НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

на тему: Оптимізація процесу роботи госпіталів

Виконав (-ла): студент (-ка) IV курсу, групи ІО-72
(шифр групи)
Бутенко Владислав Олександрович
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н. Антонюк Андрій Іванович
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) професор, д.т.н. Сімоненко В. П. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент
(підпис)

Київ – 2021 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2021 р.

_

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

_____ *Бутенка Владислава Олександровича*_____

1. Тема проєкту *Отмізація процесу роботи госпіталів*
керівник проєкту Антонюк Андрій Іванович, доцент, к.т.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від _____ 2021 року №_____

2. Термін здачі студентом закінченого проєкту _____

3. Вихідні дані до проєкту *технічна документація. теоретичні дані*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Опис предметної області, і сучасних методів розв’язання поставленої задачі, опис та порівняння систем оптимізації роботи госпіталів, загальна структура системи.

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень)
Діаграма класів, Структурна схема, Схема алгоритму моделі.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В. П.		

7. Дата видачі завдання_____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2021-15.12.2021*	
2.	*Вивчення та аналіз завдання*	*15.12.2021-15.03.2021*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2021-25.03.2021*	
4.	*Розробка структур окремих підсистем*	*25.03.2021-5.04.2021*	
5.	*Програмна реалізація системи*	*5.04.2021-15.04.2021*	
6.	*Оформлення пояснювальної записки*	*15.04.2021-20.05.2021*	
7.	*Захист програмного продукту*	*25.04.2021*	
8.	*Передзахист*	*23.05.2021*	
9.	*Захист*	*15.06.2021*	

Студент-дипломник _____
(підпис)

Керівник проєкту _____
(підпис)

Анотація

В даній бакалаврській дипломній роботі була реалізована система електронної черги з автоматичною пріоритизацією вхідного потоку пацієнтів, яка оптимізує роботу госпіталя, медичного центру, або будь-якого іншого закладу охорони здоров’я.

Система дозволяє автоматизувати розподілення пацієнтів по вільним потужностям медичного закладу та формувати електронну чергу, враховуючи пріоритет кожного пацієнта, який був призначений йому при первинному огляді. Дана електрона черга надає можливість зменшити час контакту між хворими, і таким чином обмежити розповсюдження хвороби. Серверна частина цього програмного продукту була написана на мові програмування Java.

Аннотация

В данной бакалаврской дипломной работе была реализована система электронной очереди с автоматической приоритизацией входного потока пациентов, которая оптимизирует работу госпиталя, медицинского центра или любого другого учреждения здравоохранения.

Система позволяет автоматизировать распределения пациентов по свободным мощностям медицинского учреждения и формировать электронную очередь, учитывая приоритет каждого пациента, который был назначен ему при первичном осмотре. Данная электронная очередь дает возможность уменьшить время контакта между больными, и таким образом ограничить распространение болезни. Серверная часть этого программного продукта была написана на языке программирования Java.

Annotation

In this bachelor's graduate work, an electronic queue system with automatic prioritization of the input flow of patients was implemented, which can potentially

optimize the work operation of a hospital, a medical center or any other healthcare institution.

The system makes it possible to automate the distribution of patients according to the free capacities of a medical institution and to form an electronic queue, taking into account the priority of each patient which was assigned to him during the initial examination. This electronic queue makes it possible to reduce the time of contact between patients and thus limit the spread of the disease. The server side of this software product was written in the Java programming language.

Відомість дипломного проєкту

№ З.п.	Формат	Позначення	Найменування	Кілк. листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	*А4*	*ІАЛЦ.467200.001 ВП*	Відомість проєкту	1	
3	*А4*	*ІАЛЦ.467200.002 ТЗ*	Технічне завдання	4	
4	*А4*	*ІАЛЦ.467200.003 ПЗ*	Пояснювальна записка	82	
5	*А4*	*ІАЛЦ.467200.004 Д1*	Додаток 1	1	
6	*А1*	*ІАЛЦ.467200.005 Д2*	Додаток 2	1	
7	*А1*	*ІАЛЦ.467200.006 Д3*	Додаток 3	1	

Зм.		№ документа	Підп.	Дата	*ІАЛЦ.467200.001 ВП*		
Розробив		Бутенко В.О.			*Оптимізація процесу роботи госпіталів*	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Антонюк А.І.					
Н.контр.		Сімоненко В. П.				*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-72*	
Затв.		Стіренко С.Г,					

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи

освітньо-кваліфікаційного рівня бакалавр

на тему: «Оптимізація процесу роботи госпіталів»

Київ – 2021 р.

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

В рамках данної дипломної роботи є дослідження існуючих систем оптимізації роботи госпіталів на базі електронної черги, та створення окремого варіанту даної системи.

Область застосування: дана система призначена для будь-яких закладів надання медичних послуг, яким необхідно розподіляти вхідний потік пацієнтів по потужностях закладу. Цільові об’єкти – медичні центри, державні та приватні лікарні.

Наше технічне завдання також поширюється на курс «Оптимізація інформації у комп’ютерних системах». Область використання: практичне використання студентами та викладачами в освітньому процесі університету.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфіційноосвітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є створення серверної частини системи, яка буде розподіляти вхідний потік пацієнтів по потужностям медичного закладу, в залежності від пріоритету кожного пацієнта.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є електронна науково-технічна література про програмування, серверну архітектуру та алгоритми обробки пріоритезованих об’єктів; публікації в Інтернеті та окремих статтях, що висвітлюють дані питання.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробляємого продукту

- Розробка серверної архітектури відповідно сучасних тенденцій
- Розробка алгоритму ефективного розподілення вхідного потоку об'єктів згідно їх пріоритетів.
- Створення клієнтської частини для демонстрації роботи системи

5.2. Вимоги до програмного забезпечення

- Операційна система Windows або Linux
- Java 8, або новіші версії
- СКБД MySQL 5.0, або новіші версії
- Середа розробки Intellij Idea 2019, або новіші версії

5.3. Вимоги до апаратної частини.

- Процесор Intel або AMD.
- Оперативна пам’ять 4ГБ або більше.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	20.12.2020
Складання і узгодження технічного завдання	25.12.2020
Створення структури системи і окремих модулів	12.02.2021
Тестування окремих модулів системи	25.04.2021
Допрацювання, налагодження і виправлення помилок	10.05.2021
Оформлення документації дипломної роботи	20.05.2021

Пояснювальна записка
до дипломного проєкту
на тему: «Оптимізація процесу роботи госпіталів»

Київ - 2021 р.

ЗМІСТ

ВСТУП

Автоматизація та діджиталізація з кожним роком проникають все глибше у всі сфери людського життя, змінюючи та удосконалюючи вже застарілі бізнес процеси, які ще декілька років тому сприймалися як інноваційні. В усьому світі вже починається четверта промислова революція, яка за допомогою таких технологій як штучний інтелект (AI), робототехніка, великі дані (Big Data), інтернет речей (IoT), розвитку нанотехнологій та біотехнологій змінить світ не менше, ніж його змінив винахід парової машини, масова електрифікація або комп'ютеризація.

Власне здоров'є – це перша та найважливіша потреба будь-якої людини і саме в наш час галузь охорони здоров'я знаходиться на найвищому рівні розвитку за всю історію людства. До того ж, вона є як надзвичайно затребуваною, так і досить гнучкою, щоб в короткий час імплементувати в себе найновітні технології, бізнес-процеси та останні досягнення в медицині. На думку сучасних вчених та медиків, ця сфера знаходиться на етапі активного розвитку на даний момент, і в найближчому майбутньому ми станемо свідками виникнення великої кількості дизруптивних інновацій у сфері надання медичних послуг[1].

Ще у 2005 році ВООЗ випустила резолюцію щодо електронного здоров'я (WHA58.28 eHealth Resolution)[2], в якій відзначила наступне:

- Потенційний вплив, який можуть здійснити досягнення в галузі інформаційно-комунікційних технологій на надання медичних послуг, охорону здоров'я та діяльність, пов'язану зі здоров'ям на користь як країн з низьким, так і з високим доходом;
- Досягнення в галузі інформаційних та комунікаційних технологій значно підвищили очікування ВООЗ та можуть бути використані для вдосконалення сфери охорони здоров'я.

- Визначити, що сформована ВООЗ стратегія eHealth буде служити основою для діяльності ВООЗ щодо електронного здоров'я;
- Підкреслити, що електронне здоров'я – економічно ефективне та безпечне використання інформаційних та комунікаційних технологій на підтримку охорони здоров'я та галузей, які пов'язані зі здоров'ям.

Крім того, під час пандемії коронавірусної інфекції COVID-19 було виявлено, що існуючі підходи та технології надання невідкладної першої медичної допомоги під час надкритичного завантаження закладів є неефективними, що призводило до іноді нераціоанльного завантаження апаратів штучної вентіляції легень (ШВЛ) та великої смертності серед інфікованих.

Як корисна й перспективна галузь в сфері діджиталізації був обраний напрямок оптимізації роботи закладів надання медичних послуг через створення та імплементування системи електроних черг з пріорітетами, яка у реальному часі зможе правильно завантажувати усі можливі ресурси відділення та надавати необхідні медичні послуги спочатку тим, хто найбільше в них потребується.

Система, що розробляється, має наступні особливості:

- Система має чітке уявлення про теперешнє і майбутнє завантаження будь-якого свого ресурса (кабінета, врача, устаткування) та формує рішення про направлення пацієнта з найбільшою ефективністю.
- Кожний вхідний агент (пацієнт) має свій, призначений на первинному обслідуванні, пріоритет, який застосовується системою для формування персональної черги.
- Усі дані про пацієнта та результати його обслідування заносяться в електрону базу відділення, що повністю виключає заповнення медицинських карточок та значно зменшує документообіг.

Мета роботи. Створення системи електронної черги для закладів надання медичних послуг для оптимізації ефективності їх роботи та завантаженності.

Задачі роботи.

1. Дослідження проблеми, аналіз та оцінка можливих підходів до розв’язання даної задачі.
2. Аналіз існуючих рішень, оцінка їх ефективності, визначення переваг та недоліків.
3. Визначення основних функцій розроблюваної системи.
4. Побудова структури розроблюваної системи.
5. Розробка алгоритму функціонування.
6. Розробка системи електронної черги.
7. Тестування програми.
8. Розробка документації для користувачів.

Призначення. Система підходить для будь-яких закладів, які діють у сфері надання медичних послуг та мають безсистемний вхідний потік пацієнтів, котрий необхідно розподілити по наявним ресурсам заклада. Насамперед це:

- Приватні та державні клініки
- Приватні та державні медичні центри.
- Інші заклади, які надають не такий широкий спектр медицинських послуг, проте теж потребують раціонального розподілення клієнтів.

Імплементування даної системи дозволить вищеназваним закладам покращити завантаженість медичного персоналу та обладнання, пріоритезувати вхідний потік пацієнтів та надавати медичну допомогу насамперед тим, хто найбільш потребує її.

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

На сьогоднішний день ми бачимо чіткі тенденції діджиталізації сфери охорони здоров'я – на ринку з'являються компанії-розробники спеціалізованих систем та технологій, які оптимізують існуючі бізнес-процеси в закладах надання медичних послуг. Основні споживачі в Україні представлені двома групами:

Перша група – часні клініки та медичні центри. Вони активно придбають нові технології або імплементують спеціалізовані системи для того щоб покращити якість виконуваних робіт та послуг, що надаються, і, як результат - збільшити майбутні доходи та окупити витрати на придбані технології.

Друга група – державні госпіталі та клініки. Вони, на сьогодняшній день, не так активно здійснюють технологічні реформи, адже будь-які реформи державної організації фінансуються з державного бюджету. Проте, позитивні тендеції все ж існують. Первинна мережа лікарень у 2018 році була комп'ютеризована лише на 3%. На кінець 2019 року – комп'ютаризація торкнулася 97% лікарень. Це значно зменшило документообіг та дозволило зберігати дані пацієнтів в єдиному цифровому реєстрі[1]. В лютому 2021 року Кабінет Міністрів України разом із Міністерством Цифрової Трансформації сформував стратегію цифрової трансформації на наступні 3 роки. Усього було затверджено 94 проекта, серед яких є проекти для лікарень[2]. Отже, можна очікувати, що в найближчі три роки серед державних закладів охорони здоров'я збільшиться попит на проекти цифрової трансформації.

1.1 Актуальність впровадження системи електронної черги

Згідно зі звітом компанії Qminder[3], яка займається розробкою та імплементуванням QMS-систем (Queue management system – система управління чергою), заклади, що імплементують таку систему, отримують значні переваги, а саме:

- Мінімізація часу очікування пацієнтів, і як наслідок – зменшення черг в лікарнях.
- Збір даних для аналізу – система не тільки опитмізує потік пацієнтів, але й записує завантаженість закладу в кожен період часу. Аналізує ці дані, можна зробити висновки про швидкість надання різних послуг всередині лікарні та, за потребою, провести зміни у роботі закладу щоб покращити швидкість або якість надання послуг.
- Зменшення часу контакту між хворими та здоровими пацієнтами – завдяки електронній черзі, відвідувачі центрів охорони здоров'я будуть контактувати на протязі значно меншного періоду часу, що знизить ймовірність зараження здорових пацієнтів хворими.
- Аналіз відгуків пацієнтів – після надання послуг, пацієнту буде запропоновано пройти невелике опитування про якість отриманих послуг. Використовуючи ці дані, управління лікарні або медичного центру може робити висновки про робочий персонал та існуючи підходи лікування та, за потребою, вносити зміни, щоб більшість відвідувачів були задоволені.
- Планування завантаження персоналу та об'єктів всередині лікарні.
- Уникнення помилок – при використанні правильно налаштованої системи значно зменшується ймовірність виникнення помилки через людський фактор – наприклад, неправильний напрямок пацієнта до лікаря, якого немає на робочому місці.
- Зменшення документообігу – весь процес лікування записується у цифровий реєстр заклада, через який можна моментально отримати історію хвороби будь-якого пацієнта. Через це відпадає потреба у веденні документації (наприклад, медичних карток), що значно прискорює прийом лікаря.

1.2 Приклади реалізації

На даний момент існують готові рішення, розроблені компаніями або стартапами, які діють переважно у сфері eHealth. На ринку України найбільш розповсюджені наступні рішення:

1.2.1 Система електроної черги QualityQlick

Система працює за наступним алгоритмом[4]:

1. Клієнт обирає послугу на електронному терміналі, отримує талон з номером і кодом у черзі.
2. Оператор викликає клієнтів за допомогою віртуального пульта, коли необхідний кабінет звільниться. Після виклику, номер клієнта з’являється на інформаційному табло.
3. Послуги можуть являти собою маршрут, наприклад: «Кабінет 2» - «Кабінет 9» - «Віконце 3». На кожен етап може даватися часовий норматив надання послуги.
4. Після отримання послуги, для оцінки якості обслуговування клієнту може бути запропоновано пройти експерс-опитування на сенсорному міні-комп’ютері, наприклад iPad.

Також дана система дозволяє адміністратору налаштовувати перелік співробітників, ролі, послуги, маршрути, нормативи часу та алгоритми експрес-опитуван, формувати звіти з експортом даних в Excel.

Основні елементи системи можна побачити на Рис 1.1.

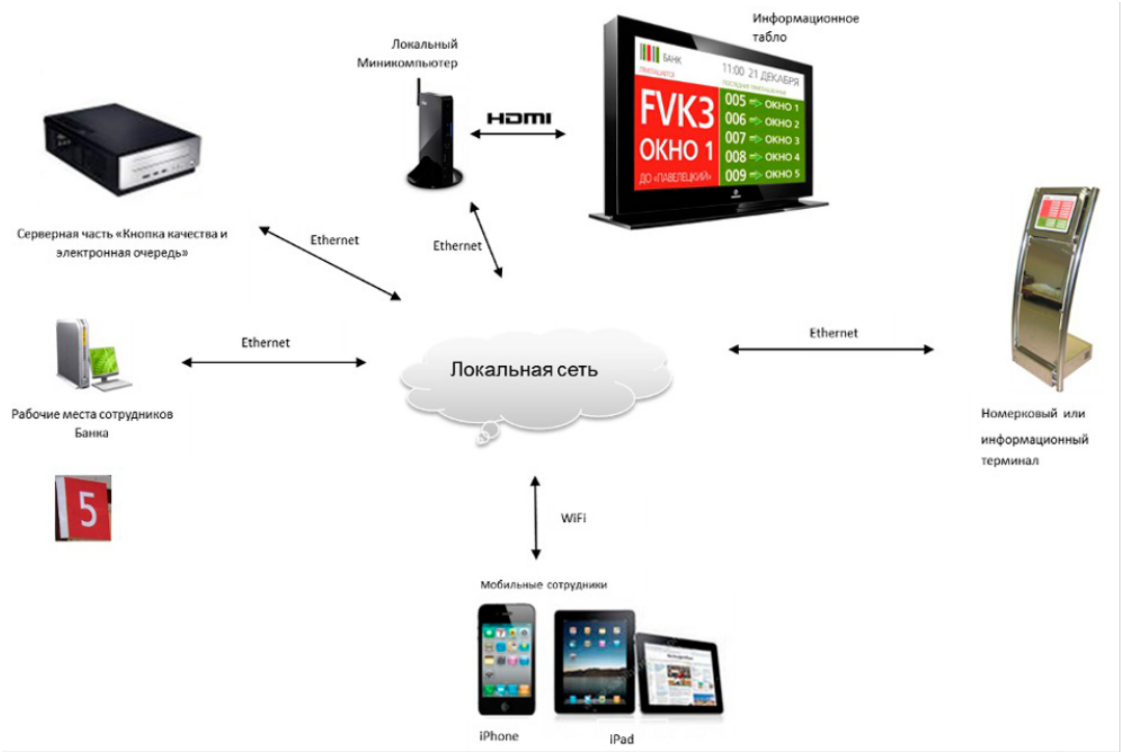


Рис. 1.1 Елементи системи QualityQlick

1.2.2 Електрона черга AKIS

Система працює за тим самим алгоритмом, що і система, описана в пункті 1.2.1, за винятком 4 – проходження експрес-опитування[5].

Основні елементи системи можна побачити на Рис. 1.2.

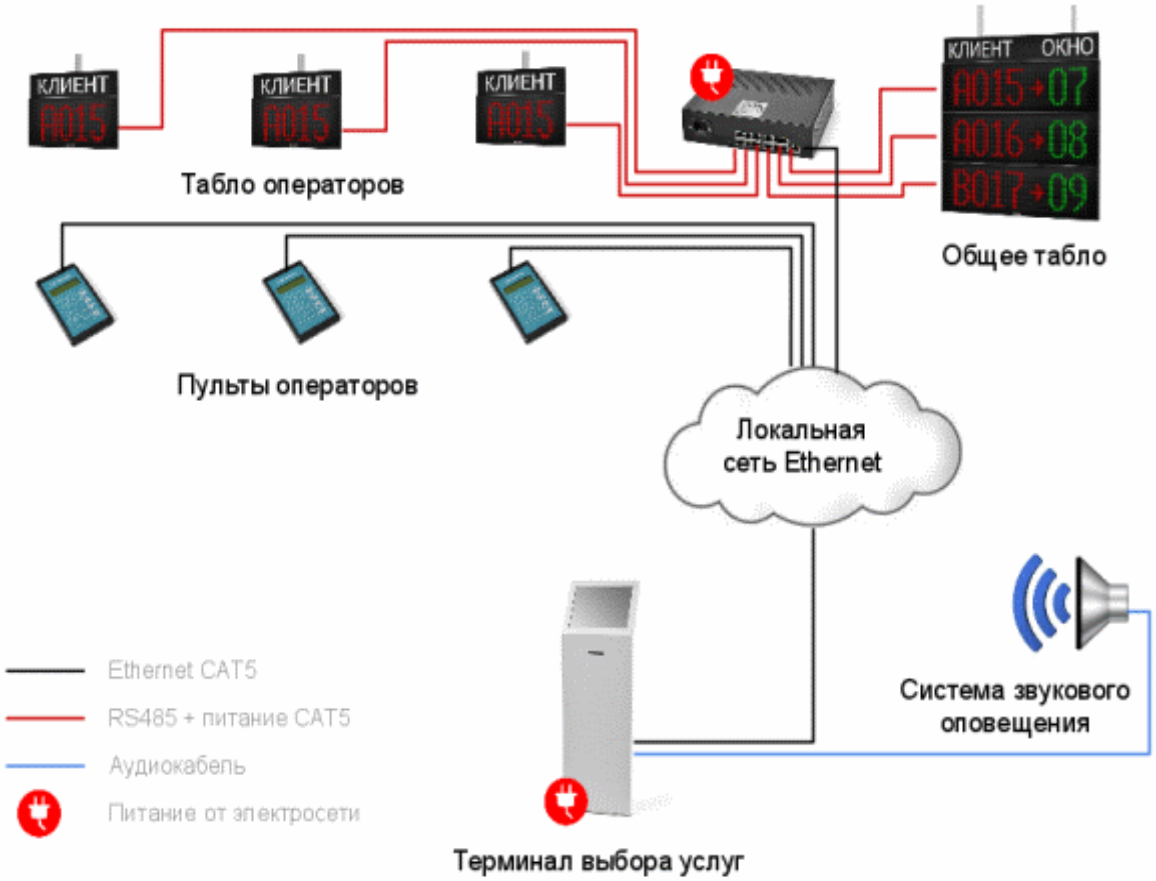


Рис. 1.2 Елементи системи AKIS

1.2.3 Система керування чергою SUO

Система працює за тим самим алгоритмом, що і система, описана в пункті 1.2.1, за винятком 4 – проходження експрес-опитування[6].

Дана система спрямована виключно на розподіл клієнтів серед кабінетів закладу та не реалізовано збереження даних для аналітики або ведення електронного реєстру.

1.3 Методології розробки та імплементації

Можемо розрязняти два основних підходи при розробці та імплементації систем керування чергою:

1 Індивідуально розроблені рішення «під ключ» - дана система розробляється для окремого медичного центру або клініки з урахуванням усієї специфіки роботи даного закладу: спектру послуг, що надається, та середній час надання цих послуг, кількість персоналу, приблизна кількість відвідувачів у розрізі днів, тіжнів та місяців, устаткування закладу та інші параметри, які відносяться лише до цього закладу. В програмі реалізуються основні модулі, які вже налаштовані під специфіку роботи усіх відділень закладу, які, на думку замовника, не будуть змінюватися у найближчому майботньому. Проте, саме рішення розробляється з урахуванням можливого майбутнього масштабування системи – збільшення кількості медичного персоналу, устаткування, можливість відкриття нових центрів надання послуг.

