Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студентки *Куряков Євгеній Сергійович*
(ПІБ)

академічної групи *121М-20-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

на тему: *Розробка ПЗ для зберігання і підвищення ефективності обробки інформації медичного закладу*

_____ *Є.С. Куряков*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	Проф. Мещеряков Л.І.			
економічний	Доц. Касьяненко Л.В.			

Рецензент				

Нормоконтролер	Доц. Реута О.В.			

Дніпро
2022

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 20 ____ Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту *121М-20-1* *Курякову Євгенію Сергійовичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Розробка ПЗ для зберігання і підвищення ефективності обробки інформації медичного закладу*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 10.12.2021 р. № 1036-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес розробки програмного забезпечення для зберігання і підвищення ефективності обробки інформації медичного закладу.

Предмет досліджень – моделі даних та методи їх систематизування і візуалізації для медичних закладах.

Мета роботи – підвищення ефективності роботи медичного закладу за рахунок кращого документообігу, економія витрат на персоналі за рахунок автоматичного збереження інформації, її менеджменту та перевірки валідності.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна результатів кваліфікаційної роботи полягає в удосконаленні системи охорони здоров'я, підвищення безпеки та якості документообігу у медичних закладах.

Практична цінність полягає у тому, що результати роботи, отримані в ході дослідження, можуть застосовуватися як основна система обробки та збереження усіх даних пацієнтів, лікарняних засобів(кількість, строк придатності, тощо) ті звітності у медичних закладах.

4 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець)**
Аналіз теми та постановка задачі	30.09.2021-12.10.2021
Аналіз предметної області, збір інформації.	13.10.2021-31.10.2021
Практична реалізація системи для зберігання і підвищення ефективності інформації у медичному закладі.	01.11.2021-16.12.2021
Написання пояснювальної записки.	17.12.2021 – 10.01.2022

Завдання видав _____ Мещеряков Л.І.

(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ Куряков Є.С. _

(підпис) (прізвище, ініціали)

Дата видачі завдання: 30.09.2021 р.

Термін подання дипломного проекту до ЕК 17.12.2021р.

РЕФЕРАТ

Пояснювальна записка: 84 с., 13 мал., 3 додатка, 17 джерел

Об'єкт дослідження: процес розробки програмного забезпечення для зберігання і підвищення ефективності обробки інформації медичного закладу.

Предмет дослідження: моделі даних та методи їх систематизування і візуалізації для медичних закладах.

Мета магістерської роботи: підвищення ефективності роботи медичного закладу за рахунок кращого документообігу, економія витрат на персоналі за рахунок автоматичного збереження інформації, її менеджменту та перевірки валідності.

Методи дослідження. Для виконання поставлених завдань були використані методи моделювання і нормалізації даних медичного закладу та методи візуалізації і маніпулювання інформацією за допомогою мови C#, Ms SQL Server, Visual studio 2019, .Net Framework, Widows Forms.

Наукова новизна результатів кваліфікаційної роботи полягає в удосконаленні системи охорони здоров’я, підвищення безпеки та якості документообігу у медичних закладах.

Практична цінність полягає у тому, що результати роботи, отримані в ході дослідження, можуть застосовуватися як основна система обробки та збереження усіх даних пацієнтів, лікарняних засобів(кількість, строк придатності, тощо) ті звітності у медичних закладах.

У розділі «Економіка» проведено розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПО і тривалості його розробки.

Список ключових слів: програмне забезпечення, бази даних, медичні заклади, пацієнти, ліки, звіти, C#, Ms SQL Server, .Net Framework, Widows Forms.

ABSTRACT

Explanatory note: 103 pages, 13 figures, 3 applications, 17 sources

Object of research: the process of developing software to store and increase the efficiency of information processing of the medical institution.

Subject of research: data models and methods of their systematization and visualization for medical institutions.

Purpose of Master's thesis: improving the efficiency of the medical institution through better paperwork, saving staff costs through automatic storage of information, its management and validation.

Research methods. Methods of modeling and normalization of medical institution data and methods of visualization and manipulation of information using C#, Ms SQL Server, Visual studio 2019, .Net Framework, Widows Forms were used to perform the tasks.

Originality of research lies to improve the health care system, improve the security and quality of paperwork in medical institutions.

Practical value of the results. Is that the results of the study can be used as the main system for processing and storing all patient data, medicines (quantity, expiration date, etc.) and reporting in medical institutions.

In the Economics section we calculated the complexity of software development and the costs of software development, the duration of the actual development are calculated.

Keywords: software, databases, medical facilities, patients, drugs, reports, C #, Ms SQL Server, .Net Framework, Widows Forms.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення;

ПК – персональний комп'ютер;

МІС – Медична інформаційна система;

ЕОМ – електронна обчислювальна машина;

СЕД – система електронного документообігу;

СУБД – система управління базами даних;

API – Application Programming Interface (програмний інтерфейс програми);

XML – Extensible Markup Language;

LINQ – Language Integrated Query;

GC – Garbage Collector.

ВСТУП

Актуальність дослідження. Ефективне інформаційне забезпечення медицини є однією з важливих передумов її якості абсолютно на всіх етапах – від первинного огляду в поліклініці до надання спеціалізованої допомоги в медичних закладах. В наші дні інформатизація охорони здоров'я означає використання інформаційних технологій, за допомогою яких значно спрощується ведення обліку медичних послуг та управління медичною інформацією, а також надає можливість виконувати необхідні дії в електронному вигляді. Як показує всесвітня практика – впровадження Медичної інформаційної системи (МІС) та використання її лікарями дозволяє зекономити до 20% часу в порівнянні з використанням лікарями паперових бланків. В Україні за всю історію її незалежності, було декілька спроб впровадити МІС, починаючи зі спроби в 2006 році, закінчуючи початком введенням МІС в 2017 році. Поки система МІС працює не у всіх медичних закладах країни, лікарі повинні брати складну роботу по заповненню десятків бланків на себе. Крім того впроваджені зараз МІС в основному призначені для використання сімейними лікарями, а не для лікарів - спеціалістів. Сьогодні існує десятки різних систем, які впроваджуються в тисячах українських медичних закладах, наприклад «EMCIMED», «Helsi», «Dr. Eleks» та інші. Крім проблеми надійності захисту даних, системи можуть вийти надто «технологічними», адже потребуватимуть обладнання усіх медичних закладів країни комп'ютерами з доступом в інтернет та встановлення на них складного програмного забезпечення.

Мета дослідження – підвищення ефективності роботи медичного закладу за рахунок кращого документообігу, економія витрат на персоналі за рахунок автоматичного збереження інформації, її менеджменту та перевірки валідності.

Завдання дослідження. Для досягнення поставленої мети в роботі сформульовані і вирішені такі завдання:

1. Викласти основні моделі та їх властивості притаманні медичним закладам.

2. Визначити основні методи збереження та модифікації данних.
3. Визначити основні проблеми, що виникли в попередніх спробах впровадити МІС.
4. Знайти можливі шляхи оптимізації роботи з медичними даними.
5. Спроектувати та розробити відповідне програмне забезпечення, яке можна застосовувати на різних конфігураціях ОС.

Об'єкт дослідження. процес розробки програмного забезпечення для зберігання і підвищення ефективності обробки інформації медичного закладу.

Предмет дослідження. моделі даних та методи їх систематизування і візуалізації для медичних закладах.

Методи дослідження. Для виконання поставлених завдань були використані методи моделювання і нормалізації даних медичного закладу та методи візуалізації і маніпулювання інформацією за допомогою мови C#, Ms SQL Server, Visual studio 2019, .Net Framework, Widows Forms.

Наукова новизна результатів кваліфікаційної роботи полягає в удосконаленні системи охорони здоров’я, підвищення безпеки та якості документообігу у медичних закладах.

Практичне значення полягає у тому, що результати роботи, отримані в ході дослідження, можуть застосовуватися як основна система обробки та збереження усіх даних пацієнтів, лікарняних засобів(кількість, строк придатності, тощо) ті звітності у медичних закладах.

Особистий внесок автора: розробка унікальної програми для збереження та маніпулювання інформації медичного закладу.

Структура і обсяг роботи. Робота складається з вступу, трьох розділів і висновків. Містить 103 сторінки, в тому числі 69 сторінок тексту основної частини з 13 рисунками, списку використаних джерел з 17 найменуваннями на 5 сторінках, 3 додатка на 10 сторінках.

РОЗДІЛ 1.
АНАЛІЗ ТЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальний опис проблеми

Медична реформа передбачає використання інформаційних технологій для створення МІС зберігання інформації про пацієнтів. Тому обсяги інформації про пацієнтів, що зберігається в електронному вигляді, зростають. Збільшуються і потреби в функціональному наповненні цих систем.

Незважаючи на те, що зараз масово впроваджуються системи МІС по всій території України, сфера їх застосування наразі охоплює лише первинну медицину – сімейних лікарів, терапевтів і педіатрів. Для решти лікарів проблема паперових носіїв залишається актуальною. Майже половина лікарів досі використовує під час своєї роботи паперові носії.

В Україні, починаючи з 2018 року впроваджується електронна система охорони здоров'я - eHealth, яка дасть змогу забезпечити права пацієнта в частині якості надання медичних послуг, оптимізувати взаємодію пацієнта і лікаря шляхом автоматизації ведення обліку медичних послуг і управлінню медичною інформацією в електронному вигляді.

На даний час система eHealth складається з:

- Центральної бази даних — ЦБД (адміністратор ДП “Електронне здоров’я”);
- Електронних медичних інформаційних систем — МІС (системи, які дають змогу автоматизувати роботу медичних закладів з ЦБД);

Також, центральна база даних eHealth України передбачає забезпечення прозорості фінансування системи охорони здоров’я, надавати можливість працювати без паперу з поступовим переходом на електронний облік, зокрема е-рецепт, е-картка, е-направлення, для створення нових електронних сервісів сформувати бізнес-середовище, створити простів для інновацій в медицині та сприяти розвитку медичного ІТ ринку.

Швидкий доступ до всієї цієї інформації пацієнта дасть лікарю цілісну картину про його здоров'я, а повна медична історія в електронному вигляді допоможе поставити правильний діагноз. З іншого боку, записатися до лікаря на прийом, перевірити наявність ліків у медичних закладах чи аптеках, отримати рецепт на препарат —це все можливо зробити, не виходячи з дому.

Від початку реформи системи охорони здоров’я за допомогою побудови системи eHealth змінено модель фінансування медичних закладів, ефективному перерозподілу коштів згідно принципу «гроші ходять за пацієнтом» та, відповідно, збільшення доходів медичних працівників.

Ця система дозволить контролювати, наскільки ефективно витрачаються виділені державні кошти. Спочатку, як це зазначалось вище, eHealth охопить первинну медицину - сімейних лікарів, терапевтів і педіатрів. Пацієнти будуть підписувати декларації з вибраними лікарями, а лікарі - реєструвати їх в системі. Це допоможе державі оплачувати лікарю роботу з кожним пацієнтом. У 2019 році Україна знаходиться на завершальному етапі впровадження першої частини електронної системи охорони здоров’я, зокрема завершена реєстрація медичних установ та майже підписні декларації лікарів і пацієнтів.

Водночас, з боку влади постійно приймаються нормативно-правові акти щодо ухвалення Концепції розвитку екстреної допомоги. Так, в поточному році розпочалась трансформація екстреної медичної допомоги, яка дозволить принципово покращити якість та доступність екстреної медичної допомоги, щоб

в критичній ситуації кожен українець оперативно отримував необхідну допомогу.

У зв’язку з впровадженням наступного етапу реформи відбудуться оновлення алгоритмів диспетчеризації і створення нової ІТ-інфраструктури, що дасть змогу мати інформацію про стан кожного виклику в будь-який час.

Також дуже важливо підтримати на державному рівні престижність професій працівників екстреної медичної допомоги і забезпечити достойну оплату їхньої праці, облаштування сучасних диспетчерських служб по усій країні, шляхом:

- закупівлі спеціалізованого санітарного транспорту;
- оснащення для навчально-тренувальних відділів центрів екстреної медичної допомоги та медицини катастроф (ЦЕМД);
- апаратно-програмного комплексу диспетчерських служб центрів екстреної медичної допомоги та медицини катастроф.

Планується, що до кінця 2019 року в усіх регіонах України діятиме мережа сучасних диспетчерських служб, об’єднаних єдиною електронною системою. Завдяки сучасним диспетчерським, новим протоколам та центральній інформаційно-аналітичній системі, виклики оброблятимуть оперативніше, а бригади екстреної медичної допомоги швидше доїжджатимуть на місце події[1].

Водночас, існують актуальні проблеми, що вповільнюють запровадження системи, зокрема, відсутність єдиного унікального ідентифікатора для пацієнтів, недосконаліть інформаційної інфраструктури та взаємодії між загальнодержавними реєстрами, недосконаліть ряду реєстрів, недостатність фахових спеціалістів для автоматизації та управління змінами, недостатність комп’ютерного та мережевого обладнання в закладах охорони здоров’я тощо.

Зараз використовуються і впроваджуються лише окремі елементи eHealth. Досі немає розробленої та затвердженої концепції та Національної стратегії

електронної системи охорони здоров’я, чіткої затвердженої на законодавчому рівні архітектури системи та технічного завдання. Відсутні гармонізовані міжнародні та затверджені чисельні стандарти збереження, обробки та передачі медичних даних, класифікаторів та довідників. Немає затверджених єдиних функціональних вимог до медичних інформаційних систем (приватних компаній), які б проходили відповідну цим вимогам сертифікацію в залежності від рівня надання медичних послуг та після цього рекомендувалися для впровадження в медичних закладах відповідного рівня.

Сьогодні кожен лікар, окрім картки пацієнта, заповнює цілу низку іншої документації. До того ж, це зовсім не ефективно.

Проте на даний час не скасовано заповнення усіх паперових форм, оскільки серед них є статистичні дані, які потрібні для порівняння.

Також сьогодні не можна перевірити процес заповнення певних форм і те, наскільки дані, записані там, відповідають дійсності. При веденні всієї документації у електронній формі, це можна буде відслідковувати.

Підписуючи декларації з пацієнтами, сімейні лікарі, терапевти і педіатри працюють з персональними даними. Слід зазначити, що важливо, яким буде захист персональних даних пацієнтів.

Усі відомості або сукупність відомостей про пацієнта, які вносяться в декларацію про вибір лікаря, є персональними даними пацієнта (ПІБ, дата народження, реєстраційний номер облікової картки платника податків, номер та серія паспорту або інших документів, що посвідчують особу, адреса проживання та інші дані, за якими можна ідентифікувати пацієнта).

Закон України «Про захист персональних даних» від 1 червня 2010 року

№ 2297-VI[2] (далі – Закон) регулює правові відносини, пов’язані із захистом і обробкою персональних даних, і спрямований на захист

основоположних прав і свобод людини і громадянина, зокрема права на невтручання в особисте життя, у зв’язку з обробкою персональних даних.

Цей Закон поширюється на діяльність з обробки персональних даних, яка здійснюється повністю або частково із застосуванням автоматизованих засобів, а також на обробку персональних даних, що містяться у картотеці чи призначені до внесення до картотеки, із застосуванням неавтоматизованих засобів.

Згідно з формулюванням Закону персональні дані — відомості чи сукупність відомостей про фізичну особу, яка ідентифікована або може бути конкретно ідентифікована.

Отже, коли у системі почнуть працювати електронний лікарняний та електронна медична картка, персональні дані будуть оброблятися для забезпечення лікувального процесу та для покращення функціонування електронної системи охорони здоров’я.

Пацієнт, підписуючи декларацію з лікарем, людина погоджується, що його персональні дані в електронній системі будуть доступні для обробки лікарями, до яких він будете звертатися за медичною допомогою.

На персональні дані пацієнтів має поширюватись дія законодавства про лікарську таємницю і вони повинні забезпечувати захист таких персональних даних. Медичні працівники зобов'язані не допускати розголошення у будь- який спосіб персональних даних, які їм було довірено або які стали відомі у зв'язку з виконанням професійних чи службових або трудових обов'язків, крім випадків, передбачених законом.

Доступ до даних про пацієнта, що містяться в електронній системі охорони здоров'я, можливий лише у разі отримання згоди такого пацієнта (його законного представника) у письмовій формі або у формі, що дає змогу зробити висновок про надання згоди.

Без згоди доступ до інформації про пацієнта можливий лише у таких випадках:

- наявності ознак прямої загрози життю пацієнта;
- за умови неможливості отримання згоди такого пацієнта чи його законних представників;
- за рішенням суду.

За недодержання встановленого законодавством порядку захисту персональних даних, що призвело до незаконного доступу до них або порушення прав пацієнта як суб'єкта персональних даних, передбачена адміністративна відповідальність. А за порушення недоторканності приватного життя (незаконне збирання, зберігання, використання, знищення, поширення конфіденційної інформації про особу або незаконна зміна такої інформації) — кримінальна відповідальність.

Отже, електронна система має бути спроектована для роботи з персональними даними з дотриманням найкращих світових практик у сфері захисту даних, мати комплексну систему захисту інформації (КСЗІ) , а для перевірки надійності захисту даних пройти тестування щодо кібербезпеки та обов'язкову атестацію у Державній службі спеціального зв'язку та захисту інформації (ДССЗЗІ).

Варто відмітити, що сучасні МІС впроваджуються на рівні медичних закладів, тож їх використання окремими лікарями є майже неможливою. Тому варто створити таку МІС, яка б повністю задовольняла вимоги тих лікарів, для яких наразі ще не створили робочу систему. Тому виникає необхідність створення додаткових засобів, які б допомагали лікарю заповнювати електроні документи обліку пацієнтів з витратою меншого часу.

Проблема, що розглянута в даному дипломному проєкті полягає в тому, що необхідно створити ПЗ для МІС, яка буде допомагати лікарю – хірургу вести

облік пацієнтів у електронному вигляді, створювати «історію хвороби» та всі необхідні для медичного закладу документи. Відповідно до цього треба розглянути, які вимоги є до ПЗ, системи електронного документообігу, щоб розроблене ПЗ відповідало всім необхідним вимогам та буде підходити для виконання поставленої задачі.

1.2 Електронний документообіг

З моменту активного розвитку інформаційних технологій в світі, провідну роль відіграють принципи створення, збереження та передавання електронної інформації. Системи електронного документообігу (СЕД) надають змогу значно зекономити час роботи персоналу в порівнянні з паперовими документами та надають можливість автоматизованої обробки даних.

Що стосується сфери охорони здоров’я, можна стверджувати, що одужання пацієнта напряму залежить від своєчасного та коректного діагнозу. Варто відмітити, що при наявності систем електронного документообігу, пацієнт отримую набагато більше уваги від лікаря, оскільки останній більше не витрачає значну частину свого робочого часу на паперову тяганину, а більше приділяє увагу саме лікуванню пацієнта.

Перехід на електронну систему обслуговування сприяє раціоналізації роботи медичних установ, та переходу на наступний рівень надання послуг.

На сьогодні законодавчо передбачені загальні вимоги до інформації та документів у електронній системі охорони здоров'я. Створення, внесення, перегляд інформації та документів у центральній базі даних, внесення змін та доповнень до них здійснюються користувачами відповідно до прав доступу. Інформація та документи створюються та вносяться до центральної бази даних українською мовою. У разі коли використання літер української абетки призводить до спотворення інформації, можуть використовуватися латинські

літери і спеціальні символи, зокрема для запису адреси в Інтернеті та адреси електронної пошти.

Документообіг у електронній системі охорони здоров'я здійснюється відповідно до вимог законодавства про електронні документи та електронний документообіг. На всі електронні документи, що вносяться до центральної бази даних, накладається електронний підпис автора або підпис, прирівняний до власноручного підпису відповідно до закону. Інформація під час обробки та обміну в електронній системі охорони здоров'я повинна зберігати цілісність, що забезпечується шляхом захисту від несанкціонованих дій, які можуть призвести до її випадкової або умисної модифікації чи знищення, зокрема шляхом накладення електронного підпису автора або підпису, прирівняного до власноручного підпису відповідно до закону.

Для внесення інформації та документів до центральної бази даних використовуються державні класифікатори, номенклатури та довідники, затверджені в установленому законодавством порядку, зокрема спеціальні класифікації та переліки. Адміністратор здійснює технічну підтримку застосування таких класифікацій, номенклатури, довідників та переліків. Про кожен документ та інформацію, що внесені до центральної бази даних, автоматично робиться унікальний запис у відповідному реєстрі. Ідентифікатором фізичної особи є унікальний номер запису в Єдиному державному демографічному реєстрі[3]

Варто відмітити, що переваги від використання електронного документообігу досить суттєві. По-перше впровадження таких систем дозволяє економити кошти, призначені для купівлі копіювально- розмножувальної техніки й витратних матеріалів до неї, її ремонт тощо. Також це зменшує витрати на зберігання паперових документів(зокрема, фізичне звільнення місця для документів тимчасового зберігання).

По-друге, ще одним позитивним ефектом від переходу на електронний документообіг є значна економія робочого часу працівників, адже за паперового документообігу вони б витрачали більше часу на ручну обробку даних і виконання обслуговуючих функцій.

Вагомими аргументами на користь електронного документообігу є:

- підвищення швидкості пошуку документів;
- запобігання втрати важливої інформації через недбалість персоналу;
- підвищення рівня інформаційної безпеки, за рахунок шифрування даних;
- централізоване зберігання інформації в електронному архіві.

Однак в системах електронного документообігу існують і свої недоліки До основних недоліків належать:

- досить високі початкові витрати на впровадження СЕД;
- досить відчутні часові витрати: перехід на електронний документообіг неможливо здійснити за короткий проміжок часу;
- необхідність навчання і перепідготовки персоналу для роботи з СЕД;
- можливість втрати чи витоку інформації

Функціональні можливості систем електронного документообігу представлені в таблиці 1.1

Таблиця 1.1

Функції	Зміст
Реєстрація документів	Реєстрація і облік
	Ведення журналу реєстрації та обліку
Контроль виконання	Ведення контрольних завдань
	Перенесення термінів виконання завдань при необхідності
	Ведення стану виконання
Пошук	Атрибутивний пошук об’єктів документообігу
Звітність	Атрибутивний пошук об’єктів документообігу
	Журнал реєстрації подій
Адміністрування	Управління процесами
	Налагодження системи
	Розмежування повноважень

Можна зробити висновок, що впровадження електронного документообігу у лікарнях країни є обов’язковим, оскільки дозволить значно зменшити час, який потрібен лікарям для заповнення всіх потрібних бланків, актів, виписок тощо. Варто відмітити, що можлива втрата чи виток інформації є значною проблемою впровадження даної системи.

1.3 Аналіз існуючих систем

eHealth – це сукупність інформаційних сервісів в галузі охорони здоров’я та охоплює інформаційних простір різних галузей охорони здоров’я – медичну практику, управління медичними закладами, медичне право, фармацевтику, інформаційні сервіси для пацієнтів тощо.

Центральний компонент – це спеціальне програмне забезпечення, яке накопичує інформацію, що надходить з периферійного компоненту eHealth – медичних інформаційних систем або іншого програмного забезпечення – за певними правилами.

Периферійний компонент – це програмне забезпечення, зокрема медичні інформаційні системи, з якими безпосередньо працюють медичні працівники в медичних лікувальних закладах на місцях, до яких відносяться «EMCIMED»,

«Helsi», «Dr. Eleks».

Наразі існує купа МІС які доступні на ринку продуктів всім бажаючим, деякі з них являються локальними проєктами вузького спектру застосування, деякі з них являються глобальними проєктами державного значення.

Для аналізу МІС були взяті наступні програмні продукти «EMCIMED»,

«Helsi», «Dr. Eleks». Для порівняння були взяті саме ці системи, оскільки вони є одними з найрозповсюдженішими МІС в Україні.

Так, за допомогою інформаційної системи «Dr. Eleks» здійснюється реєстрація медичних закладів, лікарів первинної ланки, пацієнтів та фінансування медичного закладу, виходячи із кількості пацієнтів.

В медичній інформаційній системі «Helsi» надається сервіс запису до лікарів та зберігання медичних даних.

В медичній інформаційній системі «ЕМСіМЕД» зосереджені всі необхідні інструменти для роботи медичних працівників з центральним компонентом eHealth, а головне: електронна медична картка пацієнта, персональні дані пацієнта та його медичні записи.

На сьогодні на ринку представлені МІС проходять тестування у сімейних лікарів, терапевтів і педіатрів тощо.

Разом з цим, нажаль є купа лікарів, для яких МІС ще не адаптоване. Лікарі-хірурги - вони досі в очікуванні МІС, котра зможе задовольняти всі їх потреби. Поки на ринку немає таких МІС, лікарі змушені витрачати свій дорогоцінний час на роботу з паперовими актами, бланками, історіями хвороби.

Варто відмітити, що більша частина МІС, які представлені на ринку, не мають у відкритому доступі технічних характеристик продукту, а мають лише загальні відомості про продукт. Цей фактор перешкоджає первинному аналізу та порівнянню декількох МІС між собою. Щоб обрати оптимальний варіант потрібно витрачати кучу часу на телефонні дзвінки в компанії-постачальники, а також провести не одну особисту зустріч з їх представниками. У відкритому доступі вдалось знайти лише повні технічні характеристики системи «Helsi»

[4] та деякі технічні характеристики системи «Dr.Eleks»[5]. Що стосується системи «EMCIMED», тут все набагато складніше. У відкритому доступі майже немає технічних характеристик даного програмного забезпечення, що є, на мою думку, не допустимим для ПЗ такого масштабу. Вся знайдена інформація знаходиться на офіційному сайті системи «EMCIMED» [6].

МІС – досить складні системи, якщо розібратися в графічному інтерфейсі можна за допомогою інструкцій, які надаються до кожної системи, то самостійне встановлення та їх налаштування є неможливим. При первинному налаштуванні програми в медичних закладах обов’язково потрібен спеціаліст з команди розробників даного програмного продукту.

Деякі МІС пропонують динамічний графічний інтерфейс. Це означає, що за допомогою спеціаліста можна побудувати графічний інтерфейс, який максимально задовольняв вимоги конкретного лікаря. Це є значним плюсом для даних систем, адже від зручності залежить якість і швидкість роботи лікаря. Нажаль МІС в яких присутній динамічний інтерфейс це скоріше виняток з правил. Під час аналізу медичних інформаційних систем, було знайдено лише одну систему, яка має динамічний інтерфейс.

Варто відмітити, що в разі збою в роботі МІС, самостійне рішення проблеми є майже неможливим, оскільки може призвести до ще гірших наслідків. В таких випадках викликається спеціаліст з команди розробників системи, котрий розбирається і вирішує проблему. В свою чергу це є прямою загрозою анонімності пацієнтів.

Для порівняння було взято характеристики програм, оскільки порівняння кількісних характеристик є досить важким, враховуючи їх кількість та складність доступу до технічних характеристик певних МІС

Отже для порівняння було взято наступні характеристики: використання СУБД, надійність, робочий інтерфейс. Таблиця аналіз існуючих систем 1.2

Таблиця 1.2

	EMCIMED	Helsi	Dr. Eleks
Використання СУБД	Побудована на базі систем керування базами даних MS SQL	PostgreSQL версії не нижче 9.5 або аналог, MongoDB версії не нижче 3.6 або аналог.	Побудована на основі платформи Microsoft SQL Server 2005/ 2008/ 2012/ 2014.

Продовження таблиці 1.2

Надійність	Для зберігання і передачі необхідної інформації на робочі місця медперсоналу використовується трирівнева система: SQL-Сервер баз даних - Application Server - Client.	Дані зберігаютьсяу дата-центрі, який отримав сертифікат комплексної системи захисту інформації від Державної служби спеціального зв'язку і захисту інформації України.	Для безпечної комунікації в мережі використовується інфраструктура безпеки доменів Windows із застосуванням служби Active Directory.
Робочий інтерфейс	Див. рис. 1.3	Див. рис. 1.4	Див. рис. 1.5

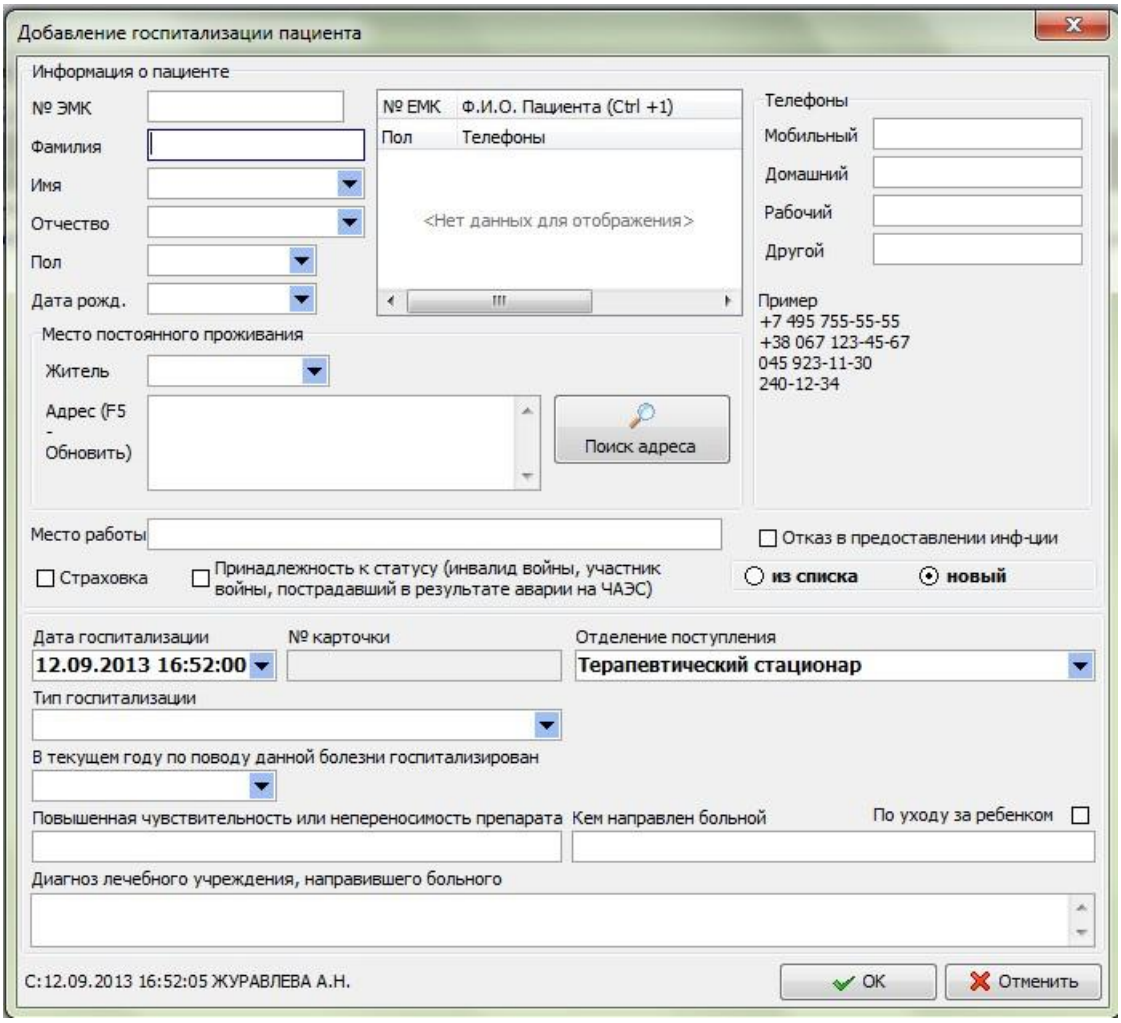

Рисунок 1.3 – Інтерфейс MIC «EMCIMED»

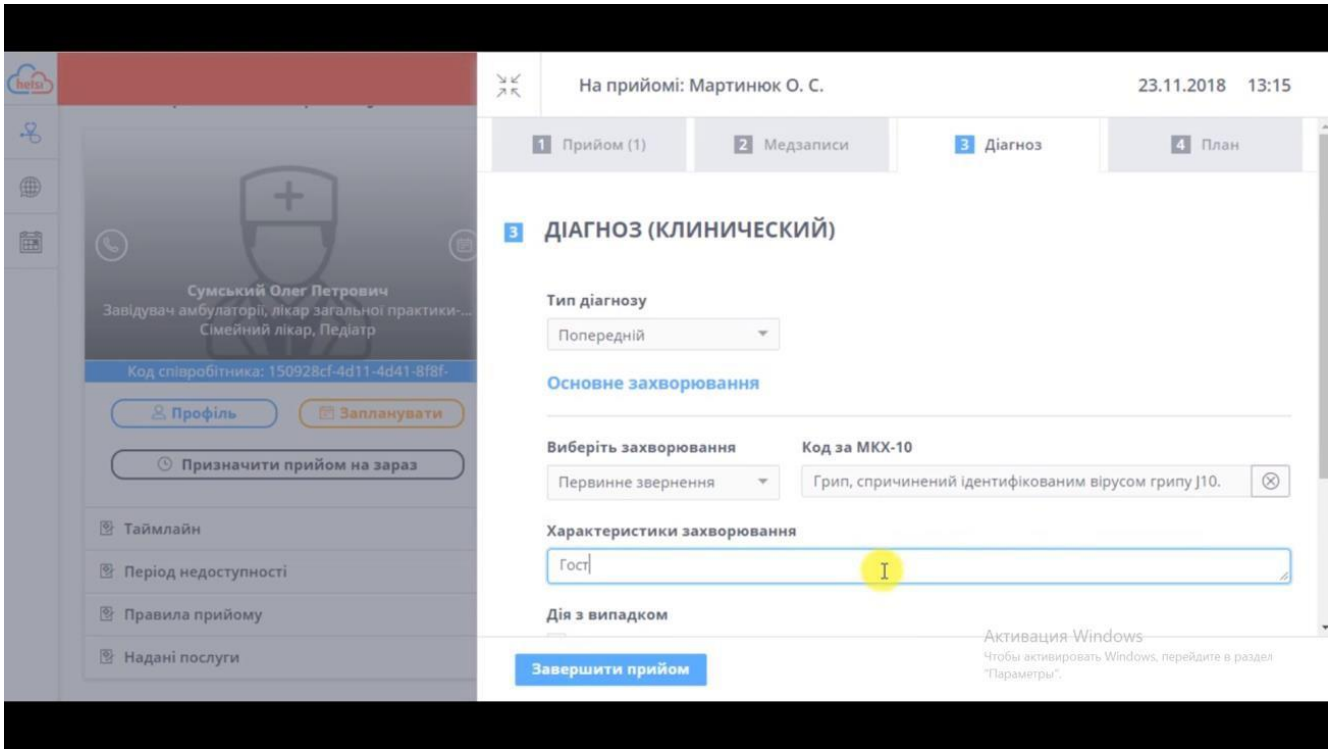

Рисунок 1.4 – Інтерфейс MIC «Helsi»

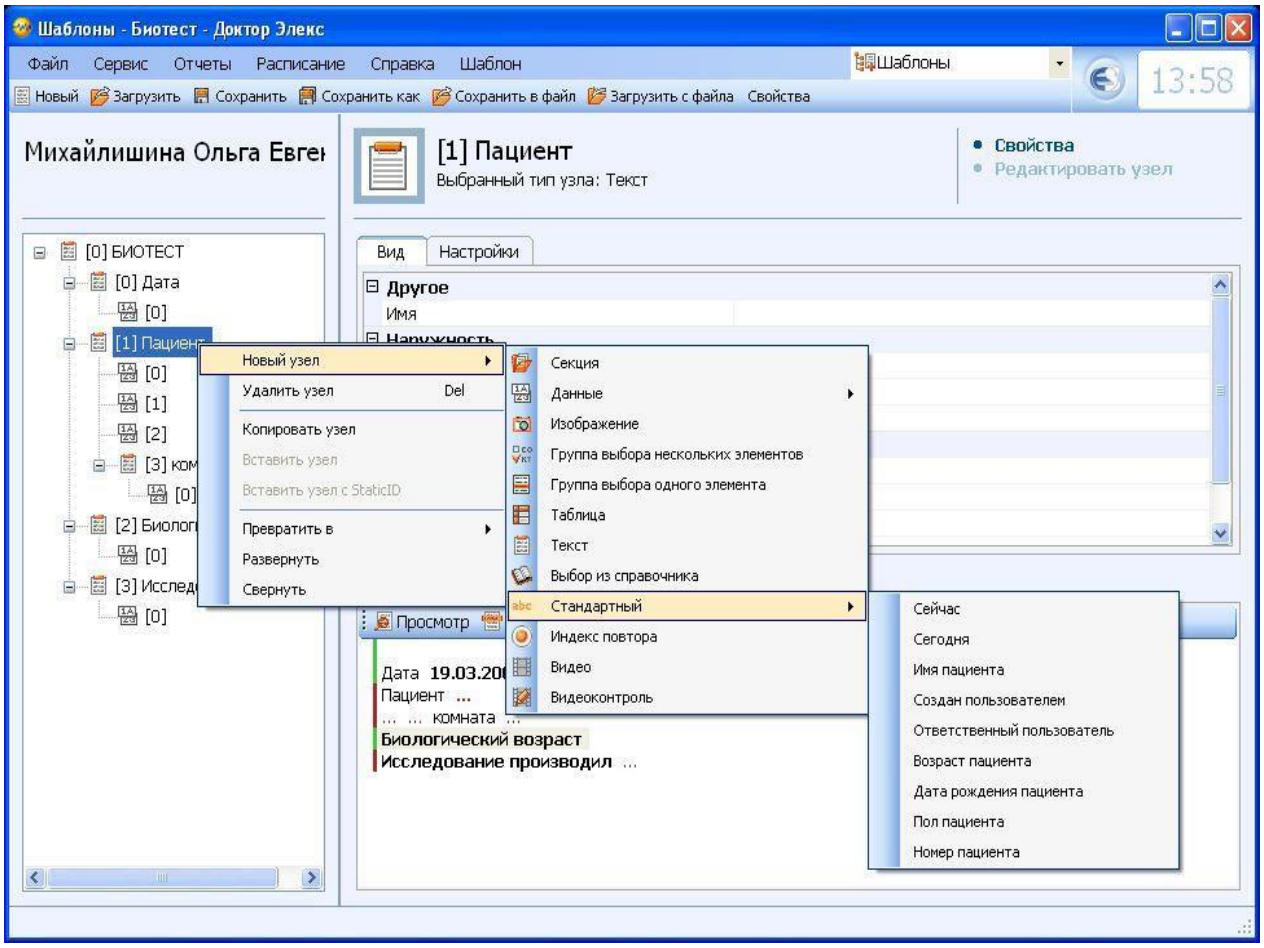

Рисунок 1.5 – Інтерфейс MIC «Dr. Eleks»

Проаналізувавши приведені данні можна прийти до висновку, що з існуючих систем, МІС «Helsi» являється оптимальною та має найменшу кількість мінусів з систем, представлених для порівняння. До плюсів можна віднести наявність у вільному доступі технічну характеристику продукту, сертифікат безпеки від Державної служби зв'язку і захисту. Гарним плюсом є те, що МІС «Helsi» існує в мобільній версії. Варто зауважити, що МІС «Helsi» має найкращий та зрозумілий інтерфейс серед усіх представлених систем

Однак, варто відмітити, що кожна з наведених для порівняння систем має свої мінуси. Наразі не розроблена така МІС, яка зможе замінити паперовий документообіг на електронний документообіг для всіх лікарів та лікарень, оскільки багато потрібних функцій ще не реалізовані в цих системах.

Питання збереження інформації досить гостро стоїть, особливо в сфері охорони здоров’я. До тих пір, поки компанії не можуть дати стовідсоткову гарантію безвідмовної роботи своїх систем, існує висока ймовірність втрати чи витоку інформації.

Таким чином, необхідно створити систему, яка б принаймні компенсувала мінуси вже існуючих систем, була зрозумілою в освоєнні та розроблена з дотриманням усіх державних стандартів та побажань лікарів.

1.4 Аналіз існуючих технологій для вирішення поставленої задачі

1.4.1 Вибір основної мови програмування

Java є мовою програмування та може працювати у будь-якій операційній системі. Java також надає віртуальну машину Java (JVM), яка дозволяє запускати код на будь-якому пристрої та будь-якій операційній системі. NET насправді є основою у найсуворішому значенні цього слова. ASP.NET використовується .NET для створення різних програм. Java – це мова програмування, а .NET – це фреймворк, який може використовувати кілька мов.

Таблиця 1.3

Основа порівняння	**Java**	**.Net**
Основна різниця	Java є мовою програмування та може працювати у будь-якій операційній системі. Java також надає віртуальну машину Java (JVM), яка дозволяє запускати код на будь-якому пристрої та будь-якій операційній системі. Тому він вважається мовою, що переноситься. Java підтримує такі мови, як Python, Ruby тощо.	.Net переважно вважається основою. Він переважно орієнтований на Windows і підтримує різні версії Windows. .Net підтримує такі мови, як C #, C ++, VB.NET і т. д. Він підтримує різні веб-служби за умовчанням, оскільки він вбудований.

Продовження таблиці 1.3

Інтегрована середа розробки	Java IDE надають редактор коду, відладчик, компілятор та такі елементи, як Maven, які допомагають легко створювати код. Існує кілька IDE, таких як Eclipse, NetBeans, IntelliJ IDEA, які полегшують процес розробки. Платформи зручні для користувача і навіть новачки можуть легко почати використовувати їх. Eclipse має багато плагінів, які дозволяють розробникам писати хороший код, а також тестувати його.	Visual studio інтегрована із .net. Він включає безліч функцій, таких як специфічні для мови середовища. .NET IDE діє як єдиний робочий простір, що має інтерфейс з декількома документами, що допомагає у процесах розробки, як редагування, компіляція тощо. буд. Основна особливість цієї IDE у тому, що вона забезпечує створення форм під час розробки. Отже, це економить час та сприяє швидкому розвитку. Допомагає в налаштуванні зовнішнього вигляду та поведінки програми за допомогою різних макросів.

