МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

Аліна САВЧЕНКО

«_____»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТРА"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: «Розробка веб-додатку для управління лікарні»

Виконав: Журавський Артем Олегович

Керівник: к.т.н., доцент кафедри КІТ Олександр Григорович Харченко

Нормоконтролер: **Ігор РАЙЧЕВ**

Київ – 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Аліна САВЧЕНКО

"___"_____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Журавського Артема Олеговича

(прізвище, ім'я, по батькові)

1.**Тема роботи:** «Розробка веб-додатку для управління лікарні», затверджена наказом ректора від "29"вересня 2023р. за № 1976/ст

2. Термін виконання роботи: : з 02 жовтня 2023р. по 31 грудня 2023р.

3. Вихідні дані до роботи: теоретичні відомості розробки веб-додатку для лікарні, мова Java, backend фреймворк Spring,

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): вступ, дослідження систем управління лікарнею, середовище розробки веб-додабку, розробка веб-додатку для лікарні.

5. Перелік обов'язкового графічного матеріалу: інформативні рисунки, графічні скріншоти роботи системи, слайди презентації в MS PowerPoint.

6. Календарний план-графік

№ п/п	**Завдання**	**Термін виконання**	**Підпис керівника**
1	Дослідження та аналіз предметної області.	02.10.23 – 05.10.23	
2	Проведення консультацій з науковим керівником	06.10.23 – 10.10.23	
3	Написання та підготовка розділу 1	11.10.23 – 13.10.23	
4	Написання та підготовка розділу 2	14.10.23 – 15.10.23	
5	Написання та підготовка розділу 3	16.10.23 – 20.10.23	
6	Оформлення пояснювальної записки	21.10.23 – 31.10.23	
7	Підготовка презентації та доповіді	01.11.23 – 20.11.23	
8	Створення доповіді та презентації	21.11.23 – 28.11.23	
9	Підготовка до захисту дипломної роботи	29.11.23 – 10.12.23	

Дата видачі завдання: 02 жовтня 2023р.

Керівник дипломної роботи: _____ Олександр ХАРЧЕНКО
(підпис керівника)

Завдання прийняв до виконання:
_____ Артем ЖУРАВСЬКИЙ
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи “Розробка веб-додатку для управління лікарні”: містить 80 сторінок, 22 рисунки, 20 наукових джерел.

Ключові слова: ЛІКАРНЯ, ВЕБ - ДОДАТОК, БЕЗПЕКА, ТЕСТУВАННЯ

Мета кваліфікаційної роботи: створення веб-додатку для управління лікарні.

Об’єкт дослідження: процес створення веб-додатку для управління організаціями.

Предмет дослідження: процес створення веб-додатку для управління лікарні.

Метод дослідження: методи порівняльного аналізу методів реалізації, обробка літературних джерел.

Результат проекту: Розроблений веб-додаток для управління лікарні.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

IT - Інформаційні технології

ООП - Об'єктно-орієнтоване програмування

SQL - Structured Query Language (мова структурованих запитів)

NoSQL - Not only SQL (нереляційна база даних)

JPA - Java Persistence API

ORM - Object-Relational Mapping (об'єктно-реляційне відображення)

CI/CD - Continuous Integration/Continuous Deployment (постійна інтеграція/постійне розгортання)

ВСТУП

У сучасному світі інформаційні технології стають невід'ємною складовою в багатьох галузях людської діяльності. Однією з таких галузей є охорона здоров'я, де ефективне управління та координація роботи медичних закладів та їхнього персоналу відіграють критичну роль у забезпеченні якості надання медичних послуг. У цьому контексті розробка веб-додатку для управління лікарні стає актуальною та важливою задачею.

Цей дипломний проект спрямований на створення інноваційного веб-додатку, який спростить та покращить процеси управління лікарні. Завдяки сучасним технологіям та інструментам розробки, я маю можливість створити зручний та ефективний інструмент для керівництва лікарнею, медичним персоналом та адміністраторами.

Цей веб-додаток буде спрямований на автоматизацію багатьох рутинних процесів, що відбуваються в лікарні, включаючи управління прийомами пацієнтів, розподіл ресурсів та медичного персоналу, ведення обліку медичних записів та багато інших аспектів лікарської практики.

У цьому дипломному проекті я детально розглянув процес розробки веб-додатку, від аналізу вимог і вибору технологічного стеку до фактичної реалізації та тестування системи. Я розгляну ключові аспекти безпеки даних, забезпечення високої доступності, а також зручного та інтуїтивно зрозумілого інтерфейсу для користувачів. Також проведу аналіз при виборі між реляційної та нереляційної базами даних для збереження інформації.

Можемо зазначити, що розробка цього веб-додатку не лише сприятиме покращенню управління лікарні, але й відкриє нові можливості для подальшого розвитку систем охорони здоров'я. Результати цієї роботи стануть важливим внеском у поліпшення медичних послуг та забезпечення більшого комфорту для пацієнтів та медичного персоналу.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ СИСТЕМ УПРАВЛІННЯ ЛІКАРНЕЮ

Дослідження систем управління лікарнею - розділ присвячений дослідженню існуючих систем управління лікарнею, а також дослідженню потреб лікарні. На основі проведених досліджень сформовані вимоги до веб-додатку.

1.1. Аналіз існуючих систем управління лікарнею

У цьому розділі докладно розглядаються існуючі системи управління лікарнею, які були піддані дослідженню в рамках практики. До цих систем належать:

- **Системи електронної медичної картки (ЕМК)**: Ці системи призначені для зберігання медичної інформації про пацієнтів та ведення обліку медичних послуг. Вони зазвичай включають в себе базу даних пацієнтів, історії хвороби, результати обстежень та інші медичні записи.
- **Системи управління пацієнтами (CPM)**: Ці системи спрямовані на реєстрацію пацієнтів, ведення їхньої історії хвороби та планування медичних послуг. Вони допомагають лікарням управляти пацієнтським потоком та оптимізувати надання медичних послуг.
- **Системи управління медичними кадрами (МКР)**: Ці системи призначені для ведення обліку медичних працівників, включаючи їхню кваліфікацію, досвід роботи та графік роботи. Вони допомагають лікарням ефективно розподіляти медичний персонал та забезпечувати якість надання медичних послуг.

- **Системи управління матеріальними ресурсами (MP)**: Ці системи призначені для обліку матеріальних ресурсів, які використовуються в лікарні, такі як ліки, медичне обладнання та реактиви.

Під час дослідження було проведено докладний аналіз функціональних можливостей існуючих систем управління лікарнею. Виявлено, що ці системи, несмотря на свою значущість, мають ряд значущих недоліків:

- **Негнучкість**: Існуючі системи не завжди можуть бути адаптовані до конкретних потреб конкретної лікарні. Вони часто мають вбудовані обмеження, які ускладнюють зміну функціоналу або процесів.
- **Необхідність внесення змін**: Існуючі системи часто потребують внесення змін, щоб відповідати новим вимогам і стандартам. Це може бути часко витратним і вимагати додаткових фінансових витрат.
- **Висока вартість**: Багато існуючих систем можуть бути дорогими в розробці, впровадженні та підтримці. Високі витрати можуть стати перешкодою для багатьох лікарень, особливо для малих та середніх медичних закладів.

Отже, аналіз існуючих систем управління лікарнею підкреслює необхідність створення нового, більш гнучкого та ефективного веб-додатку для управління лікарнею, який б враховував сучасні вимоги та потреби медичного сектору.

1.1.1. Порівняльний аналіз функціональних можливостей різних систем управління лікарнею

Порівняльний аналіз функціональних можливостей різних систем управління лікарнею є важливим етапом при виборі оптимальної системи. Він забезпечує можливість оцінити, як кожна система відповідає специфічним потребам лікарні. Ось кілька кроків, які можна виконати для такого аналізу:

1. Визначення функціональних вимог: Спочатку визначаються конкретні функціональні вимоги лікарні. Це можуть бути такі функції, як управління призначеннями, облік пацієнтів, реєстрація даних, розклад лікарів тощо.
2. Список функціональності кожної системи: Потім складається список функціональних можливостей кожної системи управління лікарнею. Це включає в себе всі доступні модулі, опції та функції, які пропонує кожна система.
3. Порівняльна таблиця функціональних можливостей: Створюється таблиця, де для кожної системи приводиться список її функціональних можливостей. Це допомагає провести прямі порівняння між системами.
4. Оцінка відповідності: Кожна функціональність оцінюється наскільки вона відповідає вимогам лікарні. Це може бути оцінка за шкалою, де 0 - функціонал відсутній, 1 - функціонал присутній, але не повністю задовольняє потреби, і 2 - повна відповідність.
5. Формування звіту: На основі порівняння формується звіт, який містить результати аналізу. Це може бути графічне представлення порівняльних характеристик або текстовий звіт з обґрунтуванням рішення та вибору найбільш підходящої системи.
6. Висновки та рекомендації: Завершується аналіз висновками та рекомендаціями. Тут робиться обґрунтування вибору певної системи на основі результатів порівняльного аналізу функціональності.

1.2. Аналіз потреб лікарні

Під час аналізу потреб лікарні в інформаційній системі для управління, було визначено кілька ключових аспектів, які важливі для ефективного функціонування медичного закладу:

- **Необхідність в єдиній інформаційній системі:**

  Лікарня має велику кількість різноманітних процесів та даних, що потребують управління. Потрібна єдина інформаційна система, яка б об'єднувала всі ці процеси та дозволяла легко обмінюватися даними між відділеннями та підрозділами. Це дозволить уникнути дублювання даних, забезпечити їхню консолідацію та забезпечити більшу точність та доступність інформації для всіх зацікавлених сторін.

- **Необхідність в адаптивній системі:**

  Медичний сектор постійно зазнає змін у зв'язку зі змінами в законодавстві, технологіях та потребах пацієнтів. Тому лікарня має потребу в інформаційній системі, яка була б гнучкою та легко адаптовувалася до змін. Це включає можливість додавання нових функціональностей, зміну робочих процесів та інтеграцію з новими технологіями без значних зусиль і витрат на розробку.

- **Необхідність в доступній системі:**

  Управління лікарнею вимагає доступності інформації для всіх співробітників, незалежно від їхньої локації та ролі. Це означає, що система повинна бути легкою у використанні та мати можливість доступу через різні пристрої, включаючи комп'ютери, планшети та смартфони. Такий підхід сприятиме покращенню координації роботи та спільної діяльності всього медичного колективу.

Аналіз цих потреб визначає важливість створення веб-додатку для управління лікарні, який буде відповідати цим вимогам та сприяти покращенню організації медичної діяльності.

1.2.1. Дослідження специфічних потреб лікарні в контексті кращого впровадження ІТ-систем

Для дослідження специфічних потреб лікарні в контексті впровадження ІТ-систем, важливо звернутися до ключових стейкхолдерів та отримати від них інформацію про конкретні потреби та вимоги. Ось кілька кроків, які можна виконати для проведення такого дослідження:

1. Збір вимог від стейкхолдерів: Проведення співбесід, опитувань або взаємодії з лікарями, медичним персоналом, адміністраторами лікарні та пацієнтами для виявлення їх потреб. Це допоможе отримати прямі відгуки та конкретні вимоги щодо системи управління.
2. Аналіз поточних проблем та обмежень: Важливо виявити, з якими проблемами стикається лікарня в даний час та які обмеження існують у поточних процесах управління.
3. Ідентифікація невпокоючих та невирішених питань: Звернення до аспектів, які можуть бути не вирішені поточними системами чи процесами управління.
4. Оцінка існуючих рішень на ринку: Проведення аналізу різних ІТ-рішень, доступних на ринку, та їх порівняння з вимогами та потребами лікарні.
5. Формування списку вимог та побажань: На основі зібраних відгуків та даних формується список вимог та побажань, які повинна вирішувати нова ІТ-система.
6. Створення концепції впровадження: На основі вимог формується концепція, яка описує, як нова система буде впроваджуватися та які вигоди вона принесе.
7. Звіт про виявлені потреби та рекомендації: На завершення проводиться підготовка звіту, в якому описуються виявлені потреби, вимоги та рекомендації щодо впровадження ІТ-системи управління для лікарні.

1.3. Аналіз інформаційних технологій в медичному секторі

У цьому підрозділі проведено глибокий аналіз інформаційних технологій, які використовуються в медичному секторі для управління лікарнями та медичними закладами. Досліджено їхній вплив на сучасну медицину та роль у покращенні процесів надання медичних послуг. Аналіз включає в себе наступні аспекти:

1. Застосування електронної медичної документації (EMR/EHR)

Аналіз систем електронної медичної документації, які дозволяють зберігати та обмінюватися медичною інформацією в електронному форматі. Досліджено переваги цих систем, зокрема швидший доступ до інформації, зменшення помилок у веденні медичної документації та підвищення координації між медичним персоналом.