Переваги даного підходу наступні:

- Така програма буде повністю враховувати усі внутрішні бізнес-процеси клієнта, що дозволить мінімізувати майбутні допрацювання.
- Розроблене рішення імплементується лише один раз із мінімальними як грошовими, так і часовими витратами.
- При такому підході мінімізується «опір персоналу» (бажання персоналу використовувати старі рішення для виконання роботи замість нового), адже рішення засновано на тих бізнес-процесах, які протікали в компнії ще до імплементації.

Недоліки такого підходу:

- Рішення «під ключ» зазвичай дорожчі, ніж загальні. Це пов’язано з додатковими часовими витратами на аналіз специфіки бізнесу клієнта, залучення бізнес-аналітиків, розробці додаткових модулів за бажанням клієнта.

- Важко враховувати можливі зміни у бізнес-процесах клієнта, які або вже відбулись, або відбудуться до дати завершення розробки.
- Створене рішення, зазвичай, буде масштабуватися, тому необхідно правильно оцінити майбутній розвиток бізнеса клієнта та заложити в програму можливість масштабування у необхідних напрямках.

2 Розробка універсальної системи електроної черги – дана система розробляється з урахуванням загальних процесів, які присутні у більшості закладів, що надають медичні послуги. Проте, через специфічність роботи різних компаній, врахувати всі нюанси неможливо. Отже, розроблена система може покрити більшість потреб клієнта без допрацюівань (Out of the box), але, на вимогу окремих клієнтів, система може бути допрацьована для врахування усіх нюансів роботи клієнта.

Плюси даного підходу:

- Готову програму можна дистрибутувати широкому спектру клієнтів, бізнес-потреби яких частково або повністю покриті створеним рішенням.

Недолікі такого підходу наступні:

- При недостатньому соответствии програми бізнес-процесам клієнта необхідно будет витрачати час для її доробку під окремого клієнта, що результує у додактову ціну системи, що імплементується.
- Імплементація рішення проходить важче, «опір персоналу» вищий через вирогідну зміну існуючих бізнес-процесів після запуску системи.
- Майбутнє масштабування може бути неможливим у дистрибутивній версії, , що може призвести або до додаткових витрат на додання

нового функціоналу у майбутньому, або й зовсім до відмови клієнта від створеного рішення.

Можна зробити висновок, що єдиної методології розробки такої системи не існує. Менеджмент компанії сам вирішує який спосіб застосовувати, враховуючи специфіку ринку.

ВИСНОВКИ ДО РОЗДІЛУ 1

В результаті аналізу предметної області та існуючих рішень було встановлено, що, на даний момент, існують програмні рішення електронної черги. Проте, рішення, в якому була б реалізована система автоматичної пріоритизації пацієнтів в залежності від його стану, не було реалізовано в жодному рішенні. Медичний заклад, в якому буде встановлена система електронної черги з такою функціональністю може значно покращити ключові показники обслуговування клієнтів та врятувати багато життів. Для демонстрації можливостей і переваг такого підходу, було прийняте рішення розробити такий функціонал в рамках даної дипломної роботи.

РОЗДІЛ 2
ОПИС ТЕХНОЛОГІЙ ТА АЛГОРИТМІВ ДЛЯ РОЗРОБКИ СИСТЕМИ

2.1 Опис алгоритму функціонування електронної черги з пріоритетами

В основі розроблюваної системи знаходиться логіка розподілу пацієнтів за потужностями клініки, в залежності від пріоритету пацієнта. Для ефективного визначення даного пріоритету необхідно розглянути, обрати і реалізувати алгоритм черги з пріоритетами. В даному розділі будуть розглянуті різні варіанти алгоритмів черги з пріоритетами.

Черга з пріоритетами (англ. priority queue) — це структура даних, що призначена для обслуговування множини елементів, кожний з яких додатково має свій "пріоритет", пов'язаний з ним. У пріоритетній черзі першим обслуговується елемент, який має найвищий пріоритет, відповідно елемент, що має найнижчий пріоритет буде обслугований останнім. У деяких реалізаціях, якщо два елементи мають однаковий пріоритет, вони подаються відповідно до порядку, в якому вони були закладені, в той час як в інших реалізаціях упорядкування елементів з однаковим пріоритетом не визначено[7].

Хоча черги з пріоритетами часто реалізуються купами, вони концептуально відрізняються від них. Черга пріоритетів - це абстрактне поняття, як "список" або "карта"; так само, як список може бути реалізована зв'язаним списком або масивом, черга з пріоритетом може бути реалізована купою або іншими методами, наприклад таким як невпорядкований масив.

Черга з пріоритетами повинна принаймні підтримувати такі операції:

- *is_empty* – перевіряє, чи є черга порожнею.
- *insert_with_priority* – додання елементу до черги (массиву) з вказанням його пріоритету. Також можливий другий варіант цієї

операції – *insert_without_priority*, або *insert_with_default_priority* – додання елементу до черги, коли його пріоритет ще невідомий. В такому випадку, елемент все одно додається до черги, але йому встановлюється пріоритет за замовчуванням – наприклад, 0.

- *pull_highest_priority_element* – вилучення елемента з найбільшим пріоритетом із черги та повернення його.
- *pull_lowest_priority_element* – вилучення елемента з найменшим пріоритетом із черги та повернення його.
- *find_max* – пошук елемента з найбільшим пріоритетом та повернення його індексу.
- *find_min* – пошук елемента з найменшим пріоритетом та повернення його індексу.
- *delete_max* – пошук елемента з найбільшим пріоритетом та видалення його.
- *delete_min* – пошук елемента з найменшим пріоритетом та видалення його.
- *change_priority* – зміна значення пріоритету певного елементу. Як аргумент може бути переданий або сам елемент, або його індекс в черзі.
- *merge* – об’єднання двох пріоритетних черг, зберігаючи оригінальні черги.
- *meld* – об’єднання двох пріоритетних черг, руйнуючи оригінальні черги.
- *split* – розділення початкової пріоритетної черги на дві або більше частин.

Операції *Pull_highest_priority_element* (*або Pull_lowest_priority_element*) можуть мати інші назви - *pop_element, get_maximum_element* (*get_minimum_element*) або *get_front (most) _element*.

До того ж, при реалізації данного алгоритму часто використовується операція peek (або у цьому контексті вона може мати назви find_max або find_min). Ця операція повертає елемент з найвищим (або найнижчим) пріоритетом, але не змінює чергу та не видаляє елемент, що повертається із черги. Така операція майже завжди виконується за час O(1) – тобто кількість операцій, необхідних для повернення максимуму або мінімуму із черги, завжди дорівнює 1.

Це відбувається через те, що черга є заздалегідь відсортованою за зменшенням (або зростанням) пріоритетів, і для повернення елементу з найбільшим (або найменшим) пріоритетом достатньо лише знайти та повернути перший (або останній) елемент. Ця операція та її продуктивність можуть мати вирішальне значення при розробці багатьох програм пріоритетних черг, адже вона гарантує майже моментальне виконання операції повернення максимуму або мінімуму, не зважаючи на розмір черги та величину пріоритетів.

Стеки та звичайні черги відрізняються від черг з пріоритетами. У черзі пріоритетів порядок є внутрішнім - це залежить від значення пріоритетму того елемента, що був вставлений в масив. У стеку або звичайній черзі порядок є зовнішнім: порядок елементу визначається від того порядку, в якому значення було додано. З точки зору поведінкового підтипу, черга не є підтипом черги пріоритетів, а черга пріоритетів не є підтипом черги. Жоден з них не може бути замінений іншим, адже вони реалізують різну функціональність та використовуються для різних цілей [8].

Розглянемо різні варіанти реалізації черг з пріоритетами.

2.1.1 Наївна реалізація (англ. Naive implementation)

«Наївна реалізація» - це поширений термін із програмування, який є досить нечітким. По суті він значить, що для вирішення комплексної проблеми

застосовується дуже прямий та простий підхід, який не враховує усіх унікальних особливостей цієї проблеми. Це не значить, що рішення є неробочим і його не можна використовувати, проте для оптимізації роботи і для уникнення майбутніх помилок підхід бажано змінити.

Як наївну реалізацію черги з пріоритетами можно розглянути використання звичайної черги і при додаванні нового елемента класти його в кінець, а при запиті елемента з максимальним пріоритетом проходити по всьому списку. Тоді операція insert буде виконуватися за O(1), а extract_Min або extract_Max за O(n).

Ця реалізація є робочою якщо розмір черги є невеликим та операції повернення елементів виконуються нечасто. Проте ця реалізація стає неефективною при великій кількості елементів черги, адже тоді буде виконуватися багато операцій extract_Min або extract_Max, при яких ми будемо втрачати багато часу на проходження всієї черги.

Реалізація на псевдокоді може виглядати наступним чином:

Додавання нового елемента:

```
insert(node) {
  list.append(node)
}
```

Повернення елементу із найбільшим пріоритетом:

```
pull() {
  foreach node in list {
    if highest.priority < node.priority {
      highest = node
```

```
    }
  }
  list.remove(highest)
  return highest
}
```

В іншому випадку можна зробити навпаки – використовувати звичайну чергу, але при доданні кожного нового елементу з власним пріоритетом заново відсортовувати усю чергу так, щоб першим елементом у черзі знаходився елемент з найбільшим пріоритетом. Тоді, при запиті елемента із найбільшим (або найменшим) пріоритетом, достатньо буде повернути перший (або останній) елемент черги. При такому підході час виконання сортування-вставки дорівнює O(n), а час повернення елементу з найбільшим (або найменшим) пріоритетом дорівнює O(1). Це підхід ефективен при великій кількості операцій повернення елементу з найблішим (або найменшим) пріоритетом, адже незважаючи на розмір черги ця процедура буде виконана за одну операцію. Проте, якщо у програмі необхідно часто додавати нові елементи в чергу, необхідно буде кожен раз заново сортувати масив, що є досить затратною операцією.

Реалізація на псевдокоді може виглядати наступним чином:

Додавання нового елемента:

```
insert(node) {
  foreach (index, element) in list {
    if node.priority < element.priority {
      list.insert_at_index(node,index)
```

```
    }
  }
}
```

Повернення елементу із найбільшим пріоритетом:

```
pull() {
    highest = list.get_at_index(list.length-1)
    list.remove(highest)
    return highest
}
```

Кожен із цих підходів має свої переваги і недоліки. Перший варіант реалізації слід використовувати коли є велика потреба у частому доданні нових елементів до черги, а пошук та повернення елементу з найбільшим (найменшим) пріоритетом виконується досить рідко.

Другий варіант реалізації слід використовувати в тому випадку, коли є потреба у частому пошуку елементу з найбільшим (найменшим) пріоритетом, а додання нових елементів відбувається нечасто.

Але всеодно розглянуті вище реалізації пріоритетної черги є «наївними реалізаціями» і мають значні недоліки при виконання деяких операцій, або коли розмір черги стає критично великим. Для кращої продуктивності пріоритетні черги реалізують за допомогою куп, що дозволяє виконувати операції вставки і видалення за O(log n). Використання спеціальних куп, таких як Двобатьківська купа або 2-3 купа, дозволяє ще більше поліпшити асимптотику деякий операцій.

2.1.2 Купа та двобатьківська купа (англ. Bi-parental heap або Beap)

Спочатку розглянемо як працює структура даних типа «купа» в програмуванні.

Купа – це структура даних, яка підтримує наступні операції [9]:

1. Додати елемент в структуру даних.
2. Вилучити зі структури даних найбільший (або найменший) елемент. Об’єкт, що вилучається, буде видалений зі структури

При цьому в цієї структурі можуть зберігатися однакові елементи.

Якщо реалізувати таку структуру на базі списку, то додавати елементи можна в кінець списку за час O(1), але пошук найбільшого елемента буде займати O(n), якщо n - число елементів в списку, так як доведеться переглядати всі елементи списку і вибирати з них найбільший (або найменший).

Якщо ж зберігати елементи в списку упорядкувавши їх за зростанням, то вилучення найбільшого елемента займатиме O(1), але додавання елемента в список буде займати час O(n), так як доведеться зсувати елементи, які знаходяться у списку, для того щоб вставити новий елемент на правильне місце.

Спеціальна структура даних «Купа» (англ. Heap) дозволяє виконувати ці операції за O(log n) час, завдяки чому, при занадто великих значеннях n, операції вилучення та додавання елементів виконуються швидше.

У купі елементи зберігаються у вигляді двійкового дерева, тобто у елементів є два нащадка - лівий і правий. У вершині купи знаходиться один елемент, у нього - два нащадка на наступному рівні, у них, в свою чергу, ще два нащадка на третьому рівні (разом - 4 елементи на третьому рівні) і так далі. Рівні заповнюються в порядку збільшення номера рівня , а сам рівень заповнюється зліва направо. У елементів останнього рівня немає жодного

нащадка. Також в купі може бути один елемент, у якого тільки один нащадок (лівий).

При цьому для елементів купи існує наступна властивість - кожен з елементів купи більше своїх нащадків, або дорівнює їм. Зокрема це означає, що в вершині купи зберігається найбільший елемент.

Приклад купи з 9 елементів наведена на Рис. 2.1.

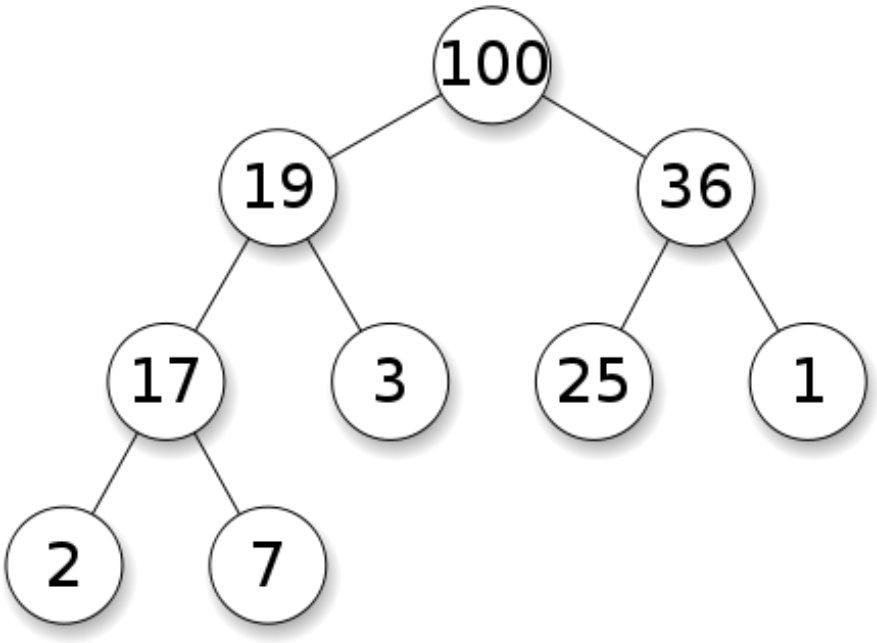


Рис. 2.1 Графічне зображення купи з 9 елементів

Всі ці елементи зручно зберігати в списку, починаючи з кореневого елемента. Для простоти нумерації пропустимо, що у списку існує нульовий елемент. Тоді вершина купи буде зберігатися в елементі списку з індексом 1. Інші елементи купи зберігаються поспіль в елементах списку з індексами 2, 3, 4 і так далі. Тобто, купа, що наведена на малюнку вище, будет зберігатися у списку наступним чином [100, 19, 36, 17, 3, 25, 1, 2, 7].

Можна побачити, що у елемента $H_{[i]}$ лівим нащадком є елемент $H_{[2 * i]}$, а правим нащадком - елемент $H_{[2 * i + 1]}$. А батьківським елементом для $H_{[i]}$ є елемент $H_{[i // 2]}$.

Одне з найбільш відомих застосувань купи - сортування за допомогою купи або пірамідальна сортування (англ. Heapsort). У даному сортуванні з елементів списку спочатку будується купа, а потім елементи по одному видаляються з купи - спочатку найбільший елемент, потім - найбільший з решти і так далі. При цьому купу можна зберігати там же, де зберігаються елементи самого списку. Таким чином, пірамідальне сортування має складність O(n*log n), але при цьому не вимагає додаткової пам'яті (як сортування злиттям) і не є ймовірнісної (як швидке сортування Хоара).

Також за допомогою купи можна організувати структуру даних «чергу з пріоритетами». У черзі кожному елементу зіставляється пріоритет - деяке ціле число. При видаленні елемента з черги видаляється не той елемент, який був доданий раніше (як у звичайній черзі), а елемент з найбільшим пріоритетом. Тобто елементи в черзі з пріоритетами можна зберігати в купі, порівнюючи їх при цьому за пріоритетом.

У черзі з пріоритетами також є операція зміни пріоритету елемента. Для цього реалізовані дві функції - збільшення та зменшення пріоритету. При збільшенні пріоритету елемент піднімається вгору, тому ця функція реалізована аналогічно операції додавання елемента. При зменшенні пріоритету елемент спускається вниз, як в операції видалення елемента.

Після того, як ми розглянули структуру даних типа «купа», можна розглянути різні варіації даної структури даних. Перший з них – Bi-parental heap.

Bi-parental heap, або **Beap**, яка ще називається двобатьківскою купою - це неявна структура даних, яка дозволяє ефективно вставляти та шукати

елементи, не вимагаючи при цьому більше місця, ніж самі елементи даних. Ефективний пошук та вставка підтримуються розумним розташуванням елементів у масиві, що підтримує структуру beap, та розумними алгоритмами, що використовують це розташування [10].

У Bi-parental heap кожний вузол, як правило, має двох батьків (якщо він не є першим або останнім) і двох дітей (якщо це не на останньому рівні). На відміну від купи, Beap дозволяє сублінійний пошук. Цей алгоритм був представлений Ян Манро та Хендрою Сувандою. Пов'язана структура даних - таблиця Юнга.

Графічно Beap можна представити у вигляді графа, що зображений на Рис. 2.2.

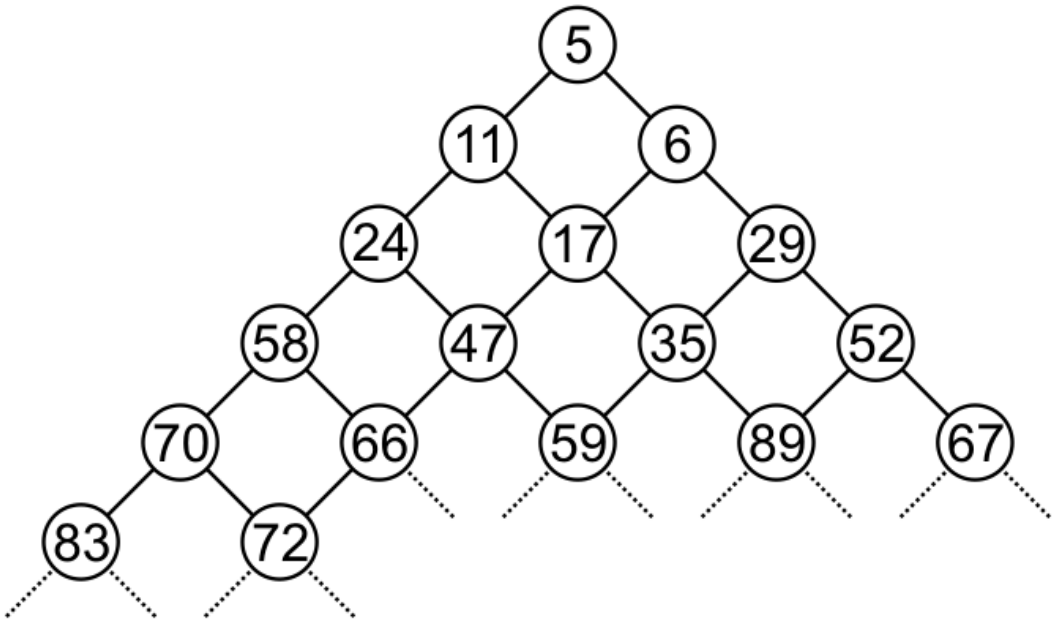


Рис. 2.2 Графічне зображення Beap

Висота структури становить приблизно √n. Крім того, якщо припустити, що найнижчий рівень повністю заповнений, кількість елементів на цьому рівні також дорівнює √n. Завдяки цим властивостям усі основні операції CRUD (C - Create – створення, R – Read – читання, U – Update – оновлення, D – Delete - видалення) виконуються всередньому за час O(√n). Знайти елементи в купі

можна за час O(n) у гіршому випадку. Видалення та вставка нових елементів передбачає зміщення елементів вгору або вниз (приблизно так, як у купі), для того щоб елементи з найбільшим пріоритетом знаходились вгорі купи, а елементи з найменшим пріоритетом – внизу. Додатковою перевагою Bi-parental heap є те, що цей алгоритм забезпечує постійний час доступу O(1) до найменшого елемента та O(√n) час для максимального елемента.

При роботі цього алгоритму використовується AVL-дерево, що призводить до того, що використання пам’яті становить Θ(n). Це досягається завдяки тому, що AVL-дерево - це перш за все двійкове дерево пошуку, ключі якого задовольняють стандартній властивості: ключ будь-якого вузла дерева не менш будь-якого ключа в лівому субдереві даного вузла і не більше будь-якого ключа в правому субдереві цього вузла.

Особливістю AVL-дерева є те, що воно є збалансованим в наступному сенсі: для будь-якого вузла дерева висота його правого субдерева відрізняється від висоти лівого субдерева не більше ніж на одиницю. Доведено, що цієї властивості досить для того, щоб висота дерева логарифмічно залежала від числа його вузлів: висота h AVL-дерева з n ключами знаходиться в діапазоні від log2 (n + 1) до 1.44 log2 (n + 2) - 0.328. А так як основні операції над двійковими деревами пошуку (пошук, вставка і видалення вузлів) лінійно залежать від його висоти, то отримуємо гарантовану логарифмічну залежність часу роботи цих алгоритмів від числа ключів, що зберігаються в дереві.