Продовження таблиці 1.3

Продуктивність, сумісність та інші фактори.	Нижче наведені фактори, які відіграють важливу роль у продуктивності Java: 1) Java є мовою, що інтерпретується, і, отже, код не перетворюється на машинну мову до тих пір, поки не буде виконаний. 2) LINQ недоступний з JAVA 3) Java 7 може використовувати оператор switch для рядкової змінної, але не для більш старих платформ	Нижче наведені фактори, які відіграють важливу роль у продуктивності .NET: 1) .Net компілюється та запускається в операційній системі, в якій розгорнуть код. 2) LINQ дозволяє користувачеві писати запити безпосередньо замість використання процедур, що зберігаються. 3) C # дозволяє використовувати оператор switch для рядкової змінної. 4) C# підтримує підтримку родових даних

Продовження таблиці 1.3

Продовження таблиці 1.3

Переваги та недоліки. Збіжності	Java є мовою переноситься і, отже, він може бути запущений в будь-якій операційній системі. Ви можете легко створити програму або сайт на будь-якій платформі. Java зазвичай викликає базу даних рідше, оскільки всі її доменні об'єкти вже є. Багато інтегрованих роз'ємів доступні для Java, і вони легко доступні через Інтернет. Крім того, код, написаний на Java, обернено сумісний і може бути легко перенесений з однієї операційної системи в іншу. Одним із недоліків Java є безпека. Безпека здебільшого ставиться під загрозу у рішеннях, які залежать від платформи. Крім того, Scala несумісна з Groovy і, отже, потребує покращених навичок розробки. Крім того, мультиплатформенність робить Java особливою, з іншого боку, вона робить Java повільнішим.	Microsoft надає повне уніфіковане середовище як 1) .Net також надає платформу .Net. Він забезпечує повну яке надає кошти для збирання масштабованість, яка надає бізнес-логіки та даних. всі інструменти та IDE через 2) Це також забезпечує кращу мережу Microsoft. платформу як платформу та Недоліком .Net є те, що забезпечує автоматичний збір підтримка доступна лише сміття. платформи Entity. Це обмежує об'єктно-реляційну підтримку. Керований код працює повільніше, ніж власний код у цьому середовищі. Розробники повинні залежати від Microsoft у всіх оновленнях, функціях та її покращеннях.
	1) Java має тривіневу архітектуру, яка дозволяє розробникам створювати багаторівневі веб-програми. 2) Це одна з найкращих мов для корпоративної веб-розробки.	

Java і .Net часто йдуть пліч-о-пліч у світі програмування. Платформи Java та .Net виконують подібні функції. У той час як Java та .NET часто називають лише фреймворками.

Основні відмінності між Java та .Net пояснюються в наступних пунктах:

- Java переважно мова програмування, а .Net - це фреймворк. Java можна розглядати як платформу із такими середовищами JE, як J2EE. JEE і т. д. У нього є середовище виконання Java, в якій є JVM, як компілятор часу і засіб запису байт-коду. Це мова з численними бібліотеками, які можна використовувати у будь-якій операційній системі. .Net, з іншого боку, використовує операційну систему Windows, корпоративні сервери Windows. Його фреймворк складається з Common Language Runtime, бібліотек класів фреймворку та ASP.NET. У цього також є власні послуги будівельного блоку і візуальна студія.
- .Net має власну платформу Microsoft, яка буде безпечнішою, ніж будь-яка платформа з відкритим вихідним кодом. Безпека є серйозною проблемою, яку легко розв'язують розробники Asp.Net як закрита платформа. Для Java через брак ресурсів може виникнути проблема безпеки. Common Language Runtime .Net краще, ніж JVM, оскільки JVM просто перетворює код на байт-код для базових операційних систем. Це не призначено для обчислювальних машин і трохи повільніше, ніж CLR. Java має багато IDE для свого програмування, таких як Eclipse, NetBeans і т.д. Net, з іншого боку, має Visual Studio.

1.4.1 Визначеня з основною СУБД

Різниця між SQL Server та PostgreSQL:

Обидва ці типи SQL можуть виникнути питання про те, в чому різниця між SQL Server і PostgreSQL. Microsoft SQL Server - це система управління та аналізу

баз даних, яка в основному використовується для електронної комерції, бізнесу та різних рішень для сховищ даних. PostgreSQL, з іншого боку, є вдосконаленою системою управління об'єктно-реляційними базами даних, яка забезпечує підтримку розширеного підмножини стандартів SQL, включаючи різні транзакції, зовнішні ключі, підзапити, тригери та різні типи і функції користувача.

Обидва SQL Server та PostgreSQL є популярним вибором на ринку; Давайте обговоримо деякі основні відмінності між SQL Server і PostgreSQL:

Підтримка CSV: Postgres знаходиться на вершині гри, коли справа доходить до підтримки CSV. Він надає різні команди, такі як "копіювати в" та "копіювати з", які допомагають у швидкій обробці даних. Він також надає корисні повідомлення про помилки. Якщо при імпорті виникне невелика проблема, він видасть помилку та зупинить імпорт. З іншого боку, сервер SQL не підтримує ні імпорт даних, ні їх експорт.

Кросплатформність. У сучасному світі кросплатформність мови або будь-якої програми справді дуже важлива. З появою технологій та IT-індустрії стало дуже важливо бути доступним через Linux та UNIX, оскільки вони є системами з відкритим вихідним кодом. Сервер SQL - це продукт Microsoft, заблокований постачальником, і його можна запускати лише у системах Microsoft. Postgres можна використовувати в Linux, BSD, Solaris, а також Windows. Особливості процедурної мови: PL/PGSQL – це рідна процедурна мова, що надається Postgres, яка має різні сучасні функції. Він підтримує тип даних JSON і, отже, має максимальну потужність і безліч гнучких можливостей, включених в один пакет. На додаток до цього користувач може легко використовувати Python, Perl, R, Java, PHP з SQL, оскільки вони підтримуються як процедурні мови в Postgres. MS SQL сервер також забезпечує підтримку процедурної мови як інтегровану функцію. Але ця функція є трохи брудна, повільна і має погані функції. Він також має невеликі помилки з різними помилками, які завжди відбуваються.

Регулярні висловлювання: Postgres надає величезну кількість регулярних виразів як основу для аналітичної роботи. SQL-сервер, з іншого боку, має подібну підрядок patindex, яка не така хороша в порівнянні з висловлюваннями, наданими Postgres.

Таблиця 1.4

Основа порівняння SQL Server із PostgreSQL	SQL Server	PostgreSQL
Основна різниця	SQL-сервер - це система управління базами даних, яка переважно використовується для електронної комерції та надає різні рішення для сховищ даних.	PostgreSQL - це вдосконалена версія SQL, яка забезпечує підтримку різних функцій SQL, таких як зовнішні ключі, підзапити, тригери та різні типи і функції користувача.

Продовження таблиці 1.4

Оновлювані види	Подання можуть бути оновлені, навіть якщо поновлено 2 представлення таблиці. Якщо таблиці мають різні ключі і оператор оновлення не включає більше однієї таблиці, він буде оновлений автоматично. Користувач також може	Подання PostgreSQL можуть оновлюватися, але не автоматично, на відміну від сервера SQL. Користувач повинен написати правила для різних уявлень, щоб оновити їх. Крім того, складні види можуть бути легко створені.

	використовувати тригери для оновлення складних уявлень.	
Обчислювані стовпці	Сервер SQL надає стовпці, що обчислюються, але уявлення переважно обчислюваних стовпців. Обчислювані стовпці мають дуже обмежене використання, оскільки вони не здатні утримувати різні згортки.	PostgreSQL не надає стовпці, що обчислюються. PostgreSQL, з іншого боку, має функціональні індекси, які працюють просто як уявлення.

Продовження таблиці 1.4

Копіювання	SQL-сервер може реплікувати всі види даних. Це може бути доставка журналів, дзеркальне відображення, моментальні знімки, транзакції та злиття тощо. буд., і навіть можуть мати передплатників, не заснованих на Windows Server.	Реплікація в Postgres виконується у формі звітів і має бути найменш полірованою. Хоча є різні сторонні варіанти на вибір з тих, які є безкоштовними і не безкоштовними. PostgreSQL 8.4 або пізніша версія може мати вбудовану функцію реплікації.
Підтримка збережених процедур та збережених функцій різними мовами	SQL-сервер підтримує цю функцію. Це може бути зроблено з будь-якою мовою, яка відповідає CLR, таким як VB, C #, Python і т. д. Щоб це зробити успішно, користувач спочатку повинен спочатку скомпілювати код у все.	Тут немає потреби спочатку створювати нудно. Користувач, який створив код може легко побачити, що робить код. Нестача сервера повинна містити мову, яка використовується середовищем.

Продовження таблиці 1.4

Динамические действия в SQL	SQL сервер не підтримує цю функцію. Але замість цього користувач може використовувати процедуру, що зберігається, і викликати її з операторів вибору, так що це набагато більше обмежує, ніж PostgreSQL.	PostgreSQL дійсно надає цю можливість, і, просто використовуючи оператори вибору, користувач може виконувати практично всі операції, легко витягувати та виконувати всі інші завдання.
Матеріалізовані уявлення	Так, він надає можливості для запуску матеріалізованих уявлень. Однак, функціонування залежить від того, де виконується запит. Це може бути SQL Express, робоча група тощо.	Postgres не дає можливості для запуску матеріалізованих уявлень. Натомість у них є модуль з ім'ям mat views, який допомагає розбудовувати будь-які матеріалізовані уявлення.
Чутливість до регістру	За замовчуванням SQL-сервер вважається нечутливим до регістру, але якщо користувач хоче змінити те саме, він може зробити це, опустившись до рівня стовпця.	За замовчуванням PostgreSQL чутливий до регістру, і його важко зробити нечутливим. До нього можуть бути внесені зміни, але вони не розкриті і не відповідають ANSI

1.4.2 Технологія для створення графічного інтерфейсу

Щоб створити GUI програми в Microsoft .NET, потрібно використовувати Windows Forms. Windows Forms – новий стиль побудови програми на базі класів .NET Framework class library. Вони мають власну модель програмування, яка більш досконала, ніж моделі, що базуються на Win32 API або MFC, і вони виконуються в керованому середовищі .NET Common Language Runtime (CLR). Ця стаття дає уявлення про те, що таке Windows Forms, розглядаючи її від моделі програмування до Microsoft Intermediate Language та JIT-транслятора.

Ви вже багато чули, що Microsoft .NET – нова платформа, яка базується на Windows. Це ціла нова парадигма програмування, яка змінить шлях, яким ви думаєте про написання програм для Windows. Вона реалізована на бібліотеці класів .NET Framework class library та містить більш єдину модель програмування, покращений захист та багатші можливості для написання повнофункціональних веб-додатків. І це тільки початок.

Windows Forms – одна з найцікавіших можливостей Microsoft .NET. Якщо ви знайомі з MFC (або Windows API), то Windows Forms є гарним початком для роботи з .NET Framework class library, тому що вона дозволяє писати традиційні GUI додатки з вікнами, формами і т.п. речами. Одного разу, почавши працювати з Windows Forms, ви зможете швидко зрозуміти .NET Framework.

Головна вигода від написання Windows-додатків з використанням Windows Forms - це те, що Windows Forms гомогенізують (створюють однорідніщу (гомогеннішу) структуру) програмну модель і усувають багато помилок і протиріч від використання Windows API. Наприклад, кожен досвідчений програміст під Windows знає, що деякі стилі вікна можуть застосовуватися лише до вікна, коли вже створено. Windows Forms значною мірою усувають таку суперечність. Якщо ви хочете існуючому вікну встановити стиль, який може бути присвоєний тільки в момент створення вікна, то Windows Forms спокійно знищить вікно і знову створить його із зазначеним стилем. Крім того, .NET Framework class library набагато багатший, ніж Windows API, і коли

ви писатимете програми, використовуючи Windows Forms, ви отримаєте в розпорядження більше можливостей. Написання програми за допомогою Windows Forms потребує менше коду, ніж програми, які використовують Windows API або MFC.

Інша вигода від Windows Forms - ви використовуєте той самий API, незалежно від мови програмування, яку ви вибрали. У минулому вибір мови програмування керував вибором API. Якщо ви програмували у Visual Basic, ви використовували один API (реалізований мовою Visual Basic), тоді як програмісти C використовували Win32 API, а програмісти C++, взагалі кажучи, використовували MFC. MFC-програмісту було важко перейти на Visual Basic і навпаки. Але тепер такого немає. Всі програми, які використовують Windows Forms, використовують один API із .NET Framework class library. Знання одного API достатньо дозволить програмісту писати програми фактично будь-якою мовою, яку він вибере.

Windows Forms не менше, ніж сучасна модель програмування для GUI додатків. На відміну від моделі програмування Win32, в якій багато що йде ще від Windows 1.0, нова модель була розроблена з урахуванням усіх сучасних вимог. Ціль цієї статті полягає в тому, щоб познайомити читача з моделлю програмування Windows Forms. Щоб компілювати та виконувати приклади коду, наведеного далі, на вашому комп'ютері має бути встановлений пакет Microsoft .NET Framework SDK (.NET Framework SDK Beta 1 доступний на сайті Microsoft).

Модель програмування Windows Forms:

У Windows Forms термін "форма" – синонім вікна верхнього рівня. Головне вікно програми – форма. Будь-які інші вікна верхнього рівня, які має програму - також форми. Вікна діалогу також вважаються формами. Незважаючи на назву, програми, які використовують Windows Forms, не виглядають як форми. Подібно

до традиційних Windows-додатків програми здійснюють повний контроль над подіями у власних вікнах.

Програмісти бачать Microsoft .NET через лінзу .NET Framework class library. Уявіть MFC на порядок більше і ви отримаєте точну картину про ширину та глибину .NET Framework class library. Щоб полегшити протиріччя в позначеннях і надати організацію багатьом сотням класів, .NET Framework class library розбита на ієрархічні розділи на ім'я. Кореневий розділ, System, визначає фундаментальні типи даних, що використовуються всіма програмами .NET.

Програми, які використовують Windows Forms, використовують класи System.WinForms. Цей розділ включає такі класи, як Form, який моделює поведінку вікон чи форм; Menu, яке представляє меню; Clipboard, який дозволяє програмам Windows Forms використовувати буфер обміну. Він також містить численні класи, що надають засоби керування, наприклад: Button, TextBox, ListView, MonthCalendar і т.д. Ці класи можуть бути включені в додаток або з використанням лише імені класу або з використанням повного імені, наприклад: System.WinForms.Button.

В основі майже кожної програми, написаної із застосуванням Windows Forms, - похідний клас від System.WinForms.Form. Зразок цього класу представляє головне вікно програми. System.WinForms.Form має безліч властивостей та методів, які мають багатий програмний інтерфейс до форм. Бажаєте знати розміри клієнтської області форми? У Windows ви б викликали функцію API GetClientRect. У Windows Forms потрібно використовувати властивості ClientRectangle або ClientSize.

Програми, засновані на Windows Forms, які використовують кнопки, списки та інші типи компонентів Windows, використовують класи керування System.WinForms, які значно спрощують програмування керування. Бажаєте створити стилізовану кнопку із зображенням у вигляді фону? Немає проблем. Увімкніть потрібне зображення в об'єкт System.Drawing.Bitmap та призначте

його властивості кнопки BackgroundImage. Як щодо керування кольором? Ви коли-небудь намагалися налаштувати колір тла текстового поля? У Windows Forms це просто: потрібно просто привласнити колір властивості BackColor, все відстальне система зробить сама.

Інший важливий "будівельний" блок програми, який використовує Windows Forms - клас System.WinForms на ім'я Application. Цей клас містить статичний метод Run, який завантажує програму та відображає вікно.

Ви скажете: якщо програми, які є Windows Forms, не обробляють повідомлення, як вони відповідають на введення користувача або знають коли малювати? Багато класів мають віртуальні методи, які можна перевизначити... Наприклад, System.WinForms.Form містить віртуальний метод OnPaint, який викликається, коли клієнтська область форми потребує оновлення. OnPaint - один із багатьох віртуальних методів, який можна перевизначити у похідному класі для формування інтерактивних форм.

Інша важлива грань моделі програмування Windows Forms - механізм, який форми використовують для відповіді на введення в меню, засобів керування та інших елементів GUI програми. Традиційні Windows-програми обробляють повідомлення WM_COMMAND та WM_NOTIFY використовуючи події процесу Windows Forms. У C# та іншими мовами, які підтримують .NET Common Language Runtime (CLR), події - члени типу першого класу нарівні з методами, полями та властивостями. Фактично всі керуючі класи (control classes) Windows Forms (а також багато некеруючих класів) створюють події. Наприклад, кнопка (примірник System.WinForms.Button) після натискання створює подію Click.

Висновки до першого розділу

Наразі в нашій країні існує багато медичних інформаційних систем, які використовуються в багатьох медичних закладах. Велика кількість лікарів використовують такі системи в своїй роботі, що значно економить їх час та дозволяє приділяти набагато більше уваги пацієнтам. Серед існуючих медичних інформаційних систем багато лікарів зможуть знайти таку, яка буде відповідати їх вимогам.

Серед проаналізованих систем варто звернути увагу на медичну інформаційну систему «Helsi». Згідно наведеного аналізу, ця система являється ідеальним варіантом для лікарів первинної медицини.

Але все ще є велика кількість лікарів, які не мають змоги використовувати МІС під час своєї роботи через наступні причини:

- Відсутність обладнання (ПК, доступ до інтернету)
- Відсутність універсальних МІС для всіх галузів медицини
- Складність налаштування існуючих МІС
- Велика вартість ПЗ

Враховуючи вищесказане, потрібно або допрацьовувати існуючі МІС, або створити нове ПЗ, яке буде виконувати функції МІС в тих галузях медицини, для яких сучасні системи наразі непридатні.

РОЗДІЛ 2.
Моделювання та розробка архітектури МІС

2.1 Моделювання та нормалізація БД

Однією з найголовніших частин будь якого ПЗ є данні. Адже саме для маніпулювання даними було створене програмування і знаючи не можна не приділити багато уваги розробці правильного методу зберігання та обробки даних.

2.2.1 Перша нормальна форма

Відношення знаходиться в першій нормальній формі (скорочено 1НФ), якщо всі його атрибути атомарні, тобто якщо жоден з його атрибутів не можна розділити на простіші атрибути, які відповідають якимось іншим властивостям сутності, що описується.

Називатимемо вихідне відношення основним, а значення неатомарного атрибуту - підлеглим.

Щоб нормалізувати вихідне ставлення, атрибути якого неатомарні, необхідно поєднати схеми основного і підлеглого відносин. Крім того, якщо, наприклад, таблиця, що відповідає ненормалізованому відношенню, вже міститься в БД і заповнена інформацією, завдання ускладнюється тим, що значення неатомарного атрибуту може у свою чергу містити кілька кортежів.

Слід пояснити сказане з прикладу. Розглянемо ставлення, що має атрибути Код співробітника, ПІБ, Посада, Проекти. Очевидно, що один працівник може

працювати над кількома проектами. Припустимо, що проект описується ідентифікатором, назвою та датою здачі.

Легко помітити, що не всі атрибути цього відношення є атомарними (неподільними). Зокрема, атрибут «Проекти» можна розділити на три простіші атрибути: «Код проекту», «Назва», «Дата здачі», а значення цього атрибуту для співробітника Іван Іванович Іванов містить кілька кортежів — інформацію про три проекти.

Примітка: з певної точки зору атрибут ПІБ можна також вважати неатомарним і в такому випадку його також слід розділити на простіші, як Прізвище, Ім'я, По батькові.

Тепер настав час розглянути алгоритм нормалізації відношення до 1НФ.

1. Створити нове відношення, схема якого буде отримана шляхом злиття основної та підлеглої схем вихідного відношення до однієї.
2. Для кожного кортежу вихідного відношення включити в нове стільки рядків, скільки кортежів міститься у підпорядкованому відношенні цього кортежу.
3. Заповнити значення атрибутів нового відношення, які відповідають атрибутам підпорядкованого відношення.
4. Заповнити рядки нового відношення значення атомних атрибутів вихідного.

Застосуємо цей алгоритм до наведеного вище відношення. Схема нового відношення складатиметься з 6 атрибутів: "Код співробітника", "ПІБ", "Посада", "Код проекту", "Назва", "Дата здачі". Для одного єдиного кортежу заданого відношення, додамо в нове три рядки, по одному для кожного проекту (за кількістю кортежів у підпорядкованому відношенні). Тепер можна заповнити значення розділених атрибутів кортежами із підлеглого відношення. Потім перенесемо в кожен із цих рядків значення атомарних атрибутів: «Код

співробітника», «ПІБ», «Посада» (як Ви вже здогадалися, всі три рядки будуть містити однакові значення цих атрибутів).

2.2.2 Друга нормальна форма

Мета приведення до першої нормальної форми (1NF) - дати можливість побудови висловлювань даних за допомогою формул логіки предикатів першого порядку. Насправді це дає можливість побудови запитів вибірки за умовами, оскільки значення мають однакову природу.

Наприклад, якщо у відносини 'Сім'я' є атрибут 'Діти', ми можемо легко порівняти два рядки 'Вася' та 'Аня' і визначити їх лексикографічний порядок. Але порівнювати рядки 'Вася' і 'Аня, Саша' та визначати їхній порядок безглуздо, а спроба вирішити цю проблему без декомпозиції, призведе або до запровадження правил порівняння скалярів і списків, або до ускладнення мови формул. Логіки предикатів першого порядку тут замало. Тому 1NF вимагає, щоб усі значення були простими значеннями домену.

Тобто перша НФ має справу зі структурою значень записів.

Друга (і третя) НФ має справу вже з ключами та залежностями у схемі. Перелічимо її цілі із поясненнями.

Головною метою приведення до другої нормальної форми є бажання позбутися надмірності зберігання даних і як наслідок уникнути аномалій модифікації цих даних (аномалій зміни, вставки та видалення)

Другий по порядку, але не за значенням, метою нормалізації в 2NF є максимально розбити модель даних на окремі відносини, щоб їх можна було комбінувати та використовувати у виразах новими, не передбаченими спочатку способами.

Мінімізувати зусилля щодо зміни схеми у разі потреби. Чим менше залежностей усередині схеми, тим менше змін у ній буде потрібно при зміні моделі даних.

Зрозумілість схеми для користувача. Чим тримати всі дані в одній великій таблиці, простіше уявити дані як кілька пов'язаних та логічно розділених відносин. Це простіше читати, сприймати, проектувати та підтримувати. Зрештою, будь-яка модель даних починається на дошці чи папері у вигляді кружечків, блоків та ліній, які так люблять малювати діти та програмісти.

2.2.3 Третя нормальна форма

Перед тим як переходити до процесу приведення таблиць бази даних до третьої нормальної форми, необхідно, щоб ці таблиці вже знаходилися у другій нормальній формі, докладно процес приведення таблиць бази даних до другої нормальної форми, а також усі вимоги, що пред'являються до другої нормальної форми, ми розглядали у попередній статті – друга нормальна форма (2NF).

Після того як таблиці бази даних знаходяться у другій нормальній формі, ми можемо починати наводити базу даних до третьої нормальної форми та розглядати відповідні вимоги.

Вимоги третьої нормальної форми (3NF)

Вимога третьої нормальної форми (3NF) у тому, щоб у таблицях була відсутня транзитивна залежність.

Транзитивна залежність – це коли неключові стовпці залежить від значень інших неключових стовпців.

Якщо у першій нормальній формі нашу увагу було націлено на дотримання реляційних принципів, у другій нормальній формі в центрі нашої

уваги був первинний ключ, то у третій нормальній формі вся наша увага приділена стовпцям, які є первинним ключем, тобто. неключовим стовпцям.

Щоб нормалізувати базу даних до третьої нормальної форми, необхідно зробити так, щоб у таблицях були відсутні неключові стовпці, які залежать від інших неключових стовпців.

Іншими словами, неключові стовпці нічого не винні намагатися грати роль ключа в таблиці, тобто. вони дійсно повинні бути неключовими стовпцями, такі стовпці не дають можливості отримати дані з інших стовпців, вони дають можливість подивитися на інформацію, що міститься в них, тому що в цьому їх призначення.

Головне правило третьої нормальної форми (3NF) звучить так:

Таблиця має містити правильні неключові стовпці

2.2 Моделювання сучасного та зручного графічного інтерфейсу

UX це User Experience (дослівно: «досвід користувача»). Тобто це те, який досвід/враження отримує користувач з вашого інтерфейсу. Чи вдається йому досягти мети і наскільки просто чи складно це зробити.

A UI — це User Interface (дослівно «інтерфейс користувача») — те, як виглядає інтерфейс і те, які фізичні характеристики набуває. Визначає, якого кольору буде ваш «виріб», чи зручно людині потраплятиме пальцем у кнопочки, чи читатиме текст і тому подібне…

UX/UI дизайн — це проектування будь-яких інтерфейсів, в яких зручність використання так само важлива як і зовнішній вигляд.

Що таке UX та UI дизайн, тобто

Прямий обов'язок дизайнера UX/UI — це, наприклад, «продати» товар чи послугу через інтерфейс. Саме на основі роботи UX/UI дизайнера користувач

приймає рішення: "Бути чи не бути?" Подобається чи не подобається. Купити чи не купити.

Насправді цілі дизайнера можуть бути різними. Не обов'язково щось продавати. Але я спеціально не хочу використовувати надто абстрактні фрази, щоб цей текст був зрозумілий новачкам; щоб стиль викладу не перетворився на хрестоматію з мови програмування зразка 90-х років.

UX/UI дизайн не відноситься лише до смартфонів та веб-сайтів. Більше того, професія UX/UI дизайнера існувала з давніх-давен. Просто раніше вона так не називалася. Точніше раніше вона взагалі ніяк не називалася, а була частиною інших професій.

UX/UI дизайн на прикладі точильного каміння

Отож, коли винахідник чергового точильного каменю думав:

чи буде він сидіти і сам натискати педаль або він спростить механізм, але приставить раба який розкручуватиме колесо рукою, то на той момент він був UX дизайнером.

А та людина, яка думала, якої величини буде камінь, якого кольору вибрати дерево для підставки і чим скріпити дерев'яні жерди (цвяхами чи шкіряними джгутами?) і якою довжиною буде ручка, був UI дизайнером.

І той спосіб, яким би ви заточували меч, називався б інтерфейс.

Різниця між UX і UI в тому, що UX дизайнер планує те, як ви будете взаємодіяти з інтерфейсом і які кроки вам потрібно зробити, щоб зробити щось. А UI дизайнер вигадує, як кожен із цих кроків буде виглядати. Як бачите з прикладів вище, UX та UI так тісно пов'язані, що іноді грань між поняттями змивається. Тому і UX, і UI зазвичай займається один дизайнер та його професія пишеться через.

UI/UX дизайн - це зараз одна з найпопулярніших професій у цифровій індустрії. Скільки часу вона буде затребувана, залежить від розвитку цієї галузі. І, зважаючи на все, вона тільки набирає обертів.

UX та UI – це не тренди. Технологи розвиваються. Попит на сайти зростає. Цифрові програми з'являються як гриби. А інструменти дизайну та розробки спрощуються настільки, що вже практично будь-яка людина без знання програмування може "на коліні" зробити сайт-візитку. Ось тільки цей сайт має якось виглядати. І не просто як абстрактний каркас із тексту та кнопок. Тут програмістам і потрібна допомога дизайнера UX/UI.

Поділ на веб-дизайнерів та UX/UI дизайнерів з'явився з розвитком інтернету. Згодом знадобилися вужчі фахівці, які робили б інтерфейси саме для веб-сайтів.

Так, UI/UX дизайн — це більш широке та ємне поняття, ніж веб-дизайн.

Висновки до другого розділу

У цьому розділі було висвітлено два наважливих аспекта розробки ПЗ: створення швидкої та гнучкої бази даних та розробка сучасного і приємного інтерфейсу користувача.

Ми виявили важливі властивості даних, а саме:

- Перша нормальна форма (1НФ, 1NF) це властивість відношення у реляційній баз даних. Відношення знаходиться в першій нормальній формі тоді і тільки тоді, коли домен кожного атрибута містить лише нероздільні значення, а значення кожного атрибута містить лише одне значення з цього домену. Перша нормальна форма є істотною властивістю відношення у реляційній базі даних. Нормалізація баз даних - це процес представлення бази

даних з точки зору відносин у стандартних нормальних формах, де перша нормальна є мінімальною вимогою.

- Друга нормальна форма (2НФ, 2NF) — нормальна форма, що використовується для нормалізації баз даних. 2НФ первісно була визначена 1971 року Едгаром Коддом. Щоб перебувати в другій нормальній формі, таблиця, що перебуває в першій нормальній формі, має відповідати додатковим критеріям. А саме: 1НФ таблиця перебуватиме в 2НФ тоді й лише тоді, коли для будь-якого потенційного ключа K і будь-якого атрибута A, який не є частиною потенційного ключа, A залежить саме від цілого потенційного ключа, а не від його частини. Тобто, 1НФ таблиця перебуває в 2НФ тоді й тільки тоді, коли всі її неключові атрибути функціонально залежні від потенційного ключа в цілому. У разі, якщо 1НФ таблиця не має складних потенційних ключів (таких, що складаються більш ніж з одного атрибута), тоді вона автоматично перебуватиме в 2НФ.
- Третя нормальна форма (3НФ) — нормальна форма використовна в нормалізації баз даних. 3НФ первісно була визначена 1971 року Едгаром Коддом. За Коддом таблиця знаходиться в 3НФ тоді й лише тоді, коли виконуються наступні умови: відношення R (таблиця) знаходиться в 2НФ, кожен неключовий атрибут відношення R нетранзитивно (безпосередньо) залежить від кожного потенційного ключа в R. Неключовий атрибут R — атрибут, що не є частиною будь-якого потенційного ключа. Транзитивною називають таку функціональну залежність, в якій X → Z (X визначає Z) непрямо, а через X → Y і Y → Z (і неправильно, що Y → X).

Та також встановили важливість підходу з використанням UX/UI для розробки сучасних додатків

UX - це взаємодія користувача з продуктом. Це про те, як зробити так, щоб кожен відкрив програму і відразу розібрався, як вона працює.

Будь-який проект починається з опрацювання UX-частини. На цьому етапі дизайнери досконально студіюють продукти конкурентів, визначають потреби користувачів, їх проблеми, а потім приходять до розуміння, як найкраще їх вирішити. Такі UX-дослідження дозволяють продумати продукт, створити та протестувати прототипи та відмовитися від свідомо неробочих рішень.

UI це про те, як продукт виглядає, яким його сприймає користувач. Головне завдання — створити інтерфейс, який виділятиметься на фоні конкурентів. Дивлячись на який користувач подумає: «Млинець, хто так круто зробив?» Для цього в хід йдуть кольори, типографіка, інфографіка та анімація.

На етапі створення UI дизайнер також може працювати над логотипом та брендингом (фірмовим стилем) — усім, що так чи інакше відображає цілісність продукту, його позиціонування. UI прототипування, анімація та адаптивність - це ті аспекти, які забезпечують комфортну взаємодію з продуктом на будь-яких девайсах.

Виходить, що різниця між UI та UX полягає у робочих процесах. А ціль одна.

РОЗДІЛ 3.
ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Програмна реалізація функціональності управління звітами

Так як одна з основних можливостей MIC це манипулювання звітами багато часу було преділено саме розробці цього функціоналу і додано понад 30 видів репортів.

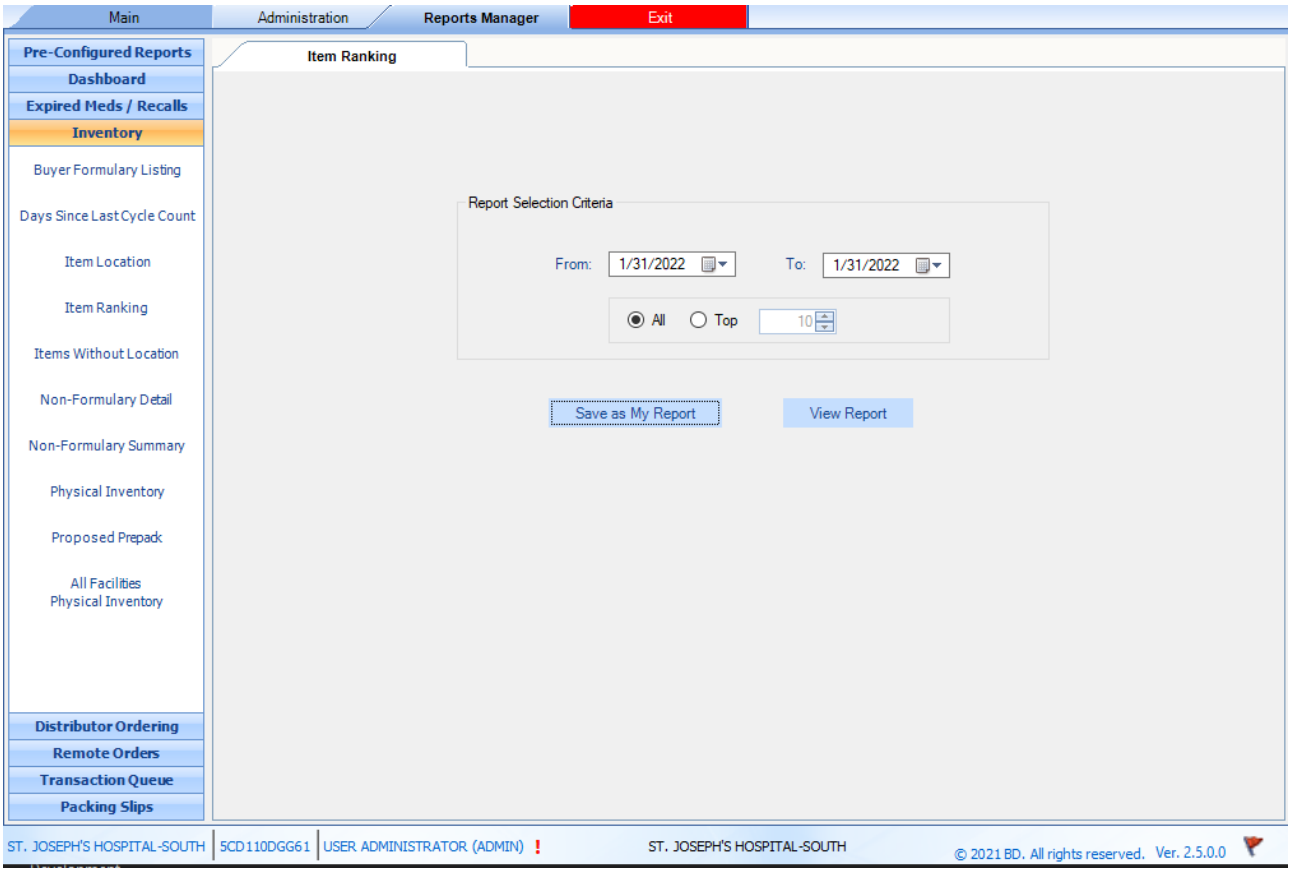

Рис 3.1 Форма з вибором бажаного виду звіту

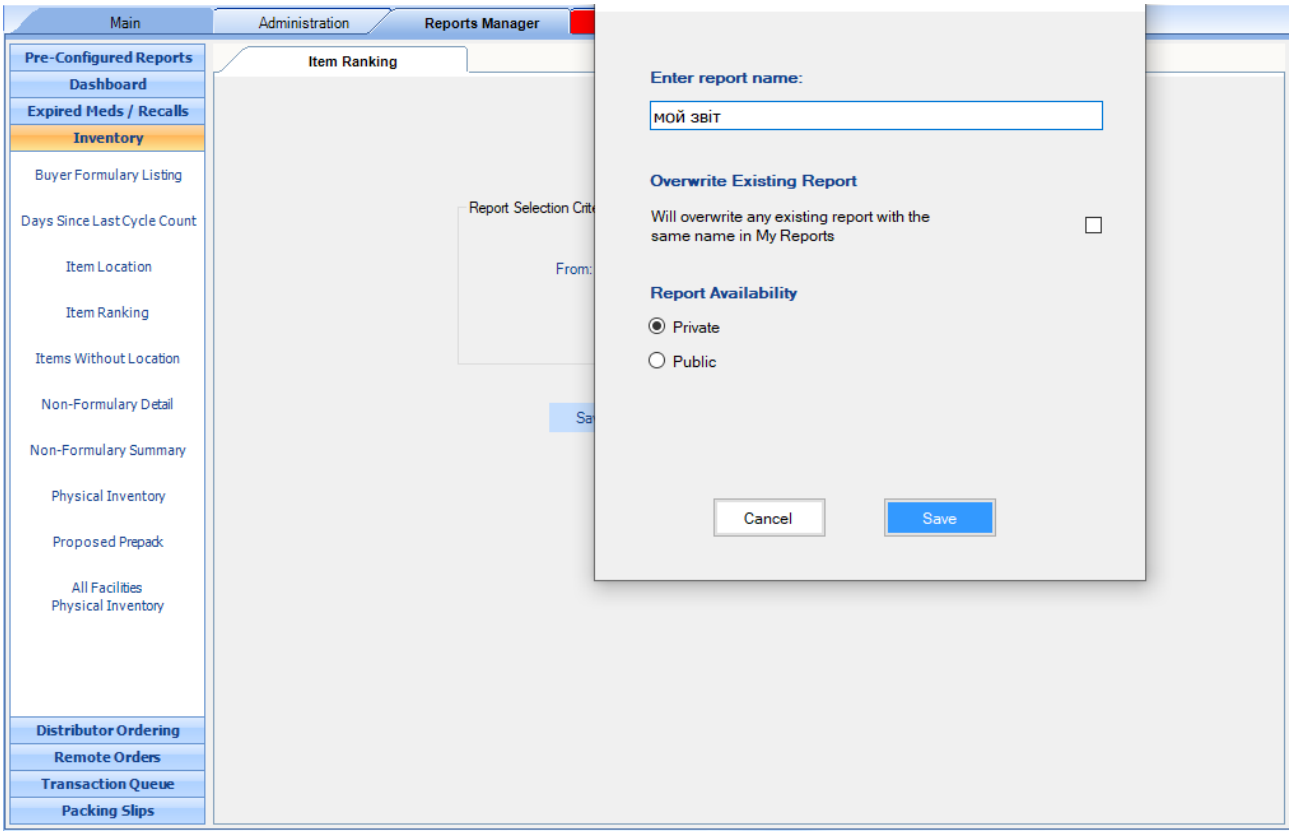

Рис 3.2 Вікно з можливістю створення і збереження нового звіту

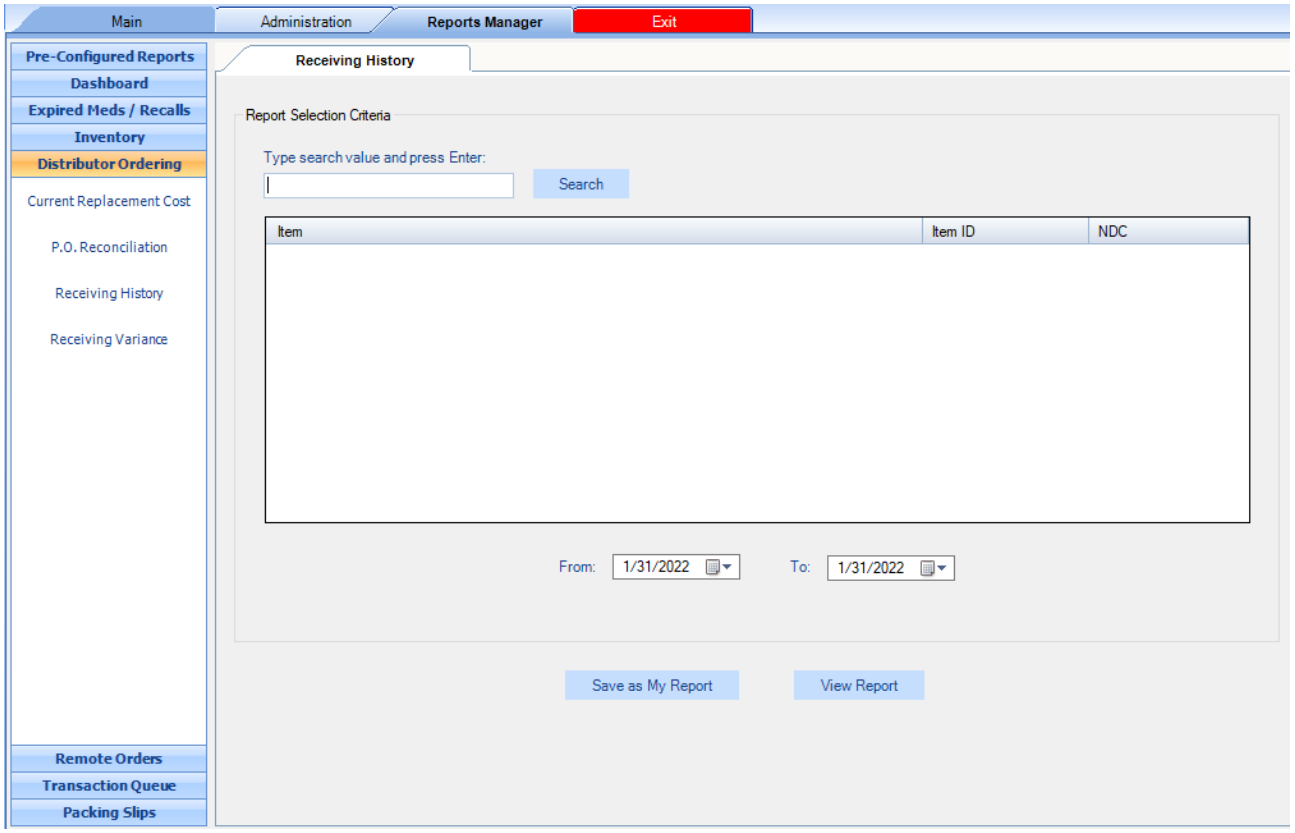

Рис 3.3 Сторінка з можливістю додавання звіту про постановку нових ліків

3.2 Опис основного функціоналу додатку

3.2.1 Головна сторінка

На головній сторіцні роспологається найважливіші і найвживаніші посилання, такі як: вибір ліків, повернення ліків, список усіх повернень, тощо.

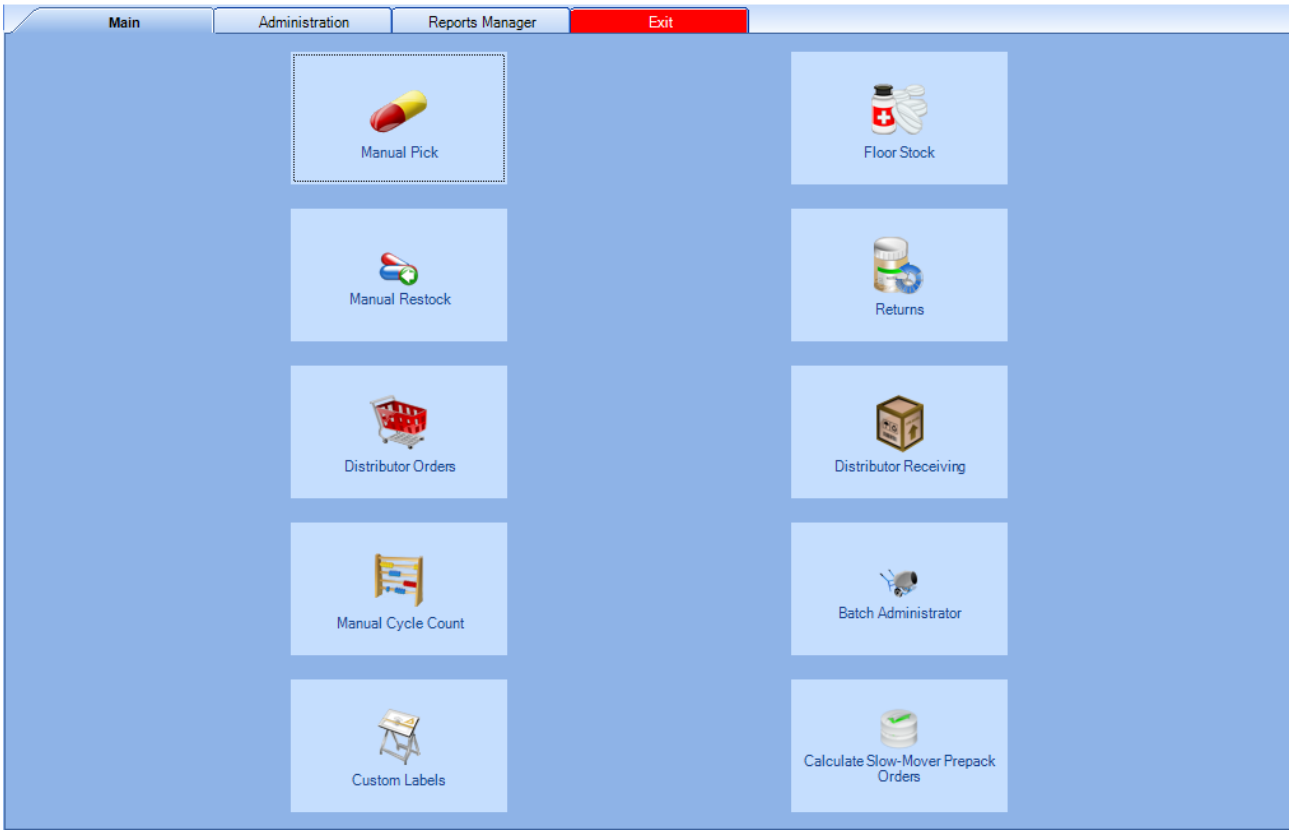

Рис 3.5 Головна сторінка

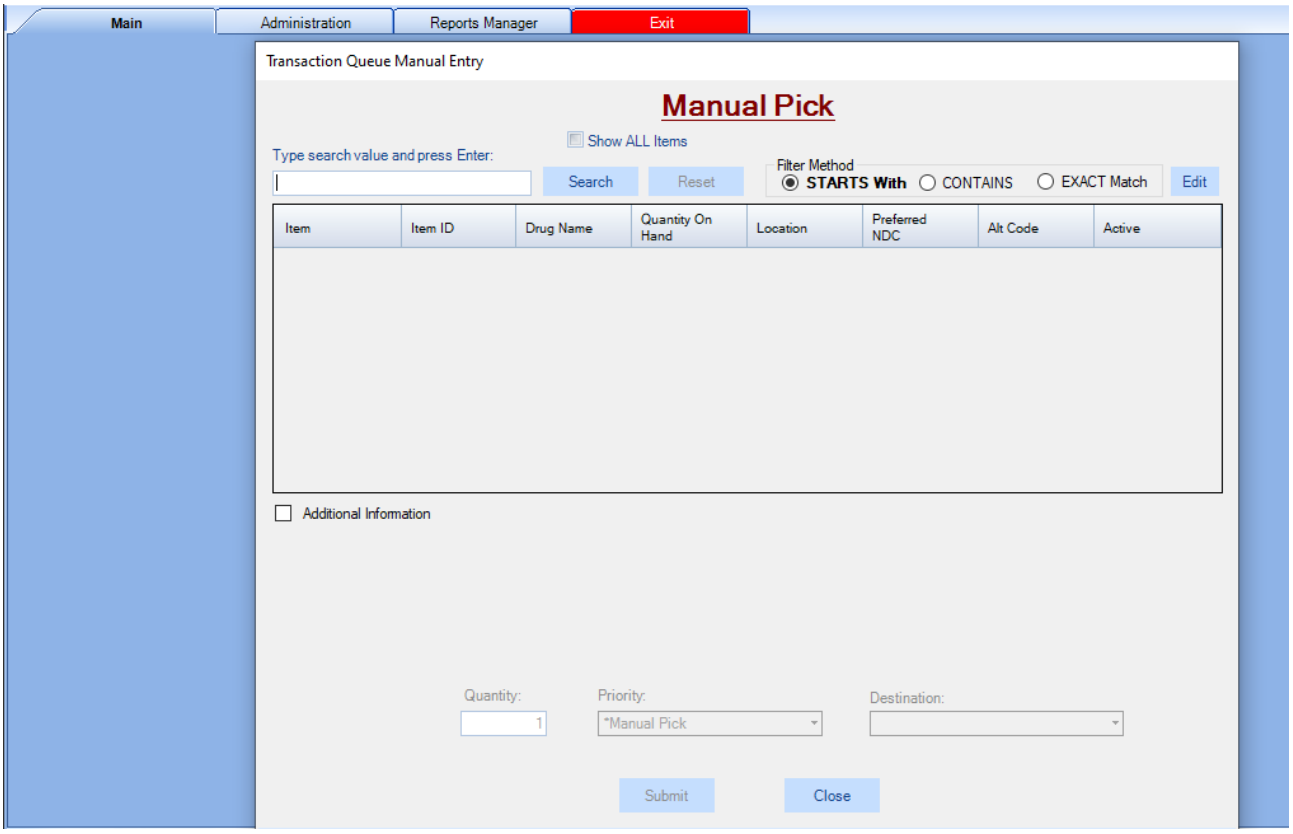

Рис 3.6 Форма ручного вибору ліків

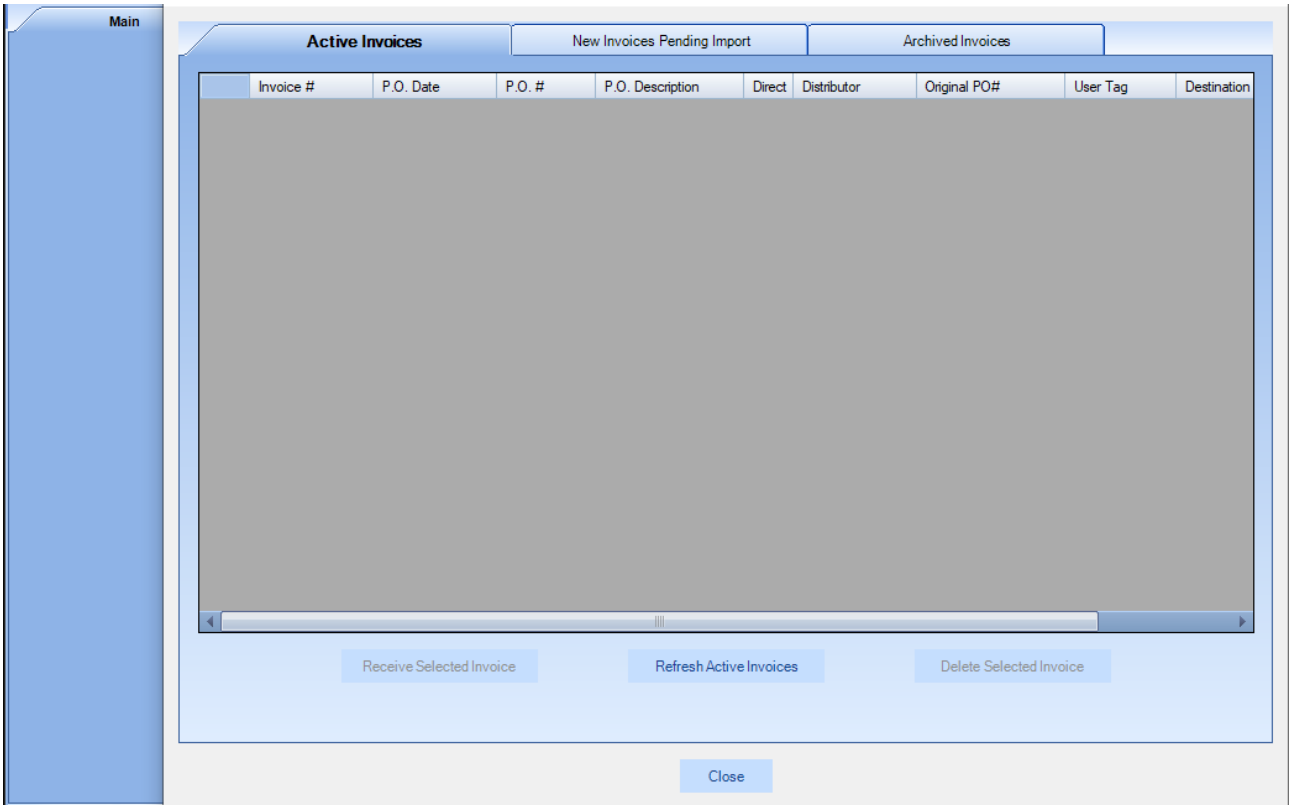

Рис 3.7 Форма для відображення заказів на ліки, медичне обладнання, тощо.

3.2.2 Можливість авторизації у додаток та система ролей

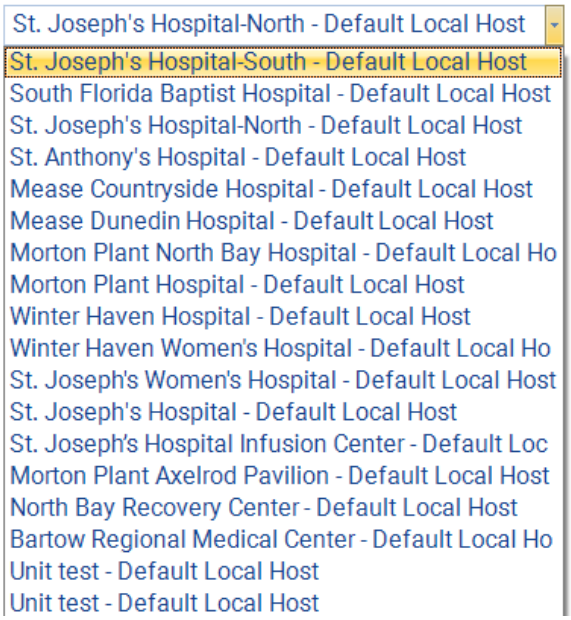

Рис 3.7 Можливість вибору певного медичного закладу з одного конгломерату

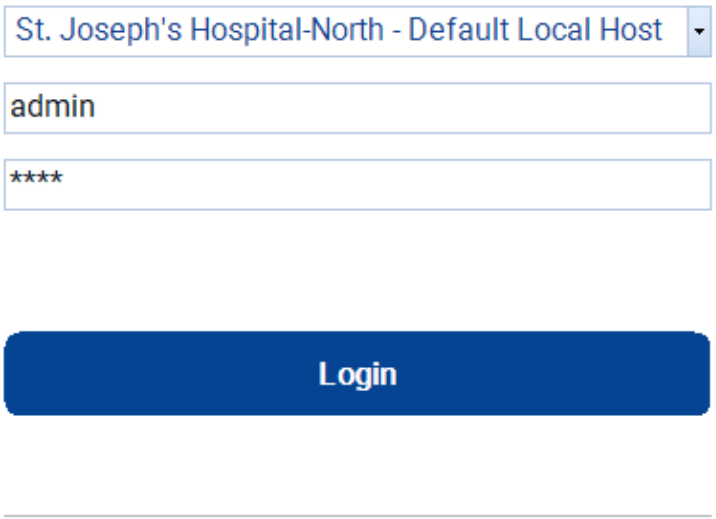

Рис 3.8 Форма для логіну у додаток

3.2.3 Можливості адміністратора MIC

Ні для кого не секрет, що адміністратор має абсолютні права у будь якій системі і моя не стала випадком.

Адміністратор в ній може добавляти нових користувачів, змінювати налаштування звітів, змінювати налаштування SMTP серверу і безліч іншого.

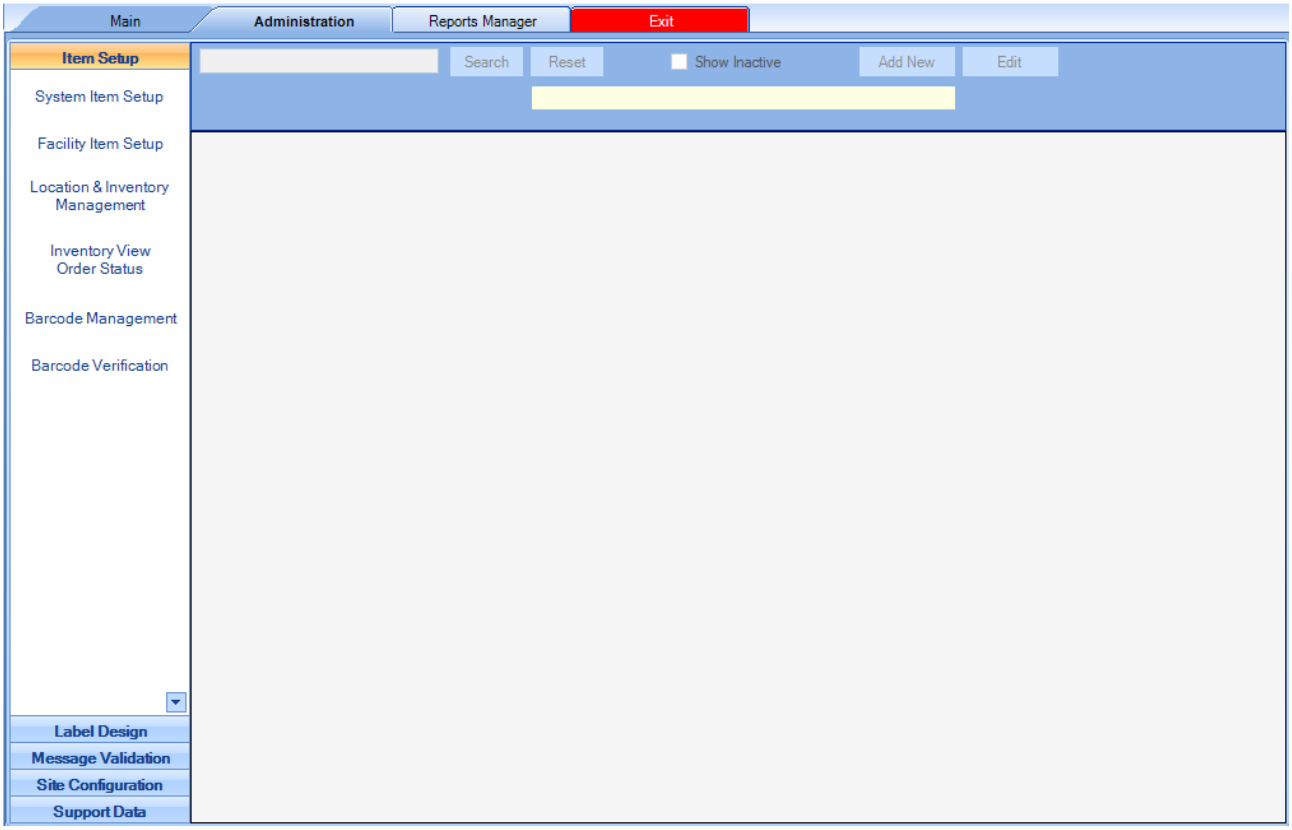

Рис 3.9 Начальна сторінка адміністратора

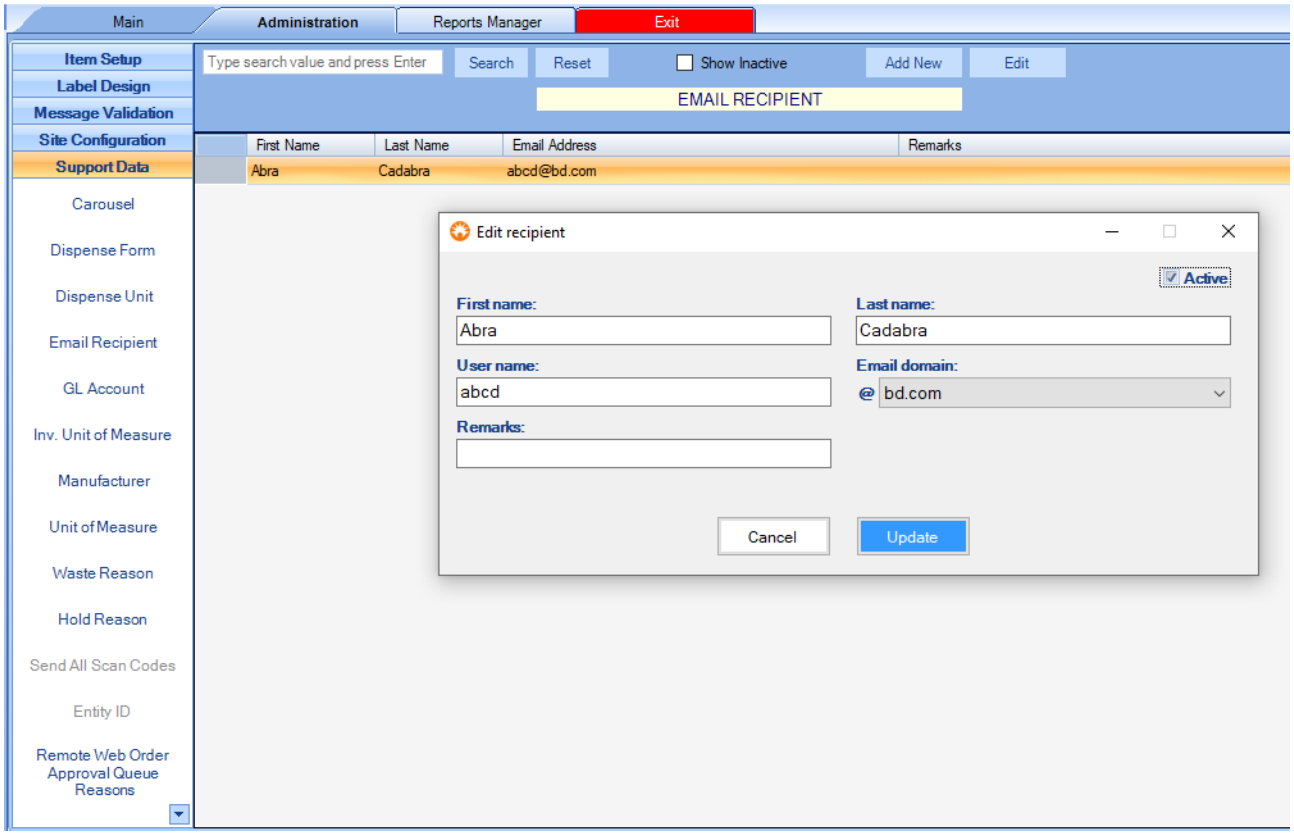

Рис 3.10 Можливість додавання нових користувачів

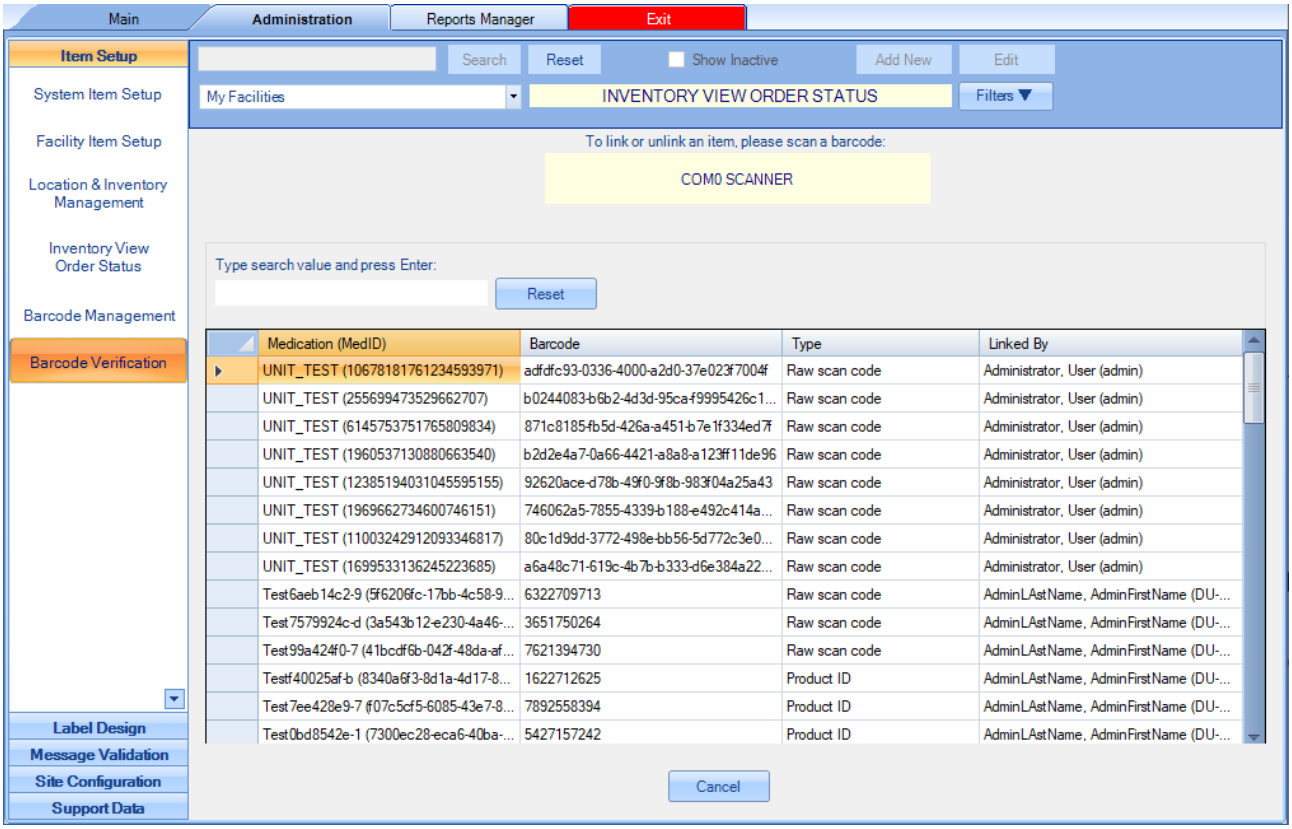

Рис. 3.11 Сканер для зчитування Штрих-кодів і менеджменті ліків

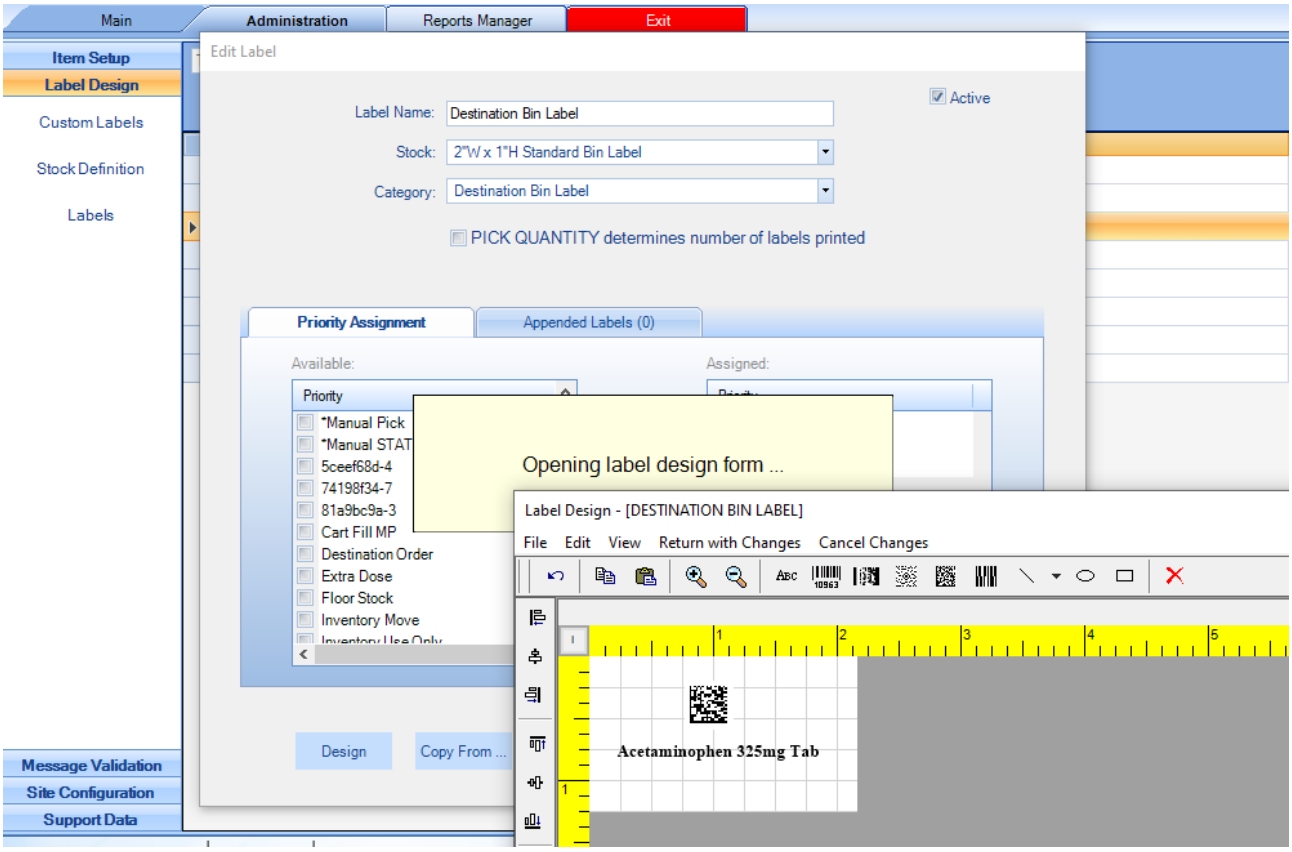

Рис 3.12 Дизайнер для створення власних штрих кодів та QR кодів

Висновки до третього розділу

В даному розділі була проведений детальний розбір функціоналу додатку, показана послідовність виконання та викликів вікон програми.

В даному проекті було використано принцип послідовного виклику вікон, щоб інтерфейс був простий в користуванні, зрозумілий та підходив під вимоги користувача.

Було реалізовано функцію авторизації та реєстрації користувача в додатку, щоб забезпечити надійність існуючих даних пацієнтів.

Було показано функціонал та можливості адміністратора системи.

Наразі основною ступеню розвитку додатку є його локалізація та додавання функціоналу для детального маніпулювання пацієнтами.

РОЗДІЛ 4.
ЕКОНОМІКА

4.1. Визначення трудомісткості розробки програмного забезпечення

Одним з головних етапів при розробці ПЗ є визначення трудомісткості та розрахунок витрат на створення програмного продукту. В даному розділі буде продемонстровано приклад розрахунку витрат на розробку програми для розпізнавання неістинності міркувань.

Початкові дані:

- годинна заробітна плата програміста, грн / год — 200;
- вартість машино-години ЕОМ, грн / год — 40.

Нормування роботи в процесі створення ПЗ істотно ускладнюється в силу творчого характеру роботи програміста, тому трудомісткість розробки ПЗ може бути розрахована на основі системних моделей з різною точністю оцінки.

Трудомісткість розраховується за формулою:

$$t = t_u + t_a + t_n + t_{відл} + t_д ,$$

t_o – витрати праці на підготовку і опис поставленого завдання (50 год.);

t_u – витрати праці на дослідження алгоритму вирішення задач;

t_a – витрати праці на розробку блок-схем алгоритму;

$t_п$ – витрати праці на програмування за готовим блок-схемою;

$t_{відл}$ – витрати праці на відладку програм на ПЕОМ;

$t_д$ -– витрати праці на підготовку документації.

Сукупні витрати праці визначаються виходячи з умовного числа операторів у розроблюваному ПЗ.

Розрахунок очікуваних витрат праці:

$$Q = q \times C \times (1+ p), \text{ людино-годин,}$$

де q – передбачуване число операторів;

с – коефіцієнт складності програми;

р – коефіцієнт корекції програми в ході її розробки.

Наприклад, q буде дорівнювати 1050, с – 1,3, р – 0,1.

$$Q=1050\times 1,3\times(1+0,1) = 1500 \text{ людино-годин,}$$

Витрати праці на вивчення опису завдання визначаються з урахуванням уточнення опису і кваліфікації програміста за формулою:

$$t_u = \frac{QB}{(75 \dots 85)K},$$

де В – коефіцієнт збільшення витрат праці (внаслідок неповного опису завдання, В= 1,2…..1,5);

К – Коефіцієнт кваліфікації програміста, який визначається в залежності від стажу роботи за фахом (якщо стаж роботи менший 2 років, то К = 1,2).

Витрати праці на розробку алгоритму рішення задачі:

$$t_u = 1500 * 1,3 / (80 * 1,2) = 20,3 \text{ людино-годин,}$$

Витрати на складання програми по готовій блок-схемі:

$$t_a = \frac{Q}{(20 \dots 25)K} \text{ людино} - \text{годин,}$$

$$t_a = 1500 / 23 * 1,2 = 78,2 \text{ людино-годин,}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20 \dots 25)K} \text{ людино} - \text{годин},$$

$t_п$ = 15 /23 * 1,2 = 78,2 людино-годин,

Витрати праці на налагодження програми на ЕОМ:

$$t_{\text{відл}} = \frac{Q}{(4 \dots 25)K} \text{ людино} - \text{годин},$$

$t_{відл}$ = 1500 / 4 * 1,2 = 450 людино-годин

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до} \text{ людино} - \text{годин},$$

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{(15 \dots 20)K} \text{ людино} - \text{годин},$$

$t_{др}$ = 1500 / 17 * 1,2 = 105 людино-годин,

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0.75 + t_{др} \text{ людино} - \text{годин},$$

$t_{до}$ = 0,75 * 105 = 79,4 людино-годин,

$t_д$ = 105 + 79,4 = 184,4 людино-годин,

Отримуємо трудомісткість розробки програмного забезпечення:

t = 50 +20,3 + 78,2 +78,2 + 450 + 184,4 = 861,1 людино-годин.

4.2. Розрахунок витрат на створення програмного забезпечення

Витрати на створення програмного забезпечення $К_{пз}$ включають витрати на заробітну плату розробників програми ($З_{зп}$) і витрати машинного часу, необхідного для налагодження програми на ЕОМ ($З_{мв}$).

$$К_{пз} = З_{зп} + З_{мв}$$

$$З_{зп} = t * С_{пр}$$

де t – загальна трудомісткість розробки ПЗ;

$С_{пр}$ – середня годинна заробітна плата програміста.

$$З_{зп} = 861,1 * 200 = 172220 \text{ грн}$$

Витрати машинного часу, необхідного для налагодження програми на ЕОМ ($З_{мв}$) визначаються за формулою:

$$З_{мв} = t_{відл} * С_{мч}$$

где $t_{відл}$ – трудомісткість налагодження програми на ЕОМ;

$С_{мч}$ – вартість машино-години ЕОМ.

$$З_{мв} = 450\times 40 = 18000 \text{ грн.}$$

Таким чином витрати на створення програмного забезпечення, складуть:

$$К_{пз} = 172220 + 18000 = 190220 \text{ грн.}$$

Очікувана тривалість розробки ПЗ:

$$T = \frac{t}{B_k \cdot F_p}$$

где B_k – число розробників;

$F_р$– місячний фонд робочого часу (при 40-ка годинному робочому тижні $F_р$ = 176 годин).

$$T = 861,1 / 1 * 176 = 5 \text{ місяців.}$$

4.3. Маркетингові дослідження

Для розрахунку маркетингової частини на даний момент жодна з аналогічних існуючих програм з паралельними обчисленнями не задовольняє вимогам, тому що їх вихідний код невідомий, а тому складність розробки неможливо порівняти.

4.4. Економічна ефективність

Економічний ефект від впровадження ПЗ, яке використовує паралельні обчислення, очікується позитивним так як від швидкості роботи такого роду додатків залежить маса високонавантажених розрахунків у проектувальній та будівельний галузі.

ВИСНОВКИ

В даному дипломному проекті було розглянуто існуючі медичні системи, які впроваджуються на території України.

Незважаючи на те, що зараз масово впроваджується системи медичнихінформаційних систем, сфера їх застосування охоплює тільки первинну медицину – сімейних лікарів, терапевтів, педіатрів. Решта лікарів досі використовує паперові носії під час своєї роботи.

Сучасні МІС впроваджуються на рівні закладів та зовсім не пристосовані для роботи лікарів-хірургів, тож задачею дипломного проектубуло створити систему, яка компенсувала мінуси існуючих систем, була зрозумілою в освоєнні та була б розрахована на індивідуальне використанняконкретного лікаря-спеціаліста.

Ключовими вимогами до створення МІС являються: надійність збереження даних; зручність в користуванні; безвідмовна робота; швидкодія програми; простий та зрозумілий інтерфейс.

Враховуючи вимоги до створення МІС а також особисті побажання лікаря-хірурга, була запропонована система, яка описується в даному дипломному проекті. Система має інтуїтивно зрозумілий та приємний інтерфейс без зайвих функцій. Для системи було використано вбудовану систему даних, оскільки для реалізації поставленої задачі така база даних оптимально підходила.

ПЕРЕЛІК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. http://moz.gov.ua/article/reform-plan/teper-oficijno-moz-ukraini-rozpochinae-pershij-etap-transformacii-ekstrenoi-medichnoi-dopomogi-
2. Закон України від 01.06.2010, № 2297-VI "Про захист персональнихданих"
3. Порядок функціонування електронної системи охорони здоров'я, затверджений Постановою КМ України, від 25.04.2018, № 411 "Деякіпитання електронної системи охорони здоров'я"
4. https://public.docs.openprocurement.org/get/3cdfe2df19b74f10a0783132598 712a7?KeyID=52462340&Signature=ZDVuu33lp01L4AFMcJNnWaknVo3 Cc1%2FmbvhFAgjJDN9OtybMv6HX3ujL%252B1%2Fs5PnWgH%252B2oBYXXyng9GXJgMqwBA%253D%253D
5. https://helsi.me/
6. https://www.mcmed.ua/ua
7. https://www.oracle.com/technetwork/java/index.html
8. https://www.baeldung.com/java-stack-heap
9. https://www.oracle.com/technetwork/java/javase/downloads/index.html
10. ttps://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html
11. http://www.h2database.com/html/functions.html
12. https://docs.oracle.com/javafx/2/drag_drop/jfxpub-drag_drop.htm

13. https://docplayer.ru/27558273-Razrabotka-nasyshchennyh-internet-prilozheniy-s-pomoshchyu-javafx.html
14. https://www.theserverside.com/discussions/thread/46456.html
15. https://www.w3schools.com/sql/sql_injection.asp
16. http://www.tutorialspoint.com/apache_poi/apache_poi_overview.htm
17. https://poi.apache.org/

Додаток А. ЛІСТИНГ ПРОГРАМИ

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Linq;
using InventoryManager.AdminForms.PreferredOrdering;
using Toolkit;
using PharmogisticsDB;
using System.Collections;
using InventoryManager.Reports;
using System.Deployment.Application;
using System.Reflection;
using System.Threading;
using Microsoft.Win32;
using System.Data.SqlClient;
using System.Diagnostics;
using System.IO;
using InventoryManager.Class;
using Toolkit.AuthenticationManagement;
using Toolkit.ScanCodeMaintenance;
using System.Threading.Tasks;
using InventoryManager.Administration.RoutingRules.Models;
using InventoryManager.Administration.RoutingRules.Presenters;
using InventoryManager.Administration.RoutingRules.Views;
using InventoryManager.Administration.Schedules.Models;
using InventoryManager.Administration.Schedules.Presenters;
using InventoryManager.Administration.Schedules.Views;
using InventoryManager.UiFramework.Views;
using InventoryManager.Administration.Isas.Models;
using InventoryManager.Administration.Isas.Presenters;
using InventoryManager.Administration.Isas.Views;
using InventoryManager.Administration.SystemItemSetup.Models;
using InventoryManager.Administration.SystemItemSetup.Presenters;
using InventoryManager.Administration.SystemItemSetup.Views;
using InventoryManager.Models;
```

```
using InventoryManager.Notifications.Model;
using InventoryManager.Notifications.Presenter;
using InventoryManager.Notifications.View;
using Pharmogistics.Server.Shared.Common;
using InventoryManager.VendorOrder.VendorOrderCreate.Models;
using InventoryManager.VendorOrder.VendorOrderCreate.Views;
using InventoryManager.VendorOrder.VendorOrderCreate.Presenters;
using System.Data;
using AutoMapper;
using CareFusion.HostedSolutions.BarcodeParsing;
using InventoryManager.Administration.BarcodeManagement.Models;
using InventoryManager.Administration.Departments.Models;
using InventoryManager.Administration.FormularyItemList.Presenters;
using InventoryManager.Administration.FormularyItemList.Views;
using InventoryManager.Administration.ItemLocations.Binders;
using InventoryManager.Administration.ItemLocations.Models;
using InventoryManager.Administration.ItemLocations.Presenters;
using InventoryManager.Administration.ItemLocations.Views;
using InventoryManager.Administration.BarcodeManagement.Presenters;
using InventoryManager.Administration.BarcodeManagement.Views;
using InventoryManager.Administration.DispenseForm.Models;
using InventoryManager.Administration.DispenseUnit.Models;
using InventoryManager.Administration.ExternalSystemSettings.Presenters;
using InventoryManager.Administration.MMSHub.Models;
using InventoryManager.Administration.MMSHub.Presenters;
using InventoryManager.Administration.SystemItemSetup.Binders;
using InventoryManager.Administration.SupportData.Models;
using InventoryManager.Administration.WasteReason.Models;
using Pharmogistics.Server.Model.Application;
using Pharmogistics.Server.Shared.Enum;
using InventoryManager.Controls;
using Pharmogistics.Server.Shared.States;
using InventoryManager.Administration.ParLevelSettings.Models;
using InventoryManager.Administration.ParLevelSettings.Presenters;
using InventoryManager.Administration.HoldReason.Models;
using InventoryManager.Services.Scanner.Config;
using InventoryManager.Services.Scanner.Interfaces;
using InventoryManager.Services.Scanner.Service;
```

```
using InventoryManager.Administration.Manufacturer.Models;
using InventoryManager.Administration.SendInventory;
using InventoryManager.Administration.SendInventoryStatus.Models;
using InventoryManager.Administration.SendInventoryStatus.Presenters;
using InventoryManager.Administration.SendInventoryStatus.Views;
using InventoryManager.DistributorReceiving.SubstitutionOptions.Views;
using InventoryManager.DistributorReceiving.SubstitutionOptions.Presenters;
using InventoryManager.DistributorReceiving.SubstitutionOptions.Models;
using InventoryManager.Reports.ReceivingVariance.Models;
using InventoryManager.Reports.ReceivingVariance.Presenters;
using InventoryManager.Reports.ReceivingVariance.Views;
using InventoryManager.UiFramework.Response;
using Toolkit.Timezone;
using Pharmogistics.Server.Model.User;
using InventoryManager.AdminForms.LocationEntities.Presenters;
using InventoryManager.AdminForms.LocationEntities.Views;
using InventoryManager.AdminForms.LocationEntities.Models;
using InventoryManager.Administration.SupportAccount.Models;
using InventoryManager.Administration.SupportAccount.Views;
using InventoryManager.Administration.SupportAccount.Presenters;
using System.Web.UI.Design;
using DevComponents.DotNetBar;
using InventoryManager.UserReportParameters;
using InventoryManager.Reports.UserReports;
using InventoryManager.Reports.UserReports.Models;
using InventoryManager.Administration.Destination.Views.GlobalEditFormularyItems;
using InventoryManager.Administration.Destination.Presenters;
using InventoryManager.Administration.Destination.Models;
using InventoryManager.Reports.UserReports.Views;
using InventoryManager.Reports.UserReports.ViewModels;
using Newtonsoft.Json;
using Pharmogistics.Server.Model;
using InventoryManager.Administration.Destination.Views.GlobalEditUsers;
using System.ComponentModel;
using InventoryManager.AdminForms.Destinations.Models;
using InventoryManager.Reports.UserReports.UserReportParameters;
using InventoryManager.Administration.Destination.Views.GlobalEditGeneralSettings;
using Destination = InventoryManager.AdminForms.Destinations.Views.Destination;
```

```
using InventoryManager.Administration.EmailRecipients.Models;
using InventoryManager.Administration.EmailRecipients.Views;
using InventoryManager.Administration.EmailRecipients.Presenters;
using InventoryManager.Administration.EmailRecipients.ViewModels;
using DevComponents.DotNetBar.Controls;
using Pharmogistics.Server.Model.Reports.ReportParameters;
using Pharmogistics.Server.Model.Reports;
using InventoryManager.Administration.Entities;
using InventoryManager.AdminForms.IOMessageValidation.Presenters;
using InventoryManager.AdminForms.IOMessageValidation.Views;
using InventoryManager.AdminForms.PickTransactionForValidation.Models;
using InventoryManager.AdminForms.PickTransactionForValidation.Presenters;
using Pharmogistics.Server.Model.Administration.MessageValidation;
using InventoryManager.Administration.PrintLabel;
using InventoryManager.AdminForms.PickTransactionForValidation;
using InventoryManager.Extensions;
using InventoryManager.LabelPrinting.Realization;
using Pharmogistics.Server.Model.Reports.Enums;
using InventoryManager.AdminForms.MessageValidationSelection.Helpers;
using InventoryManager.UserPreferenceColumnsSetup.Views;
using InventoryManager.UserPreferenceColumnsSetup.Models;
using InventoryManager.UserPreferenceColumnsSetup.Presenters;

namespace InventoryManager
{
    public partial class Main : FormViewBase
    {
        private bool CanLoadNotifications { get; set; }

        private BindingSource wfaItemsBindingSource = new BindingSource();
        public static StaticData staticData = null;
        private ITimezoneHelper _timezoneHelper;
        private IStaticDataWrapper _staticDataWrapper;
        private static HealthCareSystemNetworkData dynamicData = null;
        public static ISessionHostConfig ActiveSessionHostFacilityData { get; private
set; }

        public static Idle AppIdle = null;
```

```
        public static bool IsAuthenticated { get; set; }
        public static bool IsAuthenticated_via_LDAP { get; set; }
        private PharmogisticsDataContext dbPharmogistics { get; set; }
        private StaticData.WinAPI winApi { get; set; }
        private string IMBackUpdDir = string.Empty;
        public bool IsStartup = true;
        public bool IsBusySendingScanCode { get; private set; }
        private bool IsCacheAuth { get; set; }

        private FormResizer FormResizer { get; set; }

        private string _itemSetupFilterExpression;
        private string _facilitySpecificItemSetupFilterExpression;

        private static readonly List<string> NonFilteredMenuOptionsList = new
List<string>
        {
            "Main.barcodeMgmtBtn",
            "Main.barcodeVerificationBtn",
            "Main.mmsHubSettingsBtn",
            "Main.parLevelSettingsBtn",
            "Main.verityLinkSettingsBtn"
        };

        private Dictionary<object, bool> SavedPanelMDViewState = new Dictionary<object,
bool>();

        #region MVP

        private ICurrentUserPermissions _currentUserPermissions;
        private MessageManagerView _messageManager;
        private ApiClient _apiClient = null;
        private IMapper _mapper;

        private SupportDataModel _supportDataModel;
        private SiteConfigurationModel _siteConfigurationModel;

        private SystemFormularyItemListPresenter _systemFormularyItemListPresenter;
        private SystemFormularyItemListView _systemFormularyItemListView;
```

```
        private FacilityFormularyItemListPresenter _facilityFormularyItemListPresenter;
        private FacilityFormularyItemListView _facilityFormularyItemListView;
        private FormularyItemLocationListPresenter _formularyItemLocationListPresenter;
        private FormularyItemLocationListView _formularyItemLocationListView;

        private IFormularyInventoryView _formularyInventoryView;

        private FacilityFormularyLocationReplenishPresenter
_facilityFormularyLocationReplenishPresenter;
        private FacilityFormularyLocationReplenishView
_facilityFormularyLocationReplenishView;

        private IFormularyInventoryPresenter _formularyInventoryPresenter;
        private IFormularyInventoryModel _inventoryModel;

        private FormularyItemListFiltersBinder _formularyItemListFiltersBinder;

        private SystemFormularyItemApiModel _systemFormularyItemModel;
        private FacilityFormularyItemModel _facilityFormularyItemModel;
        private ISystemFormularyItemPresenter _systemFormularyItemPresenter;
        private ISystemFormularyItemView _systemFormularyItemView;

        private IGlobalEditSystemFormularyPresenter _globalEditSystemFormularyPresenter;
        private IGlobalEditSystemFormularyView _globalEditSystemFormularyView;
        private IGlobalEditSystemFormularyConfirmView
_globalEditSystemFormularyConfirmView;

        private IGlobalEditFacilityItemPresenter _globalEditFacilityItemPresenter;
        private IGlobalEditFacilityItemView _globalEditFacilityItemView;
        private IGlobalEditFacilityItemConfirmView _globalEditFacilityItemConfirmView;

        private SystemPreferredOrderingModel _systemPreferredOrderingModel;
        private ICopyFacilityPresenter _copyFacilityPresenter;
        private ICopyFacilityView _copyFacilityView;

        private DepartmentApiModel _departmentModel;

        private NdcModel _ndcModel;
        private INdcPresenter _ndcPresenter;
```

```
private INdcView _ndcView;

private TranQManualEntryModel _tranQManualEntryModel;

private ItemLocationsModel _itemLocationsModel;
private IItemLocationsPresenter _itemLocationpresenter;
private IItemLocationsView _itemLocationsView;

private LocationManagementModel _locationManagementModel;
private ILocationPresenter _locationPresenter;
private ILocationView _locationView;

private IExternalLocationPresenter _externalLocationPresenter;
private IExternalLocationView _externalLocationView;

private ScheduleModel _scheduleModel;
private ScheduleListNoUiView _scheduleListNonUiView;
private IScheduleTimingView _scheduleTimingView;
private IScheduleTimingPresenter _scheduleTimingPresenter;
private IScheduleListPresenter _scheduleListPresenter;

private RemoteOrderApiModel _remoteOrderApiModel;

private RoutingRuleModel _routingRuleModel;
private RoutingRulesListView _routingRuleListView;
private IRoutingRuleView _routingRuleView;
private IRoutingRulesListPresenter _routingRuleListPresenter;
private IRoutingRulePresenter _routingRulePresenter;

private IIsaListModel _isaListModel;
private IIsaListView _isaListView;
private IIsaListPresenter _isaListPresenter;

private ILocationEntitiesListApiModel _locationEntitiesListModel;
private ILocationEntitiesListView _locationEntitiesListView;
private ILocationEntitiesListPresenter _locationEntitiesListPresenter;
//Receiving Variance
private IReceivingVarianceModel _receivingVarianceModel;
```

```
private IReceivingVariancePresenter _receivingVariancePresenter;
private IReceivingVarianceDisplayBusiness _receivingVarianceDisplayBusiness;

//Routing Rules Details
private IRoutingRulesDetailListView _routingRulesDetailListView;
private IRoutingRulesDetailListPresenter _routingRulesDetailPresenter;

//Non-Phg / External Isa
private NonPhgIsaModel _nonPhgIsaModel;
private INonPhgIsaView _nonPhgIsaView;
private INonPhgIsaPresenter _nonPhgIsaPresenter;



private LocationEntitiesApiModel _locationEntitiesApiModel;
private ILocationEntitiesView _locationEntitiesView;
private ILocationEntitiesPresenter _locationEntitiesPresenter;

//Vendor Order Create
private VendorOrderCreateModel _vendorOrderCreateModel;
private IVendorOrderCreateView _vendorOrderCreateView;
private IVendorOrderCreatePresenter _vendorOrderCreatePresenter;

//Notifications
private NotificationApiModel _notificationApiModel;
private INotificationView _notificationView;
private INotificationPresenter _notificationPresenter;
private PickCreatedNotifierModel _pickCreatedNotifierModel;
private TransactionQueueModel _transactionQueueModel;

//Scanner
private IBarcodeParser _barcodeParser;

//Dispense Form
private DispenseFormModel _dispenseFormModel;

//Dispense Unit
private DispenseUnitModel _dispenseUnitModel;
```

```
//Waste Reason
private WasteReasonModel _wasteReasonModel;

//Remote Order Approval Reason
private RemoteOrderApprovalReasonApiModel _remoteOrderApprovalReasonModel;

//Hold Reason
private HoldReasonModel _holdReasonModel;

//Manufacturer
private ManufacturerModel _manufacturerModel;

//Barcode Management
private BarcodeManagementModel _barcodeManagementModel;
private IBarcodeManagementPresenter _barcodeManagementPresenter;

//Barcode container view
private BarcodeManagementViewContainer _barcodeManagementView;
private BarcodeManagementModel _barcodeManagementContainerModel;
private IBarcodeManagementPresenter _barcodeManagementContainerPresenter;

//UnitOfMeasure
private UnitOfMeasureApiModel _unitOfMeasureApiModel;

//Facility
private FacilityApiModel _facilityApiModel;

//HealthCareSystem
private HealthCareSystemModel _healthCareSystemModel;

//MmsHubConfiguration
private MmsHubConfigurationModel _mmsHubConfigurationModel;
private IMmsHubConfigurationPresenter _mmsHubConfigurationPresenter;

//Par Level configuration
private ParLevelConfigurationModel _parLevelConfigurationModel;
private IParLevelConfigurationPresenter _parLevelConfigurationPresenter;
```

```
//Verity link
private IVerityLinkConfigurationPresenter _verityLinkConfigurationPresenter;

//SendInventory
private SendInventoryApiModel _sendInventoryApiModel;

//Inventory Unit Of Measure
private InventoryUnitOfMeasureModel _inventoryUnitOfMeasureModel;

//sentInventoryStatusHistory
private ILastSentInventoryHistoryView _lastSentInventoryHistoryView;
private LastSentInventoryHistoryApiModel _lastSentInventoryHistoryApiModel;
private ILastSentInventoryHistoryPresenter _lastSentInventoryHistoryPresenter;

//Substitution Items identification options
private ISubstitutionOptionsView _substitutionOptionsView;
private ISubstitutionOptionsPresenter _substitutionOptionsPresenter;
private ISubstitutionOptionsModel _substitutionOptionsModel;
private IUserPreferenceModel _userPreferenceModel;

// User Reports
private IUserReportsModel _userReportsModel;
private IUserReportsPresenter _userReportsPresenter;
private ISaveAsUserReportPresenter _saveAsUserReportPresenter;
private ISaveAsUserReportView _saveAsUserReportView;
private IManageReportUserModel _manageReportUserModel;

// Schedule Reports
private IScheduleReportsModel _scheduleReportsModel;
private ISMTPConfigurationModel _smtpConfigurationModel;
private IScheduleReportsPresenter _scheduleReportsPresenter;
private ISaveScheduleReportPresenter _saveScheduleReportPresenter;
private ISaveScheduleReportLayoutManager _saveScheduleReportLayoutManager;
private IScheduleReportSettingsComposer _scheduleSettingsComposer;
private IScheduleReportSettingsComposer _deliverySettingsComposer;
private RedirectScheduleReportEventArgs _currentScheduleEventArgs;

// User Reports - Navigation
```

```
        private IUserReportsRouter _userReportsRouter;
        private ReportRoute _whereToReturn;

        // Destination
        private IDestinationsApiModel _destinationsApiModel;

        // Destination / Global Edit
        private IGlobalEditModel _globalEditModel;
        private IGlobalEditPresenter _globalEditPresenter;

        // Email Recipient
        private IEmailRecipientModel _emailRecipientModel;
        private IEmailRecipientView _emailRecipientView;
        private IEmailRecipientPresenter _emailRecipientPresenter;
        private IEmailDomainModel _emailDomainModel;

        // Message Validation
        private IMessageValidationSelectionPresenter
_messageValidationSelectionPresenter;
        private IPickTransactionPresenter _pickTransactionPresenter;
        private IAllTranQWithRequestsModel _allTranQModel;
        private IPrintLabelModel _printLabelModel;
        private ILabelPreviewPresenter _labelPreviewPresenter;
        private IUserPreferenceColumnsSetupView _messageValidationUserColumnsSetupView;
        private IUserPreferenceColumnsSetupModel _messageValidationUserColumnsSetupModel;
        private IUserPreferenceColumnsSetupPresenter
_messageValidationUserColumnsSetupPresenter;



        //Generate Support Account
        private SupportAccountModel _supportAccountModel;
        private IGenerateSupportAccountView _supportAccountView;
        private IGenerateSupportAccountPresenter _supportAccountPresenter;



        #endregion

        public Main(string AppMode, bool useAppConfig)
        {
```

```
            InitializeComponent();


            SystemEvents.PowerModeChanged += new
PowerModeChangedEventHandler(SystemEvents_PowerModeChanged);


            // One time
            _apiClient = new ApiClient();
            _staticDataWrapper = new StaticDataWrapper();
            _timezoneHelper = new TimezoneHelper();
            dynamicData = new HealthCareSystemNetworkData(_apiClient);


            ActiveSessionHostFacilityData = dynamicData;


#if !(DEBUG)
            if (IsWfaEnabledOnMultipleInstance())
            {
                MessageBox.Show(
                    $"This computer {SysTools.GetMachineName()} has access to WFA. Only
one instance of Inventory Manager with WFA access allowed",
                    "Computer Access", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                Application.Exit();
                return;
            }
#endif


            // Instantiate and initialize static data class
            if (PrepStaticData(AppMode, useAppConfig))
            {
                CreateDesktopIcons();


                if (!HandleUserLogin())
                {
                    Application.Exit();
                    return;
                }


                InitializeMvp(); // One time


                SetFilterExpanded(false);
```

```
                SelectionDgv.DoubleBuffered(true);


                FormResizer = new FormResizer();

                DateFromDatePicker.MinDate = DateTime.Now.Date.AddYears(-10);

            }

            else

            {

                Application.Exit();

                return;

            }


        }


        public void UpdateFacilityData(string hostName, string connectionString)

        {

            if (HealthCareSystemNetworkData.HostConfigurationValid == false)

            {

                MessageBox.Show("Application host address configuration has not been
setup.\r\nPlease configure Application host address", "Application Setup",
MessageBoxButtons.OK, MessageBoxIcon.Hand);

                Application.Exit();

                return;

            }


            dynamicData.InitializeHost(hostName, connectionString);

        }


        private void SetExternalSystemLinkMenu()

        {

            var healthCareSystemSettings =
_healthCareSystemModel.GetHealthCareSystemSettings();


            if (healthCareSystemSettings.Status == ModelResponseStatus.Success

                && healthCareSystemSettings.Result != null

                &&
!string.IsNullOrWhiteSpace(healthCareSystemSettings.Result.VerityLinkUrl))

            {

                externalSystemLinkBtn.Links[0].LinkData =
healthCareSystemSettings.Result.VerityLinkUrl;

            }
```

```
        }

        private void externalSystemLinkBtn_LinkClicked(object sender, MouseEventArgs e)
        {
            var currentLink = externalSystemLinkBtn.Links[0].LinkData;
            if (currentLink != null)
            {
                try
                {
                    string target = currentLink as string;
                    Process.Start(target);
                }
                catch
                {
                    _messageManager.Message("Link not valid", "Verity Link");
                }
            }
        }

        /// <summary>
        /// Gets the assembly title.
        /// </summary>
        /// <value>The assembly title.</value>
        void SystemEvents_PowerModeChanged(object sender, PowerModeChangedEventArgs e)
        {
            if (e.Mode == PowerModes.Suspend)
            {
                SystemEvents.PowerModeChanged -= new
PowerModeChangedEventHandler(SystemEvents_PowerModeChanged);
                Application.Exit();
            }
        }

        private string _activeBtnText;

        private string ActiveBtnText
        {
            get { return _activeBtnText; }
```

```
            set { _activeBtnText = value; }
        }

        private string ActiveTag { get; set; }

        private string LastActiveTag { get; set; }

        private void Main_Load(object sender, EventArgs e)
        {
            if (Main.IsAuthenticated == false)
            {
                this.Visible = false;
                this.Close();
            }
            else
            {
                HideRptPanels();
                this.Visible = true;
                compNameToolStripLabel.Text = staticData.ComputerDescription;
            }
            IsStartup = false;
        }

        private void PrepareMvpFeatures()
        {
            var pharmogisticsLogger = new
Pharmogistics.Server.Shared.Logger.PharmogisticsLogger("InventoryManager");

            //initialize mvp
            _currentUserPermissions = staticData;
            _messageManager = new MessageManagerView(this);

            _mapper = AutoMapperConfiguration.GetMapper();
            //models
            _supportDataModel = new SupportDataModel(Main.staticData.Logger, _apiClient,
ActiveSessionHostFacilityData);
            _siteConfigurationModel = new SiteConfigurationModel(Main.staticData.Logger,
_apiClient, ActiveSessionHostFacilityData);
            _systemFormularyItemModel = new SystemFormularyItemApiModel(_apiClient,
ActiveSessionHostFacilityData);
```

```
            _departmentModel = new DepartmentApiModel(_apiClient,
ActiveSessionHostFacilityData);

            _facilityFormularyItemModel = new FacilityFormularyItemModel(_apiClient,
ActiveSessionHostFacilityData);

            _scheduleModel = new ScheduleModel(_apiClient,
ActiveSessionHostFacilityData);

            _routingRuleModel = new RoutingRuleModel(_apiClient,
ActiveSessionHostFacilityData);

            _isaListModel = new IsaListModel(_apiClient, ActiveSessionHostFacilityData);

            _locationEntitiesListModel = new LocationEntitiesListApiModel(_apiClient,
ActiveSessionHostFacilityData);

            _nonPhgIsaModel = new NonPhgIsaModel(_apiClient,
ActiveSessionHostFacilityData);

            _locationEntitiesApiModel = new LocationEntitiesApiModel(_apiClient,
ActiveSessionHostFacilityData);

            _systemPreferredOrderingModel = new SystemPreferredOrderingModel(_apiClient,
ActiveSessionHostFacilityData);

            _receivingVarianceModel = new ReceivingVarianceModel(_apiClient,
ActiveSessionHostFacilityData);

            //parsers

            _barcodeParser = new BarcodeParser();


            //schedules

            _scheduleListNonUiView = new ScheduleListNoUiView(SelectionDgv);

            _scheduleTimingView = new ScheduleTimingView();

            _scheduleTimingPresenter = new ScheduleTimingPresenter(_scheduleTimingView,
_scheduleModel, _messageManager, _currentUserPermissions);

            _scheduleListPresenter = new ScheduleListPresenter(_scheduleListNonUiView,
_scheduleModel, _scheduleTimingPresenter, _messageManager);


            //Routing Rules

            _routingRuleListView = new RoutingRulesListView(SelectionDgv);

            _routingRuleView = new RoutingRuleView();

            _routingRulePresenter = new RoutingRulePresenter(_routingRuleView,
_routingRuleModel, _scheduleModel, _messageManager, _currentUserPermissions);

            _routingRuleListPresenter = new
RoutingRulesListPresenter(_routingRuleListView, _routingRuleModel, _routingRulePresenter,
_messageManager);


            _routingRulesDetailListView = new RoutingRulesDetailListView();

            _routingRulesDetailPresenter = new
RoutingRulesDetailListPresenter(_routingRulesDetailListView, _routingRuleModel,
_messageManager);


            //Isas List
```

```
            _isaListView = new IsaListView(SelectionDgv);

            _isaListPresenter = new IsaListPresenter(_isaListView, _isaListModel,
_messageManager);


            //LocationEntites List

            _locationEntitiesListView = new LocationEntitiesListView(SelectionDgv);

            _locationEntitiesListPresenter = new
LocationEntitiesListPresenter(_locationEntitiesListView, _locationEntitiesListModel,
_messageManager);


            //Receiving Variance

            _receivingVarianceDisplayBusiness = new
ReceivingVarianceDisplayBusiness(_timezoneHelper, _staticDataWrapper);

            _receivingVariancePresenter = new
ReceivingVariancePresenter(_receivingVarianceView, _receivingVarianceModel,
_messageManager, _receivingVarianceDisplayBusiness);


            //Ndc

            _ndcView = new NdcView();

            _ndcModel = new NdcModel(_apiClient, ActiveSessionHostFacilityData);

            _ndcPresenter = new NdcPresenter(_ndcView, _siteConfigurationModel,
_departmentModel, _messageManager, _systemFormularyItemModel, _ndcModel);


            //Manual Entry

            _tranQManualEntryModel = new TranQManualEntryModel(_apiClient,
ActiveSessionHostFacilityData);


            //Item Setup

            _itemLocationsModel =

                new ItemLocationsModel(_apiClient, Main.staticData.Logger,
ActiveSessionHostFacilityData);


            _systemFormularyItemView = new SystemFormularyItemView();


            _globalEditSystemFormularyView = new GlobalEditSystemFormularyView();

            _globalEditSystemFormularyConfirmView = new
GlobalEditSystemFormularyConfirmView(_messageManager);


            _systemFormularyItemListView = new SystemFormularyItemListView(SelectionDgv);

            _systemFormularyItemListPresenter = new
SystemFormularyItemListPresenter(_systemFormularyItemListView, _systemFormularyItemModel,
_systemFormularyItemPresenter, _messageManager);
```

```
            _globalEditFacilityItemView = new GlobalEditFacilityItemView();

            _globalEditFacilityItemConfirmView = new GlobalEditFacilityItemConfirmView();


            _facilityFormularyItemListView = new
FacilityFormularyItemListView(SelectionDgv);

            _facilityFormularyItemListPresenter = new
FacilityFormularyItemListPresenter(_facilityFormularyItemListView,
_facilityFormularyItemModel, _messageManager);


            _copyFacilityView = new CopyFacilityView();

            _copyFacilityPresenter = new CopyFacilityPresenter(

                _copyFacilityView,

                null,

                _systemPreferredOrderingModel,

                _messageManager

                );


            //InventoryUnit of Measure

            _inventoryUnitOfMeasureModel = new InventoryUnitOfMeasureModel(_apiClient,
ActiveSessionHostFacilityData);


            // Dispense Form

            _dispenseFormModel = new DispenseFormModel(_apiClient,
ActiveSessionHostFacilityData);


            //Dispense Unit

            _dispenseUnitModel = new DispenseUnitModel(_apiClient,
ActiveSessionHostFacilityData);


            // Waste Reason

            _wasteReasonModel = new WasteReasonModel(_apiClient,
ActiveSessionHostFacilityData);


            //Remote Order Approval Reason

            _remoteOrderApprovalReasonModel = new
RemoteOrderApprovalReasonApiModel(_apiClient, ActiveSessionHostFacilityData);


            //Hold Reason

            _holdReasonModel = new HoldReasonModel(_apiClient,
ActiveSessionHostFacilityData);


            //Manufacturer
```

```
            _manufacturerModel = new ManufacturerModel(_apiClient,
ActiveSessionHostFacilityData);


            //Item Locations

            _formularyItemLocationListView = new
FormularyItemLocationListView(SelectionDgv);

            _formularyItemLocationListPresenter = new
FormularyItemLocationListPresenter(_formularyItemLocationListView,
_facilityFormularyItemModel, _messageManager);

            _locationManagementModel = new LocationManagementModel();

            _itemLocationsView = new ItemLocationsView(_messageManager);

            _locationView = new LocationView(_messageManager);

            _externalLocationView = new ExternalLocationView();

            _locationPresenter = new LocationPresenter(_locationView,
_locationManagementModel, _itemLocationsModel, _facilityFormularyItemModel,
_messageManager);

            _externalLocationPresenter = new
ExternalLocationPresenter(_externalLocationView, _messageManager);

            _itemLocationpresenter = new ItemLocationsPresenter(_itemLocationsView,
_locationPresenter, _externalLocationPresenter, _routingRulesDetailPresenter,
_itemLocationsModel, _routingRuleModel, _messageManager, _locationManagementModel);


            _facilityFormularyLocationReplenishView = new
FacilityFormularyLocationReplenishView(_messageManager);

            _facilityFormularyLocationReplenishPresenter = new
FacilityFormularyLocationReplenishPresenter(

                _facilityFormularyLocationReplenishView,

                _messageManager);


            //Inventory

            //Inventory

            _inventoryModel = new FormularyInventoryModel(_apiClient,
ActiveSessionHostFacilityData);

            _formularyInventoryView = new FormularyInventoryView();

            _formularyInventoryPresenter = new
FormularyInventoryPresenter(_systemFormularyItemModel, _inventoryModel,
_formularyInventoryView, _messageManager);


            //HealthCareSystem

            _healthCareSystemModel =

                new HealthCareSystemModel(pharmogisticsLogger, _apiClient,
ActiveSessionHostFacilityData);


            //Barcode Management Container
```

```
            _barcodeManagementView = new BarcodeManagementViewContainer();

            _barcodeManagementContainerModel = new BarcodeManagementModel(_apiClient,
pharmogisticsLogger, ActiveSessionHostFacilityData);


            _barcodeManagementContainerPresenter = new
BarcodeManagementPresenter(_barcodeManagementView, _barcodeManagementContainerModel,
_messageManager, staticData, _healthCareSystemModel, _ndcPresenter, _pharmogisticsLogger,
_ndcModel);
            _barcodeManagementView.SetupEvents();


            _systemFormularyItemPresenter =
                new SystemFormularyItemPresenter(_systemFormularyItemView,
                _systemFormularyItemModel,
                _facilityFormularyItemModel,
                _siteConfigurationModel,
                _supportDataModel,
                _ndcPresenter,
                _messageManager,
                _currentUserPermissions,
                _departmentModel,
                _facilityFormularyLocationReplenishPresenter,
                _itemLocationsModel,
                _barcodeManagementContainerPresenter,
                _ndcModel,
                _healthCareSystemModel,
                _mapper);


            _globalEditSystemFormularyPresenter = new
GlobalEditSystemFormularyPresenter(_globalEditSystemFormularyView,
                _globalEditSystemFormularyConfirmView,
                _systemFormularyItemModel,
                _messageManager,
                _supportDataModel,
                _siteConfigurationModel,
                _currentUserPermissions);


            _globalEditFacilityItemPresenter = new
GlobalEditFacilityItemPresenter(_globalEditFacilityItemView,
                _globalEditFacilityItemConfirmView,
                _facilityFormularyItemModel,
```

```
                _messageManager,
                _siteConfigurationModel,
                _currentUserPermissions);


            //ExternalIsa/NonPhgIsa
            _nonPhgIsaView = new NonPghIsaView();
            _nonPhgIsaPresenter = new NonPhgIsaPresenter(_nonPhgIsaView, _nonPhgIsaModel,
_messageManager);


            _locationEntitiesView = new LocationEntitiesView();
            _locationEntitiesPresenter = new
LocationEntitiesPresenter(_locationEntitiesView, _locationEntitiesApiModel,
_messageManager);
            //Vendor Order Create
            _vendorOrderCreateView = new VendorOrderCreateView();
            _vendorOrderCreateModel = new VendorOrderCreateModel();
            _vendorOrderCreatePresenter = new
VendorOrderCreatePresenter(_vendorOrderCreateView, _vendorOrderCreateModel,
_messageManager, pharmogisticsLogger);



            //Support Account
            _supportAccountView = new GenerateSupportAccountView();
            _supportAccountModel = new SupportAccountModel(_apiClient,
ActiveSessionHostFacilityData);
            _supportAccountPresenter = new
GenerateSupportAccountPresenter(_supportAccountModel, _supportAccountView,
_messageManager);



            //Notifications
            _notificationView = new NotificationView();
            _notificationApiModel = new NotificationApiModel(_apiClient,
ActiveSessionHostFacilityData);
            _notificationPresenter = new NotificationPresenter(_notificationView,
_notificationApiModel, _messageManager, _mapper);
            _notificationPresenter.NotificationActionEventHandler += (s, e) => {
OnNotificationActionSelected(e); };



            _pickCreatedNotifierModel = new PickCreatedNotifierModel(_apiClient,
ActiveSessionHostFacilityData);
```

```
            _transactionQueueModel = new TransactionQueueModel(_apiClient,
ActiveSessionHostFacilityData);


            //Barcode Management

            _barcodeManagementModel = new BarcodeManagementModel(_apiClient,
pharmogisticsLogger, ActiveSessionHostFacilityData);


            _barcodeManagementPresenter = new BarcodeManagementPresenter(barcodeMgmtView,
_barcodeManagementModel, _messageManager, staticData, _healthCareSystemModel,
_ndcPresenter, _pharmogisticsLogger, _ndcModel);


            //Unit Of Measure

            _unitOfMeasureApiModel = new UnitOfMeasureApiModel(_apiClient,
ActiveSessionHostFacilityData);


            //Facility

            _facilityApiModel = new FacilityApiModel(_apiClient,
ActiveSessionHostFacilityData);


            //RemoteOrder

            _remoteOrderApiModel = new RemoteOrderApiModel(_apiClient,
ActiveSessionHostFacilityData);


            //MmsHubConfiguration

            _mmsHubConfigurationModel = new MmsHubConfigurationModel(_apiClient,
ActiveSessionHostFacilityData);

            _mmsHubConfigurationPresenter = new
MmsHubConfigurationPresenter(mmsHubConfigView, _mmsHubConfigurationModel,
_messageManager);


            //ParLevelConfiguration

            _parLevelConfigurationModel = new ParLevelConfigurationModel(_apiClient,
ActiveSessionHostFacilityData);

            _parLevelConfigurationPresenter = new
ParLevelConfigurationPresenter(parLevelConfigView, _parLevelConfigurationModel,
_messageManager);


            //VerityLinkConfiguration

            _verityLinkConfigurationPresenter = new
VerityLinkConfigurationPresenter(verityLinkConfigurationView, _healthCareSystemModel,
_departmentModel, _messageManager);


            //sendInventory

            _sendInventoryApiModel = new SendInventoryApiModel(_apiClient,
ActiveSessionHostFacilityData);
```

```
            //sentInventoryStatusHistory

            _lastSentInventoryHistoryView = new LastSentInventoryHistoryView();

            _lastSentInventoryHistoryApiModel = new
LastSentInventoryHistoryApiModel(_apiClient, ActiveSessionHostFacilityData);

            _lastSentInventoryHistoryPresenter = new
LastSentInventoryHistoryPresenter(_lastSentInventoryHistoryView,
_lastSentInventoryHistoryApiModel);


            //Substitution Items identification options

            _userPreferenceModel = new UserPreferenceModel(new ApiClient(),
ActiveSessionHostFacilityData);

            _substitutionOptionsModel = new
SubstitutionOptionsModel(_userPreferenceModel);

            _substitutionOptionsView = new SubstitutionOptionView();

            _substitutionOptionsPresenter = new
SubstitutionOptionsPresenter(_substitutionOptionsView, _substitutionOptionsModel);


            // Destinations

            _destinationsApiModel = new DestinationsApiModel(_apiClient,
ActiveSessionHostFacilityData);


            //Global Edit Destinations

            _globalEditModel = new GlobalEditModel(_apiClient,
ActiveSessionHostFacilityData);


            // User Reports

            _userReportsModel = _userReportsModel ?? new UserReportsModel(_apiClient,
ActiveSessionHostFacilityData);

            _manageReportUserModel = _manageReportUserModel ?? new
ManageReportUserModel();


            if (_userReportsPresenter == null)

            {

                _userReportsPresenter = new UserReportsPresenter(userReportsView,
_userReportsModel, _messageManager);

                _userReportsPresenter.RegisterEditFormEvent += OnRegisterEditFormEvent;

                _userReportsPresenter.RedirectEvent += RoutePreconfiguredReports;

            }


            _whereToReturn = ReportRoute.UserReportsRoute;
```

```
            // Schedule Reports

            _emailRecipientModel = _emailRecipientModel ?? new
EmailRecipientModel(_apiClient, ActiveSessionHostFacilityData);

            _scheduleReportsModel = _scheduleReportsModel ?? new
ScheduleReportsModel(_apiClient, ActiveSessionHostFacilityData);

            _smtpConfigurationModel = _smtpConfigurationModel ?? new
SMTPConfigurationModel(_pharmogisticsLogger);



            if (_scheduleReportsPresenter == null)

            {

                _scheduleReportsPresenter = new
ScheduleReportsPresenter(scheduleReportsView, _scheduleReportsModel, _messageManager);

                _scheduleReportsPresenter.RedirectEvent += RoutePreconfiguredReports;

            }



            if (_saveScheduleReportPresenter == null)

            {

                _saveScheduleReportLayoutManager = new SaveScheduleReportLayoutManager();

                _scheduleSettingsComposer = new ScheduleSettingsComposer();

                _deliverySettingsComposer = new DeliverySettingsComposer();

                _saveScheduleReportPresenter = new SaveScheduleReportPresenter(

                    saveScheduleReportView,

                    _scheduleReportsModel,

                    _smtpConfigurationModel,

                    _emailRecipientModel,

                    _messageManager,

                    _saveScheduleReportLayoutManager,

                    _scheduleSettingsComposer,

                    _deliverySettingsComposer,

                    _userReportsModel

                );

                _saveScheduleReportPresenter.RegisterUserReportPreviewFormEvent +=
OnRegisterUserReportPreviewForm;

                _saveScheduleReportPresenter.ReportDetailsClick +=
OnSaveScheduleReportButtonDetailsClick;

                _saveScheduleReportPresenter.RedirectEvent += RoutePreconfiguredReports;

            }



            _currentScheduleEventArgs = null;
```

```
            // User Reports - Schedule Reports Router
            if (_userReportsRouter == null)
            {
                _userReportsRouter = _userReportsRouter ?? new UserReportsRouter(
                    _userReportsPresenter,
                    _scheduleReportsPresenter,
                    _saveScheduleReportPresenter,
                    _messageManager,
                    _pharmogisticsLogger
                );
                _userReportsRouter.ShowTab = DisplayRptPanel;
                _userReportsRouter.GetTabName = GetTabName;
            }

            //Email Recipient List
            _emailRecipientView = _emailRecipientView ?? new
EmailRecipientView(SelectionDgv);
            _emailDomainModel = _emailDomainModel ?? new EmailDomainModel();
            _emailRecipientPresenter = _emailRecipientPresenter ?? new
EmailRecipientPresenter(_emailRecipientView, _emailRecipientModel, _messageManager);

            // Message Validation
            _allTranQModel = _allTranQModel ?? new AllTranQWithRequestsModel(_apiClient,
ActiveSessionHostFacilityData);
            _printLabelModel = _printLabelModel ?? new PrintLabelModel(_apiClient,
ActiveSessionHostFacilityData);

            if (_messageValidationSelectionPresenter == null)
            {
                _messageValidationSelectionPresenter = new
MessageValidationSelectionPresenter(this.messageValidationSelectionView);
                _messageValidationSelectionPresenter.MessageValidationPicksRedirect +=
OnMessageValidationRedirect;
            }

            if (_messageValidationUserColumnsSetupPresenter == null)
            {
                _messageValidationUserColumnsSetupView = new
UserPreferenceColumnsSetupView();
                _messageValidationUserColumnsSetupModel = new
UserPreferenceColumnsSetupModel(_apiClient, ActiveSessionHostFacilityData);
```

```
                _messageValidationUserColumnsSetupPresenter = new
UserPreferenceColumnsSetupPresenter(_messageValidationUserColumnsSetupView,
_messageValidationUserColumnsSetupModel);

            }


            if (_pickTransactionPresenter == null)

            {

                _pickTransactionPresenter = new
PickTransactionPresenter(pickTransactionView, _allTranQModel,
_messageValidationUserColumnsSetupView, _messageValidationUserColumnsSetupPresenter);

                _pickTransactionPresenter.MessageValidationSelectionRedirect +=
OnMessageValidationSelectionRedirect;

                _pickTransactionPresenter.MessageValidationLabelPreviewRedirect +=
OnShowLabelPreview;

            }


            if (_labelPreviewPresenter == null)

            {

                _labelPreviewPresenter = new LabelPreviewPresenter(labelPreviewView,
_printLabelModel, _messageValidationUserColumnsSetupPresenter, _messageManager,
_pharmogisticsLogger);

                _labelPreviewPresenter.MessageValidationPicksRedirect +=
OnMessageValidationPicksRedirect;

                _labelPreviewPresenter.EditLabelRedirect += OnEditLabelWindowRedirect;

            }

        }


        private void InitializeMvp()

        {

            _formularyItemListFiltersBinder = new
FormularyItemListFiltersBinder(_siteConfigurationModel, _supportDataModel,
_itemLocationsModel);

        }


        private bool IsWfaEnabledOnMultipleInstance()

        {

            var processName = Process.GetCurrentProcess().ProcessName;


            if (Process.GetProcesses().Count(p => p.ProcessName == processName) > 1)

            {

                var macAddress = SysTools.GetMacAddress();

                var ipAddress = SysTools.GetIPAddress();
```

```
                var computer = new
ComputerClient(dynamicData.ActivePharmogisticsApiAddress)
                .GetComputer(macAddress, ipAddress, true);

                return (computer != null);
            }

            return false;
        }

        private class ListMember
        {
            public ListMember()
            {
            }

            public string Description { get; set; }
            public int ID { get; set; }
        }

        private class IsaListMember : ListMember
        {
            public bool ExternalIsa { get; set; }
        }

        protected void UserIsIdleHandler(object sender, EventArgs e)
        {
            if (Main.staticData.TimeOut > 0)
            {
                Cursor.Current = Cursors.WaitCursor;
                _pharmogisticsLogger.LogInfo(nameof(UserIsIdleHandler),
$"User:{Main.staticData.UserName} is idle for over {Main.staticData.TimeOut} seconds");
                LogOff();
                Cursor.Current = Cursors.Default;
            }
            else
                AppIdleReset();
        }
```

```
        public void LogOff()
        {
            AppIdle.timer.Stop();
            wfaBirdsEyeViewTimer.Stop();
            _pharmogisticsLogger.LogInfo(nameof(Login), $"User:{Main.staticData.UserName}
logged off");

            // Close connection to scanner
            ScannerController.CloseScanner();

            // Pre-configured reports tab reset
            ResetPreconfiguredReportTabs();

            // Message Validation
            AllowMessageValidationViewsRefresh();

            var authMgr = new
AuthenticationManager(dynamicData.ActivePharmogisticsApiAddress);
            this.Visible = false;
            var frmControls = FindAllOpenForms().ToList();
            foreach (var frm in frmControls)
            {
                if (frm == this)//Don't close the current form
                    continue;
                frm.Visible = false;
                frm.DialogResult = DialogResult.Abort;
                frm.Close();
            }
            authMgr.UnAuthenticate(ActiveSessionHostFacilityData.ActiveApiKey);

            filterControl1.ClearFilters();
            SetFilterExpanded(false);
            Main.IsAuthenticated = false;

            if (!HandleUserLogin())
            {
                Application.Exit();
```

```
                return;
            }

            AppIdleReset();
        }
        public void AppIdleReset()
        {
            AppIdle.timer.Stop();
            uint idleInterval = (uint)((Main.staticData.TimeOut * 60) * 1000);
            AppIdle = new Idle(5000, idleInterval);
            AppIdle.UserIsIdle += new EventHandler(this.UserIsIdleHandler);
        }

        private void CreateDesktopIcons()
        {
            if (ApplicationDeployment.IsNetworkDeployed)
            {
                ApplicationDeployment ad = ApplicationDeployment.CurrentDeployment;

                if (ad.IsFirstRun)
                {
                    Assembly assembly = Assembly.GetEntryAssembly();
                    string company = string.Empty;
                    string description = string.Empty;

                    if (Attribute.IsDefined(assembly, typeof(AssemblyCompanyAttribute)))
                    {
                        AssemblyCompanyAttribute ascompany =
(AssemblyCompanyAttribute)Attribute.GetCustomAttribute(assembly,
typeof(AssemblyCompanyAttribute));
                        company = ascompany.Company;
                    }
                    if (Attribute.IsDefined(assembly,
typeof(AssemblyDescriptionAttribute)))
                    {
                        AssemblyDescriptionAttribute asdescription =
(AssemblyDescriptionAttribute)Attribute.GetCustomAttribute(assembly,
typeof(AssemblyDescriptionAttribute));
                        description = asdescription.Description;
                    }
```

```
                    if (!string.IsNullOrEmpty(company))
                    {
                        string desktopPath = string.Empty;
                        desktopPath =
string.Concat(Environment.GetFolderPath(Environment.SpecialFolder.Desktop), "\\",
description, ".appref-ms");

                        string shortcutName = string.Empty;
                        shortcutName =
string.Concat(Environment.GetFolderPath(Environment.SpecialFolder.Programs), "\\",
company, "\\", description, ".appref-ms");

                        try
                        {
                            if (System.IO.File.Exists(desktopPath))
                                System.IO.File.Delete(desktopPath);

                            System.IO.File.Copy(shortcutName, desktopPath);

                        }
                        catch (Exception ex)
                        {
                            MessageBox.Show(this, "Unable to Create Shortcut" +
Environment.NewLine
                                        + Environment.NewLine + ex.Message);
                        }
                    }
                }
            }
        }
        public bool PrepStaticData(string appMode, bool useAppConfig)
        {
            // Set application name
            string appTitle = Program.AssemblyTitle;
            this.Text = staticData.ApplicationName;
            WFA_ComputerLbl.Text = staticData.ComputerDescription;
            staticData.MainForm = this;
            return true;
        }
}
```

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_Куряков.doc	Пояснювальна записка до магістерської роботи. Документ Word.
Диплом_Куряков.pdf	Пояснювальна записка до магістерської роботи в форматі PDF.
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму.
Презентація	
Презентація_Рибка.ppt	Презентація до магістерської роботи