2. Телемедицина та дистанційне нагляд

Розгляд систем телемедицини, які дозволяють здійснювати консультації та діагностику пацієнтів на віддаленій основі. Аналіз впливу цих технологій на доступність медичних послуг для віддалених та обмежених регіонів, а також підвищення ефективності консультацій та діагностики.

3. Аналіз систем управління медичними даними (HIS)

Досліджено системи управління медичними даними, які дозволяють лікарням вести облік та аналізувати клінічні дані, рецепти, розклади, історії хвороби та іншу медичну інформацію. Аналіз включає в себе ефективність таких систем у покращенні якості надання медичних послуг та сприянні прийняттю управлінських рішень.

4. Вплив інтернету речей (IoT) у медицині

Аналіз використання IoT-технологій в медицині, які дозволяють збирати дані з медичних приладів та моніторингових систем для реального часу та подальшого аналізу. Досліджено, як ці дані можуть бути використані для покращення діагностики, лікування та нагляду за пацієнтами.

5. Заходи забезпечення інформаційної безпеки в медицині

Розгляд методів та технологій забезпечення конфіденційності та цілісності медичних даних. Аналіз систем автентифікації, шифрування та інших заходів, що застосовуються в медичному секторі для захисту медичної інформації від несанкціонованого доступу.

6. Вплив інформаційних технологій на ефективність лікарень

Загальний аналіз впливу інформаційних технологій на ефективність функціонування лікарень та медичних закладів. Вивчено переваги управління медичними процесами, оптимізації ресурсів та зменшення адміністративних витрат завдяки використанню сучасних IT-рішень.

Цей аналіз інформаційних технологій в медичному секторі надає глибоке розуміння того, як сучасні технології впливають на медичну практику та допомагають покращувати доступність, якість та ефективність надання медичних послуг. Ці знання є важливими для подальшого проектування та розробки веб

1.3.1. Аналіз інноваційних рішень у використанні IT-технологій в медичному секторі

Аналіз інноваційних рішень у використанні IT-технологій в медичному секторі включає в себе дослідження передових рішень, які використовуються або можуть бути застосовані у сфері охорони здоров'я. Ось кілька ключових пунктів для такого аналізу:

1. **Електронні медичні записи (EMR/EHR):** Дослідіть системи електронних медичних записів, їх функціональні можливості та переваги для зберігання медичної інформації.
2. **Телемедицина:** Вивчіть рішення, які дозволяють здійснювати консультації та надання медичної допомоги віддалено за допомогою технологій зв'язку.

3. **Аналіз медичних даних:** Розгляньте інструменти для аналізу медичних даних, машинного навчання та штучного інтелекту для прогнозування, діагностики та управління медичними захворюваннями.
4. **Мобільні додатки для здоров'я:** Вивчіть мобільні додатки, які надають можливість відстеження стану здоров'я, планування прийому ліків, моніторингу фізичної активності та інші, які спрямовані на збереження здоров'я.
5. **IoMT (Internet of Medical Things):** Аналізуйте використання підключених медичних пристроїв та датчиків для збору даних пацієнтів та віддаленого моніторингу.
6. **Цифрові рішення для аптек:** Дослідіть технології, що полегшують управління аптечними запасами, рецептами, та наданням фармацевтичних послуг.
7. **Блокчейн в медицині:** Розгляньте можливості застосування блокчейн-технологій для забезпечення безпеки, відстеження медичних записів та ідентифікації пацієнтів.

Цей аналіз дозволить зрозуміти сучасні тренди в IT-технологіях в медичному секторі та визначити, які з них можуть бути корисними для вашого проекту управління лікарнею.

1.4. Сформування вимог до веб-додатку

На основі проведених досліджень і аналізу потреб лікарні були сформульовані вимоги до веб-додатку, який буде розроблений в рамках практики. Ці вимоги поділяються на дві основні категорії: функціональні та нефункціональні, і визначають як функціональний зміст, так і параметри функціонування системи.

Функціональні вимоги:

Реєстрація та авторизація користувачів: Веб-додаток повинен надавати можливість реєстрації нових користувачів та авторизації існуючих. Кожен користувач має мати особистий обліковий запис з власними правами доступу.

Управління пацієнтами: Веб-додаток повинен дозволяти вести облік пацієнтів, зберігати їхні медичні записи, історії хвороби, призначення та медичні послуги. Також передбачено можливість реєстрації нових пацієнтів та їхню ефективну фільтрацію.

Управління медичними працівниками: Веб-додаток повинен забезпечувати можливість ведення обліку медичних працівників, включаючи інформацію про їхню кваліфікацію, досвід роботи та робочий графік. Також необхідно передбачити можливість зміни статусу та прав доступу медичних працівників.

Управління матеріальними ресурсами: Веб-додаток повинен дозволяти вести облік матеріальних ресурсів, які використовуються в лікарні, включаючи ліки, медичне обладнання, реактиви та інше. Система має надавати звіти про залишки та витрати ресурсів.

Нефункціональні вимоги:

1. Доступність: Веб-додаток повинен бути доступний для всіх співробітників лікарні без обмежень за місцем роботи або доступу до Інтернету. Система має бути стійкою до навантаження та готовою до роботи цілодобово.
2. Безпека: Веб-додаток повинен забезпечувати високий рівень безпеки даних, включаючи захист від несанкціонованого доступу та злому. Має бути використана автентифікація та авторизація користувачів, а також шифрування даних.
3. Швидкість: Веб-додаток повинен працювати швидко і без зайвих затримок, забезпечуючи оперативний доступ до інформації та функціональності.

4. Надійність: Веб-додаток повинен бути надійним і стабільним у роботі, не допускаючи частих помилок та аварійного завершення роботи.

Ці вимоги є основою для подальшої розробки веб-додатку для управління лікарні і гарантують, що система буде відповідати потребам та очікуванням користувачів та забезпечувати ефективне управління лікарнею.

1.5. Впровадження та масштабування

В цьому розділі розглядається план впровадження вашого веб-додатку для управління лікарнею та можливості його масштабування на майбутнє. Детально описані стратегії, ресурси та технічні аспекти, які я планую використовувати для успішного впровадження та розвитку системи.

План впровадження

Впровадження вашого веб-додатку у лікарню є кроком, важливим для його успішної реалізації. Виходячи із вивчених підходів та найкращих практик, визначено план дій для впровадження:

1. Підготовчий етап:

Визначення проектної команди та обрання керівника проекту.
Оцінка ресурсів, необхідних для впровадження, включаючи фінансові, людські та технічні ресурси. Розробка графіку впровадження та визначення завдань і відповідальних за їх виконання.

2. Розгортання системи:

Налаштування тестового середовища для перевірки функціональності та забезпечення стабільності системи. Проведення тренінгів та підготовка персоналу до використання веб-додатку. Пілотне впровадження в обмеженому обсязі для оцінки продуктивності та виявлення можливих проблем.

3. Повне впровадження:

Розгортання веб-додатку на всіх підрозділах та відділеннях лікарні. Поступове введення у роботу на всіх рівнях організації, включаючи лікарів, медсестер та адміністративний персонал. Підтримка та навчання користувачів під час переходу на нову систему.

Масштабування

Важливим аспектом є можливість масштабування веб-додатку для розширення його функціональності та застосування в інших медичних закладах. Ось деякі аспекти масштабування:

1. Розширення функціональності:

Планування розширення функціональних можливостей веб-додатку для включення додаткових функцій, таких як телемедицина, аналіз даних, дистанційне нагляд і більше.

2. Підтримка більшої кількості користувачів:

Розробка стратегії масштабування, яка дозволить веб-додатку обслуговувати більшу кількість пацієнтів та медичного персоналу.

3. Розширення на інші медичні заклади:

Вивчення можливостей розгортання нашого веб-додатку в інших лікарнях або медичних закладах та розробка плану розширення.

4. Моніторинг та оптимізація продуктивності:

Постійний моніторинг продуктивності та ефективності системи з метою виявлення можливих проблем та їх вирішення.

Завдання та ресурси

Для успішного впровадження та масштабування веб-додатку передбачено наступні завдання та ресурси:

Визначення необхідних фінансових ресурсів та розробка бюджету. Формування проектної команди та обрання керівника проекту. Оптимізація існуючої інфраструктури для підтримки додаткового навантаження. Розробка документації та плану навчання для користувачів. Цей розділ містить стратегію впровадження та масштабування вашого веб-додатку, вказує на завдання та ресурси, необхідні для цього процесу. Він покликаний забезпечити успішну реалізацію проекту та його майбутнє розвиток.

1.6. Висновки до Розділу 1

У рамках Розділу 1 "Дослідження систем управління лікарнею" було проведено аналіз існуючих систем управління лікарнею, визначено потреби лікарні та сформульовані вимоги до майбутнього веб-додатку для управління лікарнею.

Під час аналізу існуючих систем було виявлено, що ці системи, хоч і мають свою значущість, мають також ряд недоліків, включаючи негнучкість, потребу в постійних змінах та високі витрати на розробку та підтримку.

Аналіз потреб лікарні підкреслив важливість створення єдиної інформаційної системи, яка об'єднуватиме всі ключові процеси управління лікарнею. Ця система повинна бути гнучкою, адаптивною та доступною для всіх співробітників лікарні.

Сформульовані вимоги до веб-додатку включають в себе як функціональні, так і нефункціональні аспекти, такі як реєстрація та авторизація користувачів, управління пацієнтами, медичними працівниками та матеріальними ресурсами, а також вимоги до доступності, безпеки, швидкості та надійності системи.

Висновки цього розділу слугують фундаментом для подальшої розробки та впровадження веб-додатку для управління лікарнею, який відповідатиме потребам сучасного медичного сектору та сприятиме покращенню якості надання медичних послуг.

1.6.1. Ідентифікація недоліків та переваг існуючих систем для вирішення потреб лікарні

Ідентифікація недоліків та переваг існуючих систем дозволяє зрозуміти, які аспекти вже існуючих рішень відповідають потребам лікарні та які покращення можна внести. Ось кілька ключових пунктів для аналізу:

1. Функціональність: Оцініть, чи відповідають функціональні можливості існуючих систем потребам лікарні. Чи забезпечується повний функціонал для управління медичними даними, пацієнтами, назначеннями та іншими ключовими аспектами?
2. Інтеграція та сумісність: Важливо врахувати можливість інтеграції існуючих систем з іншими медичними платформами та стандартами. Чи забезпечується сумісність з іншими IT-системами, що використовуються лікарнею?
3. Безпека даних: Оцініть рівень захищеності існуючих систем. Чи забезпечується надійний рівень захисту медичних даних пацієнтів? Чи відповідає цей рівень сучасним стандартам безпеки?
4. Інтерфейс користувача: Перевірте зручність та ергономіку інтерфейсу користувача існуючих систем. Чи вони інтуїтивно зрозумілі для медичного персоналу?
5. Підтримка та обслуговування: Оцініть рівень підтримки та обслуговування від виробників існуючих систем. Чи швидко реагує виробник на помилки? Чи надається своєчасна допомога та оновлення?

6. Вартість та ефективність: Порівняйте вартість впровадження та підтримки існуючих систем з їхньою ефективністю. Чи відповідає вартість функціональності та можливостям системи?
7. Задоволеність користувачів: Зберіть відгуки користувачів існуючих систем. Чи задоволені вони функціональністю, швидкодією та зручністю використання?

Аналізуючи ці аспекти, можна ідентифікувати переваги та недоліки існуючих систем, що допоможе при розробці нового веб-додатку для лікарні.

1.6.2. Пошук ефективних рішень для конкретних потреб лікарні в контексті управління даними

У пошуку ефективних рішень для конкретних потреб лікарні в управлінні даними можна використовувати наступні стратегії:

1. Аналіз вимог: Ретельно проаналізувати всі вимоги лікарні до управління даними. Це можуть бути вимоги до зберігання медичних записів, планування назначень, аналізу результатів лікування тощо.
2. Огляд ринку рішень: Провести дослідження ринку та існуючих технологій для управління медичними даними. Оцініть системи управління базами даних, інструменти для зберігання та обробки медичної інформації.
3. Кастомізація: Розглянути можливість кастомізації існуючих систем або розробки власних рішень для відповіді конкретним потребам лікарні.
4. Забезпечення безпеки: Особливу увагу слід приділити питанням безпеки даних у медичній сфері. Застосовувати високі стандарти шифрування, забезпечення конфіденційності та захисту від несанкціонованого доступу.

5. Інтеграція та сумісність: Важливо врахувати можливість інтеграції нових рішень з вже існуючими системами у лікарні. Вони повинні легко взаємодіяти для оптимізації робочих процесів.
6. Аналіз вартості: Оцінити витрати на впровадження та підтримку різних рішень. Вартість розробки та підтримки має бути в межах бюджету лікарні, але при цьому не має впливати на якість та ефективність управління даними.
7. Тестування та евалюація: Провести тестування рішень на реальних даних та сценаріях використання. Оцініть їх ефективність та придатність для вирішення конкретних задач лікарні.
8. Консультації та відгуки: Звернутись до консультантів у медичній галузі, лікарів та інших фахівців, щоб отримати відгуки та рекомендації стосовно вибору ефективних рішень для управління даними у лікарні.

1.6.3. Переваги та виклики у впровадженні нових технологій у систему управління лікарнею

Переваги та виклики у впровадженні нових технологій у систему управління лікарнею можна розглядати з кількох аспектів:

Переваги:

1. Покращення ефективності: Нові технології можуть оптимізувати процеси управління, полегшити доступ до інформації, забезпечити автоматизацію деяких операцій, що призведе до підвищення продуктивності працівників.
2. Збільшення точності та якості: Сучасні системи можуть покращити точність даних, роблячи діагностику більш точною та надійною. Це сприятиме підвищенню якості медичного обслуговування.

3. Покращення доступності даних: Використання нових технологій може допомогти у створенні централізованої бази даних, що забезпечить легкий доступ до інформації для лікарів та пацієнтів.
4. Підвищення безпеки даних: Сучасні технології забезпечують високий рівень захисту медичної інформації від несанкціонованого доступу.

Виклики:

1. Інтеграція існуючих систем: Нові технології повинні бути взаємовідповідними та легко інтегруватися з вже існуючими системами у лікарні. Це може бути складним завданням через різноманітність систем та форматів даних.
2. Тривалий процес впровадження: Впровадження нових технологій може зайняти значний час і вимагати періоду навчання персоналу та адаптації до нових систем.
3. Витрати та бюджетні обмеження: Зміна технологій може потребувати великих інвестицій. Бюджетні обмеження можуть ускладнити впровадження сучасних рішень.
4. Питання безпеки та конфіденційності: Захист даних є критично важливим у медичній галузі, тому нові технології повинні відповідати високим стандартам безпеки.
5. Опір з боку персоналу: Часом персонал може бути опірним до змін, особливо, якщо нові технології вимагають навчання та пристосування до нових робочих процесів.

Управління цими викликами та максимізація переваг допомагають побудувати більш ефективну систему управління лікарнею через впровадження нових технологій.

1.6.4. Додаткові вимоги до функціоналу веб-додатку, які деталізують пріоритети та обов'язковість функцій

1. Пріоритетність функцій: Визначення важливості різних функцій в системі. Наприклад, функції, що пов'язані з обслуговуванням пацієнтів (запис на прийом, історія лікування) можуть мати вищий пріоритет, ніж аналітичні засоби збору даних.
2. Обов'язковість функцій: Розділення функцій на обов'язкові та додаткові. Обов'язкові функції - це ті, без яких система не може ефективно працювати. Наприклад, реєстрація пацієнта та запис на прийом можуть бути обов'язковими.
3. Функціональність для користувачів: Підкреслення функцій, які забезпечують зручне користування для пацієнтів, лікарів та адміністраторів. Наприклад, зручний інтерфейс для пацієнтів або система повідомлень для спілкування між лікарями.
4. Забезпечення даних: Функції, пов'язані з безпекою та захистом особистих медичних даних пацієнтів. Це може включати обмежений доступ до конфіденційної інформації, журналювання доступу, шифрування тощо.
5. Масштабованість: Функції, які забезпечують масштабованість системи, щоб вона могла зростати та адаптуватися до змін в навантаженні чи потребах користувачів.
6. Аналітика та звітність: Функціонал для збору даних, проведення аналізу та генерації звітів, які допоможуть в прийнятті рішень у медичному закладі.

Ці додаткові вимоги допомагають чітко визначити та розподілити функціонал системи залежно від їхньої важливості та необхідності для задоволення потреб лікарні.

1.6.5. Оцінка перспектив та можливостей масштабування системи в контексті росту лікарні

1. Горизонтальне та вертикальне масштабування: Оцінка можливостей збільшення кількості серверів (горизонтальне) або покращення та оновлення серверів (вертикальне), щоб обробляти більше даних та навантаження.
2. Система управління навантаженням: Розробка механізмів для контролю за навантаженням та розподілу його між різними частинами системи, щоб уникнути перевантаження.
3. Гнучкість архітектури: Побудова системи з урахуванням гнучкості та легкості розширення, щоб додавання нових функцій чи модулів не призводило до значних змін у всій системі.
4. Оптимізація бази даних: Розгляд можливостей оптимізації бази даних для роботи з великим обсягом даних та швидкого доступу до них при збільшенні їх обсягу.
5. Автоматизація процесів: Впровадження автоматизованих засобів моніторингу, масштабування та резервного копіювання, щоб забезпечити стабільну роботу системи при збільшенні обсягу даних.
6. Забезпечення безпеки масштабування: Врахування впливу масштабування на безпеку системи та впровадження заходів для збереження цілісності та конфіденційності даних при розширенні системи.
7. Прогнозування розвитку: Розгляд потенційних шляхів розвитку лікарні та оцінка, як система може підтримати майбутні потреби та інновації.

Ці аспекти сприяють побудові масштабованої системи, яка може ефективно рости та адаптуватися до змін у медичному закладі.

РОЗДІЛ 2. СЕРЕДОВИЩЕ РОЗРОБКИ ВЕБ-ДОДАТКУ

2.1. Огляд Java як основної мови програмування

Java є об'єктно-орієнтованою мовою програмування з великою кількістю переваг у сфері веб-розробки для медичних систем. Поговоримо про деякі ключові аспекти:

Характеристики Java: Розглянемо основні характеристики мови, такі як переносимість, безпека, висока продуктивність, гнучкість і широкі можливості розширення, які роблять Java привабливою для медичної сфери.

ООП у Java: Об'єктно-орієнтоване програмування (ООП) в Java є суттєвим аспектом у розробці медичних систем. Розглянемо концепції успадкування, поліморфізму, інкапсуляції та абстракції в контексті реалізації програмного забезпечення для лікарень.

Програмування за допомогою Java у медичних додатках: Подамо приклади використання Java для створення функціоналу, який відповідає вимогам управління лікарнею: від збору та обробки даних пацієнтів до взаємодії з медичним обладнанням.

Інструменти розробки в Java: Обговоримо популярні інтегровані середовища розробки (IDE), такі як IntelliJ IDEA, Eclipse або NetBeans, які сприяють комфортній та продуктивній розробці програмного забезпечення для медичних потреб.

Ком'юніті та підтримка: Оцінимо значення широкого співтовариства розробників Java та його вплив на розвиток та підтримку медичних систем.

2.1.1. Характеристики Java: основні переваги та особливості в контексті веб-додатків для управління лікарнею

Java, як мова програмування, має низку переваг у сфері розробки веб-додатків для управління лікарнею, серед яких:

Переносимість: Java є платформонезалежною мовою, тому розроблені програми можна запускати на будь-якому пристрої з Java Virtual Machine (JVM), що робить їх доступними для використання на різних пристроях у лікарні.

Безпека: Вбудовані механізми безпеки в Java, такі як управління пам'яттю та відсічення стеку, роблять її більш надійною у відношенні до захисту від вразливостей та атак, що є важливим у медичних системах, де конфіденційність даних є пріоритетом.

Висока продуктивність: Швидкість та ефективність виконання Java роблять її відмінним вибором для розробки веб-додатків у секторі охорони здоров'я, де швидкість роботи та обробки даних є критичними.

Гнучкість та розширюваність: Java має широкий набір бібліотек та фреймворків, які сприяють швидкій розробці та розширенню функціоналу веб-додатків для відповіді на потреби лікарень.

Масштабованість: Java дозволяє створювати веб-додатки, які можна легко масштабувати для обробки більшої кількості даних чи збільшення навантаження без втрати продуктивності.

Ці характеристики роблять Java потужним інструментом у розробці веб-додатків для управління лікарнею, забезпечуючи надійність, продуктивність та гнучкість, необхідні для успішної реалізації таких систем.

2.1.2. Принципи ООП у Java та їх вплив на розробку медичного програмного забезпечення

Об'єктно-орієнтоване програмування (ООП) у Java є ключовим аспектом, що впливає на розробку медичних систем. Ось деякі принципи ООП та їх вплив:

Успадкування (Inheritance): Успадкування дозволяє створювати нові класи на основі вже існуючих, що сприяє утворенню ієрархії класів у медичній системі. Наприклад, відображення різних типів медичного персоналу (лікарі, медсестри, адміністратори) через успадкування від загального класу "Працівник".

Поліморфізм (Polymorphism): Поліморфізм у Java дозволяє однаково використовувати методи для об'єктів різних класів. Це може бути корисним у медичних системах для обробки різних типів даних (пацієнтів, медичних записів) за допомогою універсальних методів.

Інкапсуляція (Encapsulation): Цей принцип дозволяє обмежити доступ до певних частин коду та даних, що є важливим у медичних системах для забезпечення конфіденційності та безпеки медичної інформації.

Абстракція (Abstraction): Абстракція дозволяє приховати деталі реалізації, дозволяючи працювати з об'єктами на більш вищому рівні. У медичних системах це може бути використано для створення спрощеного інтерфейсу для користувача, який не потребує зайвих технічних деталей.

Спадкування інтерфейсів (Interface Inheritance): Використання інтерфейсів дозволяє створювати стандарти для класів, що реалізують ці інтерфейси. У медичних системах це може бути корисним для забезпечення відповідності до стандартів обміну медичною інформацією.

Ці принципи ООП в Java сприяють розробці більш структурованих, розширюваних та підтримуваних медичних систем, що відповідають потребам

лікарень у забезпеченні ефективного та безпечного управління медичними даними та процесами.

2.2. Java Spring як вибраний фреймворк

Java Spring - це потужний фреймворк для розробки веб-додатків, який надає велику кількість інструментів та можливостей. Ось кілька аспектів, які варто врахувати:

Ключові особливості Spring: Обговоримо основні характеристики Spring, такі як інверсія управління (Inversion of Control - IoC), контейнери інверсії управління (IoC containers), аспекти (Aspects) та універсальна підтримка управління транзакціями.

Модульність та розширюваність: Пояснимо, як Spring дозволяє побудувати додаток з набору взаємозалежних модулів, що спрощує розробку, тестування та розширення системи управління лікарнею.

Spring MVC та веб-розробка: Опишемо Spring MVC (Model-View-Controller), який дозволяє створювати веб-додатки на основі моделі MVC, що є чудовим підходом для розробки веб-інтерфейсу для систем управління лікарнею.

Spring Security та безпека: Підкреслимо важливість Spring Security у забезпеченні безпеки медичних даних у веб-додатках, обговоримо інструменти, які надає Spring для аутентифікації, авторизації та захисту даних.

Підтримка транзакцій: Обговоримо вбудовану підтримку транзакцій у Spring, яка є важливою у медичних системах для забезпечення консистентності даних під час їхнього зміщення та модифікацій.

Ці аспекти Spring роблять його потужним інструментом для розробки систем управління лікарнею, забезпечуючи безпеку, модульність та розширюваність, що є важливими аспектами у медичних програмних додатках.

2.2.1. Огляд функцій та можливостей Java Spring у розробці веб-додатків для медичного сектору

Java Spring надає широкий спектр інструментів та функцій, корисних для розробки веб-додатків у медичній сфері. Деякі з ключових можливостей включають:

Spring Boot: Це один з ключових компонентів Spring, який дозволяє швидко створювати самостійні веб-додатки. У медичному секторі це може бути важливо для швидкої розгортки та розвитку систем управління лікарнею.

Spring Data: Це модуль, який спрощує взаємодію з базами даних у Java, дозволяючи легко використовувати реляційні бази даних для зберігання медичних записів та даних пацієнтів.

Spring Security: Цей модуль забезпечує високий рівень безпеки веб-додатків, що є критично важливим у медичній сфері з огляду на конфіденційність медичної інформації.

Spring MVC: Це реалізація патерну проектування Model-View-Controller, яка дозволяє ефективно розділяти логіку, інтерфейс користувача та дані в медичних веб-додатках.

Spring AOP (Aspect-Oriented Programming): Цей підхід дозволяє вставляти код (аспекти) в певні точки програми, що може бути корисним у медичних системах для логування, аудиту та управління транзакціями.

Інтеграція з іншими технологіями: Spring дозволяє легко інтегрувати інші технології, такі як RESTful веб-служби, зовнішні API медичних систем, що робить його гнучким для співпраці з різними інструментами.

Ці функції та можливості Java Spring роблять його потужним фреймворком для розробки веб-додатків для медичного сектору, забезпечуючи безпеку, ефективність та гнучкість у розробці програмного забезпечення для управління лікарнею.