Вставки, або push-операції в Beap відбуваються в останньому доступному порожньому місці, як зображено на Рис. 2.3.

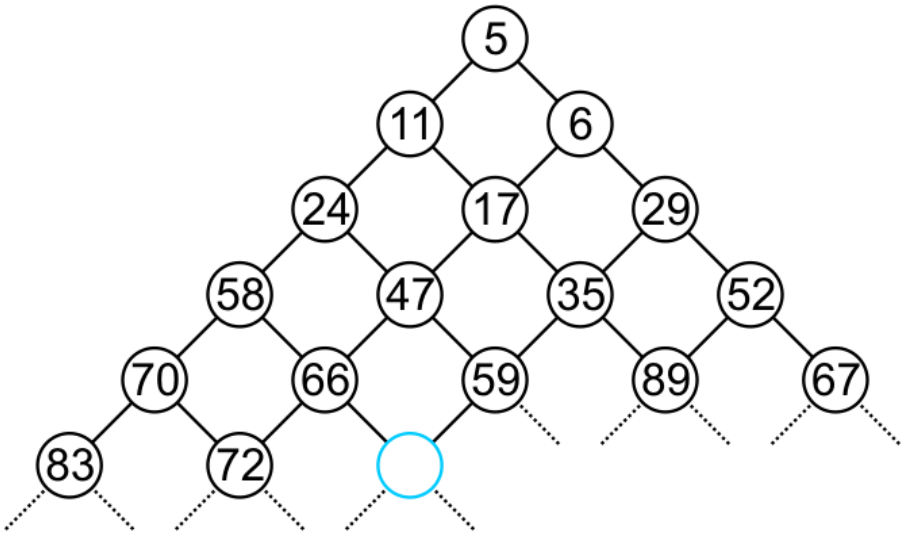


Рис. 2.3 Push-операція у Beap

Пошук може виконуватися за час виконання O(√n), як показано на Рис. 2.4, який демонструє логіку пошуку значення 44:

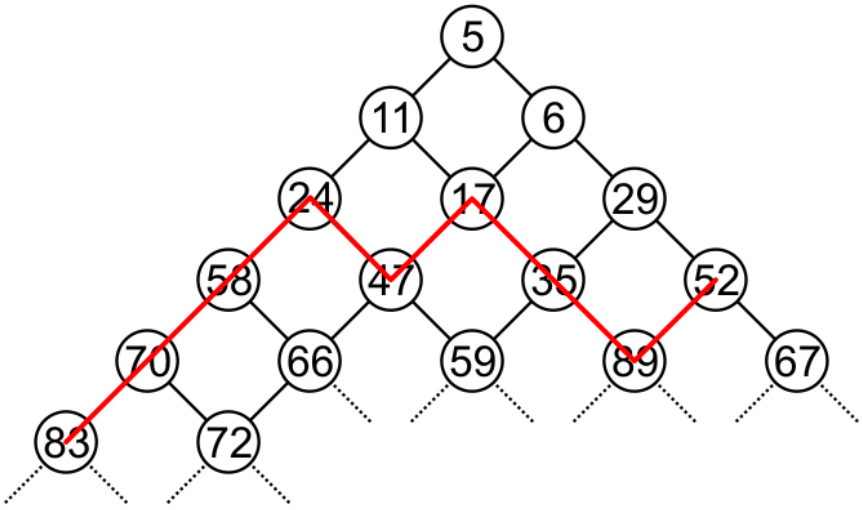


Рис. 2.4 Пошук у Beap

Логіка пошуку наступна: пошук починається з найбільшого елементу масиву. У випадку, коли елемент, що опрацьовується, більше ніж шуканий, алгоритм сходить до пов’язаного батьківского елементу, що знаходиться зверху. Цей елемент порівнюється із шуканим значенням, та, у випадку якщо вони рівні, повертається. В іншому випадку, алгоритм спускається до

дочірнього елементу (але не до того, звідки потрапили до нинішнього батьківского вузла) та повторює операцію.

2.1.3 Двійкова купа (англ. Binary heap)

Іншим варіантом реалізації пріоритетної черги може служити двійкова купа[11].

Двійкова купа (піраміда, сортуюче дерево, binary heap) - це повне бінарне дерево, де підтримується властивість порядку розміщення вершин (heap order property).

Повне бінарне дерево (Complete binary tree) – це дерево, у якому останній рівень повністю заповнений. Приклад повних бінарних дерев наведено на Рис. 2.5.

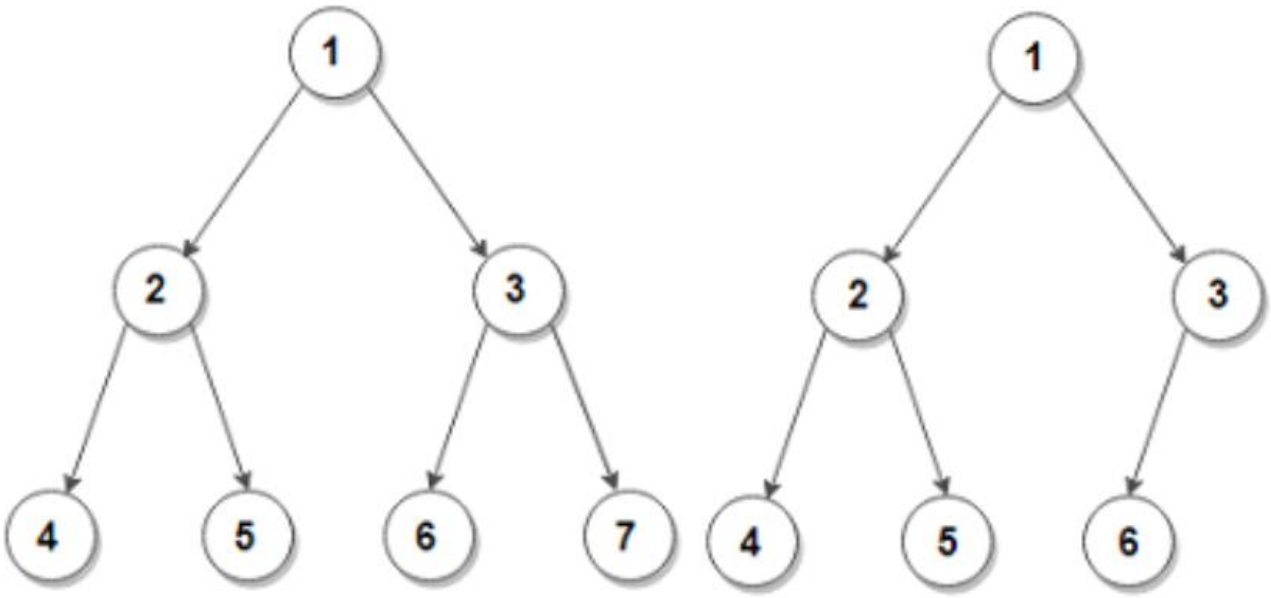


Рис. 2.5 Приклад повних бінарних дерев

Неповне бінарне дерево (Incomplete binary tree) – це дерево, у якому останній рівень повністю не повністю. Приклад неповного бінарного дерева наведено на Рис. 2.6.

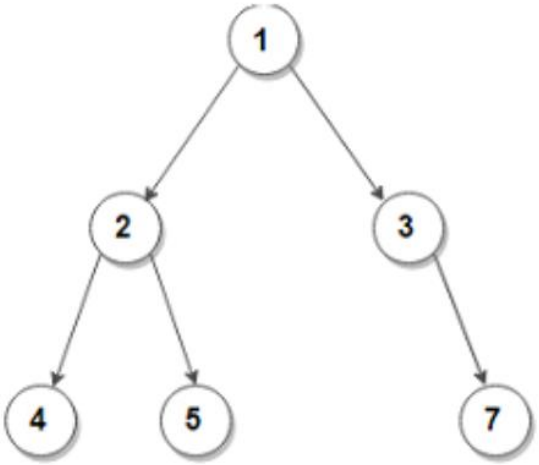


Рис. 2.6 Приклад неповного бінарного дерева

Властивість порядку розміщення вершин (heap order property) полягає в тому, що двійкова купа може бути максимальною (maximum heap) або мінімальної (minimum heap):

- maximum heap - якщо значення (пріоритет) в кожному вузлі більше або дорівнює значенням в вузлах нащадків;
- minimum heap - якщо значення (пріоритет) в кожному вузлі менше або дорівнює значенням в вузлах нащадків.

Однією з корисних властивостей повного бінарного дерева є те, що його можна представити у вигляді масиву. Наприклад, на Рис. 2.7 зображене повне бінарне дерево.

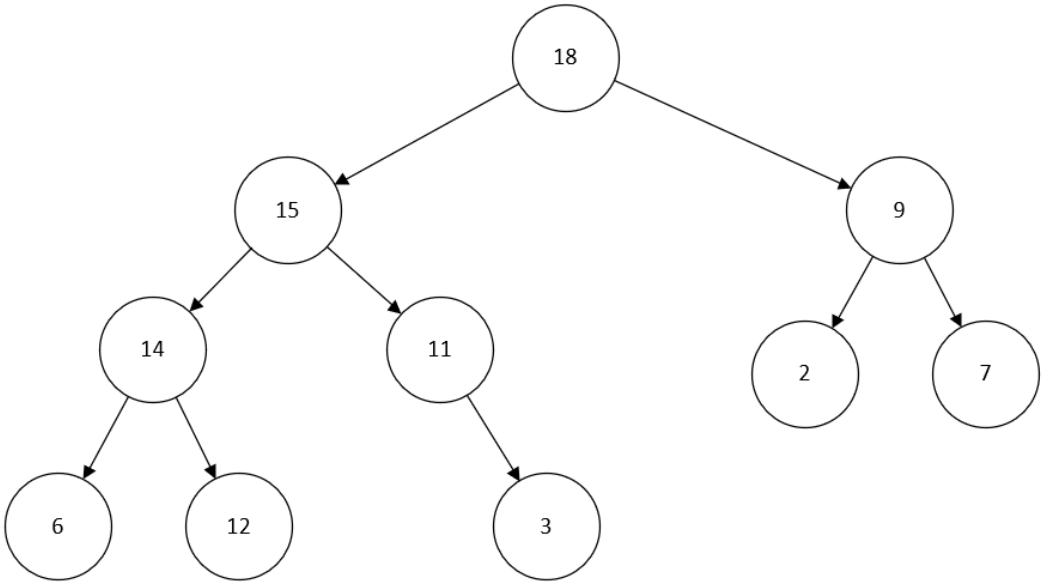


Рис. 2.7 Бінарне дерево, що можна представити у вигляді масиву

Його можна представити у вигляді масиву: [18, 15, 9, 14, 11, 2, 7, 6, 12, 3]. Перший елемент масиву відповідає кореню дерева – найбільшому (або найменшому, якщо властивість порядку дерева - minimum heap) елементу, а його дочірні елементи записуються в масив після нього.

Binary heap дуже зручно використовувати для реалізації черги з пріоритетами, адже вона може працювати як по вставці, так і по вилученню. Якщо черга працює по вставці, то при додаванні нового елемента ми шукаємо в черзі позицію, в яку буде проведена вставка. Якщо черга працює по вилученню, то при вилученні елемента з черги, ми шукаємо позицію, з якої елемент повинен бути видалений.

Операції, що повинна реалізовувати черга з пріоритетами, побудована за допомогою binary heap:

- *pq_insert (PQ, e)* - додати в чергу PQ елемент e;
- *pq_delete (PQ)* - видалити з черги PQ елемент з найвищим пріоритетом;

- *pq_peek (PQ)* - повертає із черги PQ елемент з найвищим пріоритетом (без видалення);
- *pq_size (PQ)* - повернути кількість елементів в черзі PQ;
- *pq_is_empty(PQ)* - перевірити чи є черга PQ порожнею.

Кількість необхідних залежить від кількості рівнів та кількості елементів купи. Наприклад, для виконання операції додавання елементу *pq_insert* необхідно загалом O(log n) операцій – це пов'язано з кількістю рівнів, на який елемент повинен піднятися щоб задовольнити властивості порядку розміщення вершин.

Операція повернення та видалення виконується у бінарній купі також за O(log n) операцій, через те що видалення елементу створює порожнє місце у дереві, яке повинен зайняти такий елемент, сумма нащадків якого була більше (або менша, якщо корін дерева – це елемент із найбільшим пріоритетом) цього елементу. Через це необхідно підібрати правильний елемент, який замінить видаленний та, за необхідністю, перегрупувати інші елементи дерева.

Пошук довільного елемента займає O(n) часу – як і в стандартному алгоритмі пошуку в бінарному дереві час виконання лінійно залежить від загальної кількості елементів купи.

2.1.4 Біноміальна купа (англ. Binominal heap)

Біноміальна купа - це множина біноміальних дерев з наступними обмеженнями:

1) У кожному з біноміальних дерев зберігається властивість купи.

2) Немає двох дерев однакового розміру

3) Дерева впорядковані за розміром.

Реалізація біноміальної купи будується на принципі роботи біноміального дерева. Розглянемо спочатку властивості біноміального дерева.

Біноміальне дерево – це дерево, яке задається рекуррентно:

- B_i – це B_{i-1}, в якому лівим нащадком корневого елемента є дерево B_{i-1}
- B_0 – звичайна вершина.

Приклад біноміального дерева для вершин B0, B1, B2 наведено на Рис. 2.8.

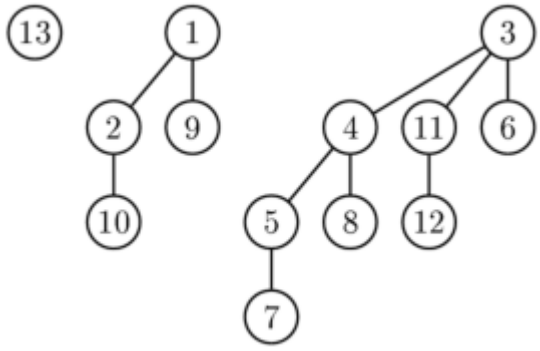


Рис. 2.8 Біноміальне дерево

Біноміальне дерево має наступні властивості:

1. Кількість вершин - 2^k;
2. Висота дерева – k;
3. Кількість вершин глибини - C^i_k (що є біноміальним коефіцієнтом);
4. Нащадки кореня – це B, B_{k-2}, …, B_0;
5. Максимальна висота вершини – O(log n), де n – кількість елементів у купі.

Сама біноміальна купа – це множина біноміальних дерев, яка зазвичай зберігається в програмі у вигляді списку за правилом «лівий син – правий брат», в якому корені біноміального дерева будуть представлені в порядку зростання висоти.

Операції біноміальної купи будуть наступні:

1. *insert (H, k)* - вставляє ключ «k» в біноміальну купу «H». Ця операція спочатку створює біноміальну купу з одним ключем «k», потім виконує об'єднання з H, у результаті чого з’являється нова біноміальна купа.
2. *getMin (H)* - виконання операції getMin () полягає в тому, щоб переглянути список коренів біноміальних дерев і повернути мінімальний ключ. Ця реалізація вимагає часу O(log n). Його можна оптимізувати до O (1), підтримуючи покажчик на мінімальний корінь ключа.
3. *extractMin (H)* - ця операція використовує операцію *union()*. Спочатку викликається *getMin()*, щоб знайти мінімальний ключ біноміального дерева, потім цей вузол видаляється і створюється нова біноміальнима купа, поєднуючи все піддерева віддаленого мінімального вузла. Після цього, викликається *union()* для H і нещодавно створеної біноміальної купи. Ця операція вимагає часу O(log n).
4. *delete (H)*: як і в Binary Heap, операція видалення спочатку зменшує ключ до мінімального, а потім викликає *extractMin()*.
5. *union()* - ця операція, що сполучає дві біноміальні купи в одну, використовується в якості підпроцесу більшістю інших операцій. Ось в чому полягає її суть: нехай є дві біноміальні купи з *H* і *H'*. Розміри дерев в купах відповідають двійковим числам m і n, тобто при наявності дерева відповідного порядку в цьому розряді числа буде стояти одиниця, в іншому випадку - нуль. При додаванні стовпчиком в двійковій системі відбуваються переноси, які відповідають злиттю двох біноміальних дерев B_{k-1} в дерево B_k. То дерево, в якому корін менше, буде враховуватися верхнім (приклад роботи операції *union()* наведено на Рис. 2.9).

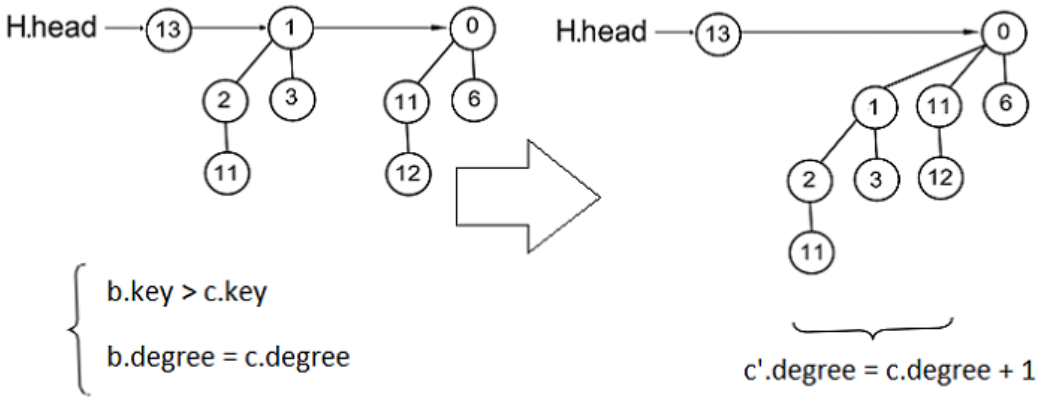


Рис. 2.9 Злиття двох дерев

Біноміальна купа підходить для реалізації пріорітетної черги для розроблюваної системи, адже основні операції відбуваються за час O(log n), що є відносно малим порівняно з іншими алгоритмами [12].

2.1.5 2-3 купа (англ. 2-3 heap)

2-3 купа (2-3 heap) - це структура даних типу дерево, яка задовольняє властивості купи. Є альтернативою купи Фібоначчі (Fibonacci heap). Складається з 2-3 дерев.

Рішення, яке пропонує реалізація 2-3 heap, полягає в тому, що на відміну від Купи Фібоначчі, 2-3 купа виконує балансування не тільки при видаленні, але і при додаванні нових елементів, знижуючи обчислювальне навантаження при добуванні мінімального вузла з купи, що є критичним у складній системі, коли при кожній операції обробляється велика кількість елементів.

Кількість нащадків вершини t довільного дерева T називається ступенем (degree) цієї вершини, а ступенем дерева називається ступінь його кореня.

2-3 деревами називаються дерева T_0, T_1, T_2, ..., визначені індуктивно. Дерево T_0 складається з єдиною вершини. Під деревом T_i розуміється дерево,

складене або з двох дерев T_{i-1} або з трьох: при цьому корінь кожного наступного стає самим правим нащадком кореня попереднього[13].

Деревами 2-3 купи назвемо дерева $H_{[i]}$. Дерево $H_{[i]}$ - це або пусте 2-3 дерево, або одне, або два 2-3 дерева ступеня i, з'єднані аналогічно деревах T_i.

2-3 купа - це масив дерев $H_{[i]}$ (i = 0..n), для кожного з яких виконується властивість купи. Візуальне представлення купи зображене на Рис. 2.10.

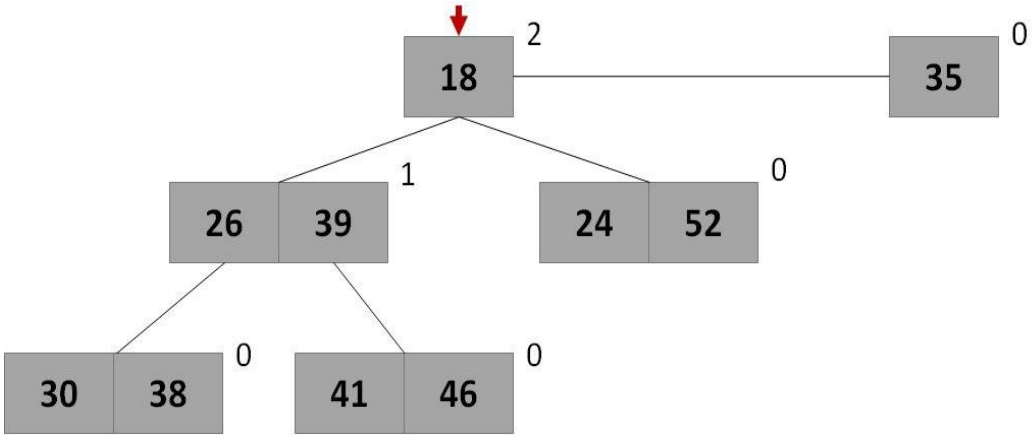


Рис. 2.10 2-3 купа

В 2-3 купі існують наступні операції:

1. *find_min()* – пошук мінімального елемента у купі. Мінімальний елемент купи знаходиться в корені одного з дерев. Щоб знайти мінімальний елемент, необхідно пройтися по деревах. Так як на мінімальному вузлі купи підтримується покажчик, ми можемо знайти цей вузол за константне час O(1);
2. *insert()* – додавання нового елемента в 2-3 купу. Сам алгоритм додавання елементу виглядає наступним чином:
    a. Для додавання нового елемента створимо дерево $H_{[0]}$, що містить одну вершину.
    b. Зробимо операцію додавання цього дерева в купу. При додаванні дерева ступеня *i* в купу можливі наступні варіанти:

i. Якщо дерева з таким пріоритетом немає, то додаємо його в купу.
ii. Інакше вилучаємо таке дерево з купи і з'єднуємо його з тим, що додається, після чого додаємо отримане дерево в купу

c. Після додавання поправляємо покажчик на мінімальний корінь

Алгоритм додавання нового елементу до купи виконується за час O(log n).

3. *delete_min()* – видалення мінімального елемента з купи. Мінімальний елемент купи знаходиться в корені одного з дерев купи, припустимо що в $H_{[i]}$ - знайдемо його за допомогою *find_min()*. Винесемо дерево H [i] з купи і видалимо його корінь, при цьому дерево розпадеться на піддерева, які ми згодом додамо в купу. Через те, що після видалення елементу необхідно правильно додати до купи усі піддерева, час виконання алгоритму становить O(log n).
4. *merge()* – об'єднання двох куп в одну. Якщо ми маємо дві купи, які потрібно об'єднати в одну, то для цього необхідно по черзі вилучати усі 2-3 дерева з другої купи і вставляти їх в першу купу. Після потрібно буде поправити лічильник кількості вузлів. Для цього необхідно підсумувати кількість елементів в першій і в другій купі і записати результат в лічильник в першій купі, а в другій купі встановити лічильник в 0. Даний алгоритм виконується за константний час O(1).

2.2 Аналіз алгоритмів автоматичної пріоритезації для системи, що розробляється

У попередньому пункті було розглянуто декілька можливих реалізацій купи, які можуть бути використані для створення пріоритетної черги – ключового елементу розроблюваної системи. Тепер потрібно обрати такий варіант, який задовільняє потреби системи. Для цього треба звернути увагу на такі характеристики вищенаведених куп, як швидкість виконання основних

операцій та складність реалізації. Загальний аналіз усіх розглянутих алгоритмів наведено у Таблиці 2.1.

Таблиця 2.1. – Порівняння алгоритмів

Назва	Операції				Складність реалізації
	insert	extract_max	decrease_key	merge	
Наївна реалізація (невідсортований масив)	O(1)	O(n)	O(n)	O(1)	Проста
Наївна реалізація (відсортований масив)	O(n)	O(1)	O(log n)	O(n+m)	Проста
Двобатьківська купа	O(√n)	O(√n)	n/a	n/a	Середня
Двійкова купа	O(log n)	O(log n)	O(log n)	O(n+m)	Середня
Біноміальна купа	O(log n)	O(log n)	O(log n)	O(log n)	Складна
2-3 купа	O(1)	O(log n)	O(log n)	O(1)	Складна

Приймаючи до уваги усі властивості кожного з вищенаведених алгоритмів, було прийнято рішення, що для поставленої задачі пріоритетна черга буде реалізована за допомогою двійкової купи (Binary heap). Ця реалізація найбільш точно відповідає поставленій задачі, адже час виконання основних операцій, таких як: додавання нового елементу, вилучення елемента

з максимальним пріоритетом та збільшення/зменшення пріоритету елемента буде виконуватися за час O(log n), що є прийнятним навіть при великих значеннях n. Така операція, як злиття двох черг, буде використовуватися порівняно рідко, тому порівняно великий час її виконання O(n + m) не буде помітно уповільнювати роботу системи.

До того ж, реалізація пріоритетної черги за допомогою двійкової купи не є досить складною задачею, що значно зменшує час на розробку системи та час на майбутню відладку програми та тестування.

2.3 Опис архітектури системи

Система, що розробляється, буде використовувати архітектуру MVC (англ. Model-View-Controller) – модель, представлення, контроллер. Дана архітектура досить широко використовується у наш час, та є однією з найсучаснійших. Розглянемо дану архітектуру більш детально.

Шаблон проектування *Модель-Представлення-Контролер* - це шаблон програмної архітектури, який побудований на основі збереження представлення даних, що використовуються при роботі програми, окремо від методів, які взаємодіють з даними.

Не дивлячись на те, що схема MVC була спочатку розроблена для персональних комп'ютерів, вона була адаптована і широко використовується веб-розробниками через точного розмежування завдань і можливості повторного використання коду. Схема стимулює розвиток модульних систем, що дозволяє розробникам швидко оновлювати, додавати або видаляти функціонал.

Графічне представлення моделі MVC наведено на Рис. 2.11.

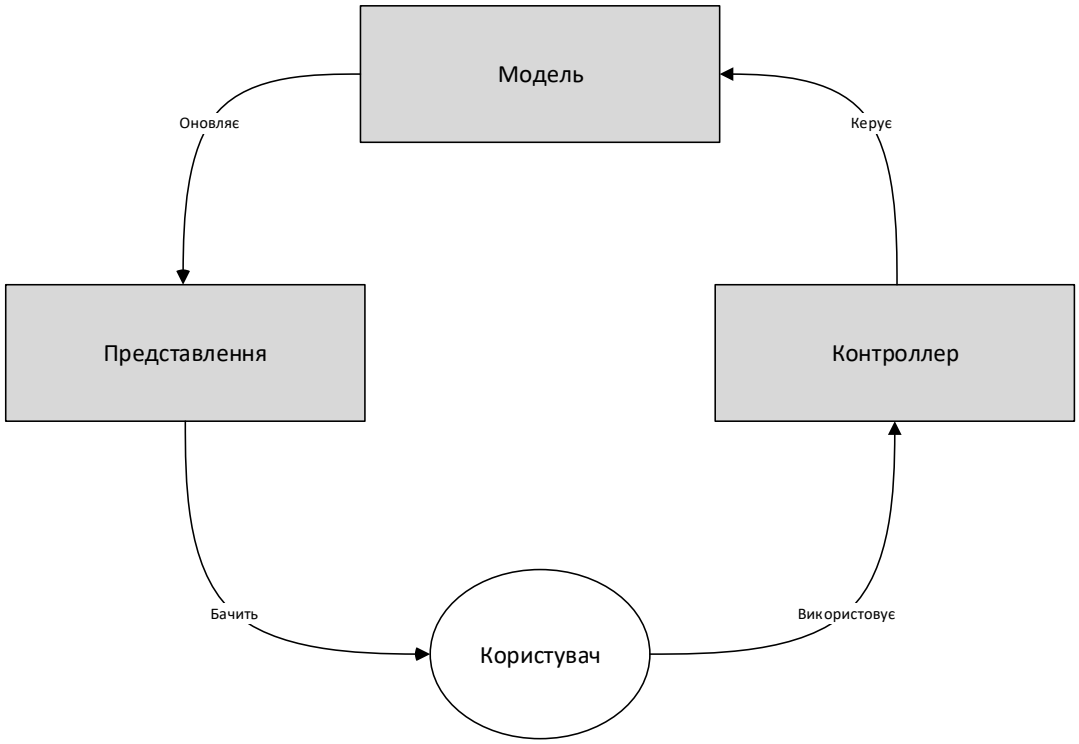


Рис. 2.11 Графічне представлення моделі MVC

Користувач бачить представлення (веб-сторінку). Взаємодіючи з веб-сторінкою, користувач вносить деякі зміни в веб-додаток. Контролер, відповідно до дій користувача, керує моделлю, вносячи відповідні зміні. Оновлена модель з’являється у представленні, що бачить користувач.

Розглянемо кожен з елементів архітектури окремо:

1. **Модель** - Моделлю називають постійне сховище даних, які використовуються в структурі. Вона повинна забезпечувати доступ до даних для їх створення, переглядку, зміни, або видалення. У загальній структурі «Модель» є зв’язуючим елементом між компонентами «Представлення» і «Контролер».
   При цьому «Модель» не має ніякого зв'язку або інформації про те, що відбувається з даними, коли вони передаються компонентам «Представлення» або «Контролер». Єдине завдання «Моделі» - обробка даних в постійному сховище, пошук і підготовка даних, переданих іншим складовим MVC.

Найчастіше це найбільш складна частина системи MVC. Компонент «Модель» - це вершина всієї структури, так як без неї зв'язок між «Контроллером» і «Представленням» неможливий.

2. **Представлення** - це частина системи, в якій даним, що були запитані у «Моделі», надається остаточний вигляд та визначається логіка їх подання. У веб-додатках, створених на основі MVC, «Представлення» - це компонент, в якому генерується і відображається HTML-код.

   Подання також перехоплює дії користувача, які потім передаються «Контролеру». Характерним прикладом цього є кнопка, що генерується «Представленням». Коли користувач натискає її, запускається дія в «Контролері».

   Компоненту «Представлення» ніколи не передаються дані безпосередньо «Контроллером». Між «Представленням» і «Контроллером» немає прямого зв'язку - вони з'єднуються за допомогою «Моделі».

3. Контролер – елемент архітектури, задача якого полягає в обробці даних, що вводить користувач, і оновленні «Моделі». Це єдина частина схеми, для якої необхідна взаємодія користувача.

   «Контролер» можна визначити, як збирача інформації, яка потім передається в «Модель» для подальшого зберігання. «Контроллер» не містить ніякої іншої логіки, крім необхідності збирати вхідні дані. «Контролер» також підключається тільки до одного «Представлення» і однієї «Моделі». Це створює систему з одностороннім потоком даних з одним входом та одним виходом в точках обміну даними.

   «Контролер» отримує завдання на виконання тільки коли користувач взаємодіє з «Представленням», і кожна його функція залежить від того, як саме користувач взаємодіяв з «Представленням».

Основною метою застосування даної концепції є відділення бізнес-логіки (моделі) від її візуалізації (представлення). За рахунок такого поділу

підвищується можливість повторного використання коду. Найбільш корисне застосування даної концепції в тих випадках, коли користувач повинен бачити ті ж самі дані одночасно в різних контекстах, або з різних точок зору. Зокрема, виконуються наступні завдання:

- До однієї моделі можна підключити кілька представлень, при цьому не змінюючи реалізацію моделі. Наприклад, деякі дані можуть бути одночасно представлені у вигляді таблиці, графіка і кругової діаграми;
- Не змінюючи реалізацію представлення, можна змінити реакції на дії користувача (натискання на кнопку, заповнення форми) - для цього досить використовувати інший контролер;
- Ряд розробників спеціалізується тільки в одній з областей: або розробляють графічний інтерфейс, або розробляють бізнес-логіку. Тому можливо добитися того, що програмісти, які займаються розробкою бізнес-логіки (моделі), взагалі не будуть знати про те, яке представлення буде використовуватися[14].

2.4 Аналіз мов програмування серверної частини

На сьогоднішній день, для створення серверної частини будь-якої програми існує декілька мов програмування. Кожна з них має свої особливості, свої переваги і недоліки. Розглянему ті мови програмування, які є одними з найпопулярніших у сучасному світі, та за допомогою яких можна буде створити серверну частину системи, відповідно до завдання.

2.4.1 Мова програмування Java

Java розробила компанія Sun Microsystems на початку 90-х років XX століття. Провідну роль у створенні мови зіграв канадський інженер Джеймс Гослінг (James Gosling). На ранніх етапах розробки мова називалася Oak. Потім його перейменували в честь сорти кави Java.

Джеймс Гослінг і його однодумці хотіли створити мову з С-подібним синтаксисом. У той же час він повинен бути простішим у порівнянні з C/C++. Розробники планували використовувати Java для програмування побутової електроніки. Проте практично відразу після випуску версії 1.0 в 1995 мова стали використовувати розробники серверного та клієнтського ПЗ.

Розглянемо більш детально особливості мови Java [15].

Java - мова програмування загального призначення. Вона відноситься до об'єктно-орієнтовних мов програмування і має сильну типізацією.

Розробники мови реалізували принцип WORA: write once, run anywhere або «пиши один раз, запускай будь-де». Це означає, що написаний на Java додаток можна запустити на будь-якій платформі, якщо на ній встановлене середовище виконання Java (JRE, Java Runtime Environment).

Це завдання вирішується завдяки компіляції написаного на Java коду в байт-код. Цей формат виконує JVM (Java Virtual Machine – віртуальна машина Java). JVM - частина середовища виконання Java (JRE). Віртуальна машина не залежить від платформи.

В Java реалізований механізм управління пам'яттю, який називається збирачем сміття або garbage collector. Розробник створює об'єкти, а JRE за допомогою збирача сміття очищає пам'ять, коли об'єкти перестають використовуватися.

Як зазначалося вище, синтаксис мови Java схожий на синтаксис інших С-подібних мов. Розглянемо деякі особливості цієї групи:

- чутливість до регістру - ідентифікатори User та user в Java - це різні сутності;

- для іменування методів використовується lowerCamelCase. Якщо назва методу складається з одного слова, воно повинно починатися з малої літери. Приклад: firstMethodName ();
- для іменування класів використовується UpperCamelCase. Якщо назва складається з одного слова, воно повинно починатися з великої літери. Приклад: FirstClassName.
- назва файлів програми має точно збігатися з назвою класу з урахуванням чутливості до регістру. Наприклад, якщо клас називається FirstClassName, файл повинен називатися FirstClassName.java;
- ідентифікатори завжди починаються з літери (A-Z, a-z), знака $ або нижнього підкреслення _;

Вище зазначено, що Java відноситься до мов програмування загального призначення. Розглянемо деякі приклади прикладного застосування мови Java.

За даними компанії Oracle, програми на Java запускаються на 3 млрд пристроїв, отже можна зробити висновок, що Java широко використовується і входить в число найбільш затребуваних мов, це не викликає сумніву.

Наприклад, переважна більшість великих компаній так чи інакше використовують Java. Дуже багато серверних додатків для корпорацій написані на цій мові. Наприклад, це програми для великих фінансових, консалтингових, страхових та аудиторських компаній, які забезпечують проведення транзакцій, фіксації торгових операцій та створюють моделі своїх клієнтів.

Популярна комп'ютерна гра Minecraft написана на Java.

Мобільна розробка - ще одна область використання Java. Цією мовою пишуть програми для пристроїв, що працюють під управлінням ОС Android.

Також Java застосовується для роботи з Big Data, розробки програм для наукових цілей, наприклад, обробки природних мов, програмування приладів - від побутових девайсів до промислових установок.

В Java також існують популярні фреймворки, такі як Spring, Hibernate, Stuts, та інші. Вони значно прискорюють розробку програмного забезпечення, адже надають вже готовий, написаний та протестований функціонал «з коробки». Такі фреймворки, як Spring та Hibernate, можна потенціально використовувати при розробці системи пріоритезації пацієнтів. Розглянемо їх більш детально.

Spring Framework - найпопулярніша середовище розробки додатків для корпоративної Java. Мільйони розробників по всьому світу використовують Spring Framework для створення високопродуктивного програмного забезпечення, що легко підтримувати.

Spring Framework - це платформа Java з відкритим вихідним кодом. Спочатку він був написаний Родом Джонсоном і вперше випущений під ліцензією Apache 2.0 в червні 2003 року.

Основні функції Spring Framework можуть бути використані при розробці будь-якого Java-додатка, але є основне використання даного фреймворка - для створення веб-додатків поверх платформи Java EE. Платформа Spring призначена для спрощення використання J2EE-розробки і просування більш ефективних методів програмування за рахунок використання моделі програмування на основі POJO (Plain Old Java Object – старий добрий Java об’єкт – мається на увазі простий об’єкт, який не перевантажений стороннім функціоналом та існує тільки для виконання одної певної задачі).

Переваги Spring наступні:

- Spring дозволяє розробникам розробляти програми корпоративного класу з використанням POJO. Перевага використання тільки POJO полягає в тому, що розробникам не потрібно використовувати EJB-контейнер, такий як сервер додатків.
- Spring організований за модульним принципом. Незважаючи на те, що існує досить велика кількість пакетів і класів, при розробці можна підключити необхідні, не перевантажуючи програму зайвими класами та методами.
- Тестування програми, написаної на Spring, дуже просто, тому що в цю середу переміщений код, що залежить від середовища. Крім того, завдяки використанню POJO, стає простіше використовувати впровадження залежностей для введення тестових даних.
- Веб-фреймворк Spring - спроектований веб-MVC-фреймворк, який надає відмінну альтернативу веб-фреймворкам, таким як Struts, та дозволяє розробляти складні веб-додатки, використовуючи архітектуру MVC.
- Spring надає узгоджений інтерфейс керування транзакціями, який може масштабуватися до локальної транзакції (наприклад, з використанням однієї бази даних), або до глобальних транзакцій.

Hibernate - це безкоштовна Java бібліотека з відкритим вихідним кодом, що представляє собою інструмент об'єктно-реляційного представлення (object-relational mapping - ORM). Його основним завданням є перетворення даних реляційної бази даних в об'єктно-орієнтовані моделі, та навпаки. Крім цього, фреймворк також надає функціонал для автоматичної побудови запитів, пошуку та вилучення даних.

Метою Hibernate є звільнення розробника мануальних завдань щодо забезпечення зберігання об'єктів в реляційної базі даних.

Hibernate може бути з легкістю впроваджений на будь-якій стадії проекту, його можна використовувати як в процесі проектування класів і таблиць, так і для роботи з уже існуючою базою даних.

Основними перевагами цього фреймворку є:

- природна модель програмування: фреймворк дозволяє розробляти класи відповідно до особливостей об'єктно-орієнтованої мови: успадкування, поліморфізму, а також, асоціацій і колекцій Java;
- прозоре збереження POJO: Hibernate не вимагає ніяких особливих інтерфейсів або базових класів для створення відповідного класу і його зв'язку з таблицями в базі даних. Абсолютно будь-який клас або структура може виступити в цій ролі;
- висока продуктивність: Hibernate підтримує «ледачу» ініціалізацію і безліч стратегій вибірок. Він не вимагає наявності спеціальних таблиць або полів в базі даних і генерує більшу частину SQL коду під час ініціалізації системи, а не під час виконання. Hibernate зазвичай забезпечує чудову продуктивність в порівнянні з прямим JDBC-кодом.
- надійність і масштабованість: Hibernate добре відомий своїми відмінними якістю і стабільністю, перевіреними використанням десятками тисяч Java-розробників. Він був розроблений, щоб працювати в кластері сервера додатків і надавати високу масштабованість архітектури.
- всебічні можливості побудови запитів: підтримка Hibernate Query Language (HQL), Java Persistence Query Language (JPAQL), критеріїв запитів, і SQL.

Можна зробити висновок, що використання Java як мови програмування для створення серверної частини системи, що розроблюється, є виправданим.

Адже, це Java – це надійна і відносно швидка мова, яка, при застосуванні таких бібліотек як Spring та Hibernate, буде відповідати всім потребам системи.

2.4.2 Мова програмування Python

Іншою мовою програмування, на якій можливо розробити серверну частину системи, є Python.

Python є інтерпретуємим, об'єктно-орієнтованою мовою програмування. Він надзвичайно простий і містить невелику кількість ключових слів, але в той же час є дуже гнучким і може використовуватися для вирішення багатьох задач – від створення серверної логіки, до аналізу великих об’ємов даних, розробки систем штучного інтелекту, та написання ботів для таких додатків як Telegram та WhatsApp. Це мова більш високого рівня, ніж Pascal, C ++ та C, що досягається, в основному, за рахунок вбудованих високорівневих структур даних (списки, словники та кортежі) [16].

В даній мові програмування є декілька виразних особливостей – по перше, це мова з динамічною типізацією. Python, на відміну від багатьох мов (Pascal, C ++, Java, та інших), не вимагає опису змінних. Вони створюються в місці їх ініціалізації, тобто при першому присвоєнні змінної будь-якого значення. Це значить, що тип змінної визначається типом значення, що присвоюється. В цьому відношенні Python нагадує мову Basic.

Тип змінної не є незмінним. Будь-яке присвоювання для неї коректно і це призводить лише до того, що типом змінної стає тип нового значення, що присвоюється.

Іншою особливістю є те, що у таких мовах як Pascal, C, C ++ організація списків представляла деякі труднощі. Для їх реалізації доводилося добре вивчати принципи роботи з покажчиками і динамічною пам'яттю. І навіть маючи високу кваліфікацію, програміст, що кожен раз заново реалізує механізми створення, роботи і знищення списків, міг легко допустити

критичні помилки, яки призводили до помилок при роботі програми. Для того, щоб полегшити роботу з такою структурою даних, як списки, були створені зручні засоби для роботи з ними. Наприклад, в Delphi Pascal є клас TList, який реалізує списки; для C ++ розроблена бібліотека STL (Standard Template Library), що містить такі структури як вектори, списки, множини, словники, стеки і черги. Однак, такі засоби є ні в всіх мовах і їх реалізаціях. У Python, таких реалізацій декілька – це кортежі (tuples), словники (dictionaries), та списки (lists), кожен з яких значно зменшує час для вирішення певної проблеми.

Переваги даної мови наступні:

- Безумовною перевагою є те, що інтерпретатор Python реалізований практично на всіх платформах і операційних системах. Першою такою мовою був C, проте його типи даних на різних машинах могли займати різну кількість пам'яті і це було значною перешкодою при написанні написанні програми, яку можна переносити на інші операційні системи. Python ж таким недоліком не володіє.
- Іншою перевагою є масштабованість мови, адже Python від самого початку розроблялась як мова з властивістю до масштабування. Це означає, що будь-який зацікавлений програміст може вдосконалювати цю мову. Інтерпретатор написаний на C і вихідний код доступний для будь-яких маніпуляцій. У разі необхідності, можна вставити його в свою програму і використовувати як вбудовану оболонку. Або ж, програміст може написати на C свої доповнення до Python та, скомпілювавши програму, отримати "розширений" інтерпретатор з новими можливостями.

- Наступна перевага - наявність великої кількості модулів, доступних для підключення, що забезпечують різні додаткові можливості. Такі модулі пишуться на мові C і на самому Python і можуть бути за необхідністю розроблені кваліфікованими програмістами. Як приклад можна навести такі модулі:
  - Numerical Python – ця бібліотека надає розширені математичні можливості, такі як маніпуляції з цілими векторами і матрицями;
  - Tkinter – модуль для побудови додатків з використанням графічного інтерфейсу користувача (GUI), що розроблений на основі широко розповсюдженого на X-Windows Tk-інтерфейсу;
  - OpenGL (або Open Graphics Library) - велика бібліотеки графічного моделювання дво- і тривимірних об'єктів. Даний стандарт підтримується, в тому числі, в таких поширених операційних системах як Microsoft Windows 95 OSR 2, 98 і Windows NT 4.0.

З іншого боку, одним з основинх недоліків мови є порівняно невисока швидкість виконання Python-програми, що обумовлено тим, що інтерпретація коду займає значну кількість часу. Однак, це з лишком окупається перевагами мови при написанні програм, які є не дуже критичними до швидкості виконання.

Для написання серверної частини додатка на Python можна використовувати фреймворк Django. Розробити серверну частину можливо і без цього фреймворка, але це займе багато часу через те, що програмісту буде необхідно писати великі об’єми повторюваного коду, та копітко настроювати взаємодію з клієнтською частиною. Цього можна уникнути, якщо використовувати даний фреймворк, адже він дозволяє уникнути вищезазначених проблем.

