КВАЛІФІКАЦІЙНА РОБОТА

Група ІПЗс-2019
Стрілецький В.Т.

2023

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Стрілецький Віталій Тарасович

УДК 004.378

Розробка веб-застосунку обліку пацієнтів стоматологічної клініки засобами Spring Framework.

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації бакалавра

Нормоконтроль

_____Стисло О.В.

(підпис, дата, розшифрування підпису)

Студент

_____Стрілецький В.Т.

(підпис, дата, розшифрування підпису)

Допускається до захисту

Завідувач кафедри

_____к.т.н., доц. Пашкевич О.П.

(підпис, дата, розшифрування підпису)

Керівник роботи

_____к.т.н., доц. Пашкевич О.П.

(підпис, дата, розшифрування підпису)

Івано-Франківськ – 2023

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук
Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»
Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____

« ___ » _____ 2023 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Стрілецький Віталій Тарасович

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Розробка веб-застосунку обліку пацієнтів стоматологічної клініки засобами Spring Framework.

керівник роботи:
Пашкевич Олег Петрович

затверджена наказом вищого навчального закладу від «11» листопада 2022 року № 155/1НВ

2. Термін подання студентом роботи 14.06.2023.

3. Вихідні дані роботи: Мова програмування Java, Spring Framework, MySQL.

4. Зміст кваліфікаційної роботи. (перелік питань, які потрібно розробити)

1. Аналіз проблеми реєстрації пацієнтів та огляд існуючих рішень.
2. Розробка архітектури клієнт-серверної аплікації для реєстрації пацієнтів.
3. Реалізація структури та функціоналу веб-аплікації

5. Дата видачі завдання 10.10.2022

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Огляд програмного забезпечення стоматологічних клінік	24.02.2023	Виконано
2.	Розробка архітектури клієнт-серверної аплікації	02.03.2023	Виконано
3.	Програмна реалізація застосунку	30.04.2023	Виконано
4.	Оформлення пояснювальної записки	02.05.2023	Виконано
5.	Оформлення графічного матеріалу та підготовка до захисту роботи	02.05.2023	Виконано

Студент _____ Стрілецький В.Т.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Пашкевич О.П.
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

У кваліфікаційній роботі розглянуто розробку програмного забезпечення для стоматологічного кабінету з використанням сучасних веб-технологій. Розроблена програма надає зручність та огляд плану прийомів для користувачів, дозволяючи переглядати, редагувати та видаляти записи до стоматолога. Крім того, програма надає стоматологам можливість переглядати записи своїх пацієнтів та видаляти їх.

У роботі детально розглянуто процес розробки програмного забезпечення, включаючи аналіз вимог, проектування бази даних, реалізацію функціональності, тестування та впровадження. Були використані сучасні технології, такі як Java, Spring Framework і база даних MySQL.

Програма надає користувачам можливість зручно керувати своїми записами, забезпечуючи гнучкість та контроль над прийомами. Також вона сприяє ефективній взаємодії між пацієнтами та стоматологами, спрощуючи процес планування та організації стоматологічного обслуговування.

Результати роботи показали, що розроблена програма успішно впроваджує функціональність для управління записами стоматологічного кабінету, сприяючи поліпшенню організації та доступності стоматологічної послуги.

КЛЮЧОВІ СЛОВА: JAVA, SPRING FRAMEWORK, MYSQL

SUMMARY

The qualification work focuses on the development of software for a dental clinic using modern web technologies. The developed program provides convenience and an overview of appointment schedules for users, allowing them to view, edit, and delete appointments with the dentist. Additionally, the program enables dentists to access and delete records of their patients.

The work extensively covers the process of software development, including requirements analysis, database design, implementation of functionality, testing, and deployment. Modern technologies such as Java, Spring Framework, and MySQL database were utilized.

The program empowers users to efficiently manage their appointments, offering flexibility and control over their schedules. It also facilitates effective interaction between patients and dentists, streamlining the process of appointment planning and dental service organization.

The results of the work demonstrate the successful implementation of the program's functionality in managing records for a dental clinic, contributing to improved organization and accessibility of dental services.

KEY WORDS: JAVA, SPRING FRAMEWORK, MYSQL

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DAO – Об’єкти доступу до даних

ORM – Об'єктно-реляційне відображення

JVM – Віртуальна машина Java

ООП – Об'єктно-орієнтоване програмування

MVC – Контролер перегляду моделі

HTML – Hyper Text Markup Language

CSS – Cascading Style Sheets

ВСТУП

Актуальність теми. Здоров'я і догляд за зубами є невід'ємною складовою загального фізичного благополуччя кожної людини. Однак, традиційний спосіб запису до стоматолога часто пов'язаний з труднощами і неефективністю, такими як незручні години роботи клініки, довгі черги та потреба особисто звертатися до реєстратури. Ці незручності можуть викликати відкладання або навіть уникання лікування, що негативно впливає на стан здоров'я пацієнтів.

У такому контексті розробка та використання стоматологічного веб-сайту з можливістю реєстрації та онлайн-запису на прийом до лікаря набувають великого значення. Цей додаток дозволить пацієнтам зручно та швидко зареєструватися, обрати зручний час відвідування, а також отримати необхідну інформацію про лікаря, послуги та ціни безпосередньо через Інтернет.

Окрім того, відкриття доступу до онлайн-сервісу з реєстрації та запису на прийом може сприяти залученню нових клієнтів до стоматологічної клініки. Завдяки зручності та широкому доступу до Інтернету, пацієнти зможуть швидко знайти інформацію про клініку, ознайомитися з відгуками і рейтингами, що позитивно вплине на їх вибір медичного закладу. Також, забезпечуючи можливість онлайн-запису, клініка підвищує рівень задоволення клієнтів, оскільки надає їм контроль над процесом запису та вибору зручного часу.

Крім того, використання стоматологічного веб-сайту з реєстрацією та записом на прийом сприяє ефективному управлінню робочим часом лікарів та персоналу клініки. Система онлайн-запису дозволяє автоматизувати процес прийому пацієнтів, розподіляючи їх по графіку роботи лікарів, уникати перекриття часу та мінімізувати очікування. Це сприяє оптимізації робочого процесу, підвищенню продуктивності та забезпеченню високої якості медичного обслуговування.

Мета роботи. Розробка клієнт-серверної веб-аплікації реєстрації пацієнтів стоматологічної клініки.

Об’єкт роботи. Процедура реєстрації та обліку пацієнтів стоматологічної клініки.

Предмет роботи. Розробка веб-застосунку обліку пацієнтів стоматологічної клініки засобами Spring Framework.

Завдання роботи. Відповідно до вибраної теми в роботі покладені такі задачі як:

- пошук існуючих на даний момент аналогів для їхнього аналізу;
- вибір мови програмування та технологій розробки
- розроблення сучасного та зручного дизайну;
- пошук даних методи створення та шляхи розповсюдження;
- проведення тестування продукту.

Методи роботи. Для вирішення поставленого завдання були використані мова програмування Java, база даних MySQL, фреймворк Spring Boot.

Результати роботи. Після виконання кваліфікаційної роботи було створено web-застосунок для стоматологічного сайту, який дозволяє зручно записуватись онлайн до стоматолога. Користувачі можуть легко вибрати бажану дату та час візиту, заповнити необхідну інформацію та отримати підтвердження. Перспективи проекту включають можливість розширення функціоналу у майбутньому.

Структура роботи. Розділи – 3. Загальний обсяг основної частини – 63 сторінок. Список використаних джерел – 20.

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМИ РЕЄСТРАЦІЇ ПАЦІЄНТІВ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд програмного забезпечення які використовують стоматологічні клініки

В мережі Інтернет можна знайти безліч стоматологічних сайтів, присвячених наданню інформації про стоматологічні клініки, послуги та лікарів. Проте, серед цих сайтів виявляється значна кількість, які не пропонують можливість онлайн-запису на прийом, і це є серйозним недоліком, який впливає на зручність та ефективність для клієнтів.

Відсутність онлайн-запису на прийом на стоматологічних сайтах ускладнює процес організації та планування візиту клієнта до стоматолога. Клієнтам доводиться зателефонувати або відвідати клініку особисто, щоб записатися на прийом. Це може вимагати додаткових зусиль і часу з боку клієнта, особливо у випадках, коли вони мають обмежений час або перебувають у віддаленому місці.

Брак онлайн-запису також може призводити до зайнятості ліній телефонів клінік та створювати незручності для клієнтів, які можуть бути вимушені чекати у черзі на телефонному зв'язку або не отримати відповідь у разі зайнятості. Це може створювати негативний враження про клініку та призвести до втрати потенційних клієнтів.

Онлайн-запис на прийом у стоматологічних сайтах є важливою функцією, яка сприяє зручності та ефективності для клієнтів. Це дає можливість клієнтам самостійно обрати зручний для них час прийому без необхідності зв'язуватися з клінікою особисто або телефонувати.

Стоматологічні сайти які були проаналізовано мною:

- citysmile.com.ua
- mainz.clinic

- ukrstom-centr.com.ua

- dent-art.com.ua

- liliaestet.com.ua

Сайт citysmile.com.ua є веб-порталом стоматологічного центру CitySmile. На головній сторінці сайту розміщена велика банерна зображення, яка привертає увагу відвідувачів. Вона передає професіоналізм і дбайливе ставлення до пацієнтів. Основне меню знаходиться вгорі сторінки, зручно розташоване і надає легкий доступ до інформації про послуги, лікарів, акції та контактну інформацію. На (рис. 1.1) зображено головна сторінка сайту.

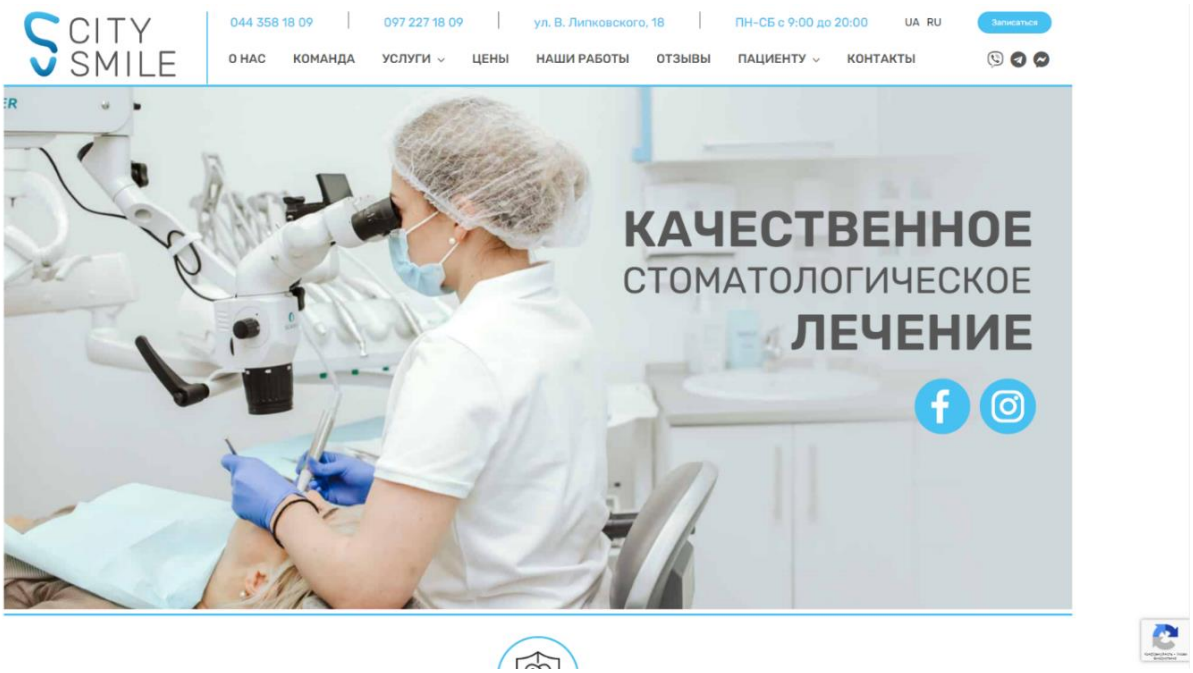

Рисунок 1.1 – Головна сторінка веб-сайту citysmile.com.ua

Сайт пропонує докладну інформацію про різні види послуг, які надає клініка. Кожна послуга має свою окрему сторінку з детальним описом процедури, фотографіями та ціновими планами. Це допомагає пацієнтам зробити інформований вибір і зрозуміти, що вони можуть очікувати від відвідування цього центру.

На сайті також є розділ, присвячений команді лікарів, де представлені фотографії і короткі біографії кожного спеціаліста. Це допомагає встановити

контакт між пацієнтом і лікарем, а також створює враження професіоналізму і надійності клініки.

Однак, сайт міг би бути більш інтерактивним і залучати відвідувачів до взаємодії. Наприклад, додавання онлайн-чату або форми для запису на прийом безпосередньо на сайт може полегшити комунікацію клієнтів з центром.

Сайт mainz.clinic є веб-порталом клініки Mainz, яка спеціалізується на наданні стоматологічних послуг. Головна сторінка містить інформацію про переваги клініки та якісне обслуговування. Дизайн сайту є сучасним і естетично приємним, з використанням яскравих фотографій і зручного розташування основного меню. На (рис. 1.2) зображено головна сторінка сайту.

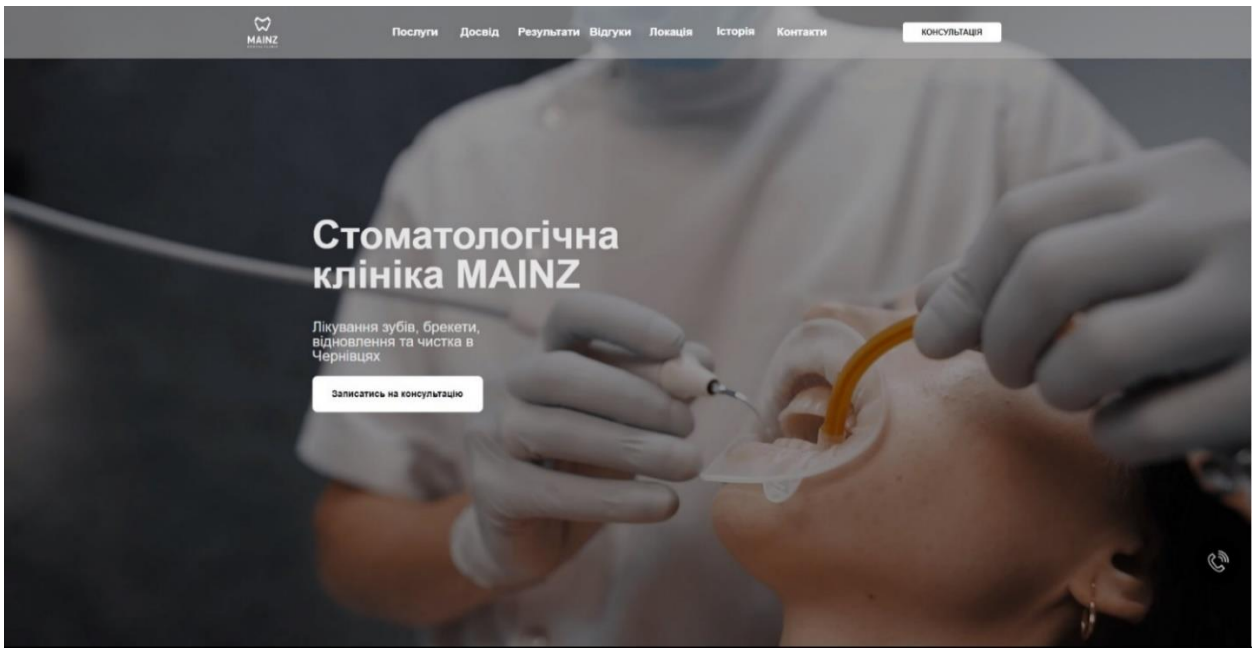

Рисунок 1.2 – Головна сторінка веб-сайту mainz.clinic

Сайт пропонує детальну інформацію про різні види послуг, які надаються в клініці Mainz. Кожна послуга має окрему сторінку з описом процедури, ілюстраціями та інформацією про вартість. Це дозволяє пацієнтам ознайомитися з усіма можливостями клініки і вибрати оптимальний варіант лікування.

Окремий розділ сайту присвячений команді лікарів. Кожен лікар представлений з фотографією, коротким описом кваліфікації та досвідом роботи.

Це дозволяє пацієнтам зробити обізнаний вибір і мати довіру до медичного персоналу.

Сайт також надає інформацію про акції та спеціальні пропозиції, що стимулює клієнтів до відвідування клініки. Наявність блогу з корисними статтями про стоматологію також сприяє підвищенню довіри до клініки та підтримує взаємодію з пацієнтами.

Однак, сайт може бути покращений шляхом додавання функціональності запису на прийом в Інтернеті. Це забезпечить зручність для клієнтів і економію їх часу. Також було б корисно включити розділ з частими запитаннями та контактною інформацією для зв'язку з клінікою.

Сайт ukrstom-centr.com.ua належить стоматологічному центру "УкрСтом-Центр". Головна сторінка сайту містить інформацію про послуги, переваги центру та команду лікарів. Дизайн сайту має приємну кольорову гаму і чітку структуру, що дозволяє легко зорієнтуватися на сайті. На (рис. 1.3) зображено головна сторінка сайту.

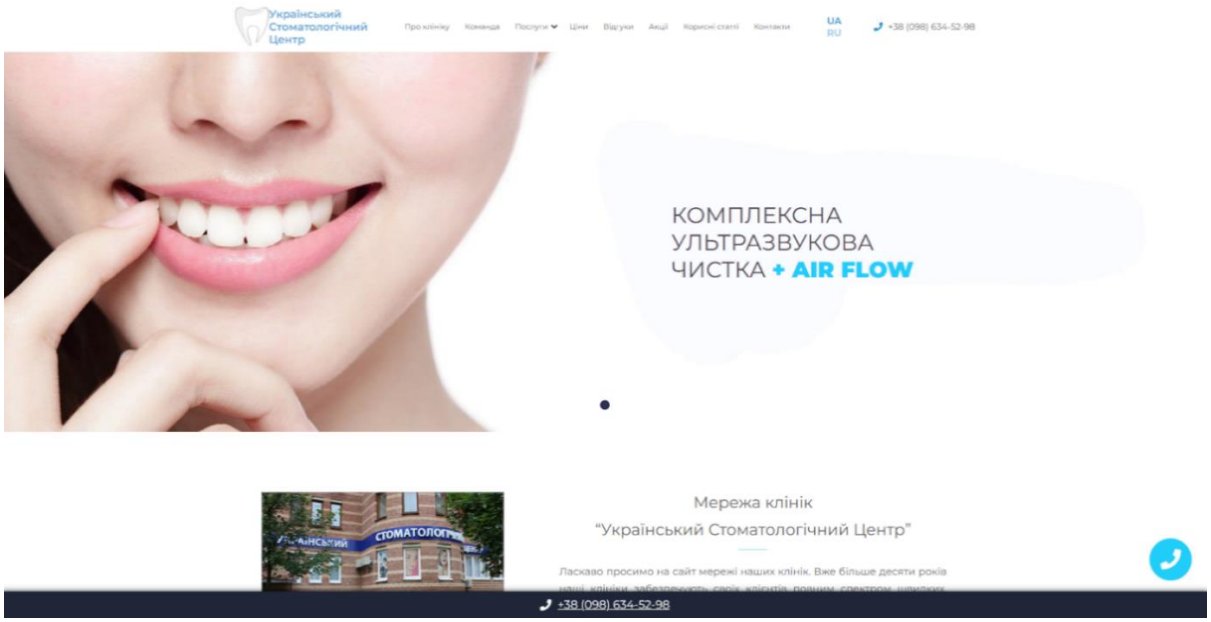

Рисунок 1.3 – Головна сторінка веб-сайту ukrstom-centr.com.ua

Сайт надає детальну інформацію про різні види послуг, які надаються в стоматологічному центрі "УкрСтом-Центр". Кожна послуга має окрему сторінку

з описом процедури, фотографіями та рекомендаціями. Це допомагає пацієнтам ознайомитися з різноманітністю лікування і вибрати оптимальний варіант.

Сайт також пропонує інформацію про команду лікарів, включаючи фотографії та короткі біографії. Це створює довіру до медичного персоналу і дозволяє пацієнтам зробити обізнаний вибір.

Наявність блогу на сайті з корисними статтями про стоматологію сприяє підвищенню інформованості пацієнтів і сприяє підтримці здоров'я ротової порожнини. Крім того, сайт має розділ з частими запитаннями, що допомагає пацієнтам знайти відповіді на свої запитання безпосередньо на сайті.

Однак, для подальшого поліпшення сайту можна розглянути додавання функціональності онлайн-запису на прийом. Це забезпечить зручність для клієнтів і заощадить їх час. Також, було б корисно включити розділ зі свідченнями задоволених клієнтів і відгуками про роботу центру, що додасть додаткову довіру до клініки.

Сайт стоматологічного центру "Dent-Art" має професійний та сучасний вигляд, що відразу привертає увагу відвідувачів. Його дизайн простий і лаконічний, з фокусом на зображеннях пацієнтів, які демонструють результати лікування. Головна сторінка містить важливу інформацію про послуги, команду лікарів та контактні дані. На (рис. 1.4) зображено головна сторінка сайту.

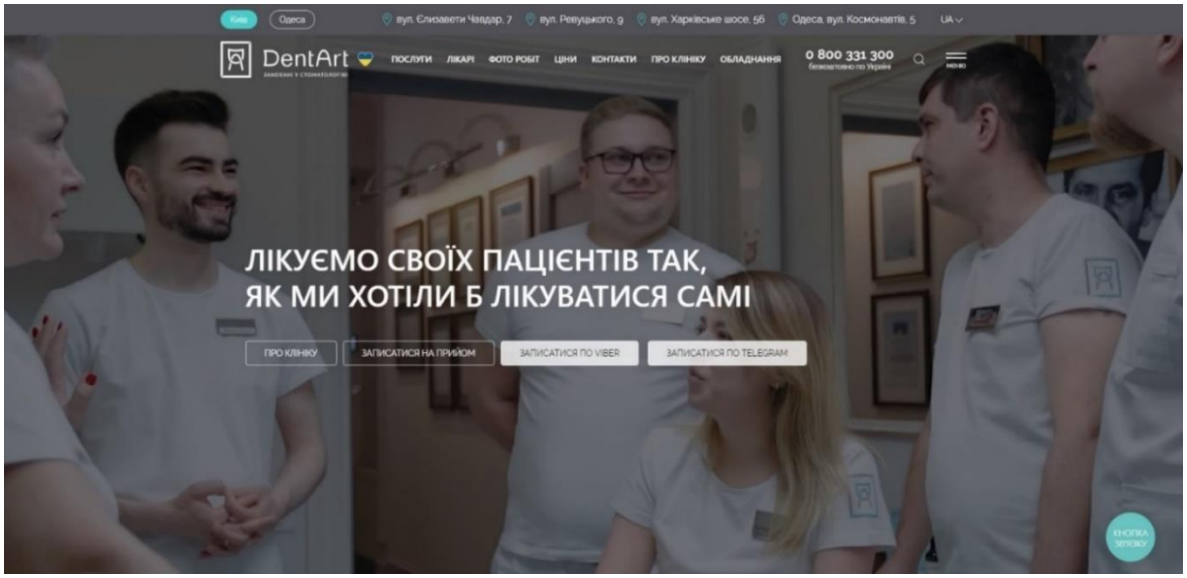

Рисунок 1.4 – Головна сторінка веб-сайту dent-art.com.ua

Сайт пропонує докладний опис різних видів стоматологічних послуг, включаючи естетичну стоматологію, ортодонтичне лікування, імплантацію і протезування. Кожна послуга має окрему сторінку з детальним описом процедури, відеоматеріалами та фотографіями. Це дозволяє пацієнтам зробити осмислений вибір і познайомитися з можливостями центру.

На сайті присутня інформація про кваліфікацію та досвід лікарів, а також їх фотографії та короткі біографії. Це допомагає пацієнтам отримати інформацію про фаховість та спеціалізацію лікарів перед візитом.

Сайт також має розділ зі статтями і порадами про догляд за зубами та порожниною рота. Це допомагає пацієнтам зберігати своє здоров'я та дізнаватися корисні поради для щоденного догляду.

Однак, для поліпшення сайту "Dent-Art" можна розглянути додавання функціоналу онлайн-запису на прийом, що забезпечить зручність для клієнтів і зменшить навантаження на адміністративний персонал.

Зокрема, сайт стоматологічного центру "Lilia Estet" пропонує широкий спектр послуг у галузі стоматології. На головній сторінці сайту можна побачити красиві фотографії усмішок пацієнтів, що підкреслює естетичний підхід центру. Дизайн сайту є сучасним та елегантним, створюючи професійний враження. На (рис. 1.5) зображено головна сторінка сайту.

Основні послуги, такі як зубне відбілювання, встановлення керамічних коронок і виготовлення вінірів, детально описані на окремих сторінках. Також присутня інформація про використовувані технології та обладнання, що гарантує якість та безпеку процедур.

Сайт надає можливість ознайомитися зі списком лікарів, які працюють у центрі, разом з їх фотографіями і короткими біографіями. Це допомагає пацієнтам вибрати лікаря, з яким вони відчувають зв'язок і довіру.

На сайті також присутній блог, де регулярно публікуються статті та корисні поради зі стоматологічного догляду. Це сприяє освіті пацієнтів та підвищенню свідомості про важливість здоров'я ротової порожнини.

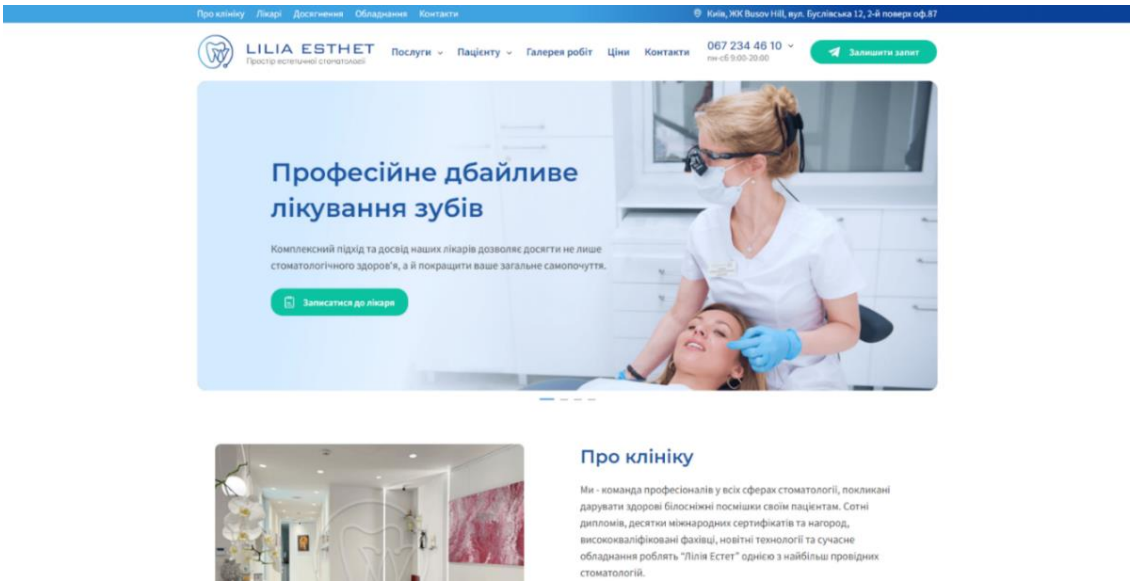

Рисунок 1.5 – Головна сторінка веб-сайту liliaestet.com.ua

У майбутньому, для подальшого розвитку сайту "Lilia Estet", можна розглянути впровадження інтерактивних елементів, таких як онлайн-консультації чи чат-підтримка, що дозволить пацієнтам отримати швидку відповідь на свої запитання та вирішити поточні питання.

1.2 Порівняння розглянутих програм аналогів.

Сайт з можливістю запису на прийом до стоматолога є значно кращим і більш зручним для клієнтів порівняно з тими, які були згадані раніше. Онлайн-запис надає клієнтам ряд переваг, які полегшують процес прийому та поліпшують їх загальний досвід.

Однією з основних переваг сайту з можливістю запису на прийом є зручність для клієнтів. Вони можуть легко зарезервувати прийом у зручний для них час, не потребуючи додаткових зусиль або відвідування клініки особисто. Це забезпечує гнучкість і зручність, особливо для тих, хто має обмежений час або перебуває у віддаленому місці. Клієнти можуть з легкістю вибрати підходящого лікаря, доступну послугу та зручний часовий інтервал просто за декілька кліків.

Крім того, сайт з можливістю онлайн-запису допомагає економити час і зусилля як для клієнтів, так і для клініки. Клієнти можуть швидко зарезервувати

прийом без потреби взаємодіяти з персоналом клініки або очікувати у черзі. Це дозволяє уникнути непотрібних затримок і стресу, пов'язаного з організацією прийому. Для клініки ж це дозволяє автоматизувати процес запису та управління прийомами, забезпечуючи більш ефективне розподілення ресурсів та уникнення конфліктів з графіком лікарів.

Також варто відзначити, що сайт з онлайн-записом на прийом покращує загальний клієнтський досвід. Клієнти отримують більше контролю над процесом прийому та можуть зручно планувати свій час і контролювати свої медичні призначення. Вони можуть вибрати оптимальний для себе час в залежності від своїх особистих потреб і розкладу. Також, завдяки можливості попереднього заповнення медичних форм або вказання особливих потреб, клієнти мають можливість підготуватися до прийому і максимально ефективно використовувати свій час у стоматологічному офісі.

Крім зручності для клієнтів, наявність функції онлайн-запису на прийом сприяє покращенню робочого процесу в стоматологічній клініці. Вона допомагає уникнути перевантаження адміністративного персоналу, зменшує кількість телефонних дзвінків і покращує управління записами на прийоми. Клініка отримує більш точну і структуровану інформацію про прийоми, що сприяє плануванню робочого часу лікарів і оптимальному розподілу ресурсів клініки.

Необхідно також зазначити, що наявність онлайн-запису на прийом є показником сучасності та інноваційності стоматологічної клініки. Вона демонструє прагнення до використання сучасних технологій для поліпшення обслуговування клієнтів і надання їм максимального комфорту. Такий підхід сприяє побудові довгострокових взаємовигідних відносин з клієнтами і створює позитивний імідж стоматологічної клініки.

Отже, сайт з можливістю онлайн-запису на прийом є кращим вибором, оскільки надає зручність, ефективність, більший контроль над власними медичними призначеннями. Клієнти можуть легко зарезервувати прийом у зручний для них час, без необхідності дзвонити або відвідувати клініку особисто. Це особливо важливо для людей з обмеженим часом або тим, хто перебуває у

віддаленому місці. Наявність функції онлайн-запису дозволяє клієнтам контролювати свій графік та легко планувати свої зустрічі з лікарем.

Додатково, сайт з можливістю онлайн-запису на прийом забезпечує економію часу та зусиль як для клієнтів, так і для стоматологічної клініки. Клієнти можуть швидко зарезервувати бажаний час без необхідності очікування або взаємодії з персоналом клініки. Це виключає потребу в телефонних дзвінках або ручному записі, що сприяє ефективнішому використанню часу для обох сторін. Крім того, автоматизований процес запису на прийом дозволяє клініці оптимізувати свою роботу, уникати конфліктів з графіком та ефективно розподіляти свої ресурси.

Окрім зручності та ефективності, наявність онлайн-запису на прийом покращує загальний досвід клієнтів. Вони отримують можливість контролювати свої медичні зустрічі і бути активними учасниками в процесі лікування. Клієнти можуть передати необхідну інформацію перед прийомом, заповнивши медичні форми заздалегідь, що дозволяє лікарю краще підготуватись до зустрічі і забезпечити якісну медичного обслуговування.

Крім того, сайт з можливістю онлайн-запису на прийом може надати додаткову інформацію клієнтам. На веб-сайті можуть бути доступні детальні описи послуг, фото лікарів. Це допомагає клієнтам зробити осмислені рішення щодо свого лікування і мати повне уявлення про послуги, які надає стоматологічна клініка.

Крім того, сайт з онлайн-записом на прийом може покращити комунікацію між клієнтами і клінікою. Клієнти можуть залишати відгуки та оцінки про свій досвід лікування, що допомагає іншим користувачам зробити вірний вибір. Також, клієнти можуть задавати запитання або отримувати консультацію через електронну пошту або онлайн-чат, що забезпечує зручну та оперативну взаємодію.

Загалом, сайт з можливістю онлайн-запису на прийом є кращим варіантом для стоматологічної клініки, оскільки він надає зручність та ефективність клієнтам, економію часу та зусиль, систематизацію процесу для клініки та

покращений клієнтський досвід. Це дозволяє клієнтам контролювати своє здоров'я та лікування, а клініці забезпечити високоякісне обслуговування і задоволення потреб своїх клієнтів.

Додатковою перевагою сайту з можливістю онлайн-запису на прийом є забезпечення конфіденційності та безпеки даних. Клієнти можуть бути впевнені, що їх особиста і медична інформація залишається конфіденційною і не потрапить до неправомірних рук. Захист персональних даних є важливою складовою для будь-якого медичного закладу, і сайт зі вбудованою системою запису на прийом може забезпечити високий рівень безпеки та конфіденційності.

1.3 Постановка задачі

У відповідності до вищенаведеного, для створення стоматологічного веб-сайту з можливістю запису до стоматолога на платформі Spring Boot потрібно виконати наступні кроки:

1. Планування та аналіз: Проведіть аналіз переваг та недоліків існуючих стоматологічних сайтів-аналогів. Визначте, які функції і можливості ви хочете реалізувати на вашому веб-сайті, зокрема функцію запису до стоматолога. Встановіть мову та технології програмування, такі як Java та Spring Boot, як основний фреймворк розробки.
2. Дизайн і інтерфейс: Розробіть сучасний та зручний дизайн веб-сайту. Використовуйте HTML, CSS для створення зручного користувацького інтерфейсу. Плануйте компоненти і сторінки, які дозволять пацієнтам легко обрати лікаря та зареєструватися на прийом.
3. Реалізація бекенду: Використовуйте Spring Boot для розробки бекенду веб-сайту. Створіть контролери та сервіси, які оброблятимуть запити на запис до стоматолога. Забезпечте взаємодію з базою даних для зберігання інформації про лікарів, пацієнтів та розклад прийому.
4. Функціонал запису на прийом: Розробіть функціонал, який дозволяє пацієнтам записуватися на прийом до стоматолога. Включіть можливість вибору

лікаря, дати та часу прийому. Забезпечте валідацію та перевірку доступності дат та часу.

5. Автентифікація та авторизація: Забезпечте механізми автентифікації та авторизації для захисту доступу до функціоналу запису на прийом.

6. Реалізація бази даних: Створіть структуру бази даних для зберігання інформації про стоматологів, пацієнтів, розкладу прийому та інших необхідних даних. Забезпечте взаємодію з базою даних з допомогою Spring Data.

7. Управління розкладом прийому: Розробіть механізм для управління розкладом прийому стоматолога. Дозвольте лікарям встановлювати свою доступність та вільний час, а пацієнтам вибирати зручний для них час з доступних варіантів.

Висновки до розділу 1

У даному розділі було проведено огляд програмного забезпечення, яке використовують стоматологічні клініки. Вивчені були п'ять сайтів стоматологічних клінік, зокрема citysmile.com.ua, mainz.clinic, ukrstom-centr.com.ua, dent-art.com.ua і liliaestet.com.ua.

Порівнявши розглянуті сайти-аналоги, можна зробити висновок, що наявність можливості онлайн-запису на прийом є серйозною перевагою, яка покращує зручність та ефективність для клієнтів.

На основі проведеного огляду та аналізу існуючих стоматологічних сайтів-аналогів виникає постановка завдання для подальшої роботи. Для створення стоматологічного веб-сайту з можливістю запису до стоматолога на платформі Spring Boot

РОЗДІЛ 2. РОЗРОБКА АРХІТЕКТУРИ КЛІЄНТ-СЕРВЕРНОЇ АПЛІКАЦІЇ ДЛЯ РЕЄСТРАЦІЇ ПАЦІЄНТІВ

2.1 Архітектура 3 рівневої веб-аплікації

Для мого сайту я використовував архітектуру 3-рівневої веб-аплікації, яка дозволяє ефективно організувати логіку та функціональність мого проекту. Ця архітектура складається з трьох основних рівнів: рівень представлення, рівень бізнес-логіки та рівень доступу до даних.

Рівень представлення (Presentation Layer): Рівень представлення відповідає за відображення інтерфейсу користувача (UI) мого сайту. Він включає HTML-шаблони, CSS-стилі і JavaScript-код, які спільно створюють користувацький інтерфейс. HTML дозволяє створювати структуру сторінок та елементів інтерфейсу, CSS відповідає за оформлення та стилізацію веб-елементів, а JavaScript-код надає інтерактивність та динамічність веб-сторінок.

Рівень бізнес-логіки (Business Logic Layer): Рівень бізнес-логіки містить основну логіку мого додатку. Він відповідає за обробку операцій, обробку даних та взаємодію з базою даних. На цьому рівні розташовуються сервіси, контролери або менеджери, які координують роботу додатку. Вони приймають запити від рівня представлення, обробляють їх, взаємодіють з базою даних, якщо потрібно, і повертають результати назад до рівня представлення. Це дозволяє розділити логіку додатку від його інтерфейсу. У моєму проекті я створив різноманітні сервіси, що відповідають за конкретні функціональність та логіку додатку. Використання рівня бізнес-логіки дозволило ефективно управляти логікою додатку та забезпечити високу якість функціональності.

Рівень доступу до даних (Data Access Layer): Рівень доступу до даних відповідає за доступ до даних та бази даних. Він включає моделі даних, репозиторії або DAO (Data Access Objects). Цей рівень виконує операції збереження, оновлення, видалення та отримання даних з бази даних. Він

дозволяє абстрагуватись від конкретної бази даних і забезпечує єдиний інтерфейс для роботи з даними. Це дає можливість змінювати або розширювати базу даних без впливу на решту додатку. У моєму проекті я використовував ORM (Object-Relational Mapping) для забезпечення зручного доступу до даних. ORM дозволяє використовувати об'єктно-орієнтовану модель даних та автоматично вирішує завдання мапування об'єктів на таблиці бази даних. На (рис. 2.1) зображено архітектура 3-рівневої веб-аплікації.

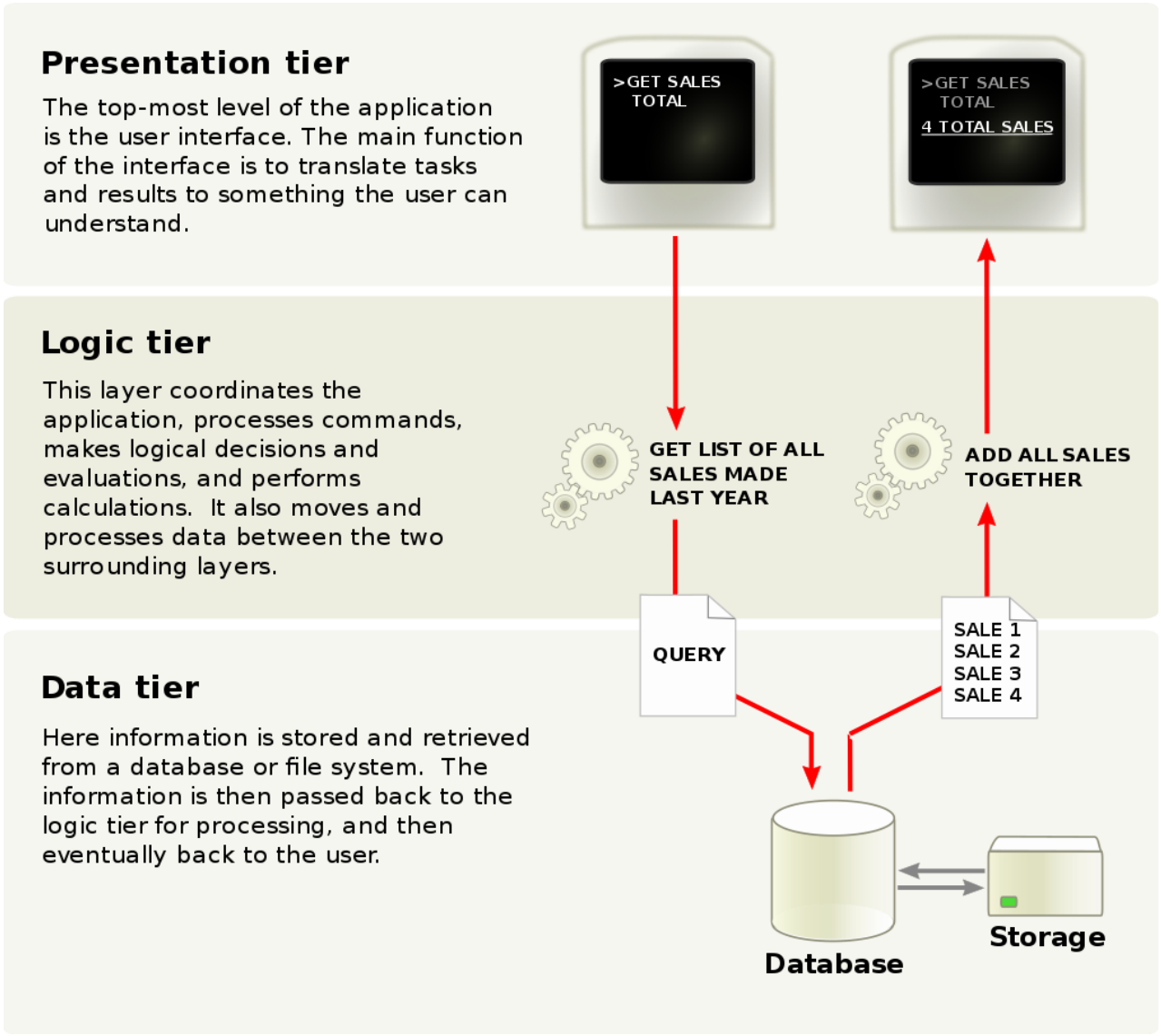

Рисунок 2.1 – Архітектура 3-рівневої веб-аплікації

Завдяки цій архітектурі мій проект отримав структуровану організацію, де кожен рівень має свої відповідальності і може бути розроблений, тестований та

модифікований незалежно від інших рівнів. Крім того, ця архітектура сприяє забезпеченню простоти, масштабованості та повторного використання коду. Розділення на рівні дозволяє команді розробників працювати над окремими компонентами системи без впливу на інші частини проекту. Крім того, це спрощує тестування, оскільки кожен рівень може бути перевірений окремо, що поліпшує стабільність та надійність додатку.

Загалом, використання архітектури 3-рівневої веб-аплікації допомогло мені створити структурований та ефективний проект з гнучкістю та масштабованістю для подальшого розвитку. Ця архітектура дозволила легко розподіляти відповідальності між різними рівнями, поліпшити керованість проектом та забезпечити зручну розробку, тестування та модифікацію окремих компонентів.

2.2 Java

Java є однією з найпопулярніших та найвикористовуваніших мов програмування в світі. Вона була розроблена компанією Sun Microsystems (пізніше придбаною Oracle Corporation) і вперше випущена у 1995 році. Java є об'єктно-орієнтованою мовою програмування, що базується на концепції "Write Once, Run Anywhere" (WORA), що означає, що програми, написані на Java, можуть працювати на будь-якій платформі, яка підтримує віртуальну машину Java (JVM).

Java має багато переваг, які сприяють її популярності та використанню у різних сферах програмування. Одна з найважливіших переваг - це портативність. Будучи мовою, що працює на віртуальній машині, Java дозволяє виконувати програми на будь-якій платформі, на якій є встановлена відповідна версія JVM. Це дозволяє розробникам писати код один раз і запускати його на різних операційних системах, таких як Windows, macOS, Linux та інші. На (рис. 2.2) зображено як працює віртуальна машина java.

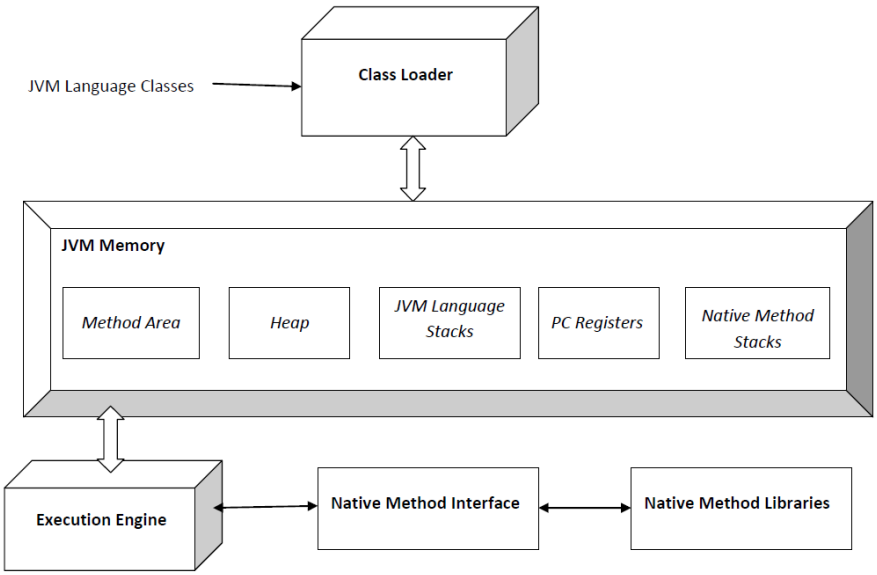

Рисунок 2.2 – Як працює віртуальна машина java

Ще одна важлива риса Java - це безпека. Java має вбудовану систему безпеки, яка дозволяє контролювати виконання програм та запобігати вразливостям і атакам. Вона включає механізми, такі як управління доступом, перевірку типів, обмеження доступу до ресурсів та інші заходи безпеки, що роблять її особливо популярною у веб-розробці та розробці платформи.

Java також відома своєю великою екосистемою та багатими бібліотеками. Існує велика кількість готових бібліотек та фреймворків, які полегшують розробку програм, забезпечують готові рішення для різних завдань і допомагають прискорити процес розробки. Наприклад, фреймворки Spring та Hibernate є одними з найпопулярніших фреймворків Java, які допомагають в розробці підприємницьких застосунків та роботі з базами даних.

Java також активно використовується у сфері мобільних додатків. Завдяки фреймворкам, таким як Android, Java стала однією з основних мов програмування для розробки додатків для платформи Android. Це дозволяє розробникам створювати мобільні додатки з використанням великої кількості готових бібліотек та інструментів, що полегшують процес розробки.

Java також займає лідируючі позиції у розробці вбудованих систем, інтернету речей (IoT), наукових обчислень, фінансових технологій та інших

областях. Вона використовується великими компаніями та установами, такими як Google, Amazon, Oracle, IBM, NASA та інші [2].

Крім того, Java підтримує багатофункціональність та розширюваність завдяки своїй об'єктно-орієнтованій природі. Розробники можуть використовувати спадкування, поліморфізм, інтерфейси, класи та інші концепції ООП для створення складних та гнучких програм. Це полегшує розподіл завдань, розвиток та підтримку коду. На (рис. 2.3) зображено структура ООП.

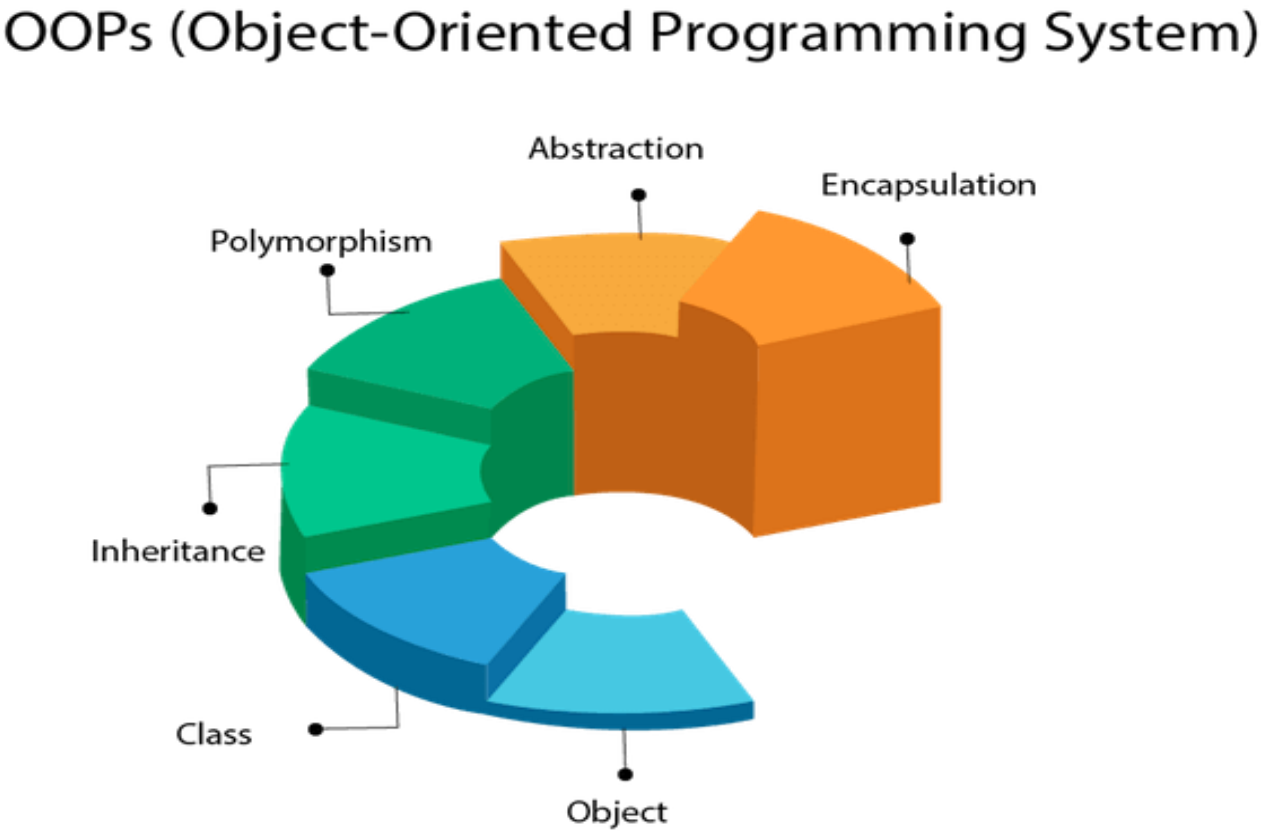

Рисунок 2.3 – Структура ООП

Узагальнюючи, Java є потужною та універсальною мовою програмування, яка використовується в різних сферах, від мобільних додатків та веб-розробки до вбудованих систем та наукових обчислень. Її портативність, безпека, розширюваність та велика екосистема роблять її популярним вибором для розробників у всьому світі [5].

2.3 Maven

Maven є потужним інструментом для управління проектами на основі Java, який допомагає впорядковувати, збирати та керувати залежностями проекту. Він забезпечує структуровану організацію проекту, автоматизоване управління залежностями, компіляцію, тестування, пакування та розгортання. Заснований на концепції конвенцій над конфігурацією (Convention over Configuration), Maven дозволяє розробникам фокусуватись на логіці додатку, а не на складних процесах налаштування та управління.

Однією з головних переваг Maven є його здатність до ефективного керування залежностями. Залежності - це бібліотеки, фреймворки або інші модулі, які використовуються у проекті. Maven дозволяє визначити залежності у файлі конфігурації проекту (pom.xml) та автоматично завантажує та встановлює їх з відповідних репозиторіїв. Це спрощує процес управління залежностями, забезпечує їх консистентність та дозволяє легко оновлювати до нових версій.

Ще одна важлива функція Maven - це здатність автоматично збирати, тестувати та пакувати проект. За допомогою команд Maven, таких як mvn compile, mvn test, mvn package, можна автоматично виконувати ці процеси. Maven дотримується конвенцій щодо структури каталогів та найменування файлів проекту, що дозволяє розробникам швидко орієнтуватись у проекті та використовувати стандартні налаштування.

Крім того, Maven надає можливість створювати та використовувати плагіни. Плагіни - це розширення Maven, які надають додаткову функціональність, наприклад, створення документації, виконання аналізу коду, генерація звітів тощо. За допомогою плагінів розробники можуть розширювати можливості Maven та адаптувати його під свої потреби.

Maven також сприяє спільноті розробників та спільному використанню коду. Він інтегрується з репозиторіями, такими як Maven Central, які містять велику кількість відкритих бібліотек та компонентів, доступних для використання. Розробники можуть легко шукати, завантажувати та

використовувати ці бібліотеки у своїх проектах. Крім того, Maven дозволяє легко публікувати власні бібліотеки в репозиторіях, що сприяє спільному використанню коду та співпраці між розробниками.

Однак, хоча Maven має багато переваг, він також має певні обмеження. Наприклад, Maven підходить найкраще для проектів на основі Java і не є найкращим вибором для проектів на інших мовах програмування. Крім того, конфігурація Maven може бути складною для початківців, особливо якщо проект вимагає специфічних налаштувань.

Узагальнюючи, Maven є потужним інструментом для управління проектами на основі Java. Він спрощує управління залежностями, забезпечує автоматизовану збірку та тестування проекту, допомагає використовувати плагіни та сприяє спільноті розробників. Його використання дозволяє покращити структуру проекту, забезпечити його стабільність та зручність у розробці [4].

2.4 Spring Boot

Spring Boot - це фреймворк, розроблений для спрощення створення, налаштування та розгортання додатків на основі Spring Framework. Він надає розширений набір інструментів і конфігураційних параметрів, що допомагають розробникам швидко розпочати роботу над своїми проектами.

Spring Boot прагне забезпечити конвенцію над конфігурацією, що означає, що розробнику не потрібно витрачати багато часу на ручну настройку. Він надає стандартні значення параметрів, які можна змінити за потреби. Це дозволяє розробникам швидше розпочати роботу над функціональністю свого додатку, замість того, щоб зосереджуватися на складних конфігураційних завданнях.

Одна з ключових особливостей Spring Boot - це автоматична конфігурація. Він здатний автоматично визначати, налаштовувати та підключати необхідні бібліотеки, сервіси та інші компоненти, що потрібні для роботи додатку. Це

зменшує необхідність у вручну налаштовуванні і допомагає знизити кількість коду, який потрібно написати.

Spring Boot також надає вбудовану підтримку для розгортання додатків. Він має вбудований веб-сервер (за замовчуванням - Tomcat), що дозволяє запускати додаток без необхідності налаштовувати окремий сервер. Це робить процес розгортання простішим і зменшує залежність від зовнішніх серверів. На (рис. 2.4) зображено технічна архітектура Spring Boot, Tomcat.

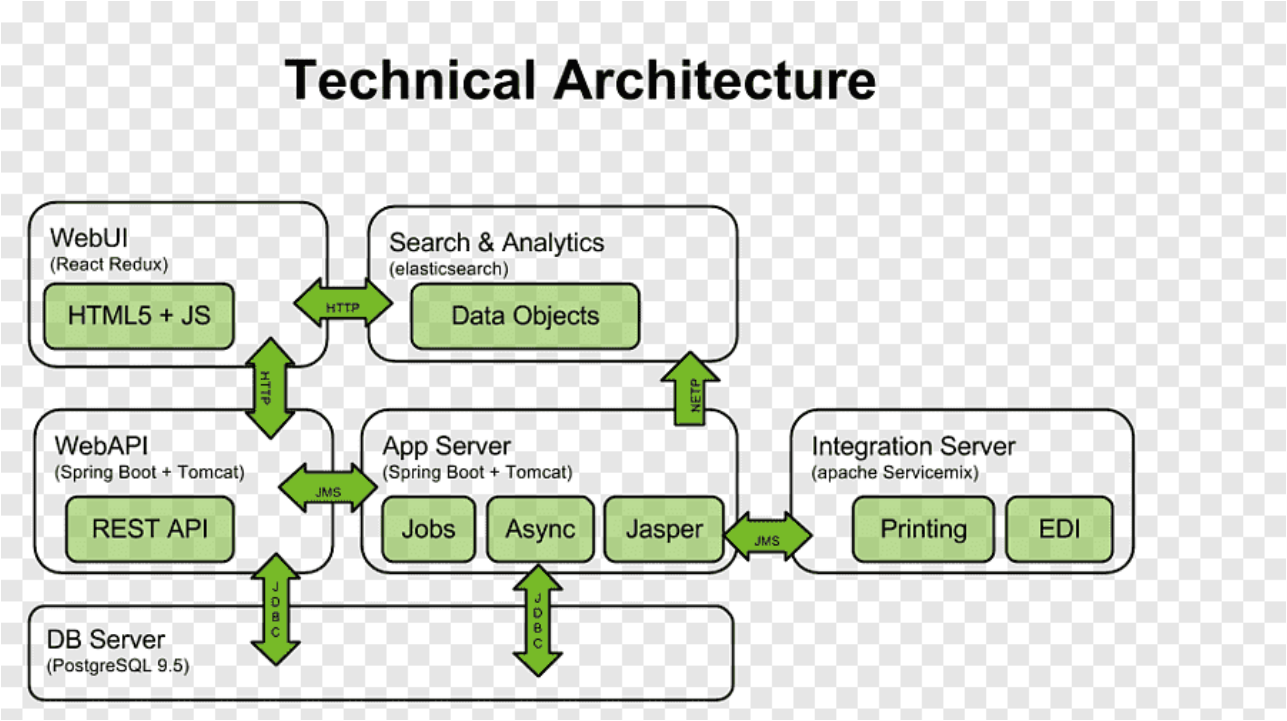

Рисунок 2.4 – Технічна архітектура Spring Boot, Tomcat

Spring Boot також надає багато інших функціональних можливостей, таких як вбудована підтримка для кешування, обробки запитів REST, роботи з базами даних та багато іншого. Він також має велику активну спільноту розробників, яка надає багато ресурсів, документацію та приклади коду для полегшення роботи з фреймворком.

Узагалі, Spring Boot є потужним інструментом для швидкого розробки додатків на основі Spring Framework. Він дозволяє розробникам зосередитися на Бізнес логіки своїх додатків, забезпечуючи швидкий старт і спрощення

конфігурації. Якщо ви шукаєте зручний спосіб розробки Java-додатків, Spring Boot може бути чудовим вибором.

Spring Boot має ще кілька важливих особливостей, які роблять його популярним серед розробників:

1. Уніфікована конфігурація: Spring Boot пропонує спрощену конфігурацію, що дозволяє вам налаштувати ваш додаток за допомогою файла application.properties або application.yml. Ви можете визначати параметри, такі як порт, шляхи до бази даних, логування та багато іншого, без необхідності писати багато коду.

2. Управління залежностями: Spring Boot пропонує удосконалений механізм управління залежностями за допомогою Maven або Gradle. Ви можете легко вказати необхідні бібліотеки та їх версії у вашому конфігураційному файлі, і Spring Boot автоматично завантажить їх та вирішить всі необхідні залежності.

3. Вбудовані сервери: Spring Boot надає підтримку вбудованих серверів, таких як Tomcat, Jetty або Undertow. Це означає, що ви можете легко запускати свій додаток без необхідності налаштовувати окремий сервер. Все, що вам потрібно, це виконати відповідну команду, і ваш додаток буде запущений.

4. Автоматична конфігурація: Spring Boot надає автоматичну конфігурацію, що дозволяє автоматично визначати та налаштовувати багато компонентів в базовому конфігураційному контексті. Наприклад, якщо ви використовуєте базу даних, Spring Boot може автоматично налаштувати підключення до бази даних на основі конфігураційних параметрів.

5. Підтримка моніторингу та управління: Spring Boot надає актовність Actuator, яка дозволяє вам моніторити та управляти вашим додатком в реальному часі. За допомогою Actuator ви можете отримати доступ до різноманітної інформації про стан додатку, такої як здоров'я, метрики, журнали, налаштування та іншого.

6. Вбудовані стандарти: Spring Boot використовує вбудовані стандарти та рекомендації для розробки додатків. Він надає консистентний підхід до структури проекту, що полегшує розуміння та підтримку коду. Крім того, Spring

Boot сприяє дотриманню кращих практик у розробці, забезпечуючи стандартизований підхід до роботи з архітектурними шаблонами та патернами проектування.

7. Велика екосистема: Spring Boot базується на Spring Framework, який має велику та активну спільноту розробників. Це означає, що ви можете легко знайти документацію, підручники, приклади коду та підтримку від спільноти. Багато сторонніх бібліотек та інструментів також підтримують Spring Boot, що дозволяє розширювати його функціональність та забезпечувати більше можливостей для вашого проекту.

8. Легкість тестування: Spring Boot надає зручність у тестуванні вашого додатку. Він має вбудовану підтримку для модульного тестування, інтеграційного тестування та тестування з використанням угод. Ви можете легко налаштовувати та виконувати різні види тестів для перевірки працездатності вашого додатку.

9. Легкість використання з іншими технологіями: Spring Boot не обмежує вас в виборі технологій та фреймворків. Ви можете легко інтегрувати Spring Boot з іншими рішеннями, такими як бази даних (наприклад, MySQL, PostgreSQL), шаблонізатори (наприклад, Thymeleaf, FreeMarker), системи кешування, інструменти безпеки та багато іншого. Це дозволяє вам побудувати потужний стек технологій для вашого додатку.

10. Моніторинг та логування: Spring Boot надає вбудовану підтримку для моніторингу та логування. Ви можете налаштувати різні рівні журналування, включаючи виведення в консоль, запис у файл або навіть інтеграцію з системами журналування, такими як Logback або Log4j. Ви також можете використовувати Actuator для отримання важливої інформації про стан додатку та метрики продуктивності.

Загалом, Spring Boot є потужним фреймворком для розробки додатків на Java, який надає зручність, продуктивність та розширюваність. Він допомагає вам прискорити процес розробки та розгортання, забезпечуючи гнучкість у виборі технологій та простоту використання. Незалежно від розміру вашого

проекту, Spring Boot може стати відмінним вибором для розробки вашого додатку [6].

2.4.1 Spring Data JPA

Spring Data JPA - це потужний модуль, наданий Spring Framework, який дозволяє розробникам легко та ефективно взаємодіяти з базами даних за допомогою Java Persistence API (JPA). Цей модуль надає високорівневий інтерфейс для зручної роботи з базами даних, що веде до скорочення кількості необхідного коду для виконання операцій з даними та спрощення розробки персистентного шару додатка.

Spring Data JPA реалізує підхід Repository, який дозволяє визначати інтерфейси репозиторіїв з методами доступу до даних. Ці методи автоматично трансформуються в SQL-запити та виконуються в контексті бази даних. Репозиторії дозволяють виконувати різноманітні операції з даними, такі як збереження, видалення, оновлення та пошук, використовуючи стандартні методи або власні запити.

Окрім того, Spring Data JPA надає можливість використовувати специфікації (Specifications) для динамічної генерації запитів. Завдяки специфікаціям можна будувати складні запити, що залежать від критеріїв, визначених в runtime. Це забезпечує більшу гнучкість при формулюванні запитів та спрощує розробку складних функціональностей.

Spring Data JPA автоматично генерує SQL-запити на основі іменованих методів репозиторіїв або анотованих запитів. Це дозволяє уникнути необхідності у власному написанні SQL-запитів, спрощує розробку та дозволяє зосередитись на бізнес-логіці додатка.

Крім того, Spring Data JPA підтримує кешування результатів запитів, що допомагає покращити продуктивність додатка шляхом зменшення кількості запитів до бази даних.

Загалом, Spring Data JPA є потужним інструментом для спрощення роботи з базами даних в Spring-додатках. Він забезпечує швидкий та зручний доступ до даних, спрощує розробку та полегшує підтримку персистентного шару додатка. З використанням Spring Data JPA розробники можуть більш ефективно працювати з базами даних та швидше розробляти функціональність, пов'язану з персистентним шаром своїх додатків [7].

2.4.2 Spring MVC

Spring MVC (Model-View-Controller) - це веб-фреймворк, що надає розробникам потужні інструменти для створення веб-додатків на основі патерна MVC. Цей фреймворк входить до складу Spring Framework і забезпечує ефективну організацію, розділення обов'язків та гнучкість при розробці веб-додатків.

Spring MVC базується на патерні Model-View-Controller, що дозволяє розділити логіку додатка на три компоненти: модель (Model), представлення (View) та контролер (Controller). Це забезпечує модульність, легкість тестування та розширення додатка.

Основні компоненти Spring MVC:

1. Контролер (Controller): Контролер відповідає за обробку вхідних запитів від користувача. Він приймає запит, взаємодіє з моделлю для отримання необхідних даних та відправляє відповідь користувачу через представлення. Контролер виконує бізнес-логіку та керує потоком додатка.
2. Модель (Model): Модель представляє дані, з якими працює додаток. Вона може включати об'єкти, які відображають структуру бази даних або інші бізнес-об'єкти. Модель забезпечує доступ до даних та методів для їх обробки.
3. Представлення (View): Представлення відповідає за відображення даних користувачу. Воно генерує HTML-сторінки, які відправляються до браузера. Представлення може використовувати шаблони для генерації сторінок та включати дані з моделі.

4. Диспетчер (Dispatcher): Диспетчер (Dispatcher) відповідає за маршрутизацію запитів та керування потоком додатка. Він приймає вхідний запит, визначає, який контролер має обробити запит, та передає йому управління. Диспетчер також може виконувати певні обробки перед та після обробки запиту.

5. Резольвери представлень (View Resolvers): Резольвери представлень відповідають за знаходження і вибір правильного представлення для відображення даних. Вони дозволяють розробникам використовувати різні формати представлень, такі як JSP, Thymeleaf, HTML і т.д.

6. Помічники контролера (Controller Helpers): Помічники контролера надають додаткові функції та допоміжні методи для контролерів. Вони спрощують обробку даних та забезпечують додаткові можливості для контролерів.

Spring MVC надає розробникам багато інших функціональних можливостей, таких як валідація даних, обробка винятків, інтернаціоналізація, безпека, кешування та інтеграція з іншими модулями Spring.

Переваги використання Spring MVC:

1. Легкість розробки: Spring MVC надає простий та зрозумілий спосіб розробки веб-додатків з використанням патерна MVC. Він пропонує конфігурацію через анотації або XML, що полегшує розгортання та налаштування додатків.

2. Гнучкість: Spring MVC надає гнучкість у виборі технологій для представлень та бази даних. Розробники можуть використовувати різні шаблони представлень, такі як JSP, Thymeleaf або FreeMarker, залежно від своїх потреб. Вони також можуть використовувати будь-яку базу даних, яку підтримує Spring.

3. Тестування: Spring MVC добре підходить для тестування, оскільки він використовує розділення логіки на модулі. Розробники можуть легко тестувати контролери, моделі та представлення окремо один від одного.

4. Інтеграція з іншими модулями Spring: Spring MVC легко інтегрується з іншими модулями Spring, такими як Spring Security для забезпечення безпеки, Spring Data для роботи з базою даних та іншими.

Загалом, Spring MVC є потужним фреймворком для розробки веб-додатків, який забезпечує ефективну організацію коду, легкість розробки та гнучкість у виборі технологій. Він є одним із популярних веб-фреймворків для розробки додатків на платформі Java [8].

2.5 Розробка структури таблиць бази даних

MySQL - це популярна система управління базами даних (СУБД), яка має багато переваг, що роблять її популярною серед розробників та організацій. На (рис. 2.5) зображено діаграма як працює MySQL з java.

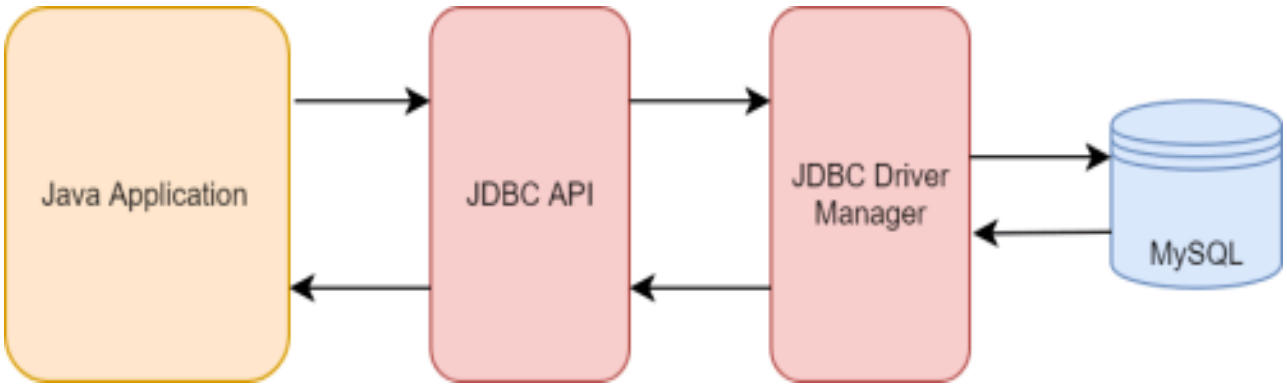

Рисунок 2.5 – Діаграма як працює MySQL з java

Ось кілька головних переваг MySQL:

1. Висока продуктивність: MySQL відомий своєю високою швидкодією та продуктивністю. Вона може ефективно обробляти великі обсяги даних і забезпечує швидкий доступ до інформації.

2. Масштабованість: MySQL підтримує горизонтальну та вертикальну масштабованість. Горизонтальна масштабованість дозволяє розширювати базу даних на кілька серверів, що забезпечує більше потужності та обробляє велику кількість запитів. Вертикальна масштабованість дозволяє збільшувати потужність сервера шляхом оновлення обладнання.

3. Гнучкість: MySQL підтримує багато різних типів даних, включаючи текст, числа, дати, зображення і багато інших. Вона також підтримує різні мови

програмування, такі як PHP, Java, Python тощо, що робить її гнучкою для використання в різних типах проектів.

4. Надійність та стабільність: MySQL володіє довідками в області надійності і стабільності. Вона має широкі можливості резервного копіювання і відновлення даних, а також механізми виявлення та виправлення помилок. Вона також забезпечує високий рівень безпеки та захисту даних.

5. Простота використання: MySQL має простий та зрозумілий синтаксис запитів, що робить його легким у використанні. Розробники швидко освоюють основи MySQL і можуть ефективно створювати запити та операції з базою даних.

6. Відкритість та безкоштовність: MySQL є відкритою системою з відкритим вихідним кодом, що дає можливість користувачам змінювати та розширювати функціонал за їхніми потребами. Крім того, MySQL є безкоштовним, що робить його доступним для використання в багатьох проектах без додаткових витрат на ліцензії.

7. Підтримка транзакцій: MySQL підтримує транзакції, що забезпечує консистентність бази даних. Це означає, що якщо транзакція не вдалась, то всі зміни, які були зроблені в базі даних під час транзакції, будуть відкатані, щоб забезпечити цілісність даних.

8. Розширені можливості реплікації: MySQL має розширені можливості реплікації, що дозволяють створювати декілька копій бази даних, які можуть працювати на різних серверах. Це забезпечує збереження резервних копій та зменшує вплив навантаження на один сервер.

Узагалі, MySQL є потужною та гнучкою системою управління базами даних, що має багато переваг та може бути використана для різних проектів, від невеликих веб-сайтів до великих корпоративних додатків. На (рис. 2.6) зображено зображений моя діаграму зв'язків у базі даних [3].

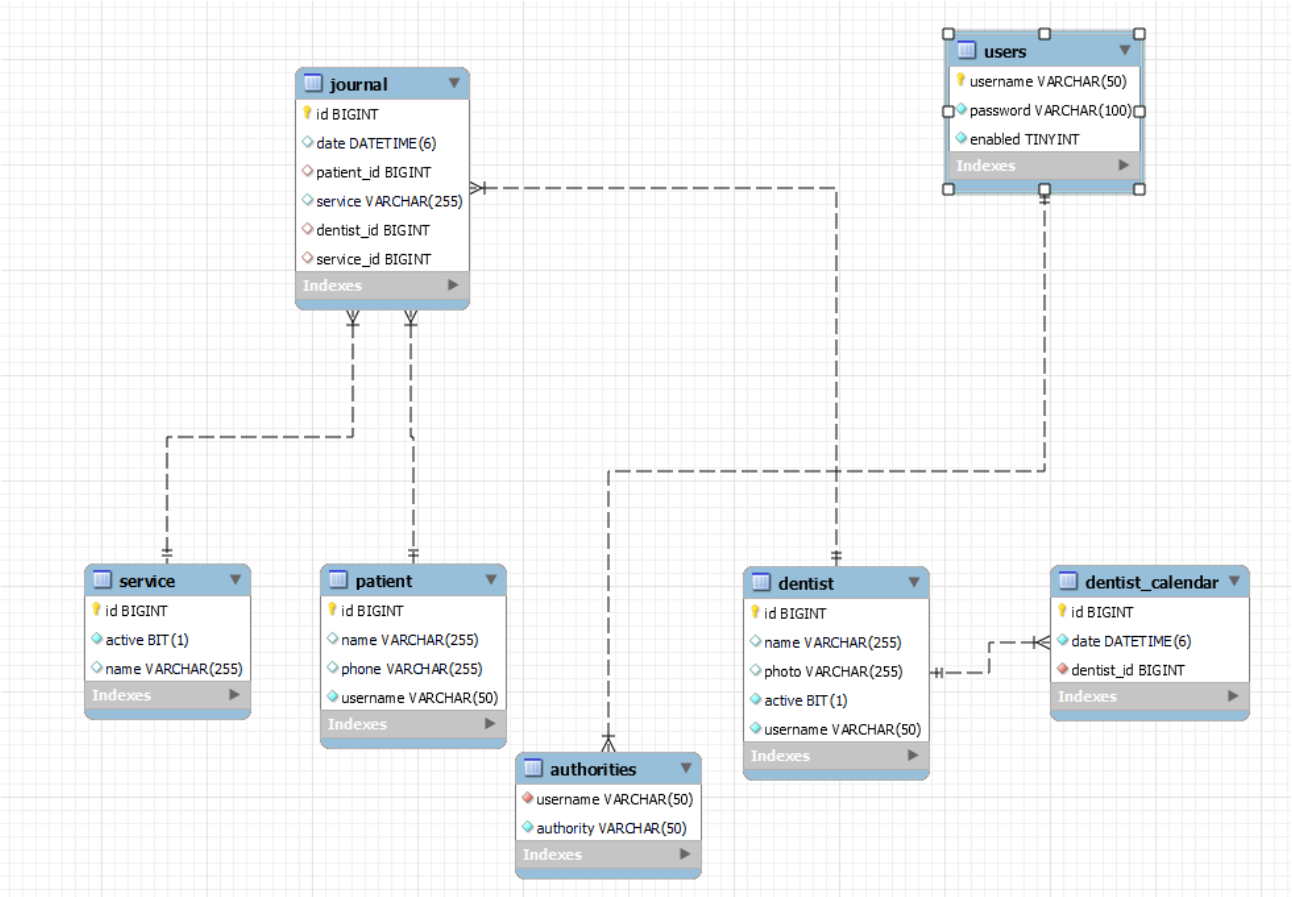

Рисунок 2.6 – Зображений моя діаграму зв'язків у базі даних

Це опис моєї бази даних, яка містить кілька таблиць із відповідними структурами і зв'язками між ними.

Таблиця users використовується для зберігання даних про користувачів. Вона містить поля username (логін користувача), password (пароль користувача) і enabled (ознака активності користувача). Поле username є первинним ключем цієї таблиці.

Таблиця authorities використовується для зберігання даних про повноваження користувачів. Вона містить поля username (логін користувача) і authority (повноваження користувача). Ця таблиця має зовнішній ключ username, який посилається на поле username в таблиці users.

Таблиця dentist містить інформацію про стоматологів. Вона має поля id (ідентифікатор стоматолога), name (ім'я стоматолога), photo (шлях до фотографії стоматолога), active (ознака активності стоматолога) і username (логін стоматолога). Поле id є первинним ключем цієї таблиці.

Таблиця service містить інформацію про послуги стоматології. Вона має поля id (ідентифікатор послуги), active (ознака активності послуги) і name (назва послуги). Поле id є первинним ключем цієї таблиці.

Таблиця patient містить дані про пацієнтів. Вона має поля id (ідентифікатор пацієнта), name (ім'я пацієнта), phone (номер телефону пацієнта) і username (логін пацієнта). Поле id є первинним ключем цієї таблиці.

Таблиця journal використовується для зберігання записів в журналі. Вона має поля id (ідентифікатор запису), date (дата запису), patient_id (ідентифікатор пацієнта, до якого стосується запис), dentist_id (ідентифікатор стоматолога, який здійснив запис), service_id (ідентифікатор послуги, наданої пацієнту). Поле id є первинним ключем цієї таблиці. Ця таблиця також має зовнішні ключі patient_id, dentist_id і service_id, які посилаються відповідно на поля id таблиць patient, dentist і service.

Таблиця dentistCalendar використовується для зберігання календарних дат стоматологів. Вона має поля id (ідентифікатор запису календаря), date (дата в календарі) і dentist_id (ідентифікатор стоматолога, до якого відноситься дата). Поле id є первинним ключем цієї таблиці, а поле dentist_id є зовнішнім ключем, який посилається на поле id в таблиці dentist.

2.5.1 Entity

У Spring Boot анотація @Entity використовується для позначення класів, які представляють сутності бази даних. Ця анотація вказує, що клас є сутністю, яка може бути збережена в базі даних. При використанні анотація @Entity клас стає персистентним, тобто об'єкти цього класу можуть бути збережені в базі даних або отримані з неї.

Ось кілька ключових понять, пов'язаних з анотацією @Entity:

1. Таблиця бази даних: Клас, позначений анотацією @Entity, відповідає таблиці бази даних. Кожен об'єкт цього класу представляє рядок або запис в таблиці.

2. Поля: Клас, який містить анотацію @Entity, повинен мати поля, які відображають колонки в таблиці бази даних. Наприклад, якщо у вас є клас User з полями id, name та email, то ці поля будуть відображені як колонки в таблиці бази даних.

3. Ключове поле: Зазвичай в кожній таблиці бази даних є одне або декілька полів, які унікально ідентифікують кожен рядок. Це називається ключовим полем. За замовчуванням, якщо у класу, позначеного анотацією @Entity, є поле з назвою "id", то воно вважається ключовим полем. Ви також можете вказати інше поле як ключове, використовуючи анотацію @Id.

4. Зв'язки між сутностями: Ви можете встановлювати зв'язки між різними сутностями бази даних, використовуючи анотації, такі як @OneToOne, @OneToMany, @ManyToOne та @ManyToMany. Ці анотації дозволяють визначати відношення між об'єктами із різних сутностей.

5. Генерація таблиць: При запуску додатка Spring Boot автоматично створює таблиці бази даних, використовуючи інформацію з анотацій @Entity. Ви можете використовувати різні режими генерації таблиць, наприклад, автоматично оновлювати схему бази даних або створювати її заново при кожному запуску.

6. CRUD операції: Після позначення класу анотацією @Entity і налаштування бази даних, ви можете виконувати операції створення, читання, оновлення та видалення (CRUD) з об'єктами цього класу. Spring Boot надає можливість автоматично генерувати SQL-запити для цих операцій на основі анотацій @Entity.

Загалом, анотація @Entity дозволяє використовувати об'єктно-реляційний відображувач (ORM) для спрощення роботи з базою даних у вашому додатку Spring Boot. Вона дозволяє вам працювати з об'єктами, а не з SQL-запитами безпосередньо, що полегшує розробку та підтримку вашого додатку [6 - 7].

2.6 Авторизація та аутентифікація за допомогою Spring Security

Spring Security - це потужна бібліотека, яка надає рішення для аутентифікації, авторизації та управління безпекою в додатках на основі Spring. Вона працює на основі принципів інверсії керування та аспектно-орієнтованого програмування, що робить її потужним і гнучким інструментом для забезпечення безпеки вашого додатка. На (рис. 2.7) зображено діаграма як працює Spring Security.

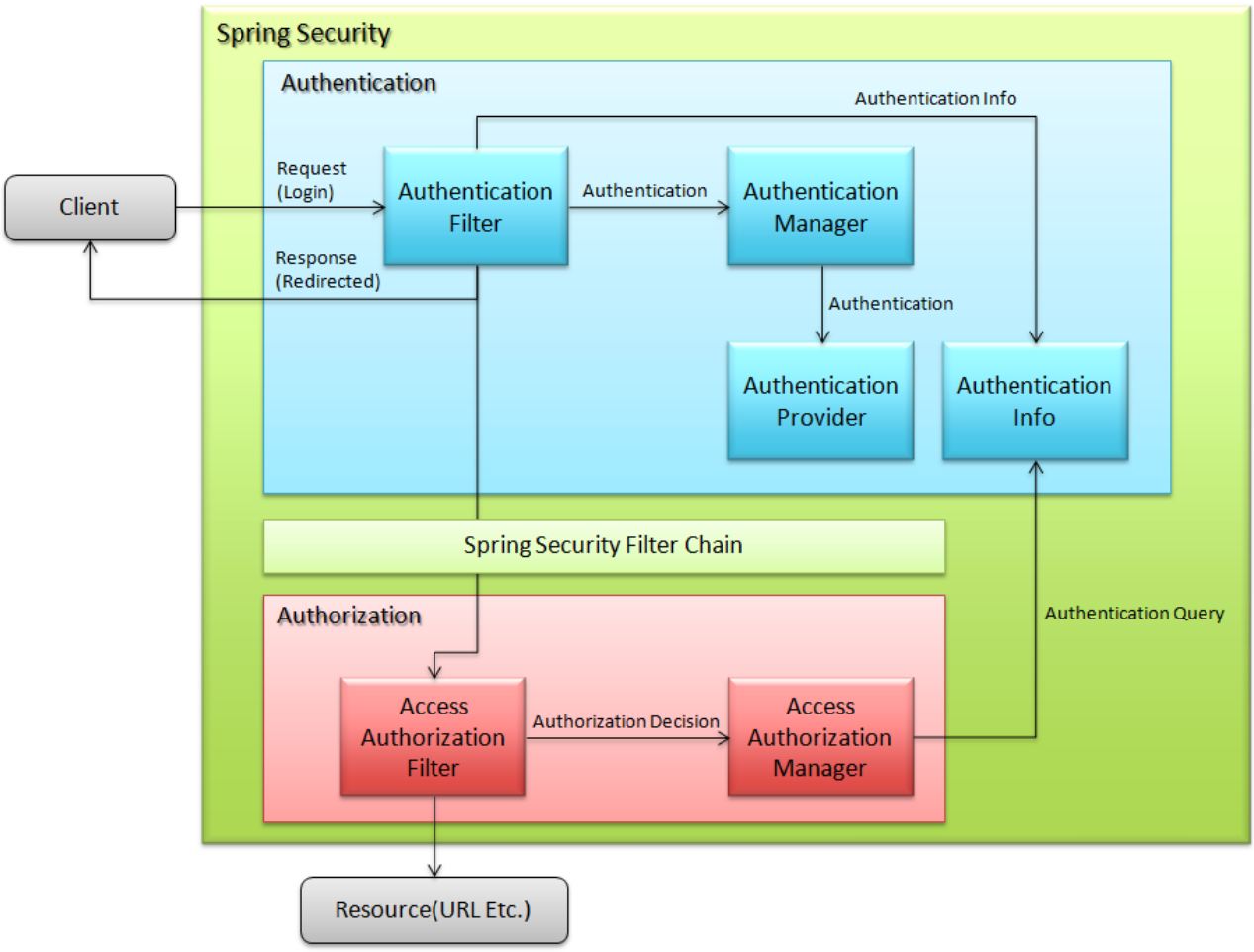

Рисунок 2.7 – Діаграма як працює Spring Security

Spring Security дозволяє вам легко захистити ваші ресурси, встановлюючи правила доступу до них. Він підтримує різні механізми аутентифікації, такі як форма входу, базова аутентифікація, аутентифікація з використанням сторонніх

постачальників (наприклад, OAuth), JWT (JSON Web Tokens) та багато інших. Ви можете вибрати підходящий механізм залежно від потреб вашого додатка.

Окрім аутентифікації, Spring Security забезпечує механізми авторизації, що дозволяють вам контролювати, які ресурси доступні різним користувачам. Ви можете встановити правила на основі ролей, дозволяючи або обмежуючи доступ до конкретних функцій та ресурсів в вашому додатку.

Одна з головних переваг використання Spring Security полягає в його інтеграції з іншими функціональними модулями Spring, такими як Spring MVC та Spring Boot. Ви можете легко інтегрувати Spring Security у вашу програму, додавши декілька залежностей та налаштувавши необхідні компоненти.

Ось деякі переваги вибору Spring Security для забезпечення безпеки:

1. Зручність використання: Spring Security надає простий та зрозумілий API, який спрощує розробку і розгортання механізмів безпеки. Ви можете легко налаштувати правила доступу, настроїти механізми аутентифікації та авторизації, а також реалізувати власні механізми захисту.

2. Гнучкість та розширюваність: Spring Security забезпечує гнучкість у виборі різних стратегій безпеки та можливість розширення його функціональності. Ви можете використовувати готові компоненти, такі як фільтри безпеки, або створювати власні реалізації, що дозволяє вам точно налаштувати безпеку відповідно до ваших потреб.

3. Інтеграція з іншими модулями Spring: Spring Security має гарну інтеграцію з іншими модулями Spring, що дозволяє вам легко поєднувати безпеку з іншими аспектами вашого додатка. Наприклад, ви можете комбінувати Spring Security з Spring MVC для захисту веб-сторінок або з Spring Data для безпечного доступу до бази даних.

4. Активна спільнота та підтримка: Spring Security має велику та активну спільноту розробників, що забезпечує високу якість документації, регулярні оновлення та підтримку. Ви можете легко знайти відповіді на свої запитання, приклади коду та поради від інших розробників.

5. Безпека з нуля: Spring Security надає повний спектр можливостей для забезпечення безпеки від самого початку вашого проекту. Ви можете додавати аутентифікацію та авторизацію з самого початку, або легко інтегрувати Spring Security у вже існуючий проект.

Вибір Spring Security для додатка дозволить забезпечити надійний та гнучкий механізм безпеки, захистити ваші ресурси та забезпечити конфіденційність інформації. Це популярний і потужний інструмент, який багато розробників вибирають для забезпечення безпеки своїх додатків на основі Spring.

Для свого сайту я релізував Spring Security таким чином:

Для початку, я створив клас WebSecurityConfig, який був відмічений анотаціями @Configuration та @EnableWebSecurity. Ці анотації дозволяють Spring Framework розпізнати цей клас як конфігураційний клас для Spring Security.

У цьому класі я використовував @Autowired для впровадження залежності DataSource, яка була необхідна для налаштування механізму аутентифікації. Я використовував dataSource для налаштування AuthenticationManagerBuilder, який відповідає за налаштування механізму аутентифікації в Spring Security.

Далі, я створив метод securityFilterChain, який був позначений анотацією @Bean. Цей метод налаштовує механізм фільтрації безпеки (SecurityFilterChain) для обробки HTTP-запитів. У тілі цього методу я використовував методи ланцюжком, що дозволяють встановити різні правила доступу до ресурсів. Наприклад, я використовував authorizeHttpRequests, щоб вказати, які запити повинні бути дозволені або обмежені. Також я налаштував механізм аутентифікації форми, вказавши сторінку входу, сторінку успішного входу та інші налаштування.

Для шифрування паролів я створив метод passwordEncoder, який повертає екземпляр BCryptPasswordEncoder. Це дозволяє захищати паролі, що зберігаються в базі даних, за допомогою алгоритму BCrypt.

З використанням анотації @Autowired я автоматично впроваджую залежність DataSource, яка є об'єктом, що забезпечує доступ до джерела даних. Це може бути база даних або інше зовнішнє джерело, яке містить інформацію про користувачів та їх облікові записи.

Метод configureGlobal використовує AuthenticationManagerBuilder для налаштування механізму аутентифікації. Виклик auth.jdbcAuthentication() вказує на використання JDBC (Java Database Connectivity) для аутентифікації, що дозволяє використовувати базу даних для перевірки ідентифікаційних даних користувачів. Вказується dataSource для встановлення з'єднання з базою даних.

Метод securityFilterChain налаштовує механізм фільтрації безпеки (SecurityFilterChain) за допомогою об'єкта HttpSecurity, який відповідає за конфігурацію правил безпеки для HTTP-запитів. В тілі методу виконується ланцюжок методів для встановлення різних правил доступу. Наприклад, метод authorizeHttpRequests вказує, які запити повинні бути дозволені або обмежені. Метод permitAll() дозволяє всім користувачам доступ до певних ресурсів, тоді як authenticated() обмежує доступ до інших ресурсів тільки аутентифікованим користувачам.

Крім того, метод formLogin() налаштовує механізм аутентифікації форми, вказуючи сторінку входу (loginPage()) та сторінку успішного входу (defaultSuccessUrl()). Метод logout() налаштовує функціонал виходу з системи, включаючи видалення інформації про аутентифікацію, видалення куків (deleteCookies()) та інвалідацію сеансу (invalidateHttpSession()).

Метод passwordEncoder() повертає екземпляр BCryptPasswordEncoder, який використовує алгоритм BCrypt для шифрування паролів користувачів. Це дозволяє забезпечити безпеку паролів, збережених у базі даних, шляхом хешування та соління.

Загалом, цей код конфігурує Spring Security для мого сайту, встановлює механізми аутентифікації та авторизації, налаштовує правила доступу до ресурсів і забезпечує шифрування паролів. Це дозволяє забезпечити надійну безпеку мого сайту та захистити конфіденційну інформацію користувачів.

Усі ці налаштування допомогли мені забезпечити безпеку мого сайту за допомогою Spring Security. Використання анотацій, залежностей та методів з ланцюжком дозволили мінімізувати кількість коду та легко налаштувати механізми аутентифікації та авторизації [10].

Висновки до розділу 2

У цьому розділі описано розробку архітектури клієнт-серверної аплікації для реєстрації пацієнтів. Для досягнення цієї мети були використані наступні технології і інструменти. Перш за все, була використана 3-рівнева веб-архітектура, яка дозволяє ефективно організувати логіку та функціональність проекту. Мова програмування Java була вибрана як одна з найпопулярніших та широко використовуваних мов програмування. Інструмент Maven допомагав управляти проектом та його залежностями. Фреймворк Spring Boot спрощував створення, налаштування та розгортання додатків, забезпечуючи розширений набір інструментів. Для взаємодії з базами даних за допомогою Java Persistence API використовувався модуль Spring Data JPA. Spring MVC надавав потужні інструменти для створення веб-додатків на основі патерна MVC. В якості системи управління базами даних була обрана MySQL.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СТРУКТУРИ ТА ФУНКЦІОНАЛУ ВЕБ-АПЛІКАЦІЇ

3.1 Розробка інтерфейсу сайту

Дизайн та інтерфейс грають важливу роль у проекті, особливо в контексті взаємодії користувача з системою. Дизайн має впливати на те, як людина сприймає та взаємодіє з машинами. Чим естетичніший, приємніший та простіший дизайн, тим більше користувачів будуть зацікавлені використовувати проект, що призведе до зростання аудиторії. Відповідно, розробникам дизайну необхідно приділяти велику увагу створенню інтерфейсу. Правильно спроектований дизайн також значно полегшить завдання верстальникам.

У графічних системах інтерфейсу користувача здійснюється використання багатовіконного режиму, змін кольору, розміру та видимості вікон (включаючи прозорість, напівпрозорість та невидимість), їхнє розташування та сортування елементів. Також передбачені гнучкі налаштування як для вікон, так і для окремих їх елементів, таких як файли, теки, ярлики, шрифти тощо. Це також включає доступність багатокористувацьких налаштувань, що дозволяють користувачам настроювати систему під свої потреби.

Дизайн та інтерфейс графічних систем є ключовими елементами, які сприяють зручності та задоволенню користувачів під час використання проекту. Вони створюють першу імпресію та впливають на загальний досвід взаємодії. Тому необхідно приділяти належну увагу розробці цих аспектів, забезпечуючи якість та привабливість для широкої аудиторії.

На (рис. 3.1) зображено зовнішній вигляд головної сторінки, яка є ключовим елементом проекту. Цей знімок екрана демонструє, яким чином була реалізована структура та вигляд сторінки. Деталі та елементи інтерфейсу на цьому зображенні надають уявлення про те, як користувачі взаємодіятимуть з проектом.

Додатково, після зображення головної сторінки, наведений фрагмент коду, який відповідає за створення цієї сторінки. Цей фрагмент коду демонструє технічну реалізацію та структуру, яка допомогла створити зображений вигляд домашньої сторінки.

Комбінація зображення вигляду сторінки та фрагменту коду надає повнішу картину про те, як була розроблена та реалізована домашня сторінка проекту. Вона дозволяє глибше зрозуміти як зовнішній вигляд, так і технічну структуру цієї важливої частини проекту.

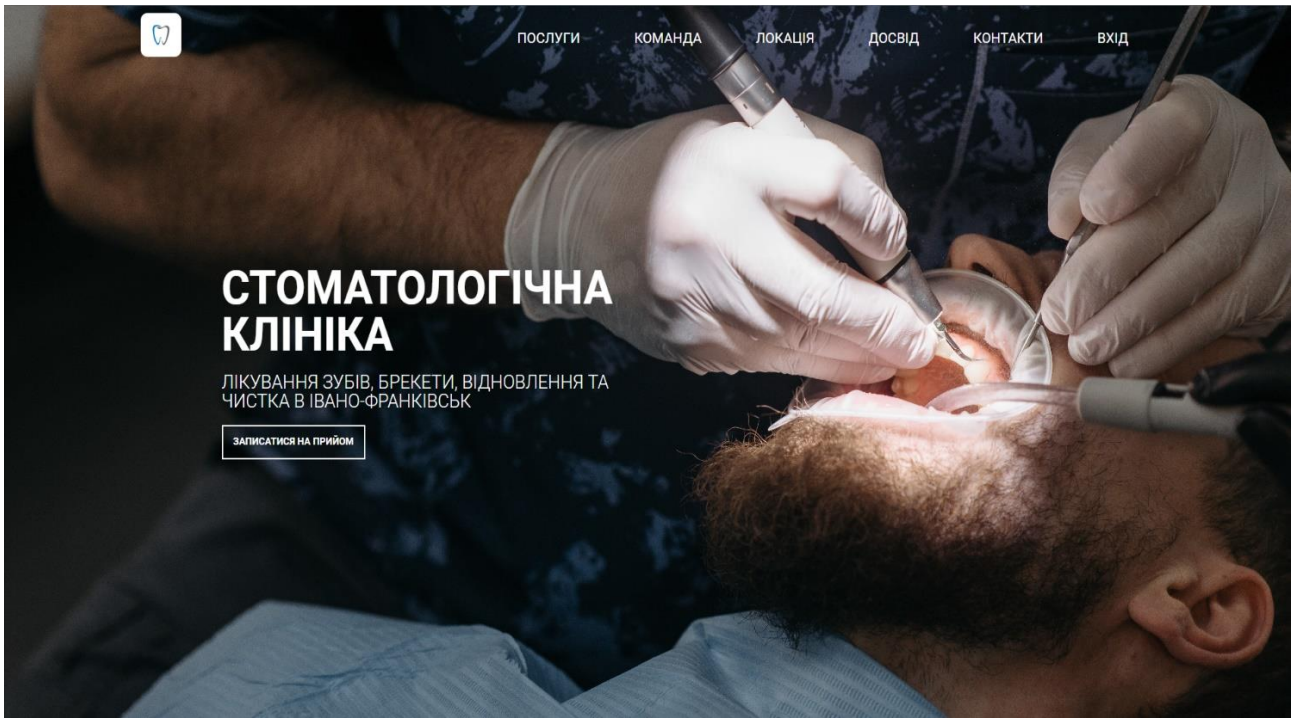

Рисунок 3.1 – Головна сторінка веб-сайту

Опис головної сторінки

Домашня сторінка містить перший компонент, який називається "Header". Цей компонент забезпечує зручну та швидку навігацію між різними сторінками. Ось фрагмент коду, який відповідає за відображення хедера:

```
<header class="header">
   <div class="container">
      <div class="header_inner">
         <div class="logo"><a href="/" class="logo-img">
            <img class="img" src="/img/logotype.jpg" width="60" height="60"></a></div>
```

```
        <nav class="nav">
          <a class="nav_link" href="#block-1">Послуги</a>
          <a class="nav_link" href="#block-3">Команда</a>
          <a class="nav_link" href="#block-4">Локація</a>
          <a class="nav_link" href="#block-2">Досвід</a>
          <a class="nav_link" href="#block-5">Контакти</a>
          <div class="" sec:authorize="hasRole('ROLE_CLIENT')">
            <a class="nav_link" href="UserPage">Мої записи</a>
          </div>
          <a sec:authorize="isAuthenticated()" class="nav_link" href=/logout>Вихід</a>
          <a sec:authorize="!isAuthenticated()" class="nav_link" href=/login>Вхід</a>
        </nav>
      </div>
    </div>
</header>
```

Після розміщення хедера ми маємо інший компонент, який представлений як div з класом "intro". Цей компонент включає фотографію, заголовок, текст і кнопку. Ось фрагмент коду, який відповідає за структуру компонента div з класом "intro":

```
<div class="intro">
    <div class="container">
      <div class="intro_inner">
        <h1 class="intro_title"> СТОМАТОЛОГІЧНА КЛІНІКА </h1>
        <h2 class="intro_title_2">лікування зубів, брекети, відновлення та чистка в Івано-
франківськ</h2>
        <div sec:authorize="!hasRole('ROLE_DENTIST')">
          <a class="btn" href="appointment">Записатися на прийом</a>
        </div>
        <div sec:authorize="hasRole('ROLE_DENTIST')">
          <a class="btn" href="DentistPage">Записи на прийом</a>
        </div>
      </div>
    </div>
</div>
```

3.2 Реалізація основного функціоналу процесу реєстрації

Одним з ключових етапів розробки стоматологічної системи є реалізація функціоналу реєстрації пацієнта на прийом до стоматолога. Цей процес передбачає взаємодію користувача з веб-інтерфейсом, створення нового запису в базі даних та його подальшу обробку, на (рис. 3.2), (рис. 3.3) зображено зовнішній вигляд сторінки для запису до стоматолога

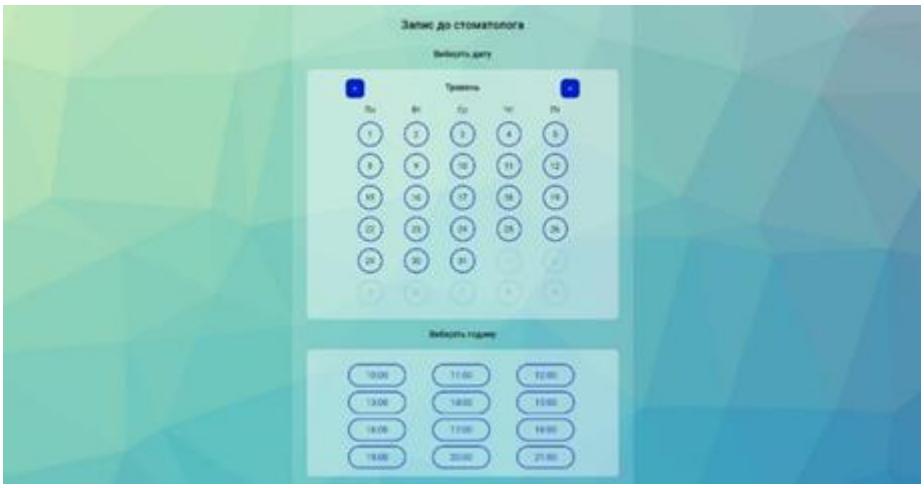

Рисунок 3.2 – Сторінки для запису до стоматолога

Рисунок 3.4 – Продовження сторінки для запису до стоматолога

Після того, як користувач вибрав дату, час, послугу та стоматолога, він натискає на кнопку "Записатися". Коли користувач натискає на цю кнопку, всі

дані збираються у формі. Для обробки цієї форми використовується наступний код, який має клас контролера AppointmentController:

```
@Controller
public class AppointmentController {
   @Autowired
   private JournalRepository journalRepo;
   @Autowired
   private DentistRepository dentistRepo;
   @Autowired
   private ServiceRepository serviceRepo;
   @Autowired
   private PatientRepository patientRepo;

   @GetMapping("/appointment")
   public String appointment(Model model) {
      model.addAttribute("dentists", dentistRepo.findByActive(true));
      model.addAttribute("services", serviceRepo.findByActive(true));
      model.addAttribute("appointmentData", new AppointmentDto());
      return "RecordDentist";
   }
```

Перша частина коду описує клас AppointmentController, який є контролером веб-додатка. Деякі ключові моменти цієї частини коду:

1. @Controller: Анотація, що позначає клас AppointmentController як контролер, який обробляє HTTP-запити.

2. @Autowired: Анотація, що позначає поля journalRepo, dentistRepo, serviceRepo та patientRepo для автоматичного впровадження залежностей (dependency injection). Це означає, що Spring автоматично надає цим полям екземпляри відповідних репозиторіїв.

3. @GetMapping("/appointment"): Анотація, яка вказує, що метод appointment обробляє HTTP GET-запити на шляху "/appointment".

4. public String appointment(Model model): Метод appointment приймає об'єкт Model і повертає рядок. В цьому методі модель розширюється шляхом додавання атрибутів dentists, services та appointmentData до моделі. Після цього відбувається повернення рядка "RecordDentist", що вказує на використання

шаблону "RecordDentist" для відображення сторінки.

```
@PostMapping("/saveAppointment")
public String saveAppointment(Principal principal, AppointmentDto appointmentData) {

    Authentication authentication = (Authentication) principal;
    final String username = ((User) authentication.getPrincipal()).getUsername();
    Patient patient = patientRepo.getPatientByUsername(username);

    final String[] dateSplit = appointmentData.getDate().split("\\.");
    final String[] timeSplit = appointmentData.getTime().split(":");
```

Друга частина коду відповідає за обробку POST-запиту на шляху "/saveAppointment" і метод saveAppointment. Деякі ключові моменти цієї частини коду:

1. @PostMapping("/saveAppointment"): Анотація, яка вказує, що метод saveAppointment обробляє HTTP POST-запити на шляху "/saveAppointment".
2. public String saveAppointment(Principal principal, AppointmentDto appointmentData): Метод saveAppointment приймає об'єкт Principal і AppointmentDto і повертає рядок. Principal містить інформацію про аутентифікованого користувача, а AppointmentDto містить дані, які передаються при записі на прийом.
3. Отримання імені користувача з об'єкта Principal і отримання відповідного об'єкта Patient з бази даних за допомогою patientRepo.
4. Розбивка дати та часу, які містяться в об'єкті appointmentData, на окремі частини за допомогою методу split.

```
    final Date date = new Calendar.Builder().setDate(
                Integer.parseInt(dateSplit[2]),
                Integer.parseInt(dateSplit[1]) - 1,
                Integer.parseInt(dateSplit[0]))
            .setTimeOfDay(Integer.parseInt(timeSplit[0]), Integer.parseInt(timeSplit[1]),
0).build().getTime();

    final Journal journal = new Journal();
    journal.setDate(date);
    journal.setDentist(dentistRepo.findById(appointmentData.getDentistId()).get());
```

```
        journal.setService(serviceRepo.findById(appointmentData.getServiceId()).get());
        journal.setPatient(patient);
        journalRepo.save(journal);
        return "redirect:/UserPage";
    }
```

Третя частина коду відповідає за створення об'єкта Date та збереження об'єкта Journal в базу даних:

1. Створення об'єкта Date за допомогою класу Calendar.Builder і встановлення дати та часу, використовуючи отримані значення з dateSplit та timeSplit.

2. Створення нового об'єкта Journal із використанням отриманих даних: дати, стоматолога, послуги та пацієнта.

3. Збереження об'єкта Journal в базу даних за допомогою journalRepo.save(journal).

4. Повернення рядка "redirect:/UserPage", що вказує на перенаправлення користувача на сторінку "/UserPage" після успішного збереження запису.

Отже, цей код виконує дії, пов'язані зі створенням об'єктів контролера, обробкою GET- та POST-запитів, отриманням даних з бази даних та збереженням запису на прийом.

Цей код містить два методи: appointment() та saveAppointment(). Метод appointment() встановлює модель зі списками стоматологів та послуг для передачі їх на сторінку "RecordDentist". Метод saveAppointment() обробляє POST-запит, збирає дані з форми та зберігає їх у репозиторії. Після збереження запису, користувач перенаправляється на сторінку "UserPage".

Після того, як користувач записався на прийом до лікаря, він отримує можливість перейти в свій власний кабінет, де може здійснити ряд дій. На (рис. 3.4) показано зовнішній вигляд сторінки кабінету користувача, де він зможе керувати своїми записами.

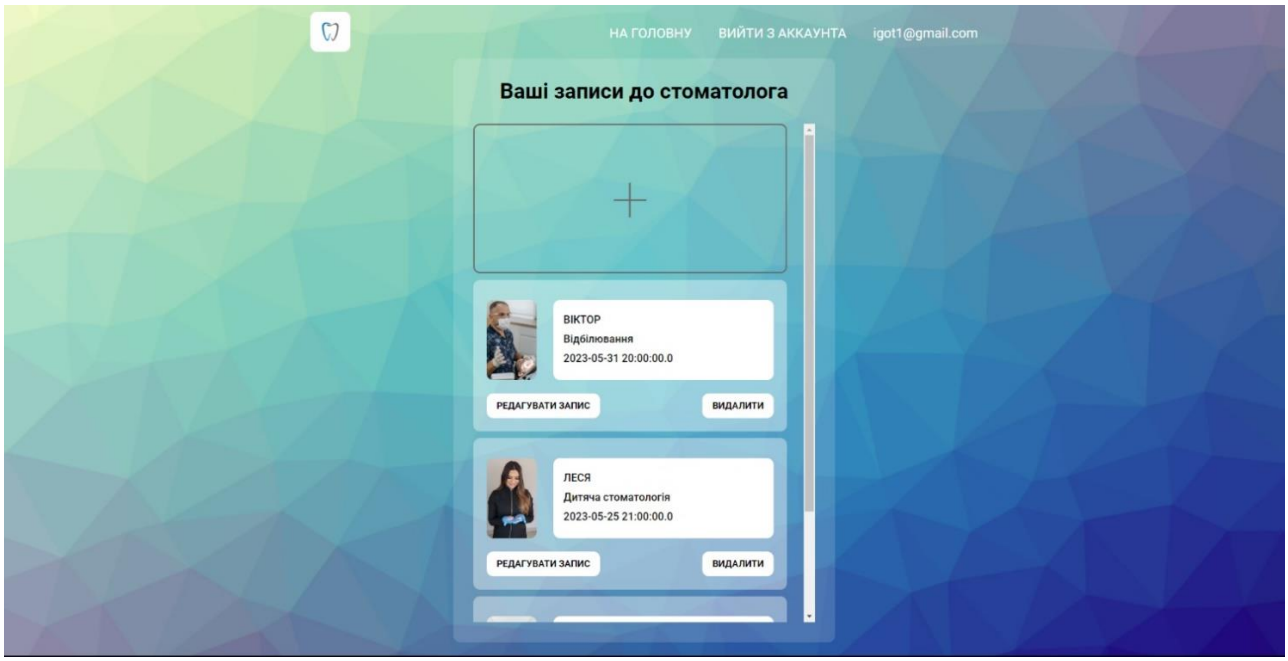

Рисунок 3.4 – Кабінет користувача

У кабінеті користувач має можливість переглянути всі свої записи до стоматолога. Він може бачити інформацію про дату та час запису, лікаря, до якого він записаний, а також надану послугу. Це дає йому зручність і огляд його плану прийомів.

Крім того, в кабінеті користувача є можливість видалити свої записи, якщо виникла необхідність змінити або скасувати прийом. Це дає користувачу гнучкість і контроль над своїми записами.

Також у кабінеті користувач може вийти зі свого облікового запису, якщо більше не потребує доступу до своїх даних. Після виходу з облікового запису користувач автоматично перенаправляється на головну сторінку, де може продовжити взаємодію з сайтом.

Нижче наведений код, який відповідає за відображення користувачеві записів у його кабінеті до стоматолога:

```
@GetMapping("/UserPage")
public String userPage(Principal principal, Model model) {
    Authentication authentication = (Authentication) principal;
    org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User) authentication.getPrincipal();
    final String username = user.getUsername();
```

```
    Patient patient = patientRepo.getPatientByUsername(username);
    final Calendar calendar = Calendar.getInstance();
    calendar.set(Calendar.HOUR, 0);
    calendar.set(Calendar.MINUTE, 0);
    calendar.set(Calendar.SECOND, 0);
    calendar.set(Calendar.MILLISECOND, 0);
    final List<Journal> patientRecords = journalRepo.findPatientRecords(patient.getId(),
calendar.getTime());
    model.addAttribute("patientRecords", patientRecords);
    return "UserPage";
}
```

У цьому коді використовується метод userPage(), який обробляє GET-запит на /UserPage. Він отримує об'єкт Principal, щоб отримати інформацію про авторизованого користувача. Після цього він отримує об'єкт користувача Patient за допомогою репозиторію patientRepo.

Далі створюється об'єкт Calendar, щоб встановити поточний час на 00:00:00, оскільки нам потрібно отримати записи на поточний день. За допомогою репозиторію journalRepo отримуються записи пацієнта за допомогою методу findPatientRecords(), передаючи його ідентифікатор та поточну дату.

Нарешті, отримані записи передаються в модель під атрибутом "patientRecords", і відбувається перенаправлення на сторінку "UserPage".

Код який відповідає за видалення запису у кабінеті користувача на сторінці:

```
@GetMapping("/UserPage/delete/{id}")
public String deleteDentist(@PathVariable String id){
    Optional<Journal> byId = journalRepo.findById(Long.valueOf(id));
    if(byId.isPresent()){
        journalRepo.delete(byId.get());
    }
    return "redirect:/UserPage";
}
```

У цьому коді використовується метод deleteDentist(), який обробляє GET-запит на "/UserPage/delete/{id}", де "{id}" - ідентифікатор запису, який потрібно видалити.

Спочатку виконується пошук запису за його ідентифікатором за допомогою методу findById() репозиторію journalRepo. Отриманий результат зберігається в об'єкт Optional<Journal> byId.

Далі перевіряється, чи присутній шуканий запис. Якщо так, то він видаляється за допомогою методу delete() репозиторію journalRepo, передаючи йому об'єкт byId.get().

На останок відбувається перенаправлення на сторінку "/UserPage" за допомогою "redirect:/UserPage".

Також і для стоматолога є його особистий кабінет, де він може переглянути, хто записався на прийом. Цей функціонал дозволяє стоматологу легко керувати своїм розкладом та планувати прийоми з пацієнтами. У кабінеті стоматолога відображається список пацієнтів, які записалися на прийом, включаючи дату та час прийому. На (рис. 3.5) зображено кабінет стоматолога.

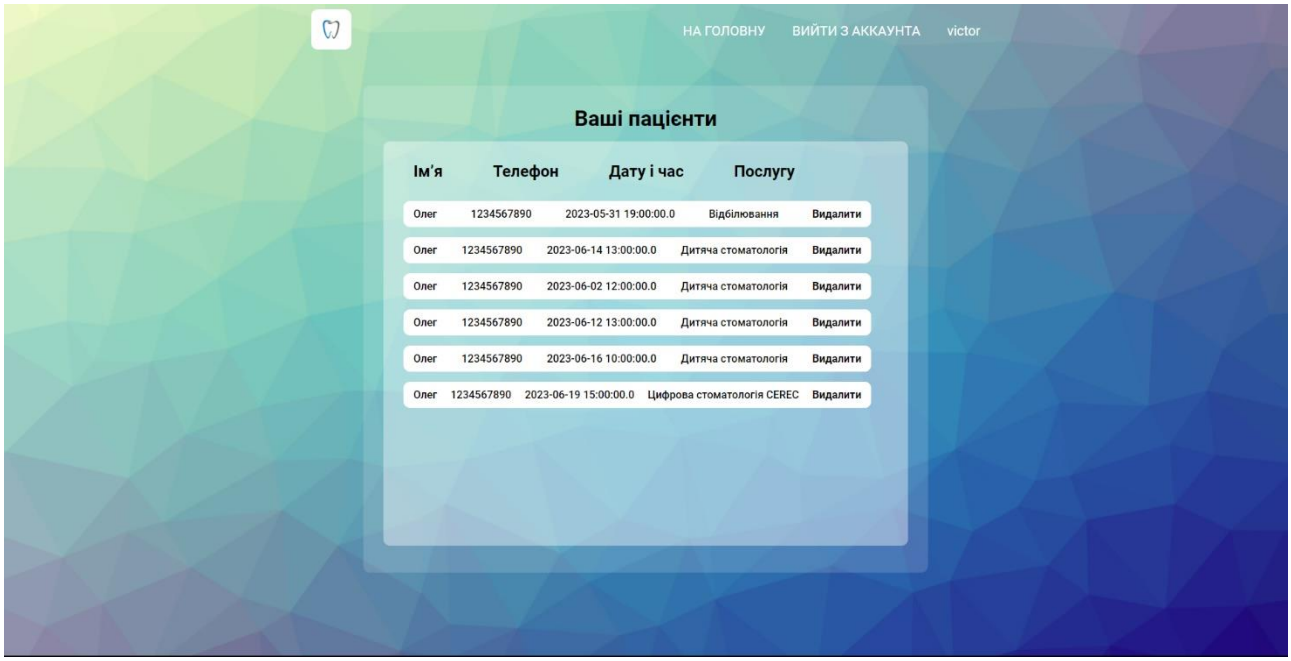

Рисунок 3.5 – Кабінет стоматолога

Нижче наведений код, який відповідає за відображення записів користувачів у кабінеті стоматолога:

```
@Controller
@RequiredArgsConstructor
```

```
public class DentistController {
    @Autowired
    JournalRepository journalRepo;
    @Autowired
    DentistRepository dentistRepository;
    @GetMapping("/DentistPage")
    public String userPage(Principal principal, Model model) {
        Authentication authentication = (Authentication) principal;
        User user = (User) authentication.getPrincipal();
        final String username = user.getUsername();
        Dentist dentist = dentistRepository.getDentistByUsername(username);
        final Calendar calendar = Calendar.getInstance();
        calendar.set(Calendar.HOUR, 0);
        calendar.set(Calendar.MINUTE, 0);
        calendar.set(Calendar.SECOND, 0);
        calendar.set(Calendar.MILLISECOND, 0);
        final List<Journal> patientRecords = journalRepo.findDentistRecords(dentist.getId(),
calendar.getTime());
        model.addAttribute("patientRecords", patientRecords);
        return "DentistPage";
    }
}
```

У цьому коді використовується клас DentistController, який є контролером для обробки HTTP-запитів на сторінку "/DentistPage".

Метод userPage() виконує GET-запит на "/DentistPage" та отримує об'єкт Principal та Model як параметри.

В цьому методі спочатку отримується об'єкт User за допомогою об'єкта Authentication та його методу getPrincipal(). Потім отримується ім'я користувача з об'єкта User.

Далі за допомогою об'єкта dentistRepository отримується об'єкт стоматолога за його іменем користувача.

Створюється об'єкт Calendar, який встановлює години, хвилини, секунди та мілісекунди на 0.

Далі виконується пошук записів користувачів, пов'язаних зі стоматологом, за його ідентифікатором та поточною датою за допомогою методу

findDentistRecords() репозиторію journalRepo. Отримані записи зберігаються в об'єкт List<Journal> patientRecords.

В кінці методу дані про записи передаються в модель за допомогою model.addAttribute("patientRecords", patientRecords).

На останок метод повертає рядок "DentistPage", який представляє назву шаблону (HTML-сторінки), яку треба відобразити користувачу.

Цей код забезпечує відображення записів користувачів у кабінеті стоматолога на сторінці "/DentistPage".

Нижче наведений код, який відповідає за видалення записів у кабінеті стоматолога:

```
@GetMapping("/DentistPage/delete/{id}")
public String deleteRecord(@PathVariable String id){
   journalRepo.deleteById(Long.valueOf(id));
   return "redirect:/DentistPage";
}
```

Цей код представляє метод deleteRecord(), який відповідає за обробку GET-запиту на шлях "/DentistPage/delete/{id}".

В цьому методі параметр id отримується з шляху, використовуючи аннотацію @PathVariable.

У тілі методу виконується видалення запису з використанням методу deleteById() репозиторію journalRepo. Параметром для цього методу є конвертоване у тип Long значення id.

Після видалення запису, метод повертає рядок "redirect:/DentistPage", що вказує на перенаправлення користувача на сторінку "/DentistPage" після успішного видалення запису [11 – 20].

3.3 Створення облікового запису пацієнта

Для цього сайту я реалізував функціонал створення облікового запису пацієнта, що дозволяє користувачам зареєструватися та мати особистий профіль. Під час реєстрації пацієнтів, вони вводять свої особисті дані, такі як ім'я, номер

телефону, адреса електронної пошти та пароль. Ця інформація зберігається у базі даних та використовується для подальшого входу в обліковий запис. Крім того, після успішної реєстрації пацієнт отримує підтвердження на електронну пошту. Таким чином, пацієнти мають можливість зручно управляти своїми записами, переглядати історію візитів та редагувати свої особисті дані. Цей функціонал сприяє покращенню комунікації між пацієнтами та стоматологічною клінікою, забезпечуючи зручний доступ до необхідної інформації та послуг. На (рис. 3.6) форма створення облікового запису.

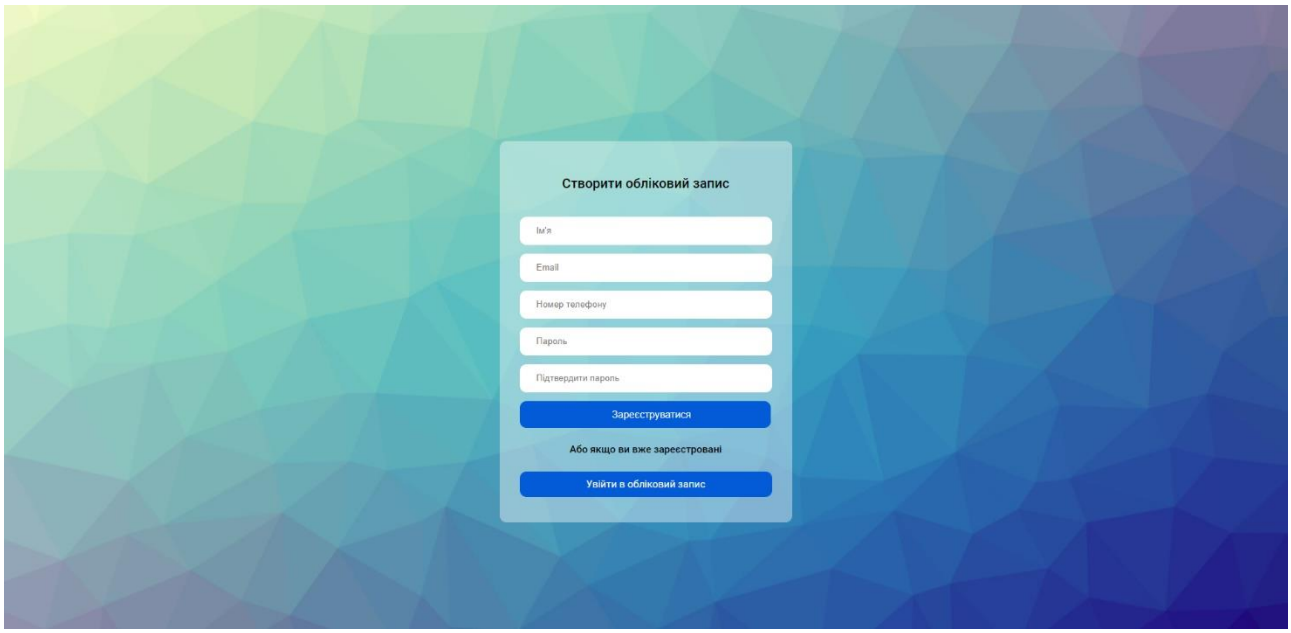

Рисунок 3.6 – Форма створення облікового запису

Нижче наведений код, який відповідає за реєстрацію користувача:

Частина 1 - Налаштування контролера та обробка GET-запиту на шляху "/SignUp":

```
@Controller
public class RegistrationController {

    @Autowired
    PatientRepository patientRepo;
    @Autowired
    UserRepository userRepo;
    @Autowired
```

```
    AuthorityRepository authorityRepo;
    @Autowired
    public PasswordEncoder passEncoder;

    @GetMapping("/SignUp")
    public String signUp(Model model){
        model.addAttribute("user", new UserDto());
        return "SignUp";
    }
```

У цій частині відбувається налаштування контролера та впровадження залежностей. Метод signUp обробляє GