ЕМК), миттєві результати аналізів та діагностики в кабінеті пацієнта. Перевагами HELSI для лікарів є зручне ведення історії хвороби пацієнтів, оперативне отримання результатів діагностики та аналізів, легке використання клінічних протоколів [13].

Не зважаючи на те, що більшість медичних закладів перейшло на інформаційну систему HELSI, вона має грубі недоліки та незадоволених клієнтів (рис. 1.5):

- Фіктивні записи в особистому кабінеті (прийоми, результати обстеження);

- Повільна робота системи, постійні збої;
- Інформація у пацієнта та лікаря відрізняється;
- Деякі пункти, що запропоновані в програмі, не відповідають дійсності.

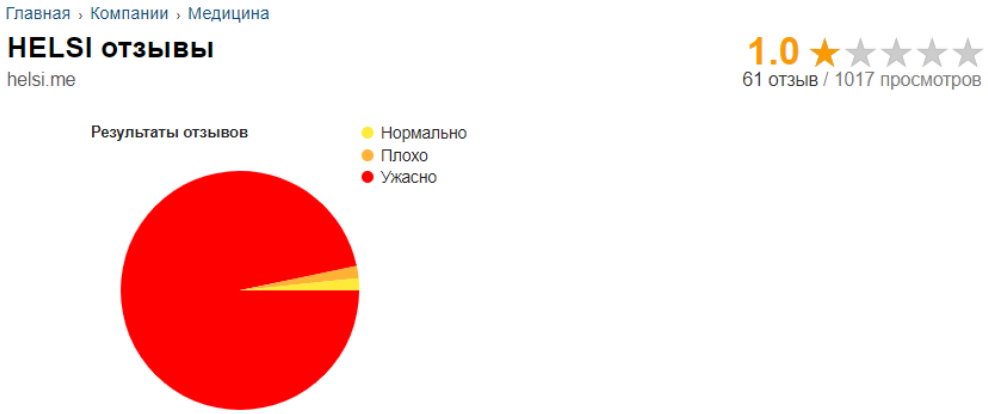

Рис. 1.5 — Результати відгуків користувачів HELSI

ASKEP.NET

Реалізує національну програму цифрової трансформації системи охорони здоров'я в якості медичної інформаційної системи, яка спеціалізується на медичному програмному забезпеченні та надає комплексні рішення для автоматизації медичних центрів, як державного так і комерційного сегменту. Одна із небагатьох систем на ринку, яка відтворює повноцінний документообіг в медичній установі; відповідає стандартам Національної служби здоров'я України, HL7 протоколу та працює із DICOM форматом.

Система використовується не лише в Україні але і у закордонних медичних закладах, є можливість коригування будь-якого поточного та додання унікального нового функціоналу під потреби закладу, пацієнти мають особистий кабінет, лікарі мають повну та оперативну інформацію про історію пацієнта. Також лікарі можуть завантажти зображення із УЗД/ЕКГ/Рентгену/КТ/МРТ до результату огляду пацієнта. Можна переглянути хто, коли і де здійснював забір аналізів, хто досліджував, результати аналізів з вказанням норм та виділенням показників, що мають

відхилення від норми. Є можливість роздрукувати або завантажити в PDF результати аналізів [14].

1С:Медицина. Лікарня та 1С:Медицина. Поліклініка

Основним існуючим рішенням в даній тематиці по праву може вважатися прикладний додаток «1С:Медицина. Лікарня».

Прикладне рішення «1С:Медицина. Лікарня» спрямоване на автоматизацію діяльності різних організацій та організаційно-правових форм медичних організацій, що надають медичні послуги в амбулаторних та стаціонарних умовах: районних, міських, районних лікарень, аптек різних спеціальностей.

Цей програмний продукт використовується для взаєморозрахунків з контрагентами, управління потоком пацієнтів та персоналізованого обліку наданої медичної допомоги. У цьому рішенні враховано всі особливості бізнес-процесів амбулаторних, клінічних та субклінічних підрозділів медичних закладів. Ця конфігурація має на меті автоматизувати діяльність таких відділів і відповідальних осіб: реєстратури, приймальні, каси, служби управління контрактами, медичного та середнього медичного персоналу, служби аналізу інформації та статистичних служб.

Окрім лікарні 1С:Медицина також має модуль «Поліклініка».

Прикладне рішення «1С:Медицина. Поліклініка» спрямоване на автоматизацію основних процесів різних організацій та організаційно-правових форм медичних установ, що надають медичні послуги в амбулаторних умовах.

Процеси медичного закладу охоплює багато структурних підрозділів, тому для автоматизованої системи важливо створити єдиний інформаційний простір медичного закладу. Прикладне рішення «1С:Медицина. Поліклініка» дозволяє створити такий інформаційний простір і розділити доступ до даних за ролями. Призначення ролей користувачам дозволяє створювати різні автоматизовані посади співробітників.

Рішення від 1С мають величезний функціональний потенціал, проте, окрім явних переваг у них існують і недоліки:

- відсутність необхідного функціоналу у безплатних версіях;
- відсутність інтуїтивно зрозумілого інтерфейсу;
- складні алгоритми пошуку, які не завжди дають необхідні результати;
- відсутність кросплатформенності.

Оскільки дане рішення є основним на даному ринку — при побудові власного продукту в першу чергу необхідно звернути увагу на виправлення саме цих недоліків.

1.3 Постановка задачі

Основна задача даної роботи полягає в організації, проектуванні і розробці веб-системи для університетської клініки СумДУ.

Для досягнення поставленої мети було проведено аналіз предметної області та аналіз недоліків існуючих інформаційних систем, що надають потрібні нам послуги.

Для кваліфікаційної магістерської роботи було визначено наявність проблем, сформульована мета та поставлено такі задачі:

1. Формулювання можливостей майбутнього додатку.
2. Вибір мови програмування для створення додатку.
3. Вибір системи управління базами даних та середовище розробки.
4. Розробка інформаційної системи-репозиторію.
5. Тестування розробленого додатку.

Для повноцінного функціонування розроблювана інформаційна система повинна мати наступний функціонал:

- механізм авторизації;
- додавання нових пацієнтів;
- перегляд історії медичних процедур;
- додавання даних до історії медичних процедур;
- пошук по прізвищу;
- перегляд списку пацієнтів.

2 ОГЛЯД ІНСТРУМЕНТАРІЮ РОЗРОБКИ

2.1 Огляд мови програмування

C# — це проста, сучасна об'єктно-орієнтована мова програмування загального призначення, розроблена Microsoft в рамках її ініціативи .NET під керівництвом Андерса Хейлсберга. На платформі .Net можна реалізувати будь-яку програму: від віконної Windows програми та сучасних веб сайтів до мобільних та хмарних додатків. Формально володіння мовою C# дозволяє розробляти програми для будь-якої платформи та операційної системи. C # ввідноситься до мов програмування високого рівня, де одній команді можуть відповідати сотні тисяч других команд. Синтаксис C# близький до C++ і Java.

C# — це добре продумана високорівнева об'єктно-орієнтована мова програмування. Кілька функцій мови C#, які забезпечують надійність: інтеграція з .NET Framework, компонентно-орієнтована, структурована мова високого рівня, сучасний синтаксис, легка для вивчення, багата стандартна бібліотека, автоматичний збір сміття [15].

Андерс Халесберг — головний дизайнер і головний архітектор C#. Він розробив Turbo Pascal, Embarcadero Delphi (раніше CodeGear Delphi, Inprise Delphi і Borland Delphi) і Visual J++. У своїх інтерв'ю та технічних статтях він заявив, що недоліки більшості основних мов програмування (таких як C++, Java, Delphi і Smalltalk) призвели до створення загальномовного середовища виконання (CLR), що в свою чергу призвело до розвитку самої мови C#.

Що стосується типів, мова C# безпечніша, ніж C++. За замовчуванням єдиними неявними перетвореннями є ті, які вважаються безпечними, наприклад цілочисельні розширення. Немає неявного перетворення між логічними значеннями та цілими числами, а також між членами перерахування та цілими числами (за винятком літералу 0, який можна неявно перетворити на будь-який тип перерахування). Будь-яке кероване перетворення має бути чітко позначено як явне чи неявне, що відрізняється від неявного конструктора копіювання C++ та оператора перетворення за замовчуванням [16].

C# має чітку підтримку коваріації та контраваріантності в загальних типах, на відміну від C++, лише тому, що семантика типу повертання віртуальних методів має певний ступінь підтримки контраваріантності.

C# не дозволяє глобальні змінні чи функції. Усі методи та члени повинні бути оголошені в класі. Статичні члени відкритих класів можуть замінювати глобальні змінні та функції.

На відміну від C і C++, локальні змінні не можуть приховувати змінні, що містяться в блоці.

Керовану пам’ять не можна звільнити; замість цього вона збирається автоматично. Збір сміття вирішує проблему витоку пам’яті та зменшує відповідальність програміста за звільнення пам’яті, яка більше не потрібна.

На відміну від C++, C# не підтримує множинну спадковість, хоча клас може реалізувати будь-яку кількість інтерфейсів. Це дизайнерське рішення, прийняте провідним мовним архітектором, щоб уникнути складності та спростити архітектурні вимоги всього CLI. При реалізації кількох інтерфейсів, які містять методи з однаковим підписом, тобто два методи з однаковими іменами приймають один і той же тип параметрів у тому самому порядку. C# дозволяє реалізувати кожен метод відповідно до інтерфейсу, який викликає цей метод, або як Java , що дозволяє реалізувати цей метод один раз і щоразу викликати його через будь-який інтерфейс класу.

Однак, на відміну від Java, C# підтримує перевантаження операторів. Тільки оператори, які найчастіше перевантажуються в C++, можуть бути перевантажені в C#.

З огляду на всі перераховані вище фактори, а саме – підтримку функцій мови, підтримувані операційні системи та простоту навчання, основною мовою розробки проекту було обрано C#.

2.2 Огляд середовища розробки

Microsoft Visual Studio — це інтегроване середовище розробки від Microsoft (IDE), що використовується для розробки графічних інтерфейсів користувача (GUI), Windows Forms, консолей, веб-сервісів та веб-додатків.

Visual Studio використовується для написання користувацького коду та керованого коду, що підтримується Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

Visual Studio містить редактор коду, що підтримує компонент розширення коду (IntelliSense), а також рефакторинг коду. Інтегрований налагоджувач запускає налагоджувач на рівні джерела та налагоджувач на рівні машини. Інші вбудовані інструменти включають аналізатори коду, розробники графічного інтерфейсу, веб-дизайнери, дизайнери класів і схем баз даних. Він використовує плагіни, які покращують функціональність майже на кожному рівні, включаючи додавання підтримки систем керування вихідним кодом (таких як Subversion і Git), а також додавання нових наборів інструментів, таких як редактори мови визначення домену та візуальні дизайнери або інші аспекти Toolkit. Розробка програмного забезпечення життєвого циклу (як клієнт Azure DevOps: Team Explorer).

Вбудовані мовні служби пропонують підтримку 36 різних добре відомих мов (наприклад, C#, C++, XML, XSLT, HTML і CSS, VB, JavaScript). За допомогою плагінів можна отримати підтримку інших мов, зокрема M, Node.js, Python і Ruby, серед інших підтримуваних мов.

Visual Studio не підтримує жодних мов програмування, рішень чи інструментів; натомість вона дозволяє підключати функції, закодовані як VSPackage. Після встановлення цю функцію можна використовувати як послугу. IDE надає три послуги: SVsSolution, що надає можливість відображати проекти та рішення; SVsUIShell, що надає функції вікон та інтерфейсу користувача; і SVsShell, що обробляє реєстрацію VSPackages. До того ж, IDE відповідає за координацію та зв’язок між службами. VSPackages — це програмні модулі, що розширюють IDE, надаючи елементи інтерфейсу користувача, послуги, проекти, редактори та дизайнери. Для доступу до VSPackages Visual Studio використовує модель компонентного об'єкту (COM). Visual Studio SDK також включає керовану платформу пакетів

(MPF), що являє собою набір керованих обгорток навколо інтерфейсів COM, що дозволяє писати пакети на будь-якій мові, сумісній з CLI. Проте MPF не надає всіх функцій COM-інтерфейсу Visual Studio. Потім можна використовувати ці служби для створення інших пакетів, що додають функціональні можливості Visual Studio IDE [17].

Дане середовище розробки було обрано по декількох критеріях:

- Підтримка мови C#.
- Інтуїтивно зрозумілий інтерфейс.
- Можливості графічного редактору, підходящого до задач.

Так як усі критерії задовільнено — середовищем розробки було обрано саме Visual Studio 2019.

2.3 Огляд додаткового інструментарію

Microsoft SQL Server — це система керування реляційними базами даних, розроблена компанією Microsoft. Як сервер баз даних — це програмний продукт, основною функцією якого є зберігання та отримання даних відповідно до вимог інших програм, може працювати на тому чи іншому комп'ютері мережі (включаючи Інтернет). Microsoft продає щонайменше десяток різних версій Microsoft SQL Server, орієнтованих на різну аудиторію та робоче навантаження, від невеликих однокомпонентних додатків до великих багатоконтурних Інтернет-додатків.

Сховище даних – це база даних, це набір таблиць із введеними стовпцями. SQL Server підтримує різноманітні типи даних, включаючи цілі, плаваючі, десяткові, символьні (включаючи рядок), Varchar (рядок змінної довжини), двійкові (для неструктурованих даних BLOB), текст (для текстових даних) та багато інших.

Переваги MS SQL Server [18,19]:

1. Покращена продуктивність - сервер MS SQL має чудові можливості стиснення та шифрування, що призводить до покращення функцій зберігання та пошуку даних.

2. Незалежність від конкретної СУБД - переважна частина текстів SQL-запитів, які містять DDL і DML, не зважаючи на наявність відмінностей в синтаксисі можуть бути досить легко перенесені з однієї СУБД в іншу.

3. Безпека - сервер MS SQL вважається одним із найбезпечніших серверів баз даних зі складними алгоритмами шифрування, що робить практично неможливим взлом рівнів безпеки, які накладаються користувачем. Сервер MS SQL не є сервером баз даних з відкритим вихідним кодом, який знижує ризик атак на сервер бази даних.

4. Декларативність - програміст описує за допомогою SQL самі дані, що потрібно витягти чи модифікувати. Як саме це зробити, вже при обробці SQL-запиту вирішує СУБД. Хоча це не є повністю універсальним принципом — програміст описує набір даних для вибірки або модифікації, але йому корисно уявляти, як СУБД інтерпретуватиме текст його запиту.

5. Відмінний механізм відновлення даних - сервер MS SQL повністю усвідомлює важливість ваших даних. Отже, MS SQL Server містить багато складних функцій, які дозволяють відновити дані, що були втрачені або пошкоджені.

T-SQL (Transact-SQL) є запатентованим розширенням процедурної мови, наданої Microsoft для SQL Server, яке надає інструкції REPL (читання-оцінка-цикл друку), що розширюють стандартний набір команд SQL для маніпулювання даними (DML) та інструкцій визначення даних (DDL), включаючи спеціальні параметри SQL Server, безпеку та управління статистикою бази даних.

Він надає ключові слова для операцій, що можна виконувати на SQL Server (моніторинг і керування сервером, створення та зміна схеми бази даних, введення та редагування даних у базі даних). Клієнт, який використовує сервер даних або керування, використовуватиме функції SQL Server, надсилаючи запити та оператори T-SQL, що потім обробляються сервером, а результат (або помилка) повертається програмі клієнта. З цією метою він створює таблицю лише для читання, з якої ви можете читати статистику сервера. Функції

керування забезпечуються визначеними системою збереженими процедурами, які можна викликати із запитів T-SQL для виконання операцій керування. Ви також можете використовувати T-SQL для створення підключених серверів. Підключені сервери дозволяють обробляти операції на кількох серверах в одному запиті. [20]

HTML – це мова розмітки, що використовується веб-браузерами для інтерпретації та об’єднання тексту, зображень та інших матеріалів у візуальні або аудіо веб-сторінки. Характеристики за замовчуванням для кожного елемента розмітки HTML визначаються в браузері, і ці характеристики можуть бути змінені або покращені веб-дизайнером за допомогою додаткового CSS. За своєю суттю HTML - це серія коротких кодів, введених у текстовий файл. Це теги, які забезпечують можливості HTML. Текст зберігається як файл HTML і переглядається через веб-браузер. Браузер читає файл і перекладає текст у видиму форму відповідно до кодів, які автор використав для написання того, що стає видимим відображенням. Написання HTML вимагає правильного використання тегів для відображення бачення автора.

Теги відрізняють звичайний текст від HTML-коду. Найпростіші теги застосовують форматування до тексту. Оскільки веб-інтерфейси повинні стати більш динамічними, можна використовувати каскадні таблиці стилів (CSS), що роблять веб-сторінки доступнішими [21].

Каскадні таблиці стилів (CSS) — це мова таблиць стилів, яка використовується для опису подання документів, написаних мовою розмітки. CSS має на меті розділити презентацію та вміст, включаючи макет, кольори та шрифти. Це поділ може покращити зручність використання вмісту, забезпечити більшу гнучкість та контроль у специфікаціях презентації, а також дозволить форматувати декілька веб-сторінок разом, вказавши відповідний CSS в окремих файлах .css, тим самим зменшуючи складність структурованого вмісту. Також дозволяє файли .css кешувати, щоб підвищити швидкість завантаження сторінки між обміном файлами та відформатованими сторінками.