Основними перевагами Django є:

- Django дозволяє уникнути написання монотонного коду, який необхідно писати при створенні будь-якого веб-додатку. Завдяки Django, створення динамічних веб-додатків з використанням Python займає набагато менше часу та дозволяє уникнути помилок людського фактору, що значно покращує роботу продукту. До того ж, даний фреймворк написаний на Python.
- Фреймворк підтримує патерн проектування MVC. Це допомагає розробникам в розділенні призначеного для користувача інтерфейсу і бізнес-логіки Django-додатків.
- Django має широкий функціонал щодо веб-безпеки, тому для розробників програмного забезпечення, що використовують Django, надається широкий вибір інструментів, які дозволяють розробити захищений сервер. У наш час, безпека серверної частини є однією з основоположних вимог, адже вона гарантує те, що треті сторони не зможуть отримати несанціонований доступ до серверу, викрасти дані користувачів, або змінити логіку роботу системи.
- Django - працює на різних операційних системах. Крім того, він підтримує взаємодію з різними базами даних.
- Додатки, створені з використанням цього фреймворку, добре піддаються масштабування. Тому розробники, які обирають Django, може бути впевненим у тому, що зможе ефективно розвивати свій проект у міру його зростання.

Як підсумок, можна зазначити, що Python, разом із використанням фреймворку Django, відповідає вимогам системи, що розробляється. За допомогою даної мови програмування можна за відносно невеликий час розробити серверну частину додатка, який буде мати необхідний функціонал

2.4.3 Мова програмування Ruby

Ruby - це ретельно збалансована мова. Її розробник Юкіхіро Мацумото (також відомий як "Matz"), об'єднав частини своїх улюблених мов (такі як Perl, Eiffel, SmallTalk і Lisp) щоб сформувати нову мову, в якій парадигма функціонального програмування збалансована принципами імперативного програмування[17].

З часу випуску публічної версії в 1995 році, Ruby привернув увагу програмістів з усього світу. У 2006 році Ruby завоював масове визнання. У найбільших компаніях по всьому світу активно діють групи розробників на Ruby, а конференції, присвячені Ruby, привертають увагу багатьох людей.

Основними перевагами мови програмування Ruby є:

- В Ruby є конструкції для обробки виключень, як в Java або Python, які дозволяють простіше працювати з помилками.
- В Ruby представлений справжній збирач сміття типу mark-and-sweep (познач і відчистити) для всіх Ruby об'ектов. Не потрібно вручну відстежувати кількість посилань в сторонніх бібліотеках. Як говорить розробник мови Matz, «Це корисніше для вашого здоров'я.»
- Масштабувати Ruby на C простіше, ніж в Perl або Python за допомогою дуже ефективного API для виклику Ruby з C. Він включає в себе виклики для вбудовування Ruby в програмне забезпечення, щоб використовувати його як скриптову мову.
- Усі необхідні бібліотеки довантажуютьс в Ruby самостійно, якщо це дозволяє операційна система.
- В Ruby реалізовані незалежні від операційної системи потоки. Таким чином, на будь-яких платформах, де запускається Ruby, програміст також може використовувати багатопоточність, не залежно від того, чи підтримує дана система потоки чи ні.

- Ruby відрізняється високою переносимістю: він був розроблений здебільшого на GNU / Linux, але працює на багатьох типах UNIX, macOS, Windows, DOS і так далі.

Крім того, за допомогою фреймворку Ruby on Rails, можна розробляти серверну частину веб-додатків. Ruby on Rails, також відомий як Rails, являє собою платформу серверних веб-додатків на основі Ruby. Rails - це середовище MVC, що надає готову структуру бази даних, веб-сторінки та веб-служби. У Ruby on Rails активно просуваються такі стандарти передачі даних, як XML або JSON, а для взаємодії з користувачем (створення представлення) - CSS, JavaScript і HTML.

Переваги Ruby on Rails схожі до переваг таких фреймворків як Spring для Java та Django для Python – він дозволяє програмістам уникати великих часових витрат на написання монотонного кода, та надає багато функіоналу для швидкого створення серверної частини «з коробки».

Хоча Ruby не є настільки ж популярною мовою як Java або Python, за допомогою неї теж можливо розробити серверну частини системи оптимізації роботи госпіталів. Завдяки зручності написання коду та фреймворку Ruby on Rails, можливо буде швидко та ефективно розробити необхідний функціонал.

2.5 Аналіз мови програмування клієнтської частини

Так як розробка серверної частини програми буде відбуватися на мові Java, то для створення клієнтської частини системи, тобто той частини, з якою буде взаємодіяти користувач, буде використаний сучасний серверний механізм шаблонів Thymeleaf.

Thymeleaf - це движок шаблонів Java XML, XHTML та HTML5, який може працювати як у web-середі(на основі сервлетів), так і окремо. Найкраще він підходить для обслуговування XHTML та HTML5 на рівні представлення (View) веб-додатків на основі MVC, але крім того, він може обробляти будь-

який XML-файл навіть в автономному середовищі. Він забезпечує повну інтеграцію із фреймворком для Java - Spring Framework.

У веб-додатках Thymeleaf є повноцінною заміною JavaServer Pages (JSP) і реалізує концепцію Natural Templates: файли шаблонів, які можна відкрити безпосередньо у браузерах, можуть без проблем відображати інформацію як веб-сторінки.

Thymeleaf - це програмне забезпечення з відкритим кодом, ліцензоване за ліцензією Apache 2.0.

Основними перевагами Thymeleaf є:

- Інтеграція зі фреймворком Spring для Java
- Мова виразів (expression language) у Thymeleaf значно більш потужна, ніж у його аналога – Java Servlet Pages (JSP). Такі фактори, як більш широкий функціонал для представлення атрибутів моделі, підтримка форму bean та інтернаціоналізація роблять Thymeleaf більш гнучким у порівнянні з JSP.
- Великий об’єм технічної документації, який дозволяє швидко вирішувати проблеми, які виникають при розробці [18].

2.6 Аналіз системи керування базами данних (СКБД)

Для зберігання даних пацієнтів та атрибутів потужностей медичного закладу, необхідно додату підтримку бази даних. Систему керування базою даних (СКБД) можна обрати одну з наступних: Oracle, MySQL, PostgeSQL, MongoDB, та інші. У системі, що розробляється, будет використана MySQL. Розглянемо цю СКБД більш детально.

Отже, MySQL - це система управління базами даних. Сама по собі база даних являє собою структуровану сукупність даних. Ці дані можуть бути будь-якими - інформація про користувачів, про товари або послуги. Для запису,

вибірки і обробки даних, що зберігаються в комп'ютерній базі даних, необхідна система управління базою даних, якою і є MySQL. Оскільки комп'ютери призначені для швидкої обробки великих обсягів даних, управління базами даних відіграє центральну роль в обчисленнях. Реалізовано таке керування може бути по-різному - як у вигляді окремих утиліт, так і у вигляді коду, що входить до складу інших додатків[19].

У реляційній базі даних дані зберігаються не усім скопом, а в окремих таблицях, завдяки чому досягається виграш в швидкості і гнучкості. Таблиці зв'язуються між собою за допомогою ключів-посилань, завдяки чому забезпечується можливість об'єднувати дані при виконанні запиту з декількох таблиць. SQL – це частина системи MySQL, яку можна охарактеризувати як мову структурованих запитів. Приклад реляційної бази даних наведено на Рис. 2.12.

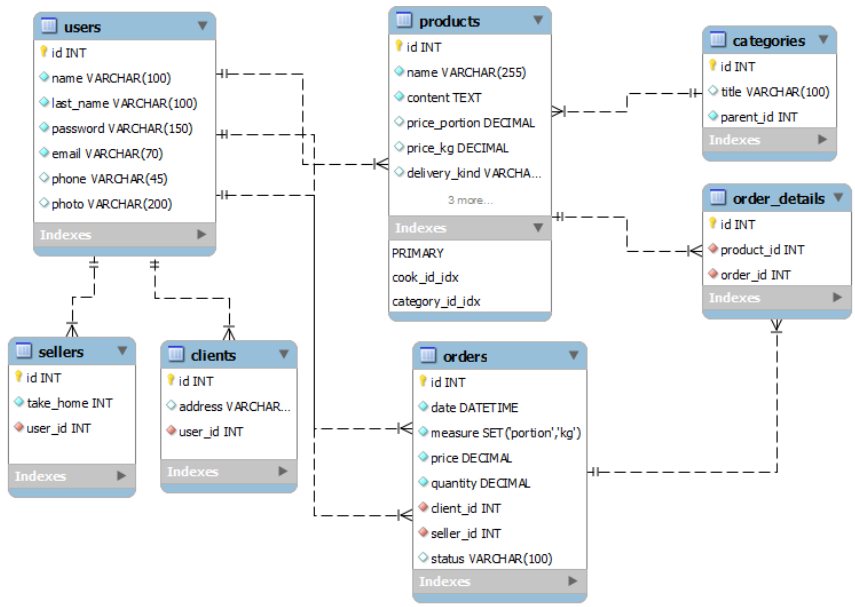


Рис. 2.12 Реляційна база даних

MySQL є дуже швидким, надійним і легким у використанні. MySQL також має ряд зручних можливостей, розроблених в спільно з користувачами.

Система, що розробляється, не є дуже вимогливою до швикдодії системи керування базою даних, тому можна було обрати будь-яку СКБД із наведених вище. Для покриття потреби зберігання даних пацієнта та атрибутів потужностей медичного закладу, буде використана MySQL. Вона є зручною для роботи і є досить швидкодійною, що підходить для системи, що розробляється.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі було розглянуто алгоритми, за допомогою яких можна найбільш ефективно розробити чергу з пріоритетами, що буде використовуватися у системі, що розробляється. Для поставленої задачі пріоритетна черга буде реалізована за допомогою двійкової купи (Binary heap). Ця реалізація найбільш точно відповідає поставленій задачі, адже такі операції, як *insert, extract_max, decrease key* виконуються за O(log n) час, що значно підвищить швидкість роботи електронної черги при великих обсягах вхідних даних.

Також було розглянуто 3 мови програмування, за допомогою яких можна розробити серверну частину системи – Java, Python та Ruby. Для розробки серверної частини було прийнято рішення використовувати мову Java, разом із фреймворком Spring, який підтримує обрану архітектуру MVC, та фреймворк Hibernate, за допомогою якого можна буде реалізувати об’єктно-реляційне відображення об’єктів системи. Для зберігання даних була обрана система керування базою даних MySQL, яка є швидкою, зручною та повністю задовільняє потреби системи у зберіганні даних.. Клієнтська частина сервісу буде розроблена із використанням HTML, CSS та Thymeleaf, який був обраний як движок шаблонів для динамічної зміни інтерфейсу користувача.

РОЗДІЛ 3
ОСНОВНІ ВИМОГИ ДО ФУНЦІОНАЛЬНОСТІ СИСТЕМИ

3.1 Вимоги до розроблюваної системи

Проаналізувавши існуючі аналоги системи електронної черги та враховуючи можливі потреби користувачів даної системи, було сформовано основні вимоги до цього додатку. Дана система повинна мати наступну функціональність:

- Можливість реєстрації пацієнта – медичний персонал повинен мати змогу зареєструвати нового пацієнта в системі. При реєєстрації необхідно внести такі дані, як: ім’я, прізвище та ім’я по батькові, вік, адресу, загальний стан пацієнта, мобільний телефон, електронну адресу, та багато іншої інформації. Також, якщо необхідно, додати документи, що підтверждують особу.
- Можливість призначити перелік процедур – система повинна мати повний перелік усіх процедур, які є в лікарні. Після реєстрації нового пацієнта, персонал медичного закладу повинен мати можливість призначити необхідні процедури пацієнту
- Можливість призначити пріоритет кожної процедури – для правильного функціонування системи, кожна процедура, до якої призначений пацієнт, повинна мати свій пріоритет. В залежності від цих значень, пацієнт буде відвідувати процедури по найбільш оптимальному шляху, що гарантує першочергове проходження найважливіших процедур та зменшення часу перебування в черзі.
- Можливість перегляду черги пацієнтів до будь-якох процедури – персонал медичного закладу повинен мати доступ до інформації про завантаженність кожної з процедур.
- Можливість переглянути інформацію про пацієнта на його особистій сторінці – ця інформація може бути корисна для лікарів, адже вона

дозволяє заздалегідь ознайомитися з пацієнтом, переглянути його історію хвороб та розробити підхід до лікування ще перед прийомом хворого.

- Можливість редагування інформації про пацієнта, його процедур і пріоритетів – ця функціональність необхідна як для виправлення помилок, що були допущені при первинному огляді пацієнта, так і для гнучкої зміни переліку процедур.
- Модуль програми, який виконує розподіл усіх пацієнтів по потужностям медичного закладу – це наріжний камінь усієї системи. Даний модуль повинен мати функціональність оптимального розподілу пацієнтів по процедурам, враховуючи нинішню завантаженість медичного закладу, так і список процедур і значення пріоритетів усіх клієнтів.
- Модуль сповіщення пацієнта – для того щоб повідомити пацієнта про те, що підійшла його черга, система повинна мати функціональність відправити пацієнту SMS-повідомлення, електронний лист або інше сповіщення на телефон.
- Можливість додання нової процедури та оновлення вже існуючих – для того щоб підтримувати актуальну інформацію про наявні процедури та додавати нові процедури, які з’являються у медичному закладі.

3.2 Бізнес-процес функціонування системи

Для більш наглядної демонстрації функціонування системи, була створена схема бізнес-процесів за допомогою графічної нотації BPMN 2.0.

Перше, що робить пацієнт, коли йому потрібна медична допомога – викликає швидку, або, якщо має можливість, сам відвідує лікарню, як зображено на Рис. 3.1.

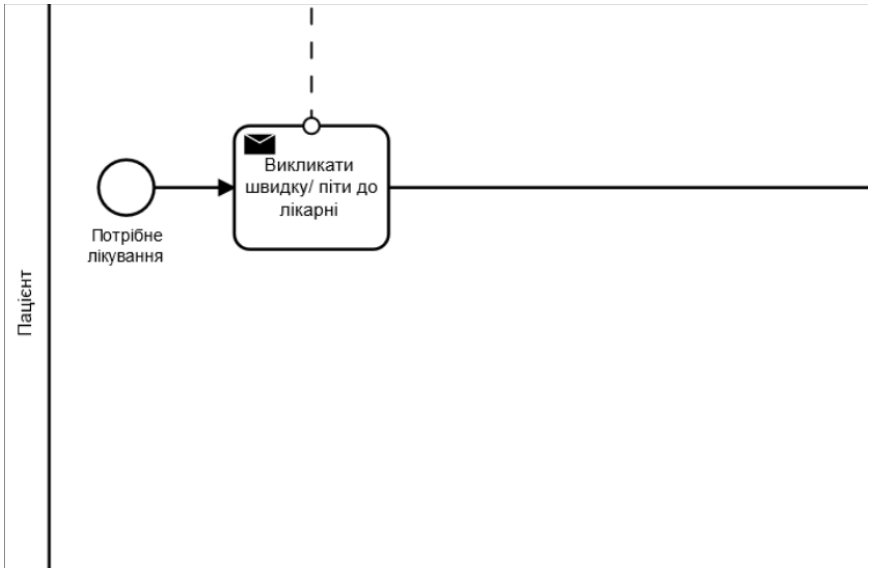


Рис. 3.1 Перші дії пацієнта

Після того, як пацієнт прибув до медичного закладу, лікар здійснює первинний огляд пацієнта та визначає список процедур. Після цього, лікар вносить інформацію про пацієнта в систему та підтверджує зміни, після чого дані записуються в базу даних медичного закладу. Далі лікар вносить інформацію про список процедур, де вказує їх перелік та відповідні пріоритети. Ця інформація також записується в базу даних. На цьому первинний огляд пацієнта завершено, обов’язки лікаря, що здійснював первинний огляд, виконано. Логіка цього процесу зображена на Рис. 3.2.

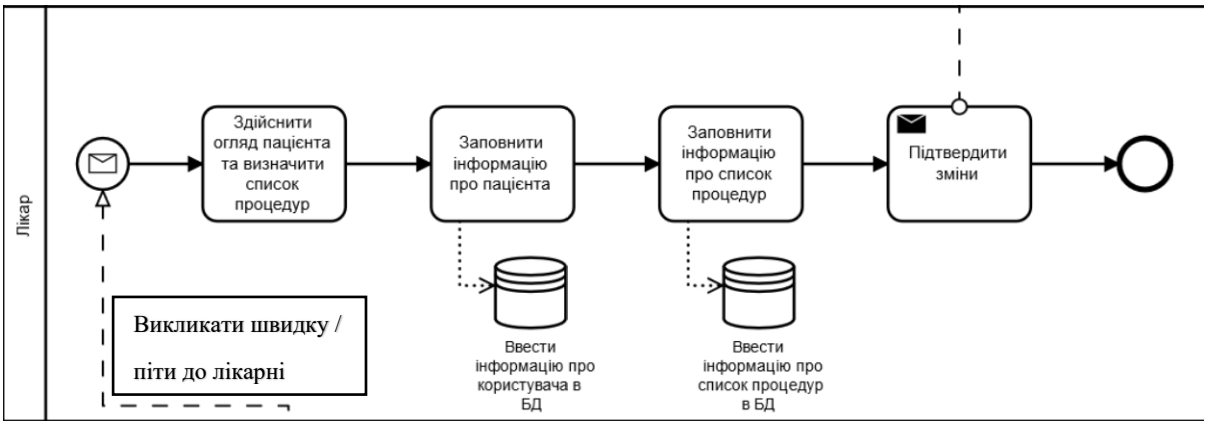


Рис. 3.2 Процес роботи лікаря

Після того, як лікар ввів інформацію про пацієнта та його процедури в систему, вона починає її обробку. Для цього система завантажує з бази даних цю інформацію, яку буде використовувати далі для оптимального додавання пацієнта до існуючих черг. Функціонал зображений на Рис. 3.3.

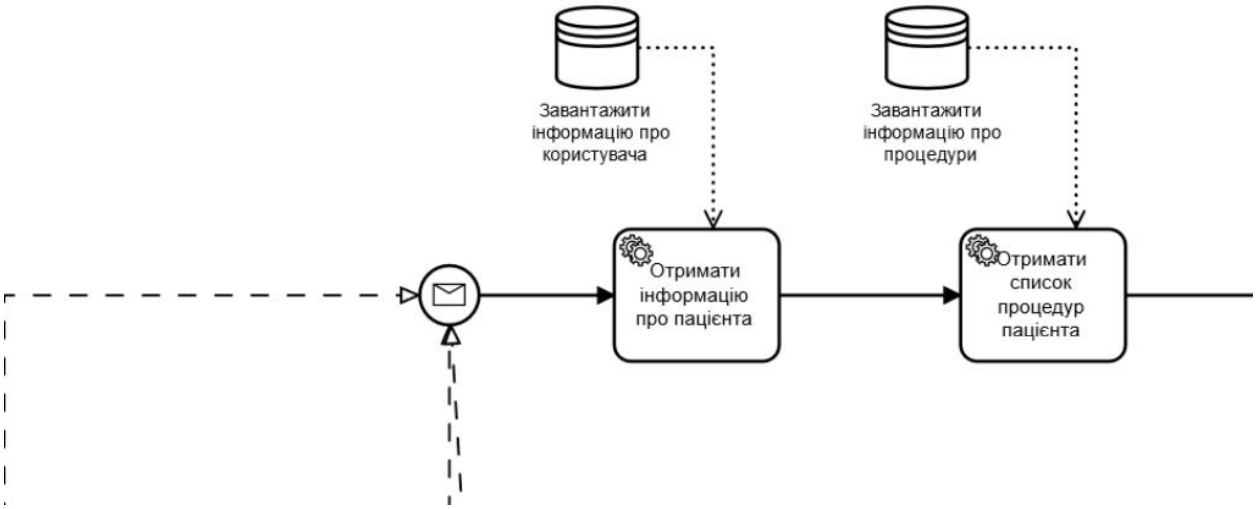


Рис. 3.3 Початок роботи системи

Після завантаження даних про пацієнта, запускається алгоритм додавання в чергу. Алгоритм графічно описаний на Рис. 3.4.

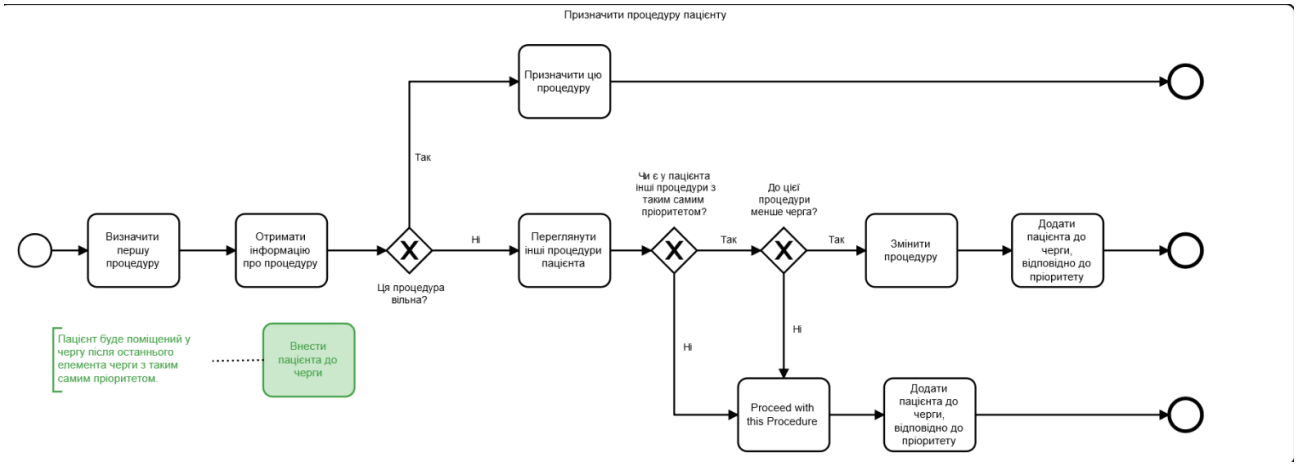


Рис. 3.4 Алгоритм визначення черги

Алгоритм працює наступним чином:

1. Визначає найпріоритетнішу процедуру пацієнта.
2. Отримує всю інформацію про дану процедуру.
3. Визначає, чи вільна ця процедура (тобто, чи є черга до неї).
4. Якщо процедура вільна, система призначає пацієнту цю процедуру та завершує роботу.
5. Якщо до процедури є черга, система перевіряє чи є у пацієнта інша процедура з таким самим пріоритетом.
6. Якщо є, то система порівнює черги, і, якщо черга до іншої процедури менша, направляє пацієнта до іншої процедури, після чого завершує роботу.
7. В іншому випадку, коли у пацієнта немає іншої процедури з таким самим пріоритетом, або до іншої процедури з таким самим пріоритетом черга більше, система залишає наявну процедуру та додає пацієнта в чергу після останньої людини з таким самим пріоритетом. Після цього алгоритм завершує роботу.

Після додавання пацієнта до черги, система надсилає сповіщення на його телефон, як зображено на Рис. 3.5.

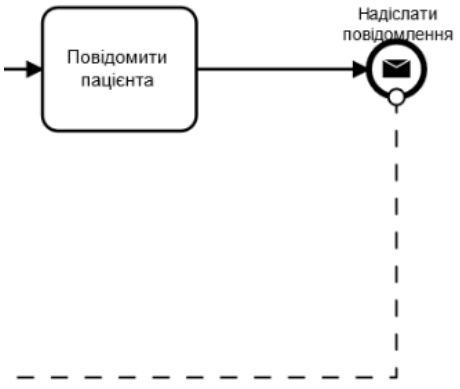


Рис. 3.5 Завершення роботи системи

Після того, як пацієнт отримав сповіщення від системи на мобільний додаток, він встає у призначену чергу. Коли очікування завершено, пацієнт

проходить процедуру, і лікар відмічає це у системі. Проводиться верифікація – якщо це була остання процедура, то пацієнту необхідно розрахуватися, якщо ні – алгоритм додавання у черги обробляє оновлені дані пацієнта, враховуючи закінчені процедури, та призначає нові. Пацієнт отримує сповіщення на телефон і переходить до наступної процедури, як зображено на Рис. 3.6.

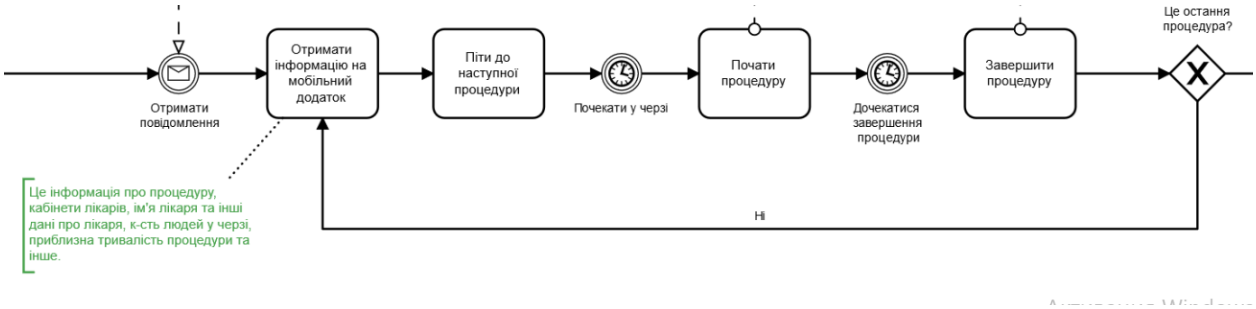


Рис. 3.6 Подальші дії пацієнта

Якщо більше процедур немає, пацієнт отримує рахунок, сплачує його та покидає медичний заклад, як зображено на Рис. 3.7.

Рис. 3.7 – Останні дії пацієнта

3.3 Опис структури системи

Структура проекту наведена на Рис. 3.8.

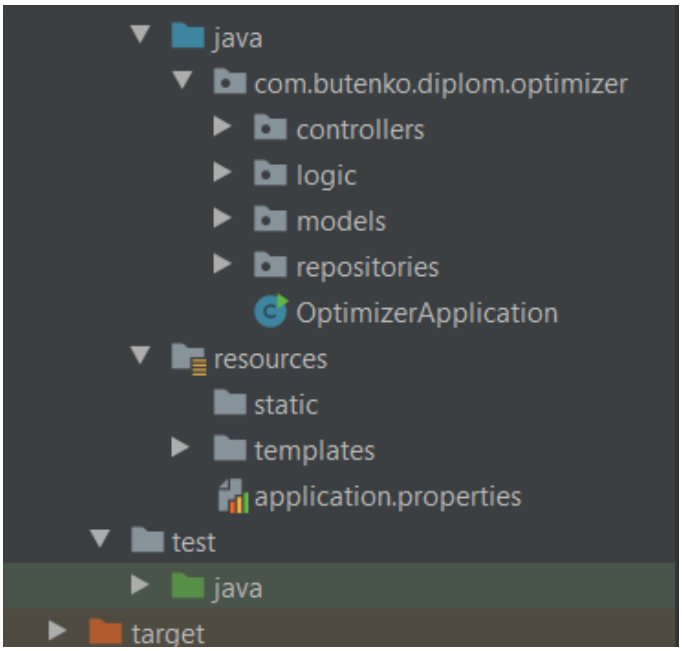


Рис. 3.8 Структура додатку

Так як програма розроблена за допомогою архітектури MVC (Model, View, Controller – Модель, Представлення, Контроллер), то структура проекту повністю відповідає цієї архітектурі.

- У пакеті *controllers* знаходяться контроллери, які отримують запити від представлень, змінюють модель та викликають нові предствлення.
- У пакеті *models* знаходяться моделі – компоненти, які надають дані та змінюють свої стани, виконуючи команди контролера.
- У пакеті logic знаходяться Java-класи, які реалізують бізнес-логіку системи, зокрема алгоритм розподілення пацієнтів по процедурах.
- У пакеті repositories знаходяться репозиторії, які запитують дані з бази даних та завантажують в неї необхідні зміни моделі.

- У пакеті templates знаходяться .html – файли, які формують представлення. Дані файли визначають який саме інтерфейс буде бачити користувач та які зміни він зможе внести до програми.

Як саме функціонує дана структура, зображено на Рис. 3.9.:

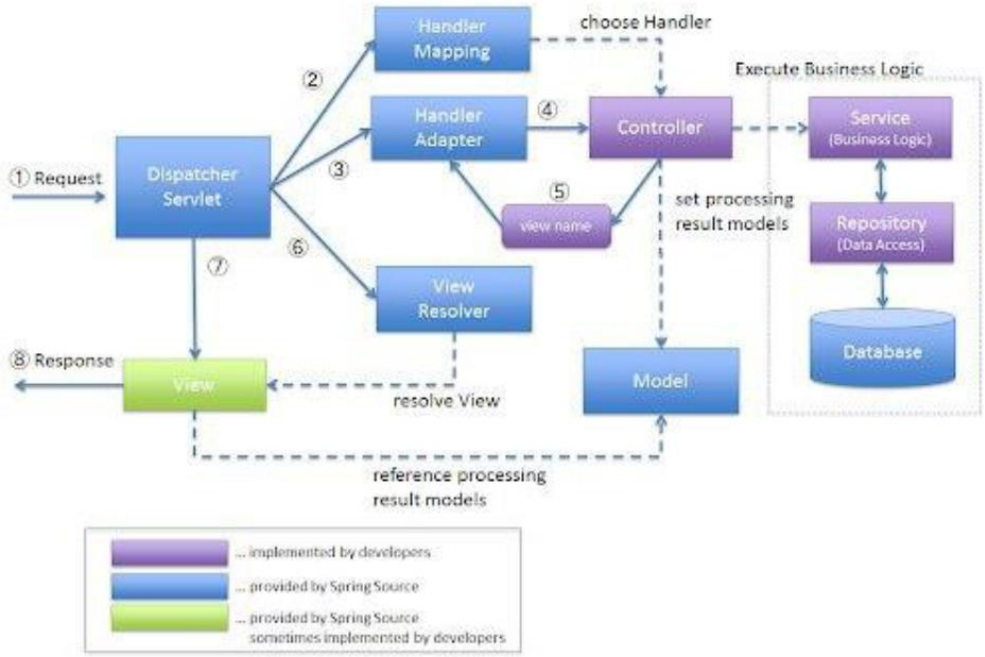


Рис. 3.9 Функціональність структури

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі було розроблено функціонал системи. Проаналізувавши існуючі рішення та можливі потреби користувачів, були сформовані основні вимоги до системи, яким вона повинна відповідати для того щоб покрити потреби як пацієнтів, так і працівників медичного закладу. Спираючись на ці вимоги, було розроблену клієнтську та серверну частину веб-додатка.

Також, за допомогою нотації моделювання бізнес-процесів BPMN 2.0, було розроблено схему бізнес-процесів системи.

Крім того було розглянуто структуру розробленого веб-додатка, який реалізований на архітектурі MVC.

РОЗДІЛ 4
ДЕМОНСТРАЦІЯ ФУНКЦІОНУВАННЯ СИСТЕМИ

4.1 Опис результату розробки

Результатом даної бакалаврської роботи є розроблена система електронної пріоритезації пацієнтів. Для цілей демонстрації було розроблено неважкий та зрозумілий користувацький інтерфейс, який дозволяє взаємодіяти з функціоналом програми та виконувати різні тести.

Перед запуском системи необхідно переконатися, що були виконані базові вимоги до апаратної та програмної частини.

На даний момент, у системі реалізований основний функціонал додавання пацієнтів, перегляду усіх пацієнтів та інформації про них, алгоритм розподілення пацієнтів в по різних чергах, в залежності від пріоритетів, та моніторинг усіх черг, із можливістю «просувати» чергу в реальному часі. Далі буде розглянуто як кожна з цих функціональностей реалізована у системі.

4.2 Демонстрація роботи системи

Після запуску додатка, за адресою localhost:8080 запускається контейнер сервлетів Apache Tomcat. Якщо користувач перейде за вказаною адресою, він побачить головну сторінку системи, яка зображена на Рис. 4.1.

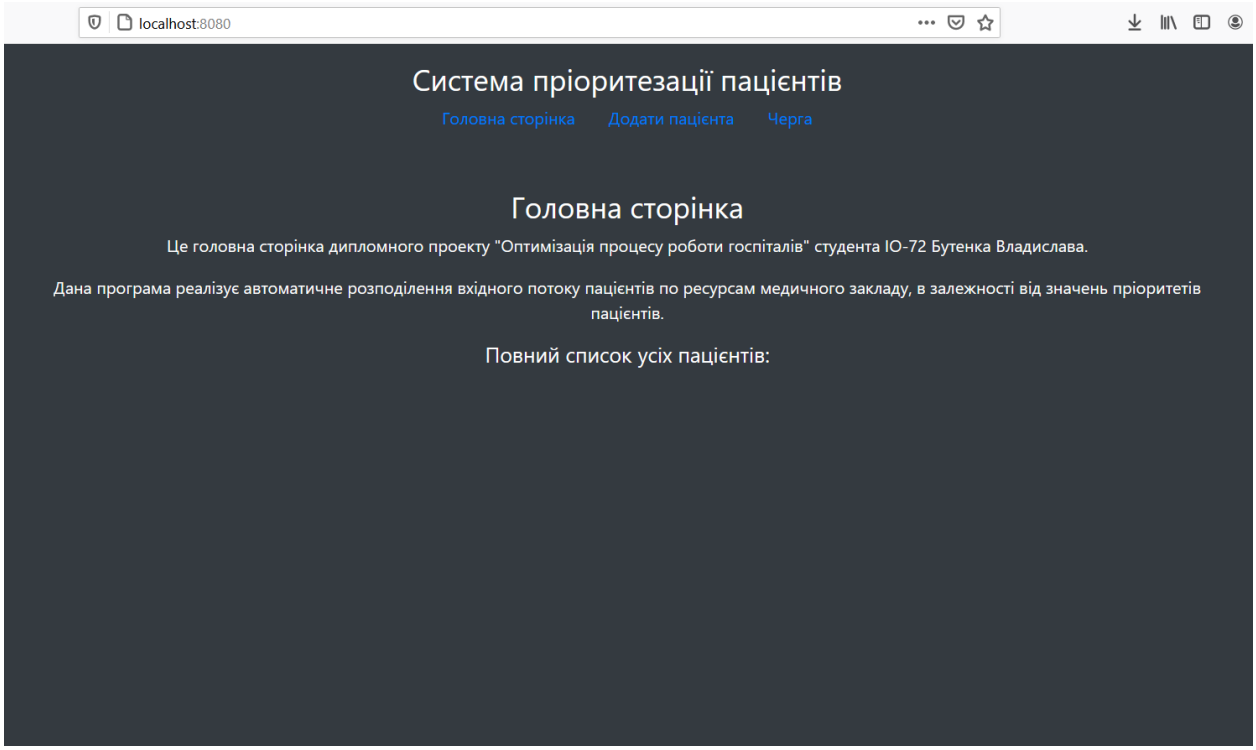


Рис. 4.1 Головна сторінка

На даній сторінці користувач може побачити меню з навігацією по веб-сайту, опис системи та повний список усіх пацієнтів. На даний момент, в систему ще не додано жодного пацієнта, тому список порожній.

Для того щоб додати першого пацієнта, користувач повинен натиснути на «Додати пацієнта» на меню навігації та перейти до відповідної сторінки, що зображена на Рис. 4.2.

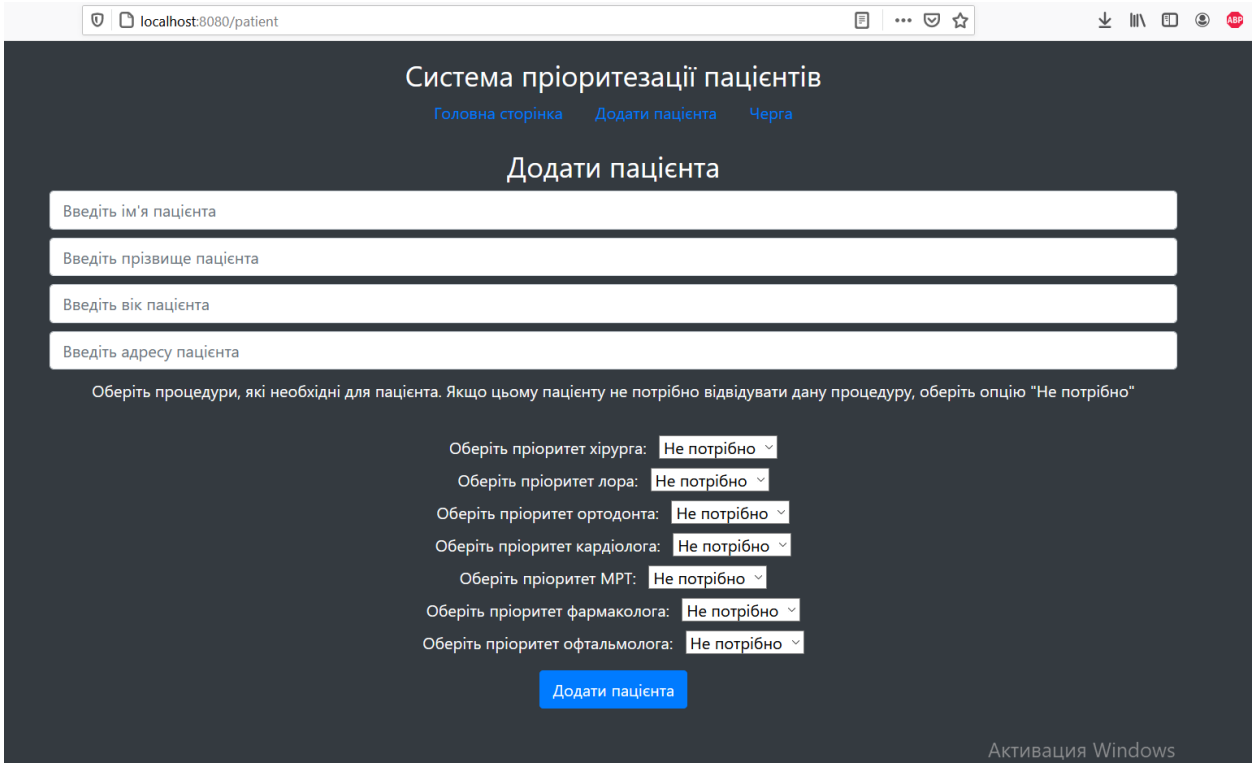


Рис. 4.2 Додавання пацієнта

На даній сторінці користувач може додати пацієнта до системи. Для цього він повинен ввести ім’я, прізвище, вік, та адресу пацієнта. Після цього, користувач може обрати які саме процедури необхідні клієнту. Для цього він повинен обрати у випадаючому списку відповідний пріоритет, як зображено на Рис. 4.3.

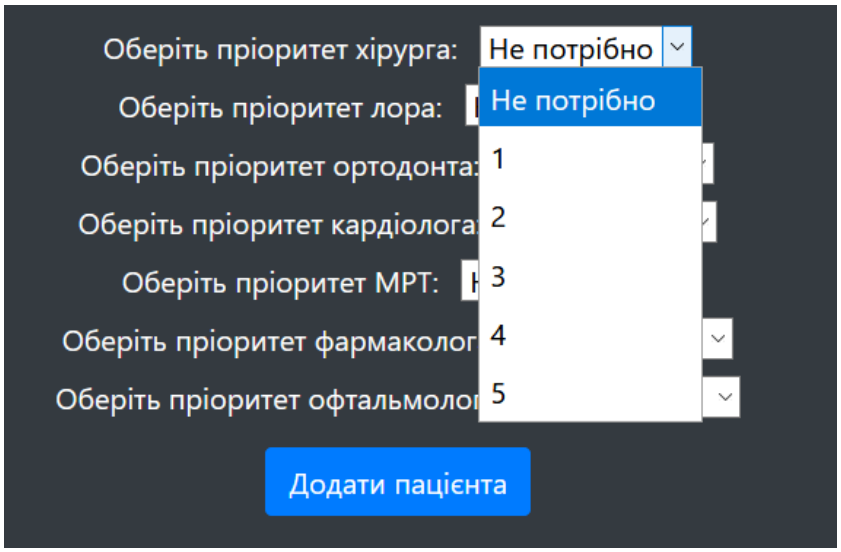


Рис. 4.3 Вибір пріоритету

Найнижчий пріоритет – 1, найвищий – 5. Якщо пацієнту не потрібна дана процедура, користувач може обрати опцію «Не потрібно», для того щоб система не враховувала цю процедуру при плануванні черг.

Для цілей тестування було обрано 7 різних процедур – прийом у хірурга, лора, ортодонта, кардіолога, процедура МРТ, консультація з офтальмологом та фармакологом. В справжній лікарні цих процедур може бути значно більше, проте для демонстрації роботи системи вистачає 7.

Після того, як користувач визначив процедури та пріоритети пацієнта, йому слід натиснути кнопку «Додати пацієнта», щоб інформація про пацієнта записалася в програму. Приклад повністю заповненої сторінки пацієнта наведено на Рис. 4.4.

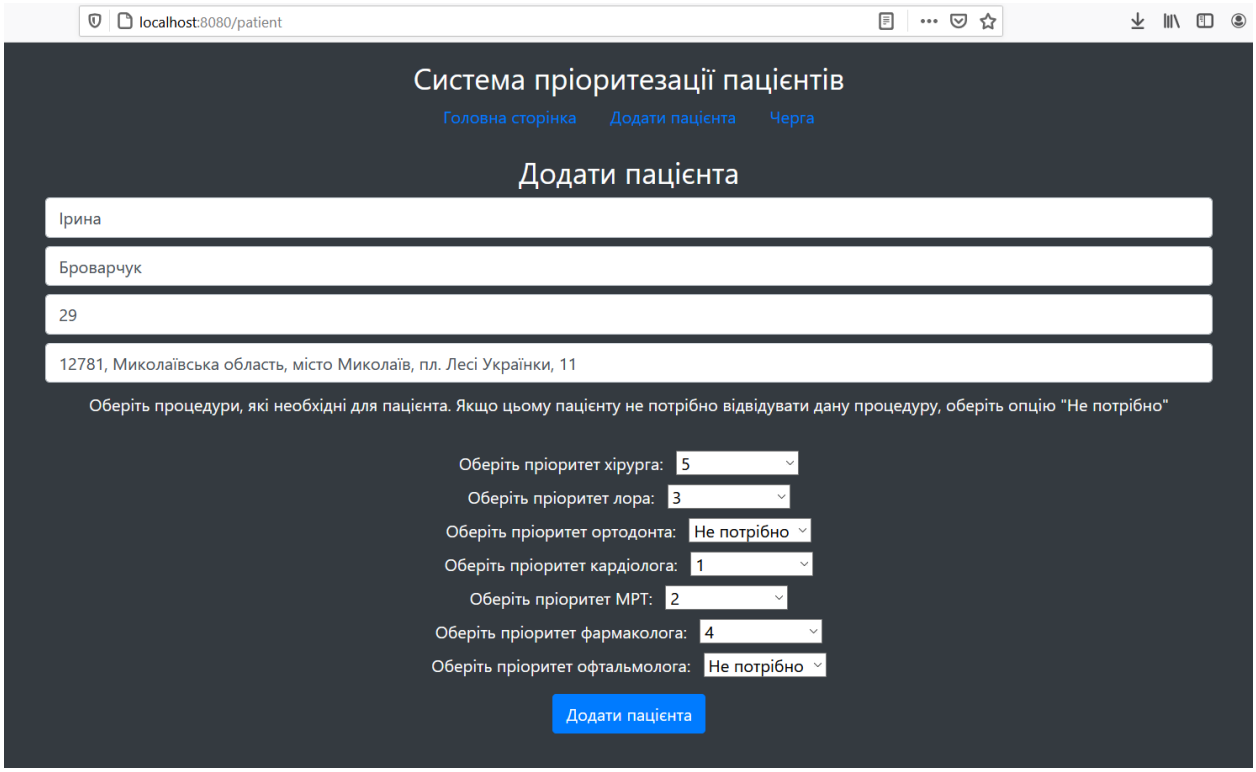


Рис. 4.4 Заповнена сторінка додавання пацієнта

Після того, як було додано першого пацієнта, головна сторінка приймає вигляд, як зображено на Рис. 4.5.

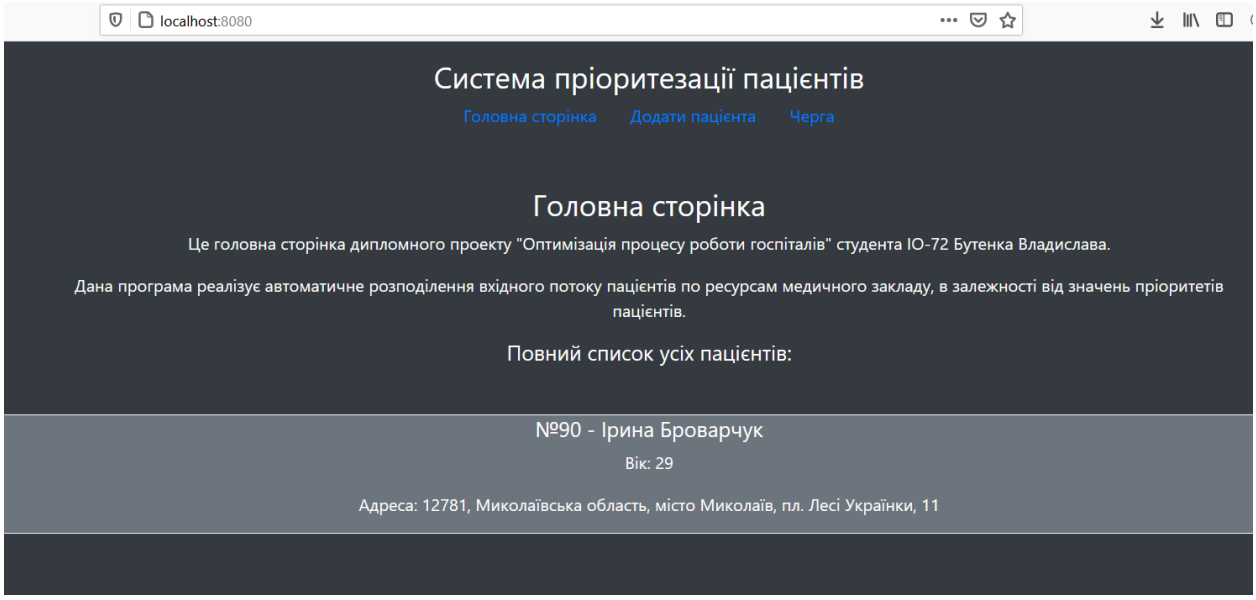


Рис. 4.5 Формування списку усіх пацієнтів

Як можна побачити, список пацієнтів на головній сторінці оновився. Теперь він складається з одного пацієнта – Ірини Броварчук.

Кожному новому пацієнтові призначається унікальний ID-номер, у випадку Ірини Броварчук – це 90. Цей номер буде використаний системою для розміщення пацієнта в існуючих чергах.

Для цілей тестування, нехай користувач додасть ще 9 пацієнтів із випадковими процедурами та пріоритетами. Тоді головна сторінка буде мати вигляд, зображений на Рис. 4.6.

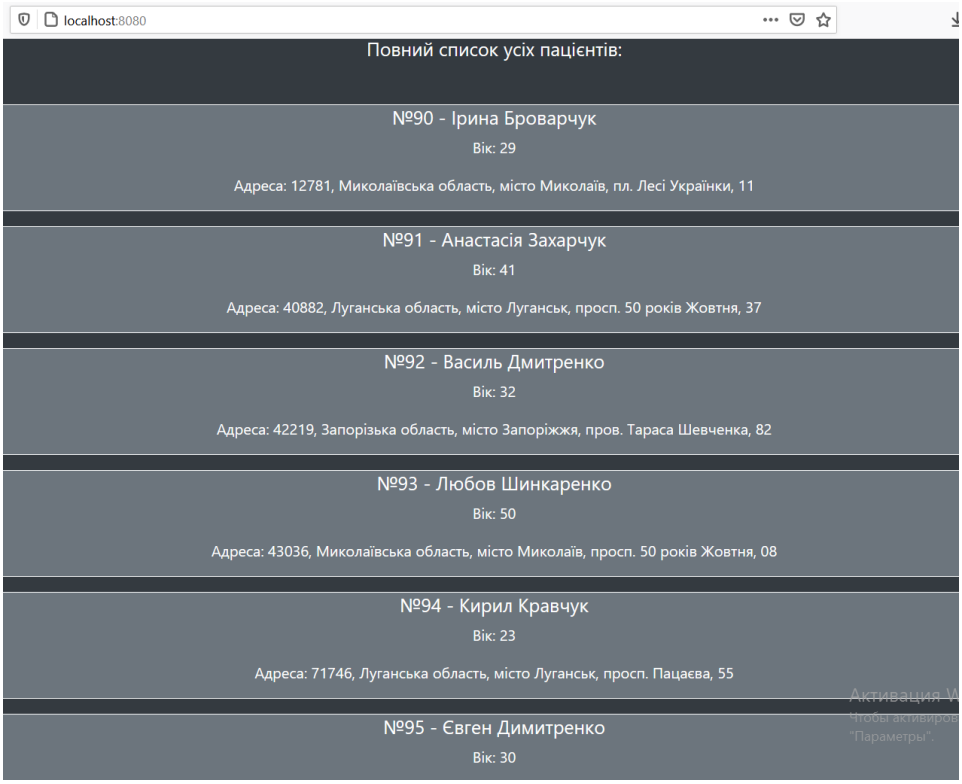


Рис. 4.6. Список із 10 пацієнтів

Повний список пацієнтів був оновлений, тепер до нього було додано усіх існуючих в системі пацієнтів.

Для того щоб переглянути усі черги, користувачеві необхідно натиснути на «Черги», що переведе його на відповідну сторінку, яка має вигляд, зображений на Рис. 4.7.

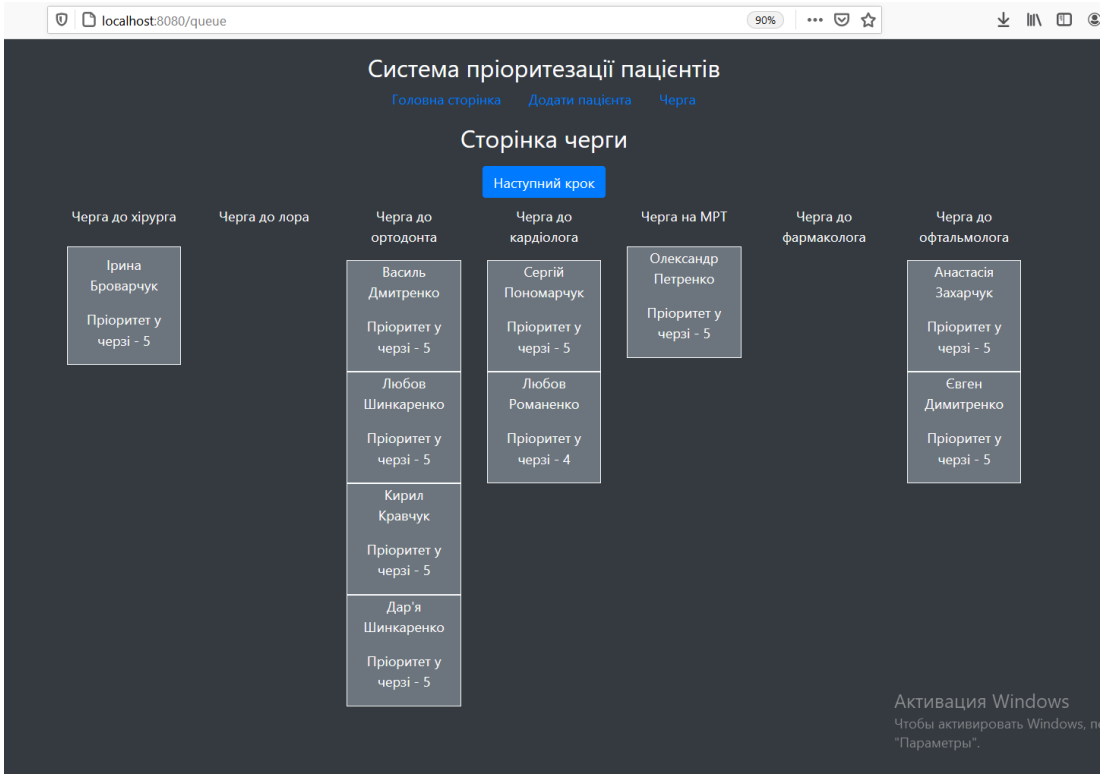


Рис. 4.7. Сторінка черги – перший такт

Як можна побачити, пацієнтов було розподілено по 7 чергах, в залежності від їхніх пріоритетів. Чим більше пріоритет пацієнта, тим ближче він знаходиться до початку черги. Пацієнти, що були додані останніми, знаходяться ближче до кінця черги.

Кнопка «Наступний крок» зрушує чергу на 1 такт. Це зроблено для імітації процесу черги у реальній лікарні. Припустимо, що проходження пацієнтом будь-якої процедури займає 1 такт. Таким чином, після натискання кнопки, кожна з 7 черг буде зрушуватися на 1 людину вперед. Ті пацієнти, що займали процедуру, після натискання кнопки завершать її, та встануть у чергу до наступної своєї процедури.

Розглянемо процес розподілу на прикладі першого пацієнта – Ірини Броварчук. Її пріоритети були наступні:

- Хірург - 5
- Лор - 3

- Ортодонт - 0
- Кардіолог - 1
- МРТ - 2
- Фармаколог - 4
- Офтальмолог - 0

Найбільший пріоритет має хірург – до нього Ірина і була направлена в першу чергу.

Після наступного такту сторінка черги має вигляд, зображений на Рис. 4.8.

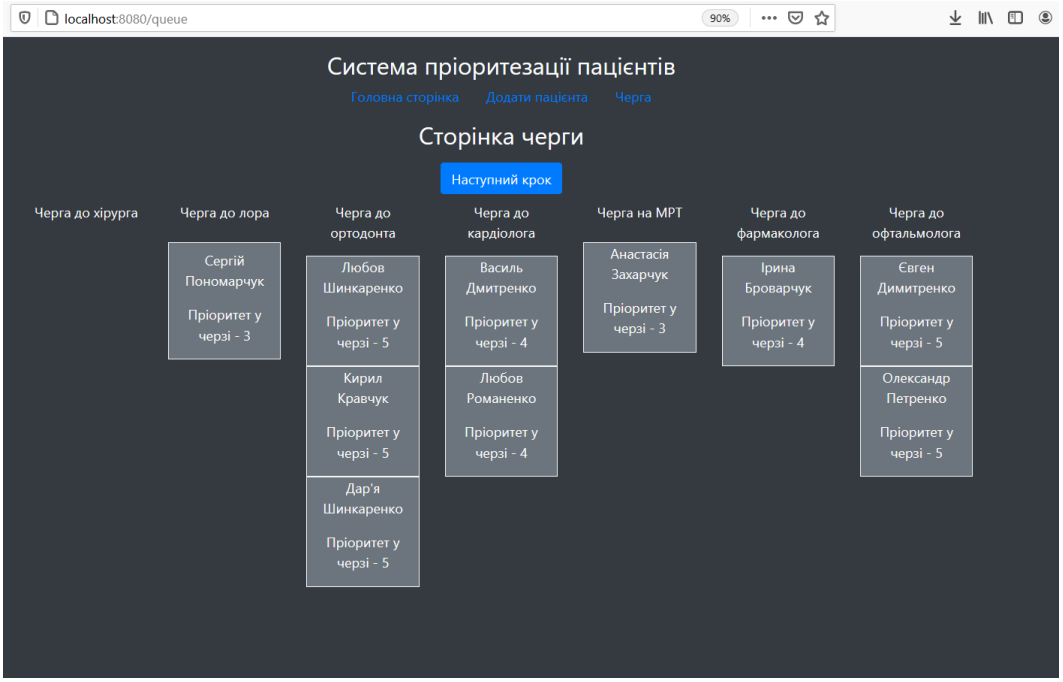


Рис. 4.8 Сторінка черги – другий такт

Наступною за значимістю процедурою для Ірини є фармаколог, тому після того, як процедура у хіргурга закінчена, система направляє Ірину до фармаколога. Нікому з інших пацієнтів на даний момент не потрібно до фармаколога, отже, Ірина знов стає першою в черзі.

Після наступного такту сторінка черги має вигляд, зображений на Рис. 4.9.

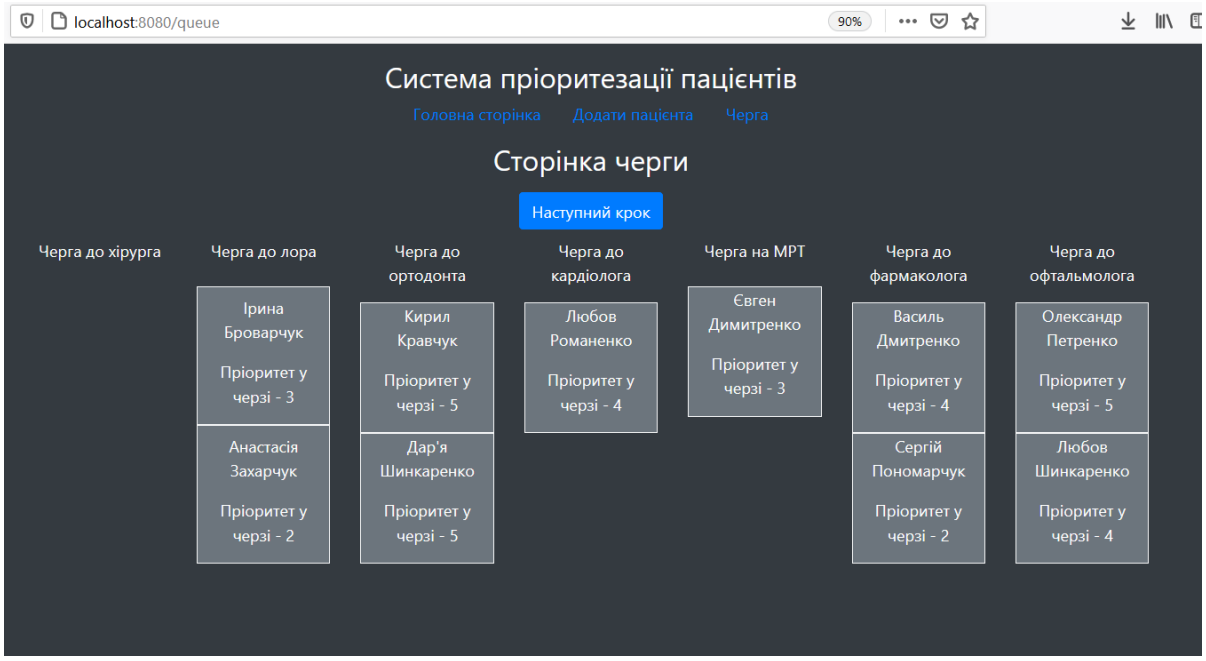


Рис. 4.9 Сторінка черги – третій такт

Система знов оновлює черги. Наступна процедура у Ірини – лор, із пріоритетом 3. У той самий час, така ж сама процедура й у Анастасії Захарчук, але з пріоритетом 2. Так як пріоритет Ірини вищий, вона визначається першою у цій черзі.

Після наступного такту сторінка черги має вигляд, зображений на Рис. 4.10.

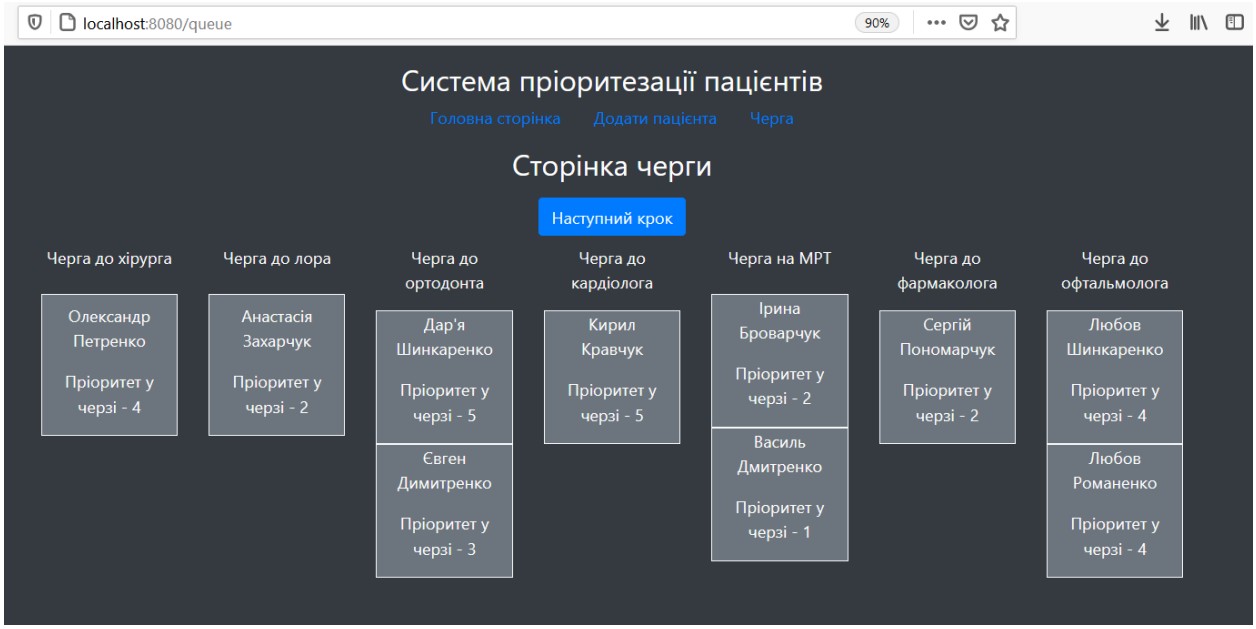


Рис. 4.10 Сторінка черги – четвертий такт

Черга оновлена, Ірина Броварчук завершила попередню процедуру та була направлена на МРТ. Василь Дмитренко мав менший пріоритет, тому система визначила Ірину, як першу в черзі.

Після наступного такту сторінка черги має вигляд, зображений на Рис. 4.11.

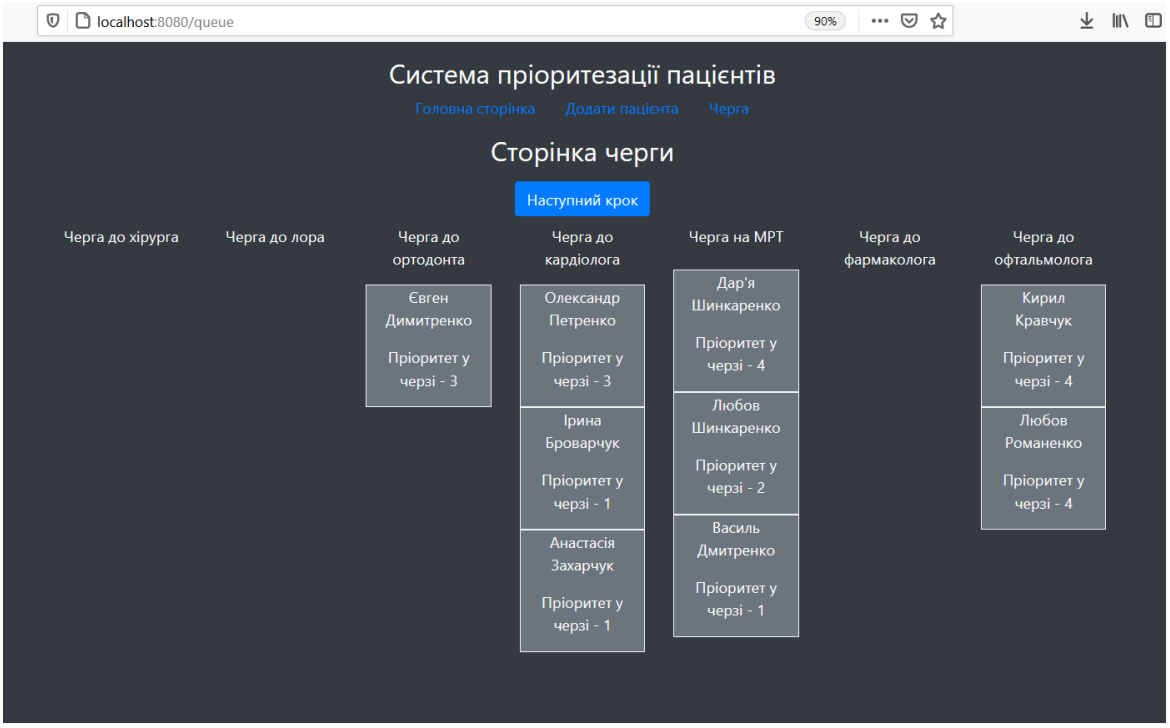


Рис. 4.11 Сторінка черги – п’ятий такт

Черга оновлена. Ірині залишилося пройти всього одну процедуру – кардіолога. Але, пріоритет Олександра Петренка вище, тому Ірина була додана в чергу після нього.

Після наступного такту сторінка черги має вигляд, зображений на Рис. 4.12.

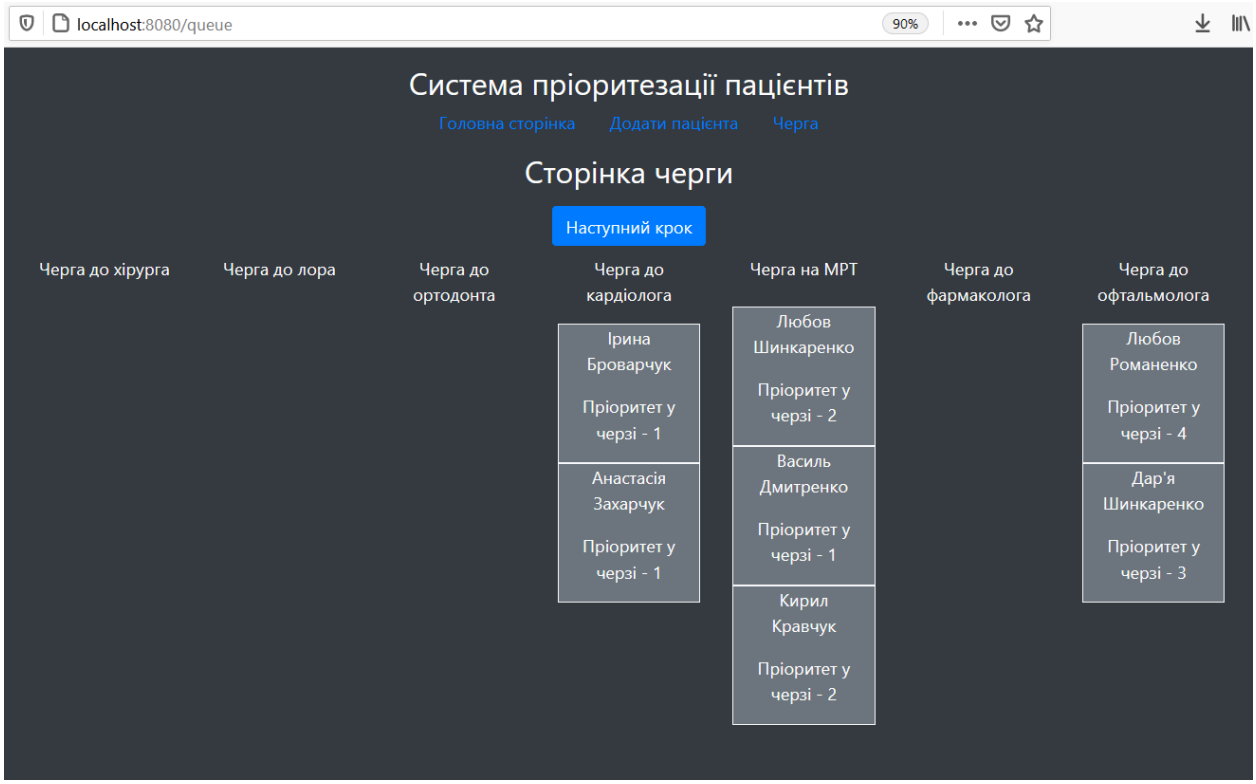


Рис. 4.12 Сторінка черги – шостий такт

Так як Олександр Петренко закінчив прийом у кардіолога, Ірина Броварчук буде записана до цієї процедури наступною.

Після наступного такту сторінка черги має вигляд, зображений на Рис. 4.13.

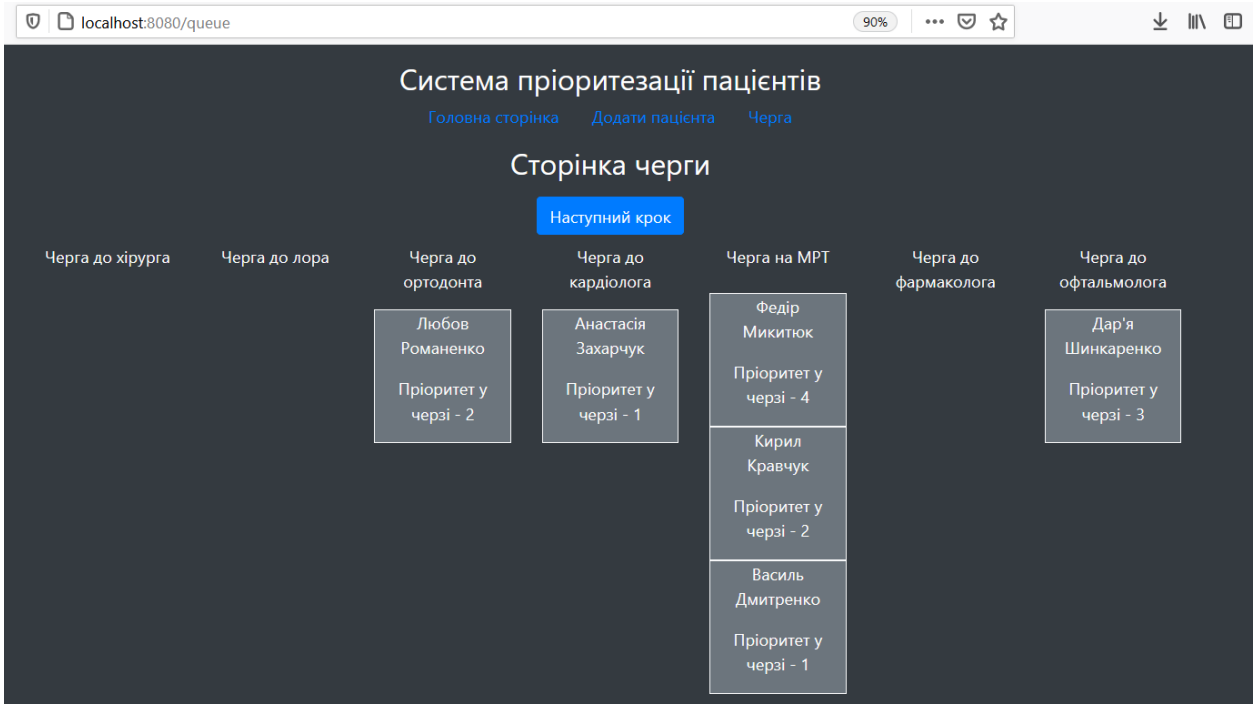


Рис. 4.13 Сторінка черги – сьомий такт

На цьому такті Ірина завершила усі свої процедури, отже вона може піти додому. Але у цей самий час був доданий ще один пацієнт – Федір Микитюк, пріоритет якого вищий за пріоритет Кирила Кравчука, який повинен був потрапити на МРТ після Любові Шинкаренко. Але система визначила, що медична допомога Федіру є більш значущою, ніж Кирилу, та змінила чергу таким чином.

Отже, можна зробити висновок, що система працює правильно і згідно з описаним вище алгоритмом.

4.2 Можливості вдосконалення системи

Для того, щоб дана система була конкурентноспроможньою, необхідно будет постійно її підтримувати і оновлювати. До даного програмного забезпечення можна додати наступний функціонал:

- Аналіз завантаженості кожної з процедур та перерозподіл пацієнтів по інших процедурах, якщо можливо.
- Відправлення сповіщень, що приходять на смартфон пацієнта.

- Створення звітів про завантаженість в розрізі кожної години часу – це буде корисною інформацією для власників медичного закладу.
- Можливість записати пацієнта на певний час у майбутньому, щоб система заздалегідь визначила найбільш оптимальний маршрут обходу медичного закладу.
- Використовувати більш сучасне апаратне та програмне забезпечення для того щоб збілшити швидкість роботи програми.

ВИСНОВКИ ДО РОЗДІЛУ 4

У четвертому розділі було продемонстровано створену систему та її повний функціонал. Під час демонстрації було створено 10 пацієнтів – кожен з котрих мав свій унікальний список процедур та пріоритетів. Після цього, було продемонстровано роботу алгоритма оптимізації черг, який, як було визначено під час тестування, реалізований правильно. Проведені експерименти підтвердили правильність обраних рішень і Мета роботи була досягнута!

Проаналізувавши роботу даної системи, можна сказати, що така система дозволить пацієнтам втрачати значно менше часу в чергах, а пацієнти, яким потрібна невідкладна медична допомога, будуть допущені до процедур в першу чергу.

Крім того, було визначено декілька можливих вдосконалень системи, адже сфера ІТ не стоїть на місці, і будь-який програмний продукт повинен постійно оновлюватися, для того щоб бути конкурентноспроможнім.

ВИСНОВКИ

Результатом даного бакалаврського проекту є реалізація системи оптимізації роботи госпіталів, за допомогою оптимізації черг.

В ході бакалаврської роботи було пройдено наступні етапи:

1. Було розглянуто актуальність даного рішення, проаналізовано існуючі приклади реалізації даної системи, та розглянуті основні методології розробки та імплементації подібних систем.
2. Проаналізувавши можливі алгоритми, за допомогою яких може бути створена найбільш ефективна електронна черга, було вирішено використовувати двійкову купу (Binary heap). Для створення серверної частини було обрано мову програмування Java, клієнтська частина була розроблена на HTML та CSS, із використанням Thymeleaf. Для зберігання даних, була обрана СКБД MySQL.
3. Було описано основні вимоги, яким потрібна відповідати система. Крім того, використовуючи нотацію моделювання бізнес-процесів BPMN 2.0, була створена графічна схема бізнес-логіки, яку система повинна підтримувати.
4. Було проведено тестування розробленої системи. При тестуванні були виправлені деякі помикли та покращена оптимізація. Як показали контрольні тестування, система працює правильно та відповідає усім вимогам.

При виконанні даного бакалаврського проекту було покращено теоретичні знання в області алгоритмів та створення клієнт-серверних додатків. Дані знання були закріплені на практиці при розробці веб-додатка системи електронної черги із пріоритетами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Цифрова охорона здоров'я у 2021 році: три напрями розвитку [Електроний ресурс]. Режим доступу - https://rb.ru/story/3-digital-healthcare-trends/ (дата звернення 12.04.2021).
2. Резолюція WHO # WHA58.28 eHealth Resolution [Електронний ресурс]. Режим доступу - https://www.who.int/healthacademy/media/WHA58-28-en.pdf?ua=1 (дата звернення 12.04.2021).
3. The benefits of QMS for hospital administration [Електронний ресурс]. Режим доступу - https://www.qminder.com/queue-management-benefits-hospital-administration/ (дата звернення 13.04.2021)
4. Компанія ТОВ «БМС» [Електроний ресурс]. Режим доступу - https://qlick.online/about-us.php (дата звернення 13.04.2021).
5. Електронна черга AKIS – сучасна система керування чергою» [Електроний ресурс]. Режим доступу - https://www.studioer.ru/ (дата звернення 13.04.2021).
6. Система керування чергою SUO [Електроний ресурс]. Режим доступу - https://zkr-systems.com.ua/hospital-integration-softwares/zkr-electronic-queue/ (дата звернення 13.04.2021).
7. Priority queues [Електроний ресурс]. Режим доступу - https://www.cs.purdue.edu/homes/ayg/CS251/slides/chap6.pdf (дата звернення 01.05.2021).
8. A priority queue with the time-finger property [Електроний ресурс]. Режим доступу - https://www.sciencedirect.com/science/article/pii/S1570866712000834 (дата звернення 01.05.2021).
9. CS241 – Lecture notes: Heap [Електроний ресурс]. Режим доступу - https://www.cpp.edu/~ftang/courses/CS241/notes/heap.htm (дата звернення 02.05.2021).

10. Bi-parental heaps (Beaps) Heap [Електроний ресурс]. Режим доступу - https://ece.uwaterloo.ca/~dwharder/aads/Algorithms/Beaps/ (дата звернення 02.05.2021).
11. Binary Heap [Електроний ресурс]. Режим доступу - https://www.geeksforgeeks.org/binary-heap/ (дата звернення 05.05.2021).
12. Binominal Heap [Електроний ресурс]. Режим доступу - https://brilliant.org/wiki/binomial-heap/ (дата звернення 05.05.2021).
13. Theory of 2-3 heaps [Електроний ресурс]. Режим доступу - https://www.researchgate.net/publication/226874995_Theory_of_2-3_Heaps (дата звернення 05.05.2021).
14. MVC Tutorial for Beginners [Електроний ресурс]. Режим доступу - https://www.guru99.com/mvc-tutorial.html (дата звернення 07.05.2021).
15. Використання Java [Електроний ресурс]. Режим доступу - https://www.java.com/ru/download/help/index_using.html (дата звернення 07.05.2021).
16. Python official website [Електроний ресурс]. Режим доступу - https://www.python.org/ (дата звернення 10.05.2021).
17. Ruby official website [Електроний ресурс]. Режим доступу - https://www.ruby-lang.org/en/ (дата звернення 10.05.2021).
18. Thymeleaf Documentation [Електроний ресурс]. Режим доступу - https://www.thymeleaf.org/documentation.html (дата звернення 12.05.2021).
19. MySQL Official website [Електроний ресурс]. Режим доступу - https://dev.mysql.com/doc/ (дата звернення 12.05.2021).

ДОДАТОК 1

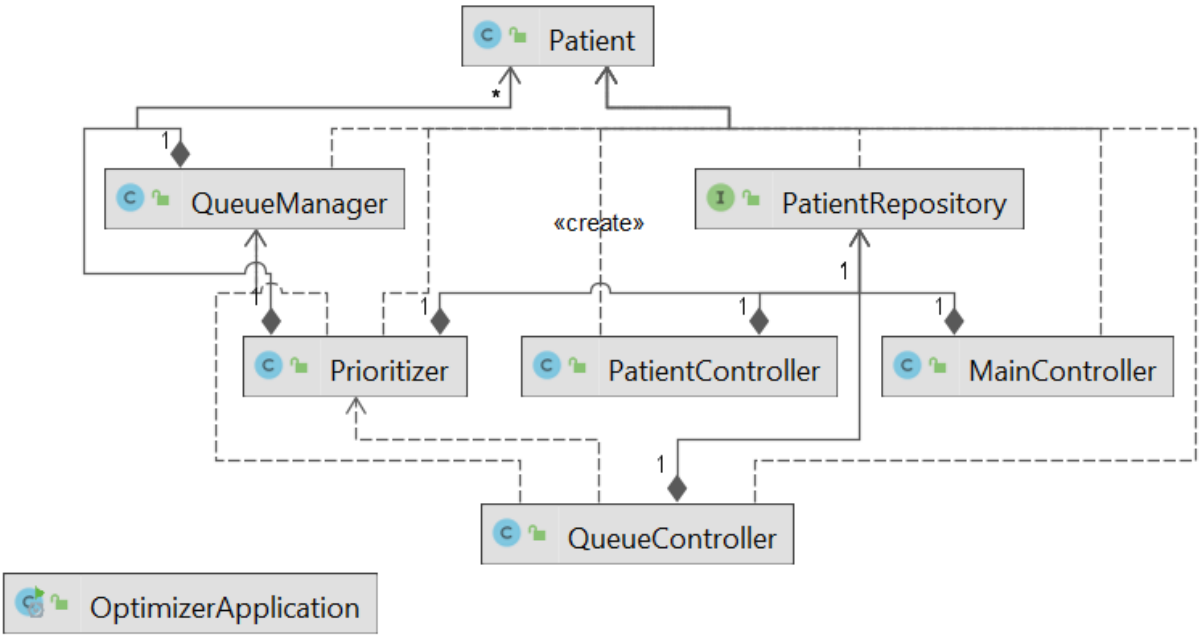
Опитмізація процесу роботи госпіталів

Діаграма класів

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2021 р.



					ІАЛЦ.467200.004 Д1		
Зм.		№ документа	Підп.	Дата			
Розробив		Бутенко В.О.			Оптимізація процесу роботи госпіталів Діаграма класів	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Антонюк А.І.					
Н.контр.		Сімоненко В. П.				НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-72	
Затверд.		Стіренко С. Г.					

ДОДАТОК 2

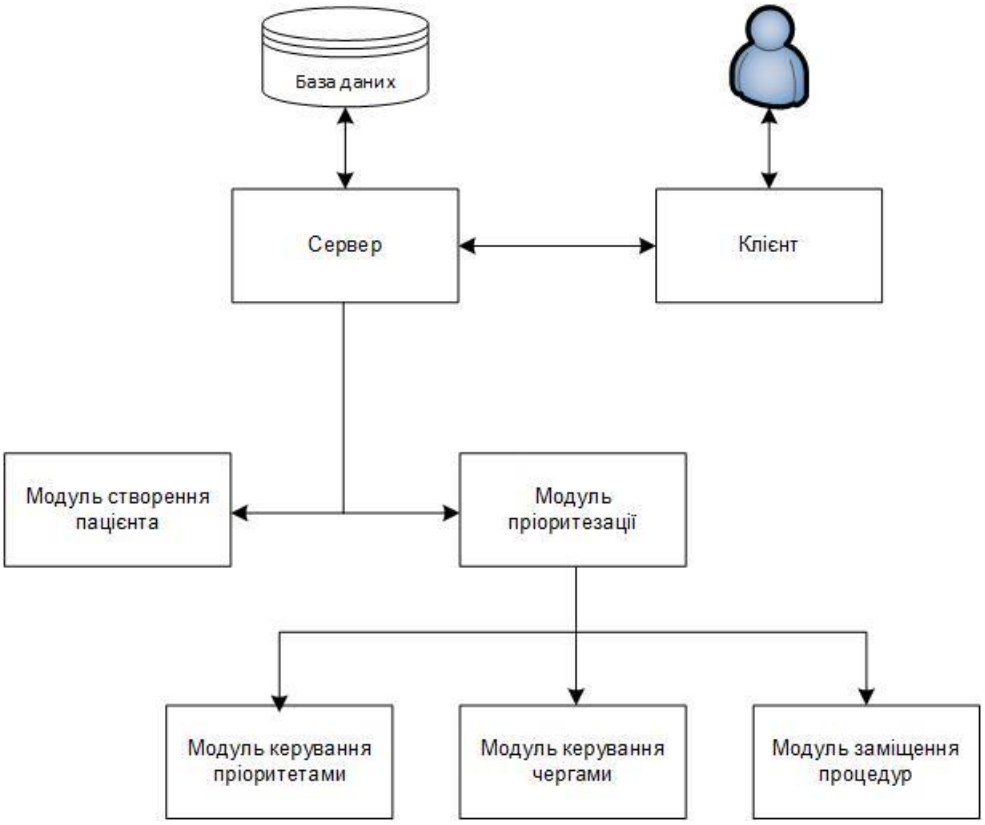
Опитмізація процесу роботи госпіталів

Структурна схема системи

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2021р.



					ІАЛЦ.467200.005 Д2	
Зм.		*№ документа*	*Підп.*	*Дата*		
Розробив		*Бутенко В.О.*			*Оптмізація процесу роботи госпіталів*	*Літ.* / *Аркуш* 1 / *Аркушів* 1
Перевірив		*Антонюк А.І.*				
Н.контр.		*Сімоненко В.П.*			*Структурна схема системи*	*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-72*
Затверд.		*Стіренко С. Г.*				

ДОДАТОК 3

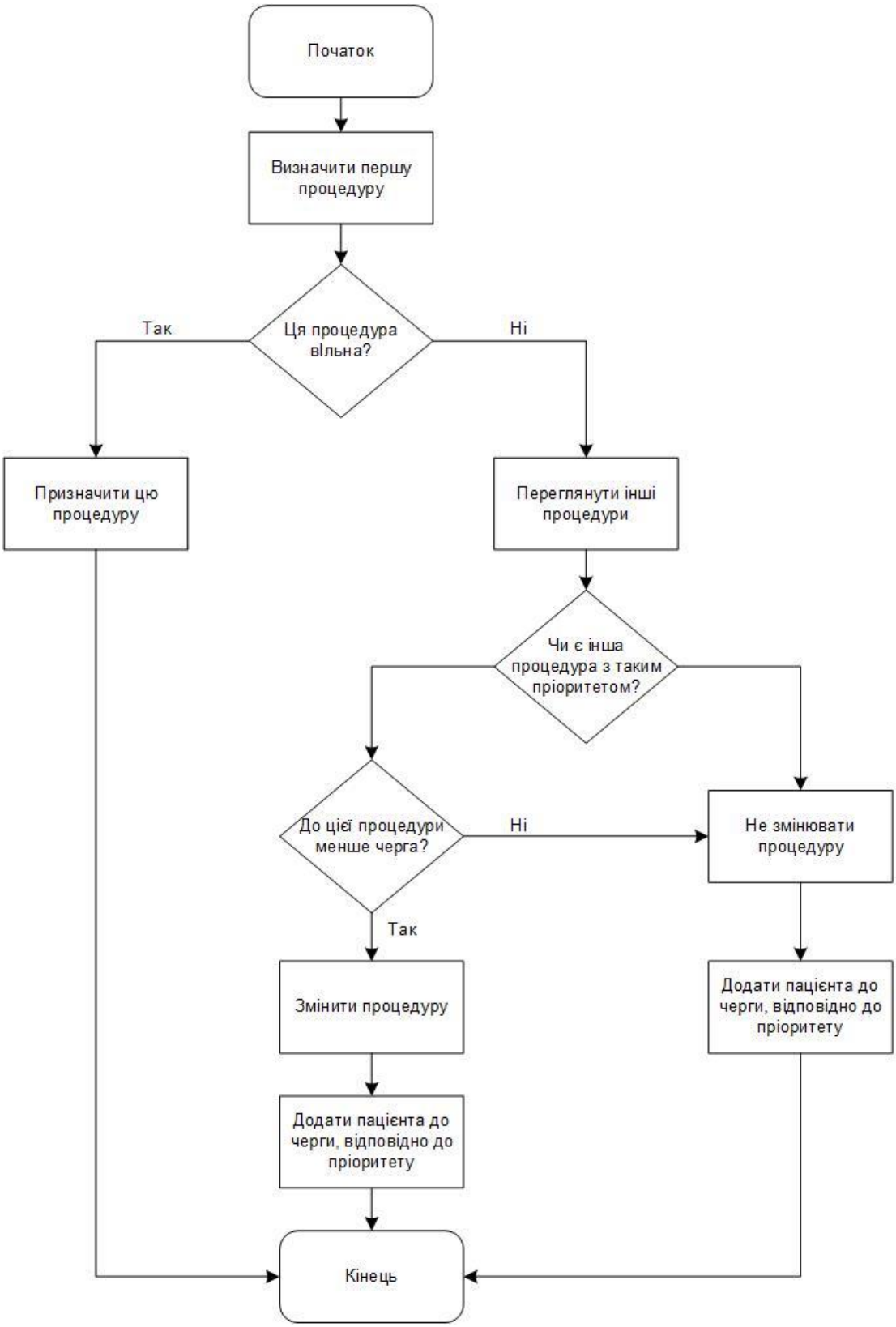
Прогнозування фродових транзакцій

Схема алгоритму моделі

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2021



					ІАЛЦ.467200.006 Д3	
Зм.		*№ документа*	*Підп.*	*Дата*		
Розробив		*Бутенко В.О.*			*Опитмізація процесу роботи госпіталів* *Схема алгоритму моделі*	*Літ.* / *Аркуш* 1 / *Аркушів* 1
Перевірив		*Антонюк А.І..*				
Н.контр.		*Сімоненко В. П.*				*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-72*
Затв.		*Стіренко С. Г.*				