2.2.2. Роль Spring у забезпеченні безпеки, масштабованості та ефективності медичних систем управління

Spring відіграє ключову роль у забезпеченні безпеки, масштабованості та ефективності медичних систем управління через такі аспекти:

Безпека засобами Spring Security: Spring Security надає низку інструментів для аутентифікації, авторизації, управління сесіями та захисту від потенційних атак. У медичних системах це дозволяє забезпечити захист медичних даних та знизити ризики порушення конфіденційності.

Масштабованість та управління засобами Spring: Spring Framework допомагає забезпечити масштабованість за допомогою своїх інструментів та підходів до управління бізнес-логікою, дозволяючи системам управління лікарнею легко розширюватися та працювати з великим обсягом даних та користувачів.

Ефективність розробки через модульність Spring: Модульність Spring дозволяє розробникам створювати підсистеми та компоненти, що полегшує розробку та тестування. Це може бути корисним у медичних системах для швидкого впровадження нових функцій та реагування на зміни вимог.

Підтримка транзакцій та консистентності даних: Spring дозволяє легко керувати транзакціями бази даних, що є важливим у медичних системах для забезпечення консистентності даних під час їхньої модифікації.

Ці аспекти роблять Spring надійним інструментом у розробці медичних систем управління, забезпечуючи безпеку, масштабованість та ефективність в управлінні лікарнею.

2.3. Вибір бази даних: PostgreSQL

PostgreSQL обраний для використання як база даних у вашому веб-додатку для управління лікарнею з наступних причин:

Надійність та стабільність: PostgreSQL відомий своєю надійністю та стабільністю. У медичних системах це критично важливо для зберігання медичних даних без ризику втрати інформації або несправностей бази даних.

Підтримка ACID: PostgreSQL гарантує дотримання властивостей ACID (Atomicity, Consistency, Isolation, Durability), що забезпечує консистентність та надійність операцій з даними, особливо важливо у медичних записах та інших критичних відносно безпеки даних областях.

Гнучкість та розширюваність: PostgreSQL має широкий функціонал та можливості для розширення, що дозволяє легко адаптувати базу даних до змінних потреб управління лікарнею.

Підтримка географічних даних: У випадку медичних систем може бути важливою підтримка географічних даних, таких як місцезнаходження лікарень, пацієнтів тощо, що PostgreSQL забезпечує.

Відкритий джерело і спільнота користувачів: PostgreSQL - це система з відкритим кодом, що означає, що вона безкоштовна та має широку спільноту розробників та користувачів, які надають підтримку та розвиток системи.

2.3.1. Переваги використання реляційних баз даних у медичних застосунках

Переваги використання реляційних баз даних у медичних застосунках доволі значущі через специфіку і вимоги до зберігання, доступу та обробки даних в цій області:

Структурованість даних: Реляційні бази даних (RDBMS) дозволяють створювати структуровані схеми для зберігання даних, такі як медичні записи, пацієнтська інформація, результати досліджень тощо. Це сприяє організації даних і полегшує їхнє управління.

Забезпечення цілісності даних: Реляційні бази даних мають вбудовану підтримку для забезпечення цілісності даних через унікальність, зовнішні ключі та обмеження цілісності, що особливо важливо у медичних системах для уникнення помилок та дублювання інформації.

Можливості стандартизації: Використання реляційних баз даних у медичних системах дозволяє застосовувати стандарти для зберігання та обміну медичної інформації, що сприяє сумісності та інтероперабельності систем.

Підтримка складних запитів та аналітики: RDBMS надають можливості для проведення складних запитів, включаючи агрегацію даних та аналітику. У медичних системах це допомагає виявляти тенденції, проводити дослідження та оптимізувати процеси лікування.

Доступність інструментів для резервного копіювання та відновлення: Багато RDBMS мають інструменти для автоматичного резервного копіювання та відновлення даних, що дозволяє забезпечити надійність і збереження інформації у разі аварій.

Відомість та підтримка: Реляційні бази даних, такі як PostgreSQL, мають широку спільноту користувачів та розробників, що сприяє розумінню та підтримці систем з використанням цих баз даних.

Ці переваги роблять реляційні бази даних привабливими для медичних застосувань через їхню надійність, структурованість, можливості аналізу даних та відповідність стандартам у сфері охорони здоров'я.

2.3.2. Репозиторії та Spring Data

Репозиторії та Spring Data відіграють ключову роль у взаємодії додатків з базою даних в середовищі Spring. Детальніше про ці поняття:

Репозиторії

У Spring, репозиторії є частиною підходу до роботи з базами даних. Вони представляють собою абстракцію для взаємодії з даними, яка дозволяє звертатися до бази даних через визначені методи. Класи-репозиторії надають інтерфейси або абстрактні класи для доступу до даних і використовуються для виконання різних операцій з даними, таких як пошук, збереження, оновлення та видалення.

Spring Data

Spring Data - це підпроєкт Spring, який надає простий та консистентний спосіб доступу до різних джерел даних в Java-додатках. Він спрощує роботу з базами даних, надаючи уніфіковані інтерфейси для різних типів даних та різних джерел даних (SQL, NoSQL тощо).

Spring Data JPA

Spring Data JPA - це частина Spring Data, яка надає абстракцію для взаємодії з реляційними базами даних через JPA (Java Persistence API). JPA є специфікацією Java для роботи з реляційними базами даних, а Spring Data JPA додає до цього великий набір зручних інструментів, які спрощують взаємодію з даними.

Основні переваги

1. Спрощення доступу до даних: За допомогою анотацій та інтерфейсів Spring Data забезпечує легкий доступ до бази даних без написання багатої підтримки для з'єднань та операцій з даними.
2. Менш коду, більше функціональності: Spring Data дозволяє використовувати менше коду для виконання типових завдань, таких як CRUD операції, роблячи розробку ефективнішою та швидшою.
3. Підтримка різних джерел даних: Spring Data може працювати з різними джерелами даних, включаючи реляційні бази даних (наприклад, MySQL, PostgreSQL) та NoSQL бази даних (наприклад, MongoDB).

Використання репозиторіїв та Spring Data у сполученні зі Spring дозволяє створювати масштабовані, ефективні та легкі для розуміння додатки з великим обсягом роботи з даними.

2.3.3. Spring Data JPA та Hibernate

Spring Data JPA та Hibernate є потужними інструментами для роботи з базами даних у Java-додатках. Дозвольте розповісти про кожен з них більш детально:

Spring Data JPA

Spring Data JPA є частиною проекту Spring Data, яка спрощує роботу з реляційними базами даних через JPA (Java Persistence API). Основні можливості:

1. Автоматична реалізація методів: За допомогою Spring Data JPA можна автоматично генерувати реалізацію методів репозиторіїв на основі іменування, що робить код більш чистим та компактним.
2. Уніфікований інтерфейс: Spring Data JPA надає уніфікований інтерфейс для роботи з різними реляційними базами даних (наприклад, MySQL, PostgreSQL).
3. Підтримка JPA анотацій: Вона дозволяє використовувати JPA-анотації для мапінгу об'єктів на таблиці бази даних.

Hibernate

Hibernate - це фреймворк для об'єктно-реляційного відображення (ORM), який реалізує JPA. Основні можливості:

1. ORM функціональність: Hibernate надає зручний спосіб роботи з об'єктами у програмі та таблицями у базі даних, забезпечуючи взаємодію через об'єктно-орієнтовані методи.

2. Кешування: Hibernate підтримує кешування, що може значно покращити продуктивність додатку, зберігаючи копії часто використовуваних даних у пам'яті.
3. Мова запитів Hibernate: Hibernate має свою власну мову запитів HQL (Hibernate Query Language), яка дозволяє виконувати запити до бази даних, використовуючи об'єктну модель замість SQL.

Спільне використання

Spring Data JPA часто використовується разом з Hibernate, оскільки вони працюють разом у складі стеку технологій для роботи з базами даних в Spring. Hibernate виступає як реалізація JPA, яку використовує Spring Data JPA для взаємодії з базою даних. Це дозволяє розробникам забезпечити зручний та ефективний доступ до даних у своїх додатках.

2.3.4. ORM (Object-Relational Mapping) в Java:

ORM (Object-Relational Mapping) - це технологія, яка дозволяє зв'язати об'єктно-орієнтовану модель програмування з реляційними базами даних. У контексті Java це використовується для спрощення роботи з базами даних через об'єктно-орієнтований підхід, коли програми використовують об'єкти для роботи з даними замість написання SQL-запитів безпосередньо.

Основні концепції ORM включають в себе:

1. Об'єктно-орієнтоване відображення: ORM надає зручний спосіб відображення даних з бази даних в об'єкти Java та навпаки. Класи Java відображаються на таблиці бази даних, а поля класів - на колонки в цих таблицях.

2. Управління відносинами: ORM дозволяє визначати відносини між об'єктами і здійснювати їх зв'язок через відносини в базі даних (наприклад, один до багатьох, багато до багатьох і т.д.).
3. Автоматичне створення SQL-запитів: ORM фреймворки, такі як Hibernate або Spring Data JPA, генерують SQL-запити автоматично на основі операцій, які ви виконуєте над об'єктами.
4. Кешування та оптимізація: ORM може автоматично кешувати дані та оптимізувати виконання запитів, що поліпшує продуктивність.
5. Підтримка різних СУБД: ORM дозволяє працювати з різними системами управління базами даних, такими як PostgreSQL, MySQL, Oracle і т.д., не змінюючи основного коду застосунку.

Використання ORM у проекті полегшує розробку, зменшує кількість написаного коду, покращує читабельність, зберігає час на написання SQL-запитів та спрощує роботу з базою даних в контексті веб-додатків для управління лікарнею.

2.4. Рішення на користь SQL перед NoSQL

Вибір SQL бази даних, такої як PostgreSQL, перед NoSQL (наприклад, MongoDB, Cassandra тощо) може бути обґрунтований для систем управління лікарнею з наступних причин:

Структурованість та консистентність даних: У медичних системах нерідко важливо зберігати дані в структурованому форматі, що сприяє консистентності та можливості ефективного проведення аналізу. SQL бази даних добре підходять для цього завдання, оскільки вони пропонують строгу схему даних та забезпечують її дотримання.

Складні запити та аналітика: У медичних системах часто потрібно проводити складні запити для аналізу та витягування певних даних. SQL бази

даних зазвичай мають потужні можливості для складних JOIN-операцій, агрегаційних функцій та іншої аналітики, що дозволяє легше та ефективніше отримувати потрібну інформацію.

Специфікації та стандарти в медичній сфері: У галузі охорони здоров'я часто існують строгі стандарти щодо обробки та зберігання даних. SQL бази даних добре відповідають цим стандартам та частіше використовуються для забезпечення відповідності нормативам.

Системи контролю доступу та безпеки: SQL бази даних мають вбудовані системи безпеки, що дозволяють точно керувати доступом до різних частин даних. Це важливо для медичних систем, де конфіденційність та цілісність даних є високопріоритетними.

Транзакційна підтримка: SQL бази даних, такі як PostgreSQL, мають вбудовану підтримку транзакцій, що дозволяє забезпечити консистентність даних під час операцій з ними, що є критичним у медичних системах.

2.4.1. Порівняння між SQL та NoSQL для веб-додатків у сфері охорони здоров'я

Порівняння між SQL та NoSQL базами даних для веб-додатків у сфері охорони здоров'я може бути корисним для визначення найбільш відповідного рішення. Розглянемо деякі аспекти порівняння:

1. Структурованість даних:

SQL (реляційні бази даних): Мають жорстко визначену схему даних, що робить їх ідеальними для сценаріїв, де потрібна точна структура даних, як у медичних системах зі стандартизованими форматами.

NoSQL: Більш гнучкі у структурі даних, дозволяють зберігати неструктуровані або поліморфні дані, що може бути корисним у випадках, коли формат даних може змінюватися.

2. Складність запитів та аналітика:

SQL: Мають потужні можливості для складних JOIN-операцій та аналітики, що сприяє проведенню детальних аналізів даних, що може бути важливим у медичних системах.

NoSQL: Зазвичай менш потужні у цьому плані, але здатні до роботи з великими обсягами даних, такими як дані сенсорів або логи.

3. Гнучкість та масштабованість:

SQL: Добре підходять для проектів з фіксованою структурою, але можуть бути менш гнучкими при потребі у розширенні.

NoSQL: Більш гнучкі у масштабуванні та здатні працювати з великими обсягами даних, що може бути корисним для медичних систем з великою кількістю пацієнтів та медичних записів.

4. Безпека та цілісність даних:

SQL: Мають довідкову систему безпеки та підтримують транзакції для забезпечення цілісності даних, що може бути критично важливим у медичних системах.

NoSQL: Менш жорстка система контролю доступу та не завжди підтримують транзакції, але можуть бути ефективними у великих розподілених системах.

5. Відповідність стандартам та регулюванням:

SQL: Частіше відповідають стандартам та вимогам регулювань у галузі охорони здоров'я, таких як HIPAA (Health Insurance Portability and Accountability Act).

NoSQL: Можуть вимагати додаткової роботи для відповідності регулятивним вимогам.

Обираючи між SQL та NoSQL, важливо враховувати потреби конкретної медичної системи, вимоги до структури даних, потужності аналітики та потреби у масштабованості та безпеці.

2.4.2. Фактори вибору SQL та його відповідність функціональним та безпековим вимогам лікарні

Контроль доступу та безпека: SQL бази даних мають розвинуті системи контролю доступу, що дозволяє регулювати, хто має доступ до якої інформації. Це важливо для забезпечення конфіденційності пацієнтських даних та відповідності регулятивним стандартам.

Система транзакцій: SQL бази даних підтримують транзакційні операції, що є критичним для забезпечення цілісності даних під час операцій з ними, таких як внесення змін до медичних записів.

Резервне копіювання та відновлення: SQL бази даних надають зручні інструменти для резервного копіювання та відновлення даних, що є важливим для запобігання втрати інформації та забезпечення неперервності роботи систем.

Адаптабельність до змін та розвиток системи: SQL бази даних забезпечують можливості для змін у схемі даних та розвитку системи, що дозволяє легко адаптувати їх до змінних потреб лікарень.

Відповідність стандартам безпеки та нормативам у галузі охорони здоров'я: Важливо, що SQL бази даних частіше відповідають стандартам безпеки, таким як HIPAA, що робить їх привабливим вибором для лікарень.

2.4.3. Транзакції в базах даних

Транзакції є ключовими управлінськими блоками в базах даних, що гарантують цілісність та консистентність даних під час виконання операцій.

Вони використовуються для забезпечення атомарності, консистентності, ізоляції та довіри (ACID) даних у базі.

1. Атомарність (Atomicity): Ця властивість гарантує, що операція або група операцій виконується повністю або не виконується зовсім. Це означає, що якщо одна частина транзакції не вдається, то всі інші зміни, внесені під час цієї транзакції, будуть скасовані.
2. Консистентність (Consistency): Транзакція переводить базу даних з одного консистентного стану в інший, що також забезпечує відповідність всіх обмежень цілісності даних.
3. Ізоляція (Isolation): Ця властивість гарантує, що одна транзакція не впливає на інші транзакції, які відбуваються одночасно. Кожна транзакція виконується самостійно і ізольовано від інших.
4. Довіра (Durability): Це означає, що зміни, внесені транзакцією, залишаються стійкими та перманентними після її успішного завершення, навіть у випадку збою системи.

В контексті лікарні, використання транзакцій у базі даних є критично важливим для забезпечення надійності, цілісності та безпеки даних. Під час збереження медичних записів, інформації про пацієнтів та операцій, транзакції допомагають уникнути помилок і забезпечити надійне збереження даних, а також управління взаємозалежними операціями у системі.

2.5. Використання Maven у проекті

Maven - це інструмент управління проектами, який дозволяє ефективно керувати залежностями, збіркою, тестуванням та розгортанням програмного забезпечення. У контексті розробки веб-додатків для лікарень, Maven може мати наступні аспекти:

Роль та переваги Maven у процесі розробки: Огляд того, як Maven спрощує управління залежностями проекту, забезпечує однорідність серед проектів, полегшує збірку та розгортання програми.

Організація структури проекту через Maven: Як Maven допомагає організувати проект, включаючи структуру каталогів, конфігураційні файли, та правильне розміщення вихідного коду.

Управління залежностями: Огляд того, як Maven дозволяє ефективно використовувати сторонні бібліотеки та компоненти в проекті через управління залежностями.

Збірка, тестування та розгортання програми за допомогою Maven: Як Maven автоматизує процес збірки, тестування та розгортання програмного забезпечення, спрощуючи ці кроки для розробників.

Переваги використання Maven для веб-додатків у сфері охорони здоров'я: Особливості Maven, які зроблять розробку та підтримку веб-додатків для лікарень більш ефективною та легкою.

Керування конфігураціями проекту: Як Maven дозволяє зберігати конфігураційні параметри у файлах та керувати ними централізовано, що спрощує управління налаштуваннями проекту.

Плагіни та розширення Maven: Огляд доступних плагінів та можливостей розширення Maven, що можуть бути корисними для розробки медичних веб-додатків, наприклад, плагіни для збірки звітів, автоматизації тестування тощо.

Управління версіями та релізами: Як Maven сприяє управлінню версіями програми, включаючи процес створення релізів та управління версіями залежностей.

Інтеграція з іншими інструментами розробки: Розгляд можливостей інтеграції Maven з іншими інструментами розробки, такими як IDE (наприклад, Eclipse, IntelliJ IDEA) чи CI/CD системами для автоматизованого впровадження.

Засоби управління життєвим циклом проекту: Як Maven допомагає управляти різними етапами життєвого циклу проекту, від розробки до тестування та розгортання.

Спільнота та підтримка: Розгляд ресурсів та підтримки, доступних у спільноті Maven, включаючи документацію, форуми та інші ресурси для підтримки розробників.

Налаштування та кастомізація Maven: Можливості налаштування Maven під конкретні потреби проекту та методи кастомізації під вимоги розробки медичних веб-додатків.

2.5.1. Роль та переваги Maven у процесі розробки веб-додатків для лікарень

Управління залежностями: Maven дозволяє визначати та керувати залежностями проекту. У медичній сфері, де потрібна велика кількість бібліотек та різноманітних компонентів, це спрощує використання та оновлення необхідних пакетів.

Спрощена збірка проекту: Maven автоматизує процес збірки проекту, що полегшує та прискорює розробку, відокремлюючи процес компіляції, тестування та пакування програми.

Стандартизація проектів: Maven використовує конвенції та стандарти, що сприяє структурованості проектів. Це особливо важливо у медичних системах, де чітка структура та правильне організоване розташування файлів мають значення.

Легкість управління багаторівневими проектами: Maven дозволяє працювати з багатьма модулями або підпроектами в одному батьківському проекті, що дозволяє ефективніше керувати складністю системи.

Інтеграція з іншими інструментами: Maven легко інтегрується з іншими інструментами розробки, такими як системи контролю версій, CI/CD системи та середовища розробки, спрощуючи роботу команди.

Підтримка та спільнота: Maven має широку підтримку та активну спільноту, що означає наявність багатьох ресурсів, документації та можливість швидко отримати відповіді на питання.

Зручний розгортання та впровадження: Maven спрощує процес розгортання та впровадження веб-додатків, забезпечуючи структурованість та готовність до впровадження програмного забезпечення.

Ці переваги дозволяють Maven бути потужним інструментом у процесі розробки веб-додатків для лікарень, забезпечуючи стабільність, продуктивність та стандартизацію розробки.

2.5.2. Організація процесу збірки, тестування та розгортання за допомогою Maven

Конфігурація збірки проекту: Опис конфігураційних файлів (pom.xml) для збірки проекту, включаючи опис залежностей, плагінів та налаштувань.

Збірка проекту: Опис етапів збірки проекту через Maven, включаючи компіляцію вихідного коду, створення пакету або war-файлу.

Автоматизація тестування: Інтеграція тестів у процес збірки для автоматизації тестування під час розробки.

Контроль якості коду: Інтеграція з інструментами контролю якості коду (наприклад, Checkstyle, FindBugs) для автоматичної перевірки якості коду під час збірки.

Створення артефактів: Підготовка готового до розгортання артефакту або пакету, готового для впровадження.

Розгортання та впровадження: Опис процесу розгортання або впровадження створених артефактів у тестові або продуктивні середовища.

Інтеграція з CI/CD системами: Використання Maven у поєднанні з CI/CD системами для автоматизації процесу збірки, тестування та розгортання.

Управління версіями та релізами: Використання Maven для керування версіями та створення релізів програмного забезпечення.

Документація та звітність: Можливості Maven забезпечують створення документації та звітів про процес збірки та тестування.

Ці аспекти дозволяють використовувати Maven як центральний інструмент для автоматизації всього процесу розробки, від збірки до впровадження програмного забезпечення.

2.5.3. Continuous Integration та Continuous Deployment (CI/CD)

Continuous Integration (CI) та Continuous Deployment (CD) є підходами до розробки програмного забезпечення, спрямованими на автоматизацію тестування, збірки та розгортання програмного коду.

1. Continuous Integration (CI): Це практика, коли розробники регулярно об'єднують свій код в спільний репозиторій. Це включає в себе автоматичну перевірку коду на помилки за допомогою тестів, запуск автоматизованих тестів при злитті коду із гілками, а також побудову проекту з метою виявлення можливих конфліктів.

2. Continuous Deployment (CD): Ця практика передбачає автоматичне розгортання коду в продуктивне середовище після проходження всіх тестів та проходження CI. Це означає, що будь-який успішний злиття коду в основну гілку (наприклад, master) спричинить автоматичне випуск нової версії програми чи оновлення.

У контексті управління лікарнею, CI/CD може забезпечити більш швидке впровадження оновлень та розробку нового функціоналу у веб-додатку. Це дозволяє автоматизувати процеси тестування та випуску, що полегшує розгортання нових функцій та зменшує час між розробкою та впровадженням змін у живе середовище. Такий підхід дозволяє забезпечити стабільність та надійність системи управління лікарнею.

2.6. Висновки до Розділу 2

Розділ, присвячений середовищу розробки для веб-додатків у сфері управління лікарнею, дав можливість глибше зануритися у вибір технологічних засобів та інструментів для побудови програмного забезпечення. Основні висновки, які можна зробити:

Вибір технологій під сферу медицини: Оглянувши Java як мову програмування та Java Spring як фреймворк, виявлено їх адаптабельність та потенціал для розробки медичних систем.

Реляційна база даних для надійності та стабільності: Вибір PostgreSQL засвідчує важливість реляційних баз даних у медичній галузі, забезпечуючи надійність та стабільність даних.

SQL у контексті потреб медичних застосувань: Рішення на користь SQL порівняно з NoSQL вибрано з урахуванням функціональних та безпекових вимог медичних установ.

Зручність Maven у процесі розробки: Maven виявився потужним інструментом для автоматизації збірки, тестування та розгортання веб-додатків, полегшуючи процес розробки.

Загальний вплив середовища розробки на продукт: Вибір технологій та інструментів має ключове значення для створення надійного та ефективного веб-додатку для управління лікарнею.

Цей розділ дозволив зрозуміти, що вибір правильних технологій та інструментів є критичним у створенні веб-додатку для медичного сектору, оскільки від цього залежить якість, безпека та надійність програмного забезпечення.

2.6.1. Узагальнення ключових аспектів середовища розробки та їх вплив на створення веб-додатку для управління лікарнею

У даному розділі розглянуто різні аспекти середовища розробки, які визначають якість та ефективність створення веб-додатків для управління лікарнею. Важливість кожного аспекту виявляється через їх вплив на розробку медичного програмного забезпечення:

Вибір мови та фреймворку програмування: Вплив Java та Java Spring на зручність розробки, швидкість створення та гнучкість в роботі з медичними даними.

Роль реляційних баз даних: Використання PostgreSQL забезпечує надійність, стабільність та консистентність медичної інформації.

Вибір SQL перед NoSQL: Розгляд вибору SQL як оптимального рішення у контексті функціональних та безпекових вимог системи управління лікарнею.

Зручність та автоматизація засобів розробки за допомогою Maven: Виявлено важливість автоматизації процесів збірки, тестування та розгортання веб-додатків для оптимізації робочого процесу.

Узагальнюючи, вибір правильних інструментів та технологій для розробки медичних веб-додатків є критичним для створення надійної, ефективної та безпечної системи управління лікарнею. Ці аспекти визначають якість, функціональність та гнучкість програмного забезпечення, що використовується у сфері медицини.

2.6.2. Забезпечення якості коду в середовищі Java

Забезпечення якості коду в середовищі Java - це важливий аспект розробки програмного забезпечення. Відповідальність за якість коду передбачає ряд практик та інструментів, спрямованих на підтримку чистоти, читабельності, ефективності та надійності кодової бази. Ось деякі з них:

1. Стандарти коду: Використання інструментів та визначення правил для однорідного форматування, коментування та іменування коду.
2. Code Review (перегляд коду): Регулярна перевірка коду членами команди для виявлення потенційних помилок, забезпечення відповідності стандартам коду та виявлення можливостей для оптимізації.
3. Automated Testing (автоматизоване тестування): Створення автоматизованих тестів для перевірки функціональності та виявлення помилок в коді.
4. Static Code Analysis (аналіз статичного коду): Використання інструментів, які аналізують код без його виконання, для виявлення потенційних проблем або невідповідностей стандартам програмування.
5. Code Metrics (метрики коду): Оцінка якості коду на основі різних показників, таких як кількість рядків, складність коду, покриття тестами та інші.
6. Continual Learning (постійне навчання): Слідкування за новими практиками, стандартами та інструментами для покращення якості коду та вдосконалення навичок команди.

Застосування цих практик допомагає створювати більш надійне, ефективне та підтримуване програмне забезпечення для управління лікарнею.

2.6.3. Інструменти профілювання та оптимізації коду в Java

Інструменти профілювання та оптимізації коду в Java дозволяють виявляти та виправляти проблеми з продуктивністю програм, знаходити місця зайвого споживання ресурсів та поліпшувати ефективність програмного забезпечення. Ось деякі з найпопулярніших інструментів:

1. VisualVM: Це інструмент для збору даних про виконання програми Java, включаючи інформацію про пам'ять, потоки, профілі роботи та моніторинг витрат ресурсів.
2. JProfiler: Надійний профілер Java, який надає детальну інформацію про виконання програм та дозволяє виявляти бутланки, витрати ресурсів та оптимізувати їх.
3. YourKit: Інструмент профілювання, який забезпечує аналіз процесів Java, знаходження утечок пам'яті та виявлення вузьких місць у виконанні.
4. Java Mission Control: Інструмент зі збору даних про виконання Java, який надає детальну інформацію про виконання програм та візуалізацію проблем продуктивності.
5. AsyncProfiler: Це легкий у використанні профілер з низьким навантаженням на систему, призначений для аналізу та оптимізації коду в реальному часі.

Ці інструменти дозволяють розробникам виявляти проблеми з продуктивністю, проводити аналіз коду та шукати способи його оптимізації, що сприяє покращенню продуктивності та ефективності програмного забезпечення.

2.6.4. Безпека даних в медичних системах

Забезпечення безпеки даних в медичних системах є важливою складовою через конфіденційність та цінність медичної інформації. Ось деякі ключові аспекти безпеки даних в медичних системах:

1. Конфіденційність інформації: Медична інформація повинна бути захищена від несанкціонованого доступу. Використання механізмів аутентифікації, авторизації та шифрування даних допомагає забезпечити конфіденційність.
2. Захист від кібератак: Медичні системи повинні мати заходи безпеки для запобігання кібератак. Це включає захист мережі, виявлення вторгнень, резервне копіювання даних і захист від вірусів та шкідливих програм.
3. Відповідність з правовими нормами: Збір, зберігання та обробка медичної інформації повинні відповідати правовим нормам та регулятивним вимогам, таким як HIPAA в США, GDPR в Європейському Союзі.
4. Аудит доступу до даних: Системи мають здатність вести журнали доступу, що дозволяє виявляти та відслідковувати дії користувачів з медичною інформацією.
5. Безпека на рівні додатків: Розробники повинні використовувати найкращі практики безпеки при програмуванні, такі як захист від SQL-ін'єкцій, XSS атак та інших потенційних вразливостей додатків.
6. Бекапи і відновлення даних: Регулярні бекапи даних та процедури відновлення допомагають у випадках аварій або втрати даних.
7. Навчання персоналу: Навчання персоналу щодо кращих практик безпеки та своєчасне оновлення є важливими для забезпечення безпеки даних в медичних системах.

Ці аспекти дозволяють створити надійну та захищену систему обробки медичної інформації, яка дотримується найвищих стандартів безпеки даних.

2.6.5. Масштабованість та витривалість додатку

Масштабованість та витривалість додатку є ключовими характеристиками, особливо для систем управління лікарнями, які повинні працювати ефективно навіть при збільшенні обсягів даних та навантаження.

1. Горизонтальне та вертикальне масштабування: Масштабування може бути реалізоване через збільшення кількості серверів (горизонтальне) або підвищення продуктивності окремих серверів (вертикальне).
2. Використання кешування: Використання кешування дозволяє зберігати частину даних у швидкодіючих системах, що прискорює доступ до них і зменшує навантаження на базу даних.
3. Автоматичне масштабування: Використання хмарних сервісів або інших інструментів, які надають автоматичне масштабування ресурсів, дозволяє додатку реагувати на змінні навантаження.
4. Резервне копіювання та відновлення: Механізми резервного копіювання та відновлення допомагають у випадку втрати даних або системних збоїв, забезпечуючи витривалість системи.
5. Тестування на витривалість: Розробка відновлюваних систем, які можуть працювати в умовах великого навантаження або випадків збоїв.
6. Моніторинг та аналіз продуктивності: Постійний моніторинг роботи системи дозволяє виявляти проблеми та вчасно реагувати на них, підтримуючи високу продуктивність.

Ці аспекти сприяють створенню масштабованих та витривалих медичних систем, які можуть ефективно функціонувати навіть у складних умовах великого обсягу даних та навантаження.

2.6.6. Оптимізація запитів до баз даних

Оптимізація запитів до баз даних є важливою складовою розробки систем, особливо у сфері управління лікарнями, де доступ до даних має велике значення для роботи з пацієнтами та їх історіями.

1. Індексація: Створення індексів на полях, за якими часто відбувається пошук або сортування, може значно покращити швидкість виконання запитів.
2. Використання оптимальних запитів: Важливо писати запити до бази даних так, щоб вони були якнайбільш оптимізованими, враховуючи обсяг даних та їхню структуру.
3. Лімітування результатів: При необхідності витягування великої кількості записів важливо використовувати механізми лімітування результатів запитів.
4. Оптимізація JOIN-операцій: Ретельне планування та оптимізація JOIN-операцій може покращити продуктивність запитів.
5. Кешування результатів: Використання кешування може зменшити час виконання повторних запитів до бази даних.
6. Моніторинг та аналіз запитів: Постійний моніторинг та аналіз запитів допомагає виявляти та виправляти ефективність запитів до бази даних.

Оптимізація запитів до баз даних дозволяє підтримувати високу продуктивність системи та забезпечує швидкий доступ до важливої інформації для лікарень та медичних установ.

РОЗДІЛ 3. РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ ЛІКАРНІ

Під час створення веб-додатку для лікарні у межах моєї дипломної роботи, я обрав монолітну архітектуру. Вибір такої архітектури зумовлений декількома ключовими факторами, які відповідають потребам саме цього проекту:

1. Простота розробки: Один великий застосунок, що включає в себе всі компоненти, може бути простіше розробити та розгорнути порівняно з багатьма мікросервісами.
2. Простота тестування та відлагодження: В монолітній архітектурі легше виконувати тестування та відлагодження, оскільки весь код знаходиться в одній програмі.
3. Легше впровадження змін: Зміни у функціоналі можуть бути впроваджені швидше через меншу кількість залежностей та взаємодій між компонентами.
4. Простота моніторингу та управління: З монолітною архітектурою легше вести моніторинг, впроваджувати управління та контролювати весь застосунок як цілісну систему.
5. Вартість розробки та інфраструктури: Монолітна архітектура може бути менш витратною з точки зору інфраструктури та управління.

3.1. Вибір інструментів розробки

3.1.1. Середовище розробки: IntelliJ IDEA

IntelliJ IDEA - це інтегроване середовище розробки (IDE) для програмістів, розроблене компанією JetBrains. Це потужний набір інструментів, який спрощує та полегшує розробку програмного забезпечення на різних мовах програмування, включаючи Java, Kotlin, JavaScript, Python та інші.

IntelliJ IDEA надає розширені можливості для розробки веб-додатків, мобільних додатків, серверних додатків та інших програмних рішень. Його функціонал включає інтелектуальний редактор коду, підтримку роботи з версійним контролем, вбудовані інструменти для рефакторингу коду, дебаггінг, автоматичні перевірки та інші корисні функції, спрямовані на підвищення продуктивності розробника.

Причини вибору:

1. Зручний інтерфейс: IntelliJ IDEA має інтуїтивно зрозумілий та дружній інтерфейс, що спрощує роботу розробника та покращує продуктивність. Його візуальні елементи, такі як допоміжні підказки, автодоповнення та інші функції, роблять робочий процес більш зручним і швидким.

2. Потужні інструменти розробки: IntelliJ IDEA надає широкий спектр інструментів для підтримки розробки, таких як рефакторинг коду, підтримка технологій, розуміння структури проектів тощо. Ці можливості допомагають розробникам швидко створювати та підтримувати високоякісний код.

3. Підтримка Java-проектів: IntelliJ IDEA забезпечує повну підтримку мови програмування Java, включаючи відладку, автоматичну перевірку коду, підказки для правильного використання API та інші корисні функції, що полегшують процес розробки на Java.

Налаштування середовища включає такі етапи:

1. Створення проекту: Вибір типу проекту (наприклад, Spring Boot), налаштування SDK (Software Development Kit) та інші параметри проекту.
2. Додавання плагінів та залежностей: Інтеграція необхідних плагінів (наприклад, Spring, Lombok) та залежностей (бібліотеки, фреймворки тощо) до проекту за допомогою Maven або Gradle.
3. Налаштування конфігурацій: Налаштування інструментів, таких як автодоповнення, відлагодження, системи контролю версій тощо, для підтримки комфортної розробки.

3.1.2. Використання Maven

Файл pom.xml: Детальне описання структури pom.xml з усіма залежностями, плагінами та конфігурацією Maven для проекту лікарні.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>artem</groupId>
    <artifactId>hospital</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>hospital</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.thymeleaf.extras</groupId>
            <artifactId>thymeleaf-extras-springsecurity6</artifactId>
        </dependency>
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <version>42.7.0</version>
        </dependency>
```

Рис. 3.1. Залежності для проекту

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludes>
                    <exclude>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </exclude>
                </excludes>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Рис. 3.2. Залежності для проекту

У цьому pom.xml визначаються основні залежності для проекту Spring Boot, такі як Spring Web, Spring Data JPA для роботи з базою даних, PostgreSQL Driver для взаємодії з PostgreSQL, а також налаштування для компіляції проекту на Java 11 та запуску Spring Boot.

3.2. Налаштування бази даних

Структура бази даних: Детальний опис структури таблиць, зв'язків та сутностей у PostgreSQL для відображення інформації про пацієнтів, лікарів та призначень.

1. Таблиця "Пацієнти" (Patients):

- id (ідентифікатор)
- ім'я
- прізвище
- адреса
- номер телефону

2. Таблиця "Лікарі" (Doctors):

- id (ідентифікатор)
- ім'я
- прізвище
- спеціальність
- номер телефону

3. Таблиця "Призначення" (Appointments):

- id (ідентифікатор)
- id_лікаря (зовнішній ключ до таблиці "Лікарі")
- id_пацієнта (зовнішній ключ до таблиці "Пацієнти")
- дата_призначення
- час_призначення
- опис

3.3. Розробка класів моделей

3.3.1. Створення моделей

Модель пацієнта: Опис класу Patient з атрибутами, які відображають основну інформацію про пацієнта.

```
package artem.hospital.model;

import ...

20 usages
@Entity
@Table(name = "patients")
@Getter
@Setter
public class Patient {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String firstName;
    private String lastName;
    private String address;
    private String contactNumber;
    private int age;
    private String diagnosis;
}
```

Рис. 3.3. Модель Пацієнта

Модель лікаря: Створення класу Doctor з відображенням основних даних про лікаря.

```java
package artem.hospital.model;

import ...

20 usages
@Entity
@Table(name = "doctors")
@Getter
@Setter
public class Doctor {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String firstName;
    private String lastName;
    private String specialization;
    private String contactNumber;
}
```

Рис. 3.4. Модель Доктора

Модель призначення: Розробка класу Appointment для відображення інформації про призначення.

```
package artem.hospital.model;

import ...

19 usages
@Entity
@Table(name = "appointments")
@Getter
@Setter
public class Appointment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "doctor_id")
    private Doctor doctor;

    @ManyToOne
    @JoinColumn(name = "patient_id")
    private Patient patient;

    private LocalDate appointmentDate;
    private LocalDateTime appointmentDateTime;
    private String reason;
}
```

Рис. 3.5. Модель Призначення

3.4. Розробка класів сервісів

3.4.1. Реалізація сервісів

Логіка для пацієнтів: Опис методів сервісу для обробки операцій з пацієнтами (створення, читання, оновлення, видалення).

PatientServiceImpl: Цей сервіс відповідає за обробку даних про пацієнтів у системі. Він надає функції для отримання списку всіх пацієнтів, знаходження пацієнта за його ідентифікатором, додавання нового пацієнта, оновлення існуючого та видалення пацієнта за його ідентифікатором.

```java
@Service
public class PatientServiceImpl implements PatientService {

    7 usages
    private final PatientRepository patientRepository;

    @Autowired
    public PatientServiceImpl(PatientRepository patientRepository) { this.patientRepository = patientRepository; }

    1 usage
    @Override
    public List<Patient> getAllPatients() { return patientRepository.findAll(); }

    1 usage
    @Override
    public Patient getPatientById(Long id) { return patientRepository.findById(id).orElse( other: null); }

    1 usage
    @Override
    public Patient addPatient(Patient patient) { return patientRepository.save(patient); }

    1 usage
    @Override
    public Patient updatePatient(Long id, Patient patient) {
        if (patientRepository.existsById(id)) {
            patient.setId(id);
            return patientRepository.save(patient);
        }
        return null;
    }

    1 usage
    @Override
    public void deletePatient(Long id) { patientRepository.deleteById(id); }
}
```

Рис. 3.6. Сервіс Пацієнта

Функціонал для лікарів: Реалізація методів сервісу для управління даними про лікарів.

DoctorServiceImpl: Цей сервіс взаємодіє з лікарями у системі. Його методи дозволяють отримати всіх лікарів, отримати лікаря за його ідентифікатором, додати нового лікаря, оновити існуючого та видалити лікаря за його ідентифікатором. Цей сервіс надає доступ до функціоналу збереження, зчитування, оновлення та видалення об'єктів типу Doctor в базі даних.

```
@Service
public class DoctorServiceImpl implements DoctorService {

    7 usages
    private final DoctorRepository doctorRepository;

    @Autowired
    public DoctorServiceImpl(DoctorRepository doctorRepository) { this.doctorRepository = doctorRepository; }

    1 usage
    @Override
    public List<Doctor> getAllDoctors() { return doctorRepository.findAll(); }

    1 usage
    @Override
    public Doctor getDoctorById(Long id) { return doctorRepository.findById(id).orElse( other: null); }

    1 usage
    @Override
    public Doctor addDoctor(Doctor doctor) { return doctorRepository.save(doctor); }

    1 usage
    @Override
    public Doctor updateDoctor(Long id, Doctor doctor) {
        if (doctorRepository.existsById(id)) {
            doctor.setId(id);
            return doctorRepository.save(doctor);
        }
        return null;
    }

    1 usage
    @Override
    public void deleteDoctor(Long id) { doctorRepository.deleteById(id); }
}
```

Рис. 3.7. Сервіс Доктора

Операції з призначеннями: Розробка функціоналу сервісу для обробки призначень.

AppointmentServiceImpl: Цей сервіс керує призначеннями. Він надає можливість отримати всі призначення, знайти призначення за його ідентифікатором, додати нове призначення, оновити існуюче та видалити призначення за його ідентифікатором.

```
@Service
public class AppointmentServiceImpl implements AppointmentService {

    7 usages
    private final AppointmentRepository appointmentRepository;

    @Autowired
    public AppointmentServiceImpl(AppointmentRepository appointmentRepository) {
        this.appointmentRepository = appointmentRepository;
    }

    1 usage
    @Override
    public List<Appointment> getAllAppointments() { return appointmentRepository.findAll(); }

    1 usage
    @Override
    public Appointment getAppointmentById(Long id) { return appointmentRepository.findById(id).orElse( other: null); }

    1 usage
    @Override
    public Appointment addAppointment(Appointment appointment) { return appointmentRepository.save(appointment); }

    1 usage
    @Override
    public Appointment updateAppointment(Long id, Appointment appointment) {
        if (appointmentRepository.existsById(id)) {
            appointment.setId(id);
            return appointmentRepository.save(appointment);
        }
        return null;
    }

    1 usage
    @Override
    public void deleteAppointment(Long id) { appointmentRepository.deleteById(id); }
}
```

Рис. 3.8. Сервіс Призначення

3.5. Розробка класів контролерів

3.5.1. Реалізація контролерів

Контролер для пацієнтів: Реалізація контролера для обробки HTTP-запитів, пов'язаних з пацієнтами.

PatientController: Цей контролер відповідає за обробку даних про пацієнтів. Він надає методи для отримання всіх пацієнтів, конкретного пацієнта, додавання, оновлення та видалення пацієнта.

```
@Controller
@RequestMapping("/patients")
public class PatientController {

    6 usages
    private final PatientService patientService;

    @Autowired
    public PatientController(PatientService patientService) { this.patientService = patientService; }

    @GetMapping
    public String getAllPatients(Model model) {
        model.addAttribute( attributeName: "patients", patientService.getAllPatients());
        return "patients";
    }

    @GetMapping("/{id}")
    public String getPatientById(@PathVariable Long id, Model model) {
        model.addAttribute( attributeName: "patient", patientService.getPatientById(id));
        return "patient";
    }

    @PostMapping
    public String addPatient(@ModelAttribute Patient patient) {
        patientService.addPatient(patient);
        return "redirect:/patients";
    }

    @PutMapping("/{id}")
    public String updatePatient(@PathVariable Long id, @ModelAttribute Patient patient) {
        patientService.updatePatient(id, patient);
        return "redirect:/patients";
    }

    @DeleteMapping("/{id}")
    public String deletePatient(@PathVariable Long id) {
        patientService.deletePatient(id);
        return "redirect:/patients";
    }
}
```

Рис. 3.9. Контролер Пацієнта

Контролер для лікарів: Розробка контролера для взаємодії з даними лікарів.

DoctorController: Цей контролер відповідає за операції з даними про лікарів. Він надає методи для відображення всіх лікарів, отримання конкретного лікаря, додавання, оновлення та видалення лікаря.

```
@Controller
@RequestMapping("/doctors")
public class DoctorController {

    6 usages
    private final DoctorService doctorService;

    @Autowired
    public DoctorController(DoctorService doctorService) { this.doctorService = doctorService; }

    @GetMapping
    public String getAllDoctors(Model model) {
        model.addAttribute( attributeName: "doctors", doctorService.getAllDoctors());
        return "doctors";
    }

    @GetMapping("/{id}")
    public String getDoctorById(@PathVariable Long id, Model model) {
        model.addAttribute( attributeName: "doctor", doctorService.getDoctorById(id));
        return "doctor";
    }

    @PostMapping
    public String addDoctor(@ModelAttribute Doctor doctor) {
        doctorService.addDoctor(doctor);
        return "redirect:/doctors";
    }

    @PutMapping("/{id}")
    public String updateDoctor(@PathVariable Long id, @ModelAttribute Doctor doctor) {
        doctorService.updateDoctor(id, doctor);
        return "redirect:/doctors";
    }

    @DeleteMapping("/{id}")
    public String deleteDoctor(@PathVariable Long id) {
        doctorService.deleteDoctor(id);
        return "redirect:/doctors";
    }
}
```

Рис. 3.10. Контролер Доктора

Контролер для призначень: Створення контролера для операцій з призначеннями.

AppointmentController: Цей контролер відповідає за обробку даних про призначення. Методи getAllAppointments, getAppointmentById обробляють GET-запити для отримання всіх призначень або конкретного призначення за його ID. Методи addAppointment, updateAppointment, deleteAppointment відповідають відповідно за додавання, оновлення та видалення призначення. Кожен з цих методів повертає рядок, який вказує, яку сторінку потрібно відобразити після обробки запиту.

```java
@Controller
@RequestMapping("/appointments")
public class AppointmentController {

    6 usages
    private final AppointmentService appointmentService;

    @Autowired
    public AppointmentController(AppointmentService appointmentService) {
        this.appointmentService = appointmentService;
    }

    @GetMapping
    public String getAllAppointments(Model model) {
        model.addAttribute( attributeName: "appointments", appointmentService.getAllAppointments());
        return "appointments";
    }

    @GetMapping("/{id}")
    public String getAppointmentById(@PathVariable Long id, Model model) {
        model.addAttribute( attributeName: "appointment", appointmentService.getAppointmentById(id));
        return "appointment";
    }

    @PostMapping
    public String addAppointment(@ModelAttribute Appointment appointment) {
        appointmentService.addAppointment(appointment);
        return "redirect:/appointments";
    }

    @PutMapping("/{id}")
    public String updateAppointment(@PathVariable Long id, @ModelAttribute Appointment appointment) {
        appointmentService.updateAppointment(id, appointment);
        return "redirect:/appointments";
    }

    @DeleteMapping("/{id}")
    public String deleteAppointment(@PathVariable Long id) {
        appointmentService.deleteAppointment(id);
        return "redirect:/appointments";
    }
}
```

Рис. 3.11. Контролер Призначення

Контролер для логіну: Розробка контролера для сторінки логіну

LoginController: Цей контролер відображає сторінку для входу користувача. Метод showLoginForm відображає сторінку входу.

```
package artem.hospital.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class LoginController {

    @GetMapping("/login")
    public String showLoginForm() {
        return "login";
    }
}
```

Рис. 3.12. Контролер Логіну

Контролер для лікарні: Реалізація контролера для основного функціоналу лікарні

HospitalController: Цей контролер відповідає за відображення основних сторінок, таких як "Про нас" (/about), "Контакти" (/contacts), "Послуги" (/services). Кожен метод відображає відповідну сторінку.

```
@Controller
public class HospitalController {

    @GetMapping("/about")
    public String showAboutPage() {
        return "about";
    }

    @GetMapping("/contacts")
    public String showContactsPage() { return "contacts"; }

    @GetMapping("/services")
    public String showServicesPage() { return "services"; }
}
```

Рис. 3.13. Контролер Лікарні

3.6. Налаштування Spring Security

3.6.1. Конфігурація безпеки

Захист доступу до ресурсів: Налаштування Spring Security для забезпечення безпеки різних частин додатку.

```
@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) {
        return http.addFilter()
                .authorizeExchange()
                .pathMatchers("/login").permitAll()
                .pathMatchers("/about", "/contacts", "/services").permitAll()
                .anyRequest().authenticated()
                .and()
                .formLogin()
                .loginPage("/login")
                .defaultSuccessUrl("/doctors")
                .permitAll()
                .and()
                .logout()
                .logoutUrl("/logout")
                .logoutSuccessUrl("/login")
                .permitAll()
                .and()
                .exceptionHandling()
                .accessDeniedPage("/access-denied")
                .and()
                .build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return BCryptPasswordEncoder();
    }
```

Рис. 3.14. Конфігурація Безпеки

@Configuration та @EnableWebSecurity: Анотації, що позначають клас як конфігурацію Spring Security.

configure(HttpSecurity http): Метод, який конфігурує поведінку безпеки.

- authorizeRequests(): Визначення правил доступу до різних ендпоінтів на основі ролей користувачів.
- antMatchers(): Специфікація URL та визначення необхідних ролей.
- anyRequest().authenticated(): Вимагає аутентифікації для всіх інших запитів.
- formLogin(): Налаштування сторінки логінування.
- permitAll(): Дозвіл для всіх користувачів на отримання доступу.
- logout(): Налаштування виходу з системи.

configureGlobal(AuthenticationManagerBuilder auth): Налаштування тестових користувачів у пам'яті.

passwordEncoder(): Бін для використання методу NoOpPasswordEncoder.

Ця конфігурація реалізує простий механізм авторизації та аутентифікації для різних ендпоінтів системи управління лікарнею.

3.7. Розробка HTML-сторінок

3.7.1. Створення шаблонів

Шаблони з використанням Thymeleaf: Розробка HTML-шаблонів для відображення інтерфейсу користувача.

Концепція сайту лікарні передбачає зручний та інформативний інтерфейс, який дозволяє користувачам знайти необхідну інформацію про лікарню, лікарів та можливість запису на призначення.

Структура сторінки:

1. Інформація про лікарню:

- Назва та логотип лікарні.
- Короткий опис про лікарню, її спеціалізації та послуги.

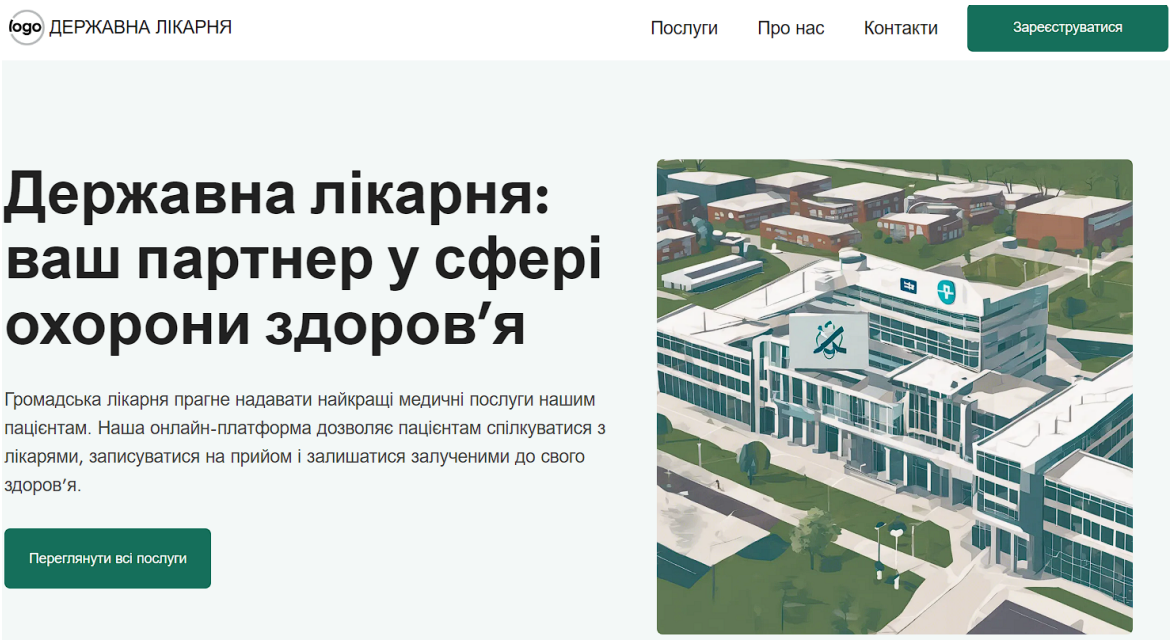

Рис. 3.15. Головна сторінка

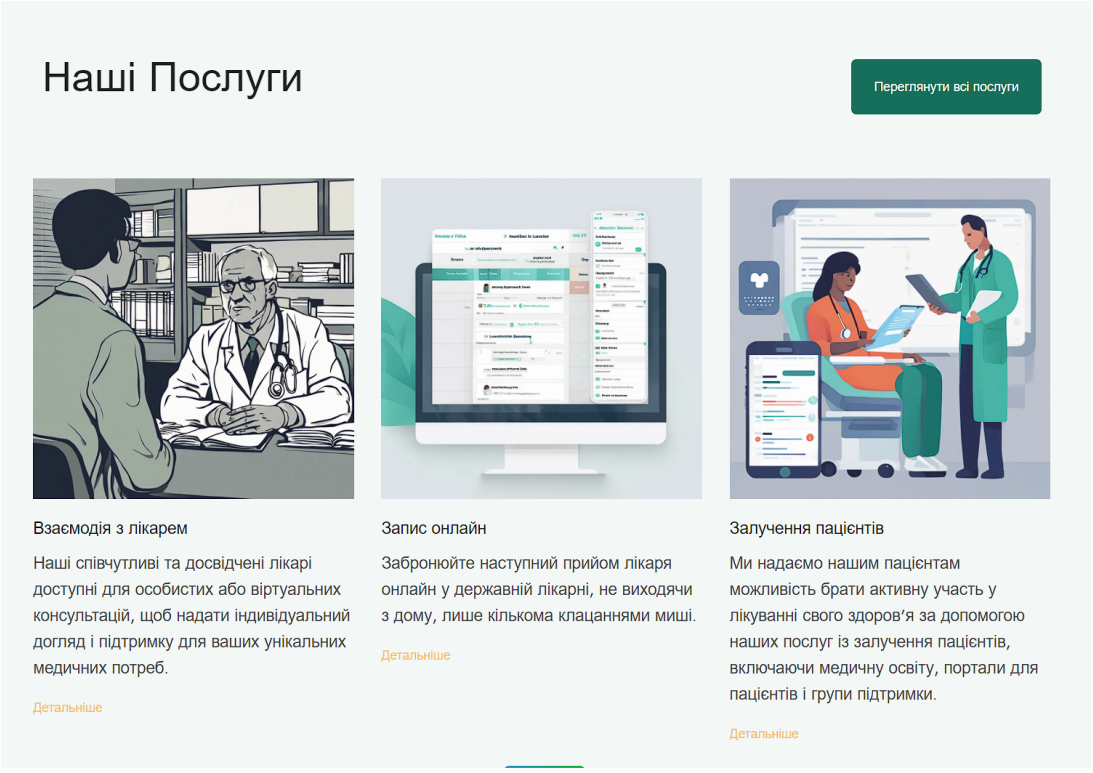

Рис. 3.16. Сторінка послуг

```
<div class="container">
    <h2>Наші Послуги</h2>
    <div class="service">
        <h2 class="service-header">Взаємодія з лікарем</h2>
        <img src="C:/artem/hospital/image_1.jpg" alt="">
        <p class="service-description">Наші співчутливі та досвідчені лікарі доступні для особистих
            або віртуальних консультацій, щоб надати індивідуальний догляд
            і підтримку для ваших унікальних медичних потреб.
        </p>
        <a href="#" class="details-link">Детальніше</a>
    </div>
    <div class="service">
        <h2 class="service-header">Запис онлайн</h2>
        <img src="C:/artem/hospital/image_2.jpg" alt="">
        <p class="service-description">Забронюйте наступний прийом лікаря онлайн у державній лікарні,
            не виходячи з дому, лише кількома клацаннями миші.</p>
        <a href="#" class="details-link">Детальніше</a>
    </div>
    <div class="service">
        <h2 class="service-header">Залучення пацієнтів</h2>
        <img src="C:/artem/hospital/image_3.jpg" alt="">
        <p class="service-description">Ми надаємо нашим пацієнтам можливість брати активну участь
            у лікуванні свого здоров'я за допомогою наших послуг із залучення пацієнтів, включаючи
            медичну освіту, портали для пацієнтів і групи підтримки.</p>
        <a href="#" class="details-link">Детальніше</a>
    </div>
</div>
```

Рис. 3.17. HTML-код сторінки послуг

2. Розділ про лікарів:

- Картки лікарів із зображенням, інформацією про спеціалізацію, досвід роботи.
- Посилання для детальної інформації про кожного лікаря (може бути окрема сторінка для кожного).

Рис. 3.18. Сторінка штату докторів

```
<div th:each="doctor : ${doctors}">
    <div class="doctor-info">
        <h2 th:text="${doctor.name}"></h2>
        <img th:src="${doctor.image}" alt="">
        <p th:text="${doctor.specialization}"></p
        <p th:text="${doctor.description}"></p>
    </div>
</div>
```

Рис. 3.19. HTML-код сторінки штату докторів

3. Запис на призначення:

   Форма для заповнення для запису на призначення: вибір лікаря, обрання дати та часу.

   Кнопка для підтвердження запису на призначення.

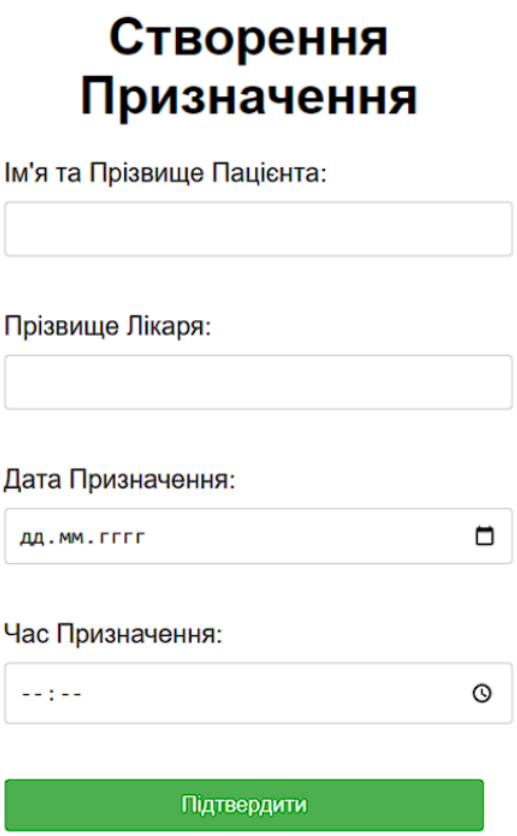

Рис. 3.20. Сторінка форми заяви призначення

```html
<div class="form-container">
    <h1>Створення Призначення</h1>
    <form action="/appointments" method="post">
        <label for="patientName">Ім'я ата Прізвище Пацієнта:</label>
        <input type="text" id="patientName" name="patientName" required><br><br>

        <label for="doctorSurname">Прізвище Лікаря:</label>
        <input type="text" id="doctorSurname" name="doctorSurname" required><br><br>

        <label for="appointmentDate">Дата Призначення:</label>
        <input type="date" id="appointmentDate" name="appointmentDate" required><br><br>

        <label for="appointmentTime">Час Призначення:</label>
        <input type="time" id="appointmentTime" name="appointmentTime" required><br><br>

        <input type="submit" value="Підтвердити">
    </form>
</div>
```

Рис. 3.21. HTML-код сторінки форми заяви призначення

3.8. Тестування та налагодження

3.8.1. Тестування функціональності

Модульні тести:

Модульне тестування - це процес перевірки роботи окремих модулів або компонентів програми. Ці тести перевіряють поведінку програмних модулів на основі їхнього інтерфейсу та специфікацій.

```java
@SpringBootTest
public class AppointmentServiceTests {

    1 usage
    @Mock
    private AppointmentRepository appointmentRepository;

    1 usage
    @InjectMocks
    private AppointmentServiceImpl appointmentService;

    @Test
    public void testGetAppointmentById() {
        Appointment appointment = new Appointment();
        when(appointmentRepository.findById(1L)).thenReturn(Optional.of(appointment));

        Appointment foundAppointment = appointmentService.getAppointmentById(1L);
        assertEquals(appointment.getReason(), foundAppointment.getReason());
    }
}
```

Рис. 3.22. Класс тестування

3.8.2. Відлагодження

Налагодження та відлагодження:

Цей етап включає в себе виявлення та виправлення помилок у розробленому додатку.

Виявлення помилок: Під час тестування або експлуатації програми можуть виникати помилки. Важливо їх знайти, використовуючи різні методи: відлагоджувальні засоби, журналізацію, спостереження за поведінкою програми тощо.

Виправлення помилок: Після виявлення помилок розробник вносить зміни у вихідний код для їх виправлення. Це може включати зміну алгоритмів, роботу з базою даних, покращення безпеки тощо.

ВИСНОВКИ

В ході дослідження та розробки системи управління лікарнею було проведено глибокий аналіз функціональних потреб лікарні та сучасних інформаційних технологій у медичному секторі. Виявлено, що ефективність управління лікарнею значно підвищується завдяки використанню спеціалізованих інструментів розробки та баз даних, які відповідають специфіці сфери охорони здоров'я.

Результатом дослідження є створення функціонального веб-додатку, який відповідає основним потребам лікарні. Використання Java разом із фреймворком Spring дозволило реалізувати ефективний і безпечний інструмент для управління лікарською установою.

Отримані результати вказують на перспективи подальшого розвитку систем управління лікарнею на основі сучасних технологій. Робота над цим проектом також підкреслила важливість безпеки даних, масштабованості та ефективності у медичних системах.

Загальною метою дослідження та розробки веб-додатку було створення інструменту, що покликаний сприяти покращенню роботи медичних установ та поліпшенню надання медичних послуг пацієнтам.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. V. K. Singh and S. M. Bhatnagar. "E-Health and Telemedicine: A Guide to the Internet for Professionals." - 2023.
2. "Healthcare Information Systems" by Aspen Publishers - 2023.
3. Mervat Abdelhak. "Health Information Management: Concepts, Principles, and Practice" - 2022.
4. Sharon B. Buchbinder and Nancy H. Shanks. "Introduction to Health Care Management" - 2021.
5. Joseph Tan. "Healthcare Information Systems and Informatics: Research and Practices" - 2018.
6. Amuthan G and Ajith Kumar. "Web Application Development with Angular and Spring Boot" - 2015.
7. "Spring Framework Documentation" [Інтернет-ресурс] / Режим доступу: https://spring.io/guides
8. "Angular Documentation" [Інтернет-ресурс] / Режим доступу: https://angular.io/docs
9. Richard Gartee. "Electronic Health Records: Understanding and Using Computerized Medical Records" - 2020.
10. Ramona Nelson and Nancy Staggers. "Health Informatics: An Interprofessional Approach" - 2017.
11. S. P. Tiwari. "Hospital Management System" - 2021.
12. Bernard J. Healey and Tina M. Barber. "Introduction to Health Care Services: Foundations and Challenges" - 2016.
13. Susan Fowler and Adam Zolyak. "Web Application Development: Concepts, Principles, and Patterns" - 2019.
14. Filip Bruun. "Developing Web Apps with Angular and TypeScript" - 2018.
15. Kathleen A. McCormick. "Healthcare Information Technology Exam Guide for CompTIA Healthcare IT Technician and HIT Pro Certifications" - 2023.
16. Martin Kleppmann. "Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems" - 2022.

17. Christian Bauer and Gavin King. "Java Persistence with Hibernate" - 2018.
18. Craig Walls. "Spring Boot in Action" - 2015.
19. Yakov Fain and Anton Moiseev. "Angular Development with TypeScript" - 2020.
20. Gerald L. Glandon, Detlev H. Smaltz, and Donna J. Slovensky. "Information Systems for Healthcare Management" - 2017.