Розділення формату та вмісту також дає змогу представити одну й ту саму сторінку розмітки в різних стилях для різних методів візуалізації. При доступі до вмісту на мобільних пристроях CSS також має альтернативні правила форматування. Каскадна назва походить від заданої схеми пріоритету, щоб визначити, яке правило стилю застосовується, коли декілька правил відповідають певному елементу. Ця каскадна схема пріоритетів є передбачуваною.

Однією з цілей CSS є надання користувачам більшого контролю над представленням. Відповідно до різних браузерів і веб-сайтів, користувачі можуть вибрати одну з різних таблиць стилів, наданих дизайнером, або видалити всі додані стилі та використовувати стиль браузера за замовчуванням для перегляду сайту [22].

Фреймворк CSS — це попередньо розроблена бібліотека, призначена для спрощення стилю веб-сторінок, що відповідають стандартам, за допомогою каскадної мови таблиць стилів. Структури CSS включають Blueprint, Bootstrap, Cascade Framework, Foundation і Materialize. Як і мови програмування та бібліотеки сценаріїв, фреймворки CSS зазвичай включають у вигляді зовнішніх таблиць .css, на які посилається HTML <head>. Вони надають безліч готових варіантів оформлення та розміщення веб-сторінок. Хоча багато з цих фреймворків було випущено, деякі автори в основному використовують їх для швидкого створення прототипів або досліджень, і вважають за краще «створювати» CSS, відповідний для кожного сайту випуску, без необхідності розробляти, підтримувати та завантажувати багато невикористаних CSS для функцій стилю сайту.

3 ПРОЕКТУВАННЯ І РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Діаграма варіантів використання

При проектуванні функціонального навантаження системи слід проаналізувати використані варіанти, що дасть повне розуміння призначення та можливої діяльності інформаційної системи в майбутньому.

Найпростіша діаграма використання — це спосіб мислення про взаємодію між користувачем і системою, який показує відносини між користувачем і різними видами використання, в яких бере участь користувач. Діаграми використання можуть ідентифікувати різні типи користувачів системи та різні моделі використання, і часто супроводжуються діаграмами інших типів. Ціль зображується колом або еліпсом.

Хоча кожну можливість можна детально розглянути за допомогою самих параметрів, використання діаграм може допомогти надати огляд системи вищого рівня.

Завдяки своїй спрощеній природі плани використання можуть бути хорошим інструментом комунікації для зацікавлених сторін. Ці малюнки намагаються змоделювати реальний світ і дозволити зацікавленим сторонам зрозуміти, як буде розроблена система. Використання діаграм передає намір системи зацікавленим сторонам більш спрощеним чином, і вони «повніше пояснюються, ніж діаграми класів».

Метою використання діаграм є показати динамічні аспекти системи. Для забезпечення повного функціонального та технічного представлення системи можна використовувати додаткові схеми та документи. Вони забезпечують спрощене та графічне представлення того, що насправді має робити система.

- Межа системи - прямокутник з ім'ям і еліпсом (прецедент). За відсутності корисної інформації її зазвичай можна опустити,
- Актор (англ. actor) — стилізована роль людини, яка являє собою набір ролей користувачів, які взаємодіють із сутностями (системами,

підсистемами, класами) (широке розуміння: люди, зовнішні сутності, класи, інші системи). Актори не можуть бути пов'язані один з одним (за винятком відносин обробки/дослідження),

- Прецедент - еліпс з написом, що вказує на виконану системну операцію (можливо, включаючи можливі варіанти), що привела до результатів, які спостерігає учасник. Назва може бути назвою або описом (з точки зору актора) системи «що» (а не «як»). Під час сценарію актори обмінюються системною інформацією. Сценарій можна відобразити в прецедентній діаграмі відео коментарів UML. Кілька різних сценаріїв можуть бути пов’язані з прецедентом.

На рисунку 3.1 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі. Адміністратор (актор) — авторизований в системі працівник клініки, який може реєструвати нового пацієнта, переглядати та редагувати його дані, а також видаляти.

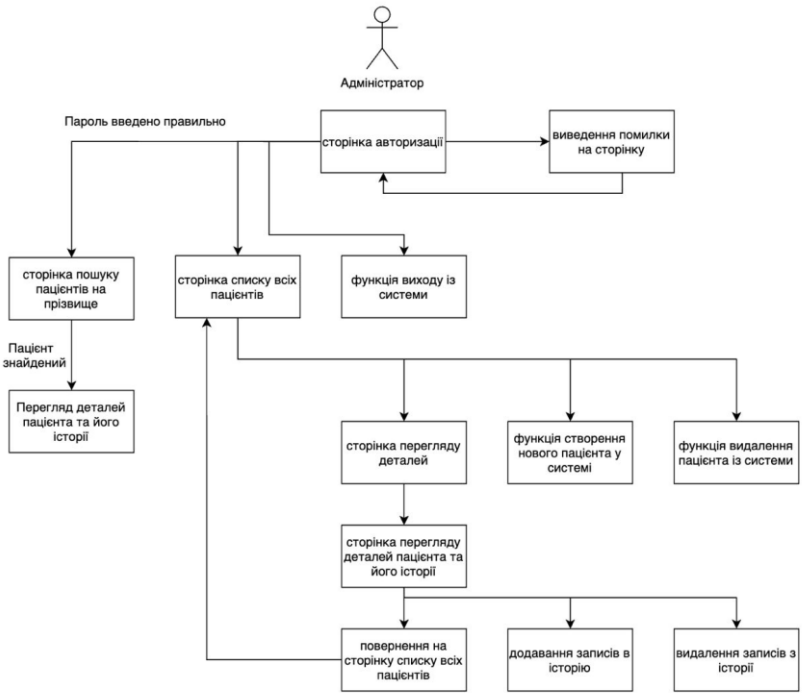

Рисунок 3.1 — Діаграма варіантів використання для користувача

3.2 Діаграма класів

При проектуванні програмного забезпечення внутрішня структура системи зазвичай відображається у вигляді діаграми класів, яка показує структуру класів і модулів проекту та взаємодію між ними.

У програмній інженерії діаграма класів уніфікованої мови моделювання (UML) — це статична структурна діаграма, яка описує структуру системи та показує системні класи, їх атрибути, операції (або методи) і зв’язки між об’єктами. Діаграми класів є основними будівельними блоками об'єктно-орієнтованого моделювання.

У моделюванні UML відносини реалізації — це відносини між двома елементами моделі, де один елемент моделі (клієнт) реалізує (реалізує або виконує) поведінку, задану іншим елементом моделі (постачальником).

Графічне представлення реалізації UML являє собою порожнистий трикутник, з’єднаний з кінцем штрихового інтерфейсу (або дерева рядків) одного або кількох реалізаторів. Проста стрілка використовується в кінці пунктирного інтерфейсу, який з’єднує його з користувачем. Реалізація може бути відображена лише на діаграмі класів або компонентів. Реалізація – це зв’язок між класами, інтерфейсами, компонентами та пакетами, які з’єднують елементи замовника з елементами постачальника. Відношення реалізації між класом/компонентом та інтерфейсом показує, що клас/компонент реалізує операції, запропоновані інтерфейсом.

Залежність — це слабкіша форма спілкування, яка вказує на те, що один клас залежить від іншого класу, оскільки він використовує його в певний момент часу. Якщо незалежний клас залежить від змінної параметра або локальної змінної методу, то один клас залежить від іншого. Це відрізняється від асоціації, де атрибут залежного класу є екземпляром незалежного класу. Іноді відносини між цими двома класами слабкі. Вони взагалі не реалізуються зі змінними-членами. Натомість вони можуть бути реалізовані як параметри функцій-членів.

Представлення асоціації UML - це лінія, що з'єднує два пов'язані класи. На кожному кінці рядка є додаткові символи. Наприклад, ми можемо використовувати стрілку, щоб вказати, що кінчик можна побачити з хвоста стрілки. Ми можемо представити право власності, помістивши кульку, і роль, яку відіграє елемент на цьому кінці, полягає в вказуванні імені ролі та кількох екземплярів сутності (діапазон об’єктів, пов’язаних з іншого кінця).

Класи сутності моделюють довгоживучу інформацію, якою обробляє система, а іноді і поведінку, пов'язану з цією інформацією. Їх не слід ідентифікувати як таблиці баз даних чи інших сховищ даних.

Вони намальовані як кола з короткою лінією, прикріпленою до нижньої частини кола. Як варіант, їх можна намалювати як звичайні класи із позначенням стереотипу «сутність» над назвою класу.

На рисунку 3.2 зображено діаграму класів, яка відображає внутрішню будову проекту.

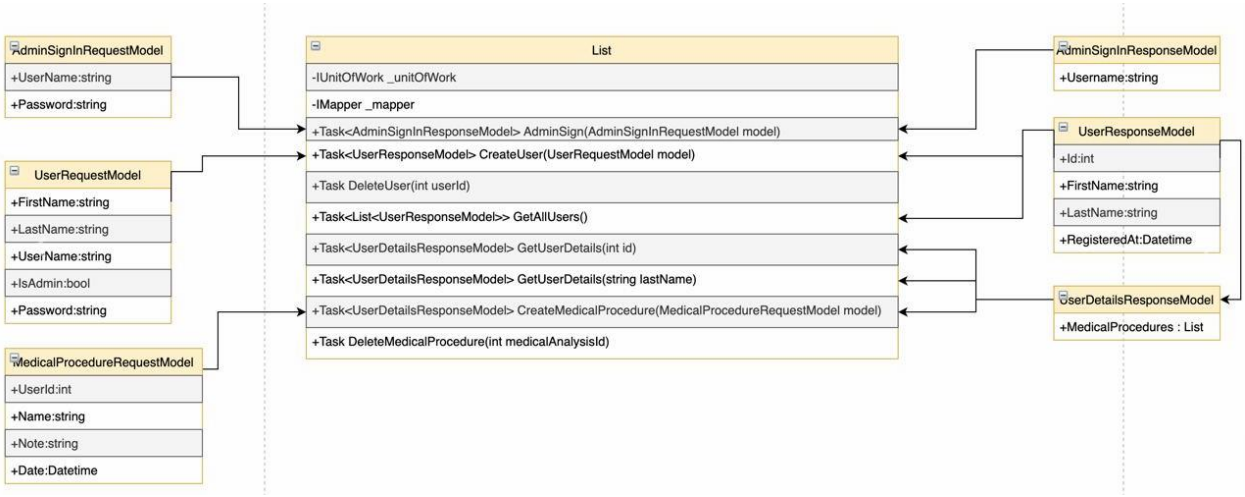

Рисунок 3.2 — Діаграма класів

3.3 Розробка структури бази даних

На рисунку 3.3 зображена схема база даних системи, яка відображає її внутрішню структуру.

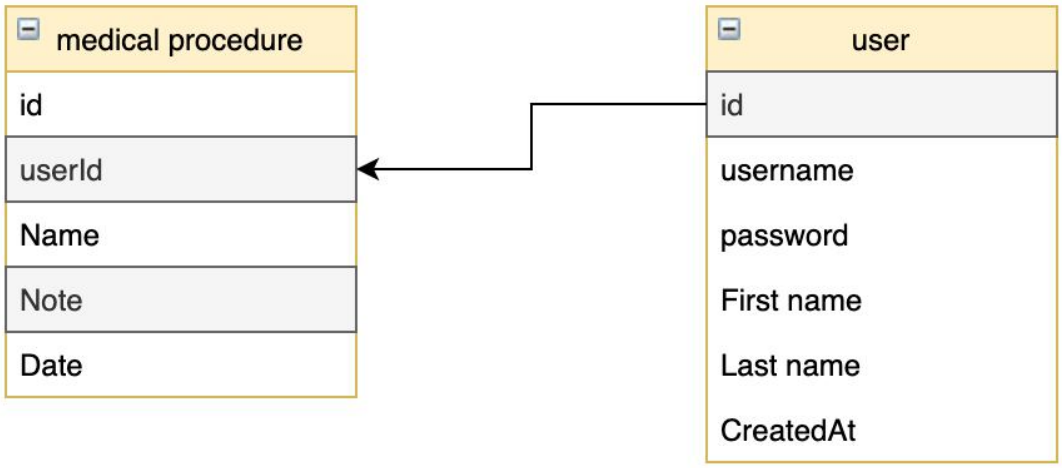

Рисунок 3.3 — Схема бази даних

Розроблена база даних складається з двох таблиць. Перша таблиця зберігає інформацію про користувача, а саме: ідентифікатор, логін, пароль, повне ім’я та дату створення. Друга таблиця представляє собою сутність медичної процедури, яка зберігає в собі дані про проведені медичні процедури і їх результати, а саме: ідентифікатор користувача, назва, результат і дату проведення.

3.4 Розробка графічного інтерфейсу користувача

Графічний інтерфейс користувача складається з 4 основних сторінок, а саме:

- Сторінка авторизації (рис. 3.4)
- Сторінка списку пацієнтів (рис. 3.5)
- Сторінка пошуку пацієнтів (рис. 3.6)
- Сторінка перегляду деталей (рис. 3.7)

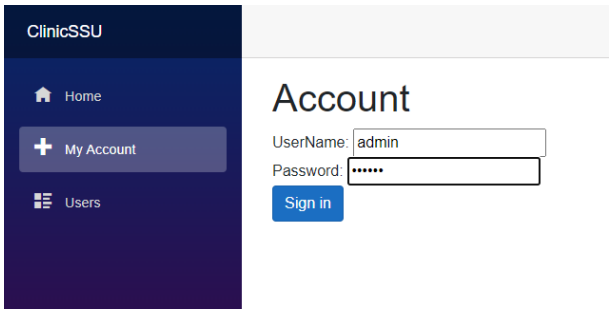

Рисунок 3.4 — Сторінка авторизації

Сторінка авторизації містить в собі 2 поля для введення логіну і паролю та кнопку авторизації, після натискання якої (при вірних даних) користувач отримує доступ до основного функціоналу проекту.

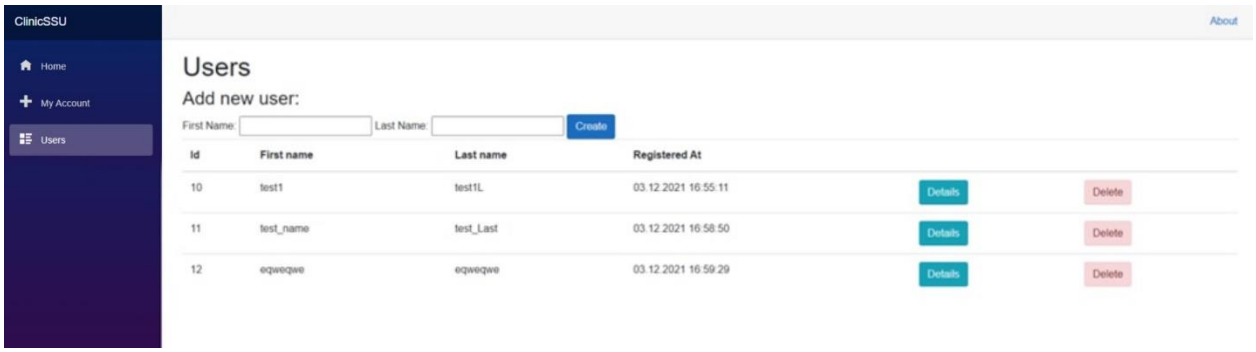

Рисунок 3.5 — Сторінка пацієнтів

Сторінка пацієнтів представляє собою список існуючих в системі пацієнтів з можливістю перегляду їх медичних процедур, видалення пацієнта або створення нового пацієнта.

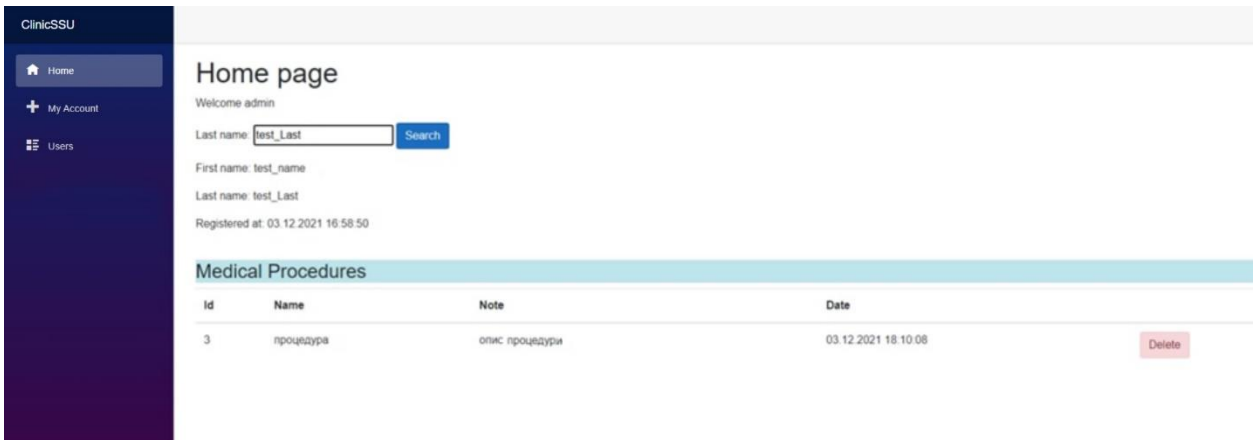

Рисунок 3.6 — Сторінка пошуку пацієнтів

Сторінка пошуку пацієнта містить в собі поле для введення прізвища пацієнта та кнопку пошуку. Після натискання кнопки пошуку, при наявності пацієнта з таким прізвищем, система виводить детальну інформацію про даного пацієнта.

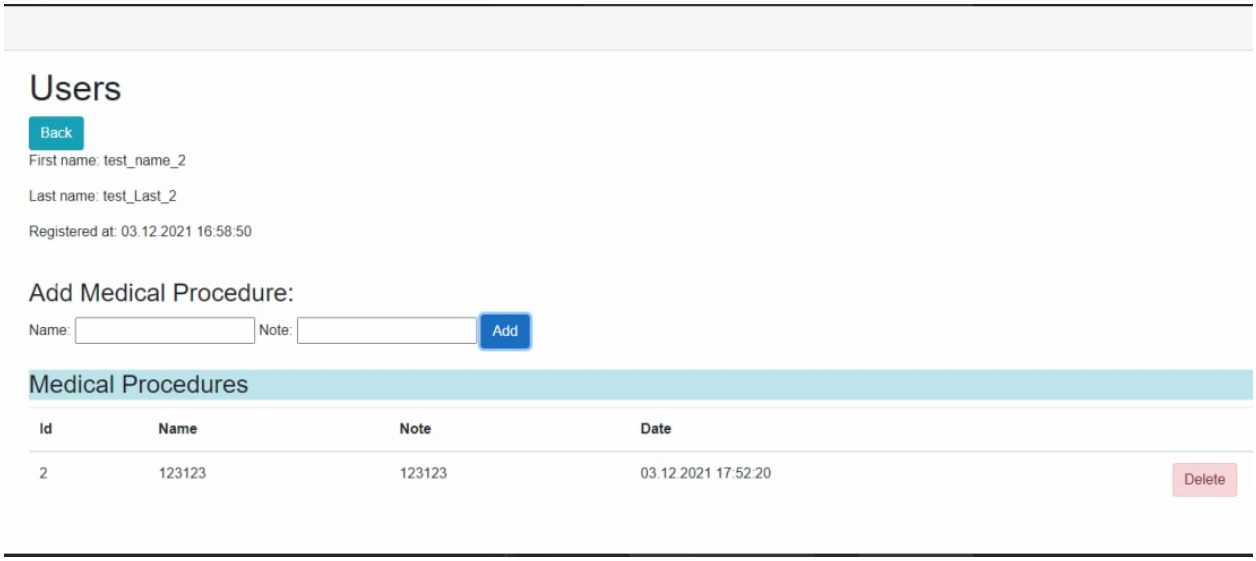

Рисунок 3.7 — Сторінка перегляду історії пацієнтів

Детальний перегляд пацієнта представляє собою список медичних процедур, в яких брав участь даний пацієнт, з можливістю додавання нових процедур, тощо.

3.5 Тестування системи

Для тестування розробленого програмного продукту було обрано методику чорного ящика. Тестування чорного ящика — це метод тестування програмного забезпечення, який перевіряє функціональність програми, не заглядаючи в її внутрішні структури чи роботи. Цей метод тестування можна застосувати практично на кожному рівні тестування програмного забезпечення: одиничному, інтеграційному, системному та приймальному. Його іноді називають тестуванням на основі специфікації.

Спеціальні знання коду програми, внутрішньої структури та знання програмування загалом не потрібні. Тестер знає, що програмне забезпечення має робити, але не знає, як воно це робить. Наприклад, тестувальник знає, що

конкретні вхідні дані повертають певний незмінний вихід, але не знає, як програмне забезпечення виробляє вихідні дані в першу чергу.

Таблиця 3.1 — Тестування додатку

№	Тест-кейс	Очікуваний результат	Отриманий результат
1	Невірні данні при авторизації	При введенні невірних даних система повідомляє користувача про те, що такого користувача не існує, або дані введені невірно	При введенні невірних даних система повідомляє користувача про те, що такого користувача не існує, або дані введені невірно
2	Пусті поля при авторизації	При спробі авторизації з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.	При спробі авторизації з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.
4	Пусті поля при реєстрації пацієнта	При спробі реєстрації пацієнта з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.	При спробі реєстрації пацієнта з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.
5	Створення процедури з пустими полями	При спробі створення процедури з пустими полями	При спробі створення процедури з пустими полями

		система повідомляє користувача про те, що ці поля необхідно заповнити.	система повідомляє користувача про те, що ці поля необхідно заповнити.
6	Створення пустого запису пацієнту	При спробі запису пацієнта з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.	При спробі запису пацієнта з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.
7	Пошук по пустому полю	При спробі пошуку по пустому полю система повідомляє про необхідність його заповнення.	При спробі пошуку по пустому полю система повідомляє про необхідність його заповнення.

Результати тестування показують, що система повністю функціональна і готова до використання в реальних умовах.

ВИСНОВКИ

За останні пару років стрімко збільшується кількість медичних закладів в Україні, що починають використовувати медичні інформаційні системи. Комп'ютер дозволяє не переписувати ті самі дані по багато разів і знаходити потрібну інформацію в зручному вигляді. Це також надає можливість контролювання лікарями своєчасності та повноти обстеження.

У ході виконання дипломної роботи було:

- проаналізовано теоретичні підходи та основні поняття предметної області;
- виконано аналітичний огляд існуючих аналогічних інформаційних систем;
- сформовано вимоги до інформаційної системи;
- проведено аналіз інструментарію розробки;
- з’ясовано особливості використання програмно-апаратних засобів Web-системи;

Розроблений додаток задовольняє всім вимогам, поставленим на етапі постановки завдання.

Головним завданням програми буде вирішення проблеми витрати часу лікаря і пацієнта, а також гарантія збереження цілісності результатів прийому.

СПИСОК ЛІТЕРАТУРИ

1. What Is an Information Repository? [Електронний ресурс] – Режим доступу: https://www.easytechjunkie.com/what-is-an-information-repository.htm – Назва з екрану.

2. Valacich J. Information Systems Today: Managing the Digital World / J. Valacich, C. Schneider., 2010. – (4th Edition)

3. Components of an Information System [Електронний ресурс] – Режим доступу: https://www.mbaknol.com/management-information-systems/components-of-an-information-system/ – Назва з екрану.

4. Website [Електронний ресурс] – Режим доступу: https://uk.wikinohr.top/wiki/Website – Назва з екрану.

5. Web development [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Web_development – Назва з екрану.

6. Information System [Електронний ресурс] – Режим доступу: https://www.wikizero.com/en/Information_systems – Назва з екрану.

7. Про затвердження переліків закладів охорони здоров'я, лікарських посад, посад фармацевтів, посад фахівців з фармацевтичною освітою (асистентів фармацевтів), посад професіоналів у галузі охорони здоров'я, посад фахівців у галузі охорони здоров'я та посад професіоналів з вищою немедичною освітою у закладах охорони здоров'я[Електронний ресурс] // Верховна Рада України : [офіційний вебпортал]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0892-02#Text, вільний. – Назва з екрану.

8. 10 Types of Healthcare Facilities [Електронний ресурс] – Режим доступу: https://www.indeed.com/career-advice/finding-a-job/healthcare-types – Назва з екрану.

9. Britannica|Hospital [Електронний ресурс] – Режим доступу: https://www.britannica.com/science/hospital – Назва з екрану.

10. Статистика ведення електронних медичних записів в ЕСОЗ [Електронний ресурс] – 2021. – Режим доступу: https://nszu.gov.ua/e-data/dashboard/emz-stats – Назва з екрану.

11. Університетська клініка СумДУ | Медичний центр в Сумах [Електронний ресурс] – Режим доступу: https://clinic.health.sumdu.edu.ua – Назва з екрану.

12. USU (Электронная история болезни) [Електронний ресурс] – 2021. – Режим доступу: http://usu.kz/elektronnaya_istoriya_bolezni.php – Назва з екрану.

13. Helsi [Електронний ресурс] – 2021. – Режим доступу: https://helsi.me/ – Назва з екрану.

14. Медична інформаційна система «Askep.net» [Електронний ресурс] – 2021. – Режим доступу: https://askep.net/ – Назва з екрану.

15. C# vs Python [Електронний ресурс] – Режим доступу: https://hackr.io/blog/c-sharp-vs-python – Назва з екрану.

16. C Sharp (мова програмування) [Електронний ресурс] – Режим доступу: https://uk.wiki2.wiki/wiki/C_Sharp_(programming_language) – Назва з екрану.

17. Microsoft Visual Studio [Електронний ресурс] – Режим доступу: uk.wiki2.wiki/wiki/Microsoft_Visual_Studio – Назва з екрану.

18. MS SQL SERVER HISTORY AND ADVANTAGES [Електронний ресурс] – Режим доступу: https://bytescout.com/blog/2014/09/ms-sql-server-history-and-advantages.html – Назва з екрану.

19. SQL [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/SQL – Назва з екрану.

20. Руководство SQL Server Transact-SQL [Електронний ресурс] – Режим доступу: https://betacode.net/10327/sql-server-transact-sql – Назва з екрану.

21. HyperText Markup Language – HTML [Електронний ресурс] – Режим доступу: https://www.investopedia.com/terms/h/html.asp – Назва з екрану.

22. CSS [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/CSS – Назва з екрану.

ДОДАТОК А

Лістинг програмного коду

```
using HospitalApp.Models.RequestModels;
using HospitalApp.Models.ResponseModels;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace HospitalApp.Services.Interfaces
{
    public interface IHospitalService
    {
        /// <summary>
        /// Admin sign in to account
        /// </summary>
        /// <param name="model"></param>
        /// <returns></returns>
        Task<AdminSignInResponseModel>
AdminSign(AdminSignInRequestModel model);


        /// <summary>
        /// Adming gets all users from database
        /// </summary>
        /// <returns></returns>
        Task<List<UserResponseModel>> GetAllUsers();


        /// <summary>
        /// admin get user details
        /// </summary>
```

```
/// <param name="id"></param>
/// <returns></returns>
Task<UserDetailsResponseModel> GetUserDetails(int id);

/// <summary>
/// admin get user details by user`s lastName
/// </summary>
/// <param name="lastName"></param>
/// <returns></returns>
Task<UserDetailsResponseModel> GetUserDetails(string lastName);

/// <summary>
/// Create new user
/// </summary>
/// <param name="model"></param>
/// <returns></returns>
Task<UserResponseModel> CreateUser(UserRequestModel model);

/// <summary>
/// Delete user by userId
/// </summary>
/// <param name="userId"></param>
/// <returns></returns>
Task DeleteUser(int userId);

/// <summary>
/// Create Medical Procedure for user
/// </summary>
```

```
        /// <param name="model"></param>
        /// <returns></returns>
        Task<UserDetailsResponseModel>
CreateMedicalProcedure(MedicalProcedureRequestModel model);


        /// <summary>
        /// Delete Medical Procedure by id
        /// </summary>
        /// <param name="medicalAnalysisId"></param>
        /// <returns></returns>
        Task DeleteMedicalProcedure(int medicalAnalysisId);
    }


namespace HospitalApp.Services.Services
{
    public class HospitalService : IHospitalService
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly IMapper _mapper;

        public HospitalService(IUnitOfWork unitOfWork, IMapper mapper, IConfiguration configuration)
        {
            _unitOfWork = unitOfWork;
            _mapper = mapper;
        }

        #region Users

        public async Task<AdminSignInResponseModel> AdminSign(AdminSignInRequestModel model)
        {
            var admin = _unitOfWork.Repository<ApplicationUser>().Get(x => x.UserName ==
model.UserName && x.Password == model.Password && x.IsAdmin).FirstOrDefault();
            if (admin == null) return null;

            var response = _mapper.Map<AdminSignInResponseModel>(admin);
            return response;
        }

        public async Task<UserResponseModel> CreateUser(UserRequestModel model)
        {
            var user = new ApplicationUser()
            {
                FirstName = model.FirstName,
                LastName = model.LastName,
```

```
            IsAdmin = false,
            CreatedAt = DateTime.Now,
            Password = model.Password,
            UserName = model.UserName,
        };

        _unitOfWork.Repository<ApplicationUser>().Insert(user);
        _unitOfWork.SaveChanges();

        var response = _mapper.Map<UserResponseModel>(user);
        return response;
    }

    public async Task DeleteUser(int userId)
    {
        var user = _unitOfWork.Repository<ApplicationUser>().Get(x => x.Id == userId &&
!x.IsAdmin).FirstOrDefault();
        if (user!= null)
        {
            _unitOfWork.Repository<ApplicationUser>().Delete(user);
            _unitOfWork.SaveChanges();
        }
    }

    public async Task<List<UserResponseModel>> GetAllUsers()
    {
        var users = _unitOfWork.Repository<ApplicationUser>().Get(x => !x.IsAdmin).ToList();

        var response = _mapper.Map<List<UserResponseModel>>(users);
        return response;
    }

    public async Task<UserDetailsResponseModel> GetUserDetails(int id)
    {
        var user = _unitOfWork.Repository<ApplicationUser>().Get(x => x.Id ==
id).Include(x=>x.MedicalProcedures).FirstOrDefault();
        if (user == null) return null;

        var response = new UserDetailsResponseModel()
        {
            Id = user.Id,
            FirstName = user.FirstName,
            LastName = user.LastName,
            RegisteredAt = user.CreatedAt,
        };

        response.MedicalProcedures =
_mapper.Map<List<MedicalProcedureResponseModel>>(user.MedicalProcedures);

        return response;
    }

    public async Task<UserDetailsResponseModel> GetUserDetails(string lastName)
    {
        var user = _unitOfWork.Repository<ApplicationUser>().Get(x => x.LastName ==
lastName).Include(x => x.MedicalProcedures).FirstOrDefault();
```

```
            if (user == null) return null;

            var response = await GetUserDetails(user.Id);

            return response;
        }

        #endregion

        #region MedicalProcedures

        public async Task<UserDetailsResponseModel>
CreateMedicalProcedure(MedicalProcedureRequestModel model)
        {
            var procedure = _mapper.Map<MedicalProcedure>(model);
            _unitOfWork.Repository<MedicalProcedure>().Insert(procedure);
            _unitOfWork.SaveChanges();

            var response = await GetUserDetails(model.UserId);
            return response;
        }

        public async Task DeleteMedicalProcedure(int medicalAnalysisId)
        {
            var procedure = _unitOfWork.Repository<MedicalProcedure>().Get(x=>x.Id ==
medicalAnalysisId).FirstOrDefault();

            if (procedure == null) return;

            _unitOfWork.Repository<MedicalProcedure>().Delete(procedure);
            _unitOfWork.SaveChanges();
        }

        #endregion

    }
}

namespace HospitalApp.Data
{
    public class BaseController
    {
        private readonly IServiceProvider _serviceProvider;

        public BaseController(IServiceProvider serviceProvider)
        {
            _serviceProvider = serviceProvider;
            using (var scope = _serviceProvider.CreateScope())
            {
                var exchangeRateService = scope.ServiceProvider.GetRequiredService<IHospitalService>();
            }
        }

        #region Users

        public async Task<AdminSignInResponseModel> AdminSign(AdminSignInRequestModel model)
        {
```

```
    using (var scope = _serviceProvider.CreateScope())
    {
        var hospitalService = scope.ServiceProvider.GetRequiredService<IHospitalService>();
        var response = await hospitalService.AdminSign(model);
        return response;
    }
}

public async Task<UserResponseModel> CreateUser(UserRequestModel model)
{
    using (var scope = _serviceProvider.CreateScope())
    {
        var hospitalService = scope.ServiceProvider.GetRequiredService<IHospitalService>();
        var response = await hospitalService.CreateUser(model);
        return response;
    }
}

public async Task DeleteUser(int userId)
{
    using (var scope = _serviceProvider.CreateScope())
    {
        var hospitalService = scope.ServiceProvider.GetRequiredService<IHospitalService>();
        await hospitalService.DeleteUser(userId);
    }
}

public async Task<List<UserResponseModel>> GetAllUsers()
{
    using (var scope = _serviceProvider.CreateScope())
    {
        var hospitalService = scope.ServiceProvider.GetRequiredService<IHospitalService>();
        var response = await hospitalService.GetAllUsers();
        return response;
    }
}

public async Task<UserDetailsResponseModel> GetUserDetails(int id)
{
    using (var scope = _serviceProvider.CreateScope())
    {
        var hospitalService = scope.ServiceProvider.GetRequiredService<IHospitalService>();
        var response = await hospitalService.GetUserDetails(id);
        return response;
    }
}
public async Task<UserDetailsResponseModel> GetUserDetails(string lastName)
{
    using (var scope = _serviceProvider.CreateScope())
    {
        var hospitalService = scope.ServiceProvider.GetRequiredService<IHospitalService>();
        var response = await hospitalService.GetUserDetails(lastName);
        return response;
    }
}

#endregion
```

```
        #region MedicalProcedures

        public async Task<UserDetailsResponseModel>
CreateMedicalProcedure(MedicalProcedureRequestModel model)
        {
            using (var scope = _serviceProvider.CreateScope())
            {
                var hospitalService = scope.ServiceProvider.GetRequiredService<IHospitalService>();
                var response = await hospitalService.CreateMedicalProcedure(model);
                return response;
            }
        }

        public async Task DeleteMedicalProcedure(int medicalAnalysisId)
        {
            using (var scope = _serviceProvider.CreateScope())
            {
                var hospitalService = scope.ServiceProvider.GetRequiredService<IHospitalService>();
                await hospitalService.DeleteMedicalProcedure(medicalAnalysisId);
            }
        }

        #endregion

    }
}



@page "/account"


<h1>Account</h1>

@if (admin == null)
{
    @if (isSignInFail)
    {
        <div class="alert alert-dismissible alert-danger">Invalid username or password</div>
    }
    <div>
        UserName: <input type="text" value=@login @onchange="@((ChangeEventArgs __e) => login =
__e?.Value?.ToString())" />
        <br>
        Password: <input type="password" value=@password @onchange="@((ChangeEventArgs __e) =>
password = __e?.Value?.ToString())" />
        <br>
        <button class="btn btn-primary" @onclick="AdminLogin">Sign in</button>

    </div>
}
else
{
    <div>
        <p>Hello @admin.Username!</p>
        <button class="btn btn-danger" @onclick="Logout">Logout</button>
```

```
    </div>
}


@code {
    private AdminSignInResponseModel admin = null;
    private bool isSignInFail;

    private string login = "";
    private string password = "";

    protected override async Task OnInitializedAsync()
    {
        var result = await BrowserStorage.GetAsync<string>("username");
        var value = result.Success ? result.Value : null;

        if (value!= null)
            admin = new AdminSignInResponseModel() { Username = value };
    }

    private async Task AdminLogin()
    {
        isSignInFail = false;
        try
        {
            admin = await Service.AdminSign(new AdminSignInRequestModel() { UserName = login,
Password= password });

        }
        catch (Exception ex)
        {
            isSignInFail = true;
        }

        if (admin == null)
            isSignInFail = true;

        if (admin!= null)
            await BrowserStorage.SetAsync("username", admin.Username);
    }

    private async Task Logout()
    {
        admin = null;

        await BrowserStorage.DeleteAsync("username");
    }

}
@page "/"

<h1>Home page</h1>

@if (admin == null)
{
    <div class="alert alert-dismissible alert-warning">
        <h4 class="alert-heading">Warning!</h4>
```

```
        <p class="mb-0">Sign in to your account for seeing users and details</p>
    </div>
}
else
{
    <p>
        Welcome @admin.Username
    </p>

    <div>
        <p>
            Last name: <input type="text" value=@LastName @onchange="@((ChangeEventArgs __e) =>
LastName = __e?.Value?.ToString())" />
            <button class="btn btn-primary" @onclick="SearchUser">Search</button>
        </p>
        @if (currentUser == null)
        {
            <p>
                Enter user last name for search
            </p>
        }
        else
        {
            <div>
                <p>First name: @currentUser.FirstName</p>
                <p>Last name: @currentUser.LastName</p>
                <p>Registered at: @currentUser.RegisteredAt</p>
                <br>

                <h3 class="table-info">Medical Procedures</h3>

                @if (currentUser.MedicalProcedures == null  || !currentUser.MedicalProcedures.Any())
                {
                    <p class="table-warning"><em>No Medical Procedures</em></p>
                }
                else
                {
                    <table class="table">
                        <thead>
                            <tr>
                                <th>Id</th>
                                <th>Name</th>
                                <th>Note</th>
                                <th>Date</th>
                                <th></th>
                            </tr>
                        </thead>
                        <tbody>
                            @foreach (var procedure in currentUser.MedicalProcedures)
                            {
                                <tr>
                                    <td>@procedure.Id</td>
                                    <td>@procedure.Name</td>
                                    <td>@procedure.Note</td>
                                    <td>@procedure.Date</td>
                                    <td><button class="btn alert-danger" @onclick="@((MouseEventArgs e) =>
DeleteProcedure(procedure.Id))">Delete</button></td>
```

```
                        </tr>
                    }
                </tbody>
            </table>
        }


    </div>
}

</div>
}


@code {
    private string LastName = null;
    private UserDetailsResponseModel currentUser = null;
    private AdminSignInResponseModel admin = null;

    protected override async Task OnInitializedAsync()
    {
        SetAdmin().ContinueWith(t => Console.WriteLine(t.Exception),
        TaskContinuationOptions.OnlyOnFaulted);
    }

    private async Task SetAdmin()
    {
        var result = await BrowserStorage.GetAsync<string>("username");
        var value = result.Success ? result.Value : null;

        if (value!= null)
        {
            admin = new AdminSignInResponseModel() { Username = value };
            StateHasChanged();
        }
    }

    private async Task SearchUser()
    {
        currentUser = null;
        if (LastName== null) return;

        currentUser = await Service.GetUserDetails(LastName);
    }

     private async Task DeleteProcedure(int id)
    {
        await Service.DeleteMedicalProcedure(id);
        currentUser = await Service.GetUserDetails(currentUser.Id);
    }


}

@page "/users"
```

```
@if (admin == null)
{
   <div class="alert alert-dismissible alert-warning">
      <h4 class="alert-heading">Warning!</h4>
      <p class="mb-0">Sign in to your account for seeing users and details</p>
   </div>

}
else
{
   <h1>Users</h1>
   @if (isDetails && currentUser!= null)
   {
      <button class="btn btn-info" @onclick="CloseDetails">Back</button>

      <div>
         <p>First name: @currentUser.FirstName</p>
         <p>Last name: @currentUser.LastName</p>
         <p>Registered at: @currentUser.RegisteredAt</p>
         <br>


         <div>
            <h3>Add Medical Procedure:</h3>
            Name: <input type="text" value=@Name @onchange="@((ChangeEventArgs __e) => Name =
__e?.Value?.ToString())" />
            Note: <input type="text" value=@Note @onchange="@((ChangeEventArgs __e) => Note =
__e?.Value?.ToString())" />
            <button class="btn btn-primary" @onclick="CreateMedicalProcedure">Add</button>

         </div>
         <br>
         <h3 class="table-info">Medical Procedures</h3>

         @if (currentUser.MedicalProcedures == null  || !currentUser.MedicalProcedures.Any())
         {
            <p class="table-warning"><em>No Medical Procedures</em></p>
         }
         else
         {
            <table class="table">
               <thead >
                  <tr >
                     <th>Id</th>
                     <th>Name</th>
                     <th>Note</th>
                     <th>Date</th>
                     <th></th>
                  </tr>
               </thead>
               <tbody>
                  @foreach (var procedure in currentUser.MedicalProcedures)
                  {
                     <tr>
                        <td>@procedure.Id</td>
                        <td>@procedure.Name</td>
```

```
                    <td>@procedure.Note</td>
                    <td>@procedure.Date</td>
                    <td><button class="btn alert-danger" @onclick="@((MouseEventArgs e) =>
DeleteProcedure(procedure.Id))">Delete</button></td>
                </tr>
            }
        </tbody>
    </table>
}


</div>

}
else
{
    <div>
        <h3>Add new user:</h3>
        First Name: <input type="text" value=@FirstName @onchange="@((ChangeEventArgs __e) =>
FirstName = __e?.Value?.ToString())" />
        Last Name: <input type="text" value=@LastName @onchange="@((ChangeEventArgs __e) =>
LastName = __e?.Value?.ToString())" />
        <button class="btn btn-primary" @onclick="CreateUser">Create</button>

    </div>

    @if (users == null)
    {
        <p class="table-warning"><em>No users</em></p>
    }
    else
    {
        <table class="table">
            <thead>
                <tr class="table-info">
                    <th>Id</th>
                    <th>First name</th>
                    <th>Last name</th>
                    <th>Registered At</th>
                    <th></th>
                    <th></th>
                </tr>
            </thead>
            <tbody>
                @foreach (var user in users)
                {
                    <tr>
                        <td>@user.Id</td>
                        <td>@user.FirstName</td>
                        <td>@user.LastName</td>
                        <td>@user.RegisteredAt</td>
                        <td><button class="btn btn-info" @onclick="@((MouseEventArgs e) =>
GetDetails(user.Id))">Details</button></td>
                        <td><button class="btn alert-danger" @onclick="@((MouseEventArgs e) =>
DeleteUser(user.Id))">Delete</button></td>
                    </tr>
                }
```

```
            </tbody>
        </table>
    }
}


}

@code {
    private bool isDetails = false;

    private string FirstName = null;
    private string LastName = null;

    private string Name = null;
    private string Note = null;

    private List<UserResponseModel> users = null;

    private AdminSignInResponseModel admin = null;
    private UserDetailsResponseModel currentUser = null;

    protected override async Task OnInitializedAsync()
    {
        var result = await BrowserStorage.GetAsync<string>("username");
        var value = result.Success ? result.Value : null;

        if (value!= null)
        {
            admin = new AdminSignInResponseModel() { Username = value };
            users = await Service.GetAllUsers();
        }
    }

    private async Task CreateUser()
    {
        if (FirstName == null || LastName == null)
            return;

        var newUser = new UserRequestModel()
            {
                FirstName = FirstName,
                LastName = LastName,
                UserName = null,
                IsAdmin = false,
                Password = null,
            };

        var user = await Service.CreateUser(newUser);
        users = await Service.GetAllUsers();
        FirstName = null;
        LastName = null;
    }

    private async Task GetDetails(int id)
    {
        isDetails =true;
```

```
        currentUser = await Service.GetUserDetails(id);
    }

    private async Task CloseDetails()
    {
        isDetails =false;
        currentUser= null;
    }

    private async Task DeleteUser(int id)
    {
        await Service.DeleteUser(id);
        users = await Service.GetAllUsers();
    }

    private async Task DeleteProcedure(int id)
    {
        await Service.DeleteMedicalProcedure(id);
        currentUser = await Service.GetUserDetails(currentUser.Id);
    }

     private async Task CreateMedicalProcedure()
    {
        if (Name == null || Note == null)
            return;

        var newProcedure = new MedicalProcedureRequestModel()
            {
                Name = Name,
                Note = Note,
                UserId = currentUser.Id,
                Date = DateTime.Now,
            };

        await Service.CreateMedicalProcedure(newProcedure);
        currentUser = await Service.GetUserDetails(currentUser.Id);
        Name = null;
        Note = null;
    }

}


namespace HospitalApp.DAL.Repository
{
    public class Repository<T> : IRepository<T> where T : class
    {
        private readonly IDataContext _context;
        private DbSet<T> _entities;

        public Repository(IDataContext context)
        {
            _context = context;
        }

        private DbSet<T> Entities
        {
```

```
    get
    {
        if (_entities == null)
            _entities = _context.Set<T>();

        return _entities;
    }
}

private void ThrowIfEntityIsNull(T entity)
{
    if (entity == null)
        throw new ArgumentNullException("entity");
}

public virtual IQueryable<T> Table
{
    get
    {
        return Entities;
    }
}

public IList<T> GetAll()
{
    return Entities.ToList();
}

public IQueryable<T> Get(Expression<Func<T, bool>> predicate)
{
    return Entities.Where(predicate);
}

public T Find(Expression<Func<T, bool>> predicate)
{
    return Entities.FirstOrDefault(predicate);
}

public T GetById(object id)
{
    if (typeof(T) == typeof(ApplicationUser))
    {
        var users = Entities.Include("Tokens");

        return ((IQueryable<ApplicationUser>)users).FirstOrDefault(w => w.Id == (int)id) as T;
    }

    return Entities.Find(id);
}

public void Insert(T entity)
{
    try
    {
        ThrowIfEntityIsNull(entity);

        Entities.Add(entity);
```

```
    }
    catch (Exception dbEx)
    {
        throw;
    }
}

public void InsertRange(IEnumerable<T> entities)
{
    try
    {
        foreach (var item in entities)
        {
            ThrowIfEntityIsNull(item);

            Entities.Add(item);
        }
    }
    catch (Exception dbEx)
    {
        throw;
    }
}

public async Task InsertAsync(T entity)
{
    await new Task(() => Insert(entity));
}

public void Update(T entity)
{
    try
    {
        ThrowIfEntityIsNull(entity);
        ((DataContext)_context).Entry(entity).State = EntityState.Modified;
    }
    catch (Exception dbEx)
    {
        throw;
    }
}

public void UpdateRange(List<T> entities)
{
    try
    {
        foreach (var entity in entities)
        {
            ThrowIfEntityIsNull(entity);
            ((DataContext)_context).Entry(entity).State = EntityState.Modified;
        }
    }
    catch (Exception dbEx)
    {
        throw;
    }
}
```

```
        public async Task UpdateAsync(T entity)
        {
            try
            {
                await new Task(() => Update(entity));
            }
            catch (Exception dbEx)
            {
                throw;
            }
        }

        public void UpdateWhere(Expression<Func<T, bool>> predicate, Expression<Func<T, T>>
newEntety)
        {
            try
            {
                Entities.Where(predicate).UpdateFromQuery(newEntety);
            }
            catch (Exception dbEx)
            {
                throw dbEx;
            }
        }

        public void Delete(T entity)
        {
            try
            {
                ThrowIfEntityIsNull(entity);

                Entities.Remove(entity);
            }
            catch (Exception dbEx)
            {
                throw dbEx;
            }
        }

        public void DeleteById(int id)
        {
            try
            {
                T entity = GetById(id);

                ThrowIfEntityIsNull(entity);

                Entities.Remove(entity);
            }
            catch (Exception dbEx)
            {
                throw;
            }
        }

        public int Count()
        {
```

```
        return Entities.Count();
    }

    public int Count(Expression<Func<T, bool>> predicate)
    {
        return Entities.Count(predicate);
    }

    public void DeleteRange(IEnumerable<T> entities)
    {
        try
        {
            Entities.RemoveRange(entities);
        }
        catch (Exception dbEx)
        {
            throw;
        }
    }

    public IEnumerable<S> ExecuteStoredProcedure<S>(string storedProcedure, Dictionary<string,
object> parameters) where S : class
    {
        try
        {
            return _context.GetDataFromSqlCommand<S>(storedProcedure, parameters);
        }
        catch (Exception dbEx)
        {
            throw;
        }
    }

    public bool Any(Expression<Func<T, bool>> predicate)
    {
        return Entities.Any(predicate);
    }

    #region Pagination

    public IQueryable<T> GetPage(int limit, int offset, string sort, bool orderByDescending)
    {
        try
        {
            var entityType = _context.Model.FindEntityType(typeof(T));

            // Table info
            var tableName = entityType.GetTableName();
            var tableSchema = entityType.GetSchema();

            Dictionary<string, string> names = new Dictionary<string, string>();

            // Column info
            foreach (var property in entityType.GetProperties())
            {
                var propertyName = property.Name;
```

```
                var columnName = property.GetColumnName(StoreObjectIdentifier.Table(tableName,
tableSchema));

                names.Add(propertyName, columnName);

                //var columnType = property.Relational().ColumnType;
            };

            var orderByStr = "";

            if (names.Any(w => string.Compare(w.Key, sort, true) == 0))
            {
                orderByStr = names.First(w => string.Compare(w.Key, sort, true) == 0).Value;
            }
            else
            {
                orderByStr = "Id";
            }
            return _entities.FromSqlRaw(string.Join(" ", "SELECT * FROM", tableName, "ORDER BY",
orderByStr, (orderByDescending ? "DESC" : "ASC"), "OFFSET", offset, "ROWS FETCH NEXT", limit,
"ROWS ONLY"));
        }
        catch (SqlException ex)
        {
            throw ex;
        }
        catch (Exception dbEx)
        {
            throw dbEx;
        }
    }

    #endregion


  }
}
```

```
<div class="top-row pl-4 navbar navbar-dark">
  <a class="navbar-brand" href="">ClinicSSU</a>
  <button class="navbar-toggler" @onclick="ToggleNavMenu">
    <span class="navbar-toggler-icon"></span>
  </button>
</div>

<div class="@NavMenuCssClass" @onclick="ToggleNavMenu">
  <ul class="nav flex-column">
    <li class="nav-item px-3">
      <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
        <span class="oi oi-home" aria-hidden="true"></span> Home
      </NavLink>
    </li>
    <li class="nav-item px-3">
      <NavLink class="nav-link" href="account">
        <span class="oi oi-plus" aria-hidden="true"></span> My Account
      </NavLink>
    </li>
```

```
        <li class="nav-item px-3">
            <NavLink class="nav-link" href="users">
                <span class="oi oi-list-rich" aria-hidden="true"></span> Users
            </NavLink>
        </li>
    </ul>
</div>

@code {
    private bool collapseNavMenu = true;

    private string NavMenuCssClass => collapseNavMenu ? "collapse" : null;

    private void ToggleNavMenu()
    {
        collapseNavMenu = !collapseNavMenu;
    }
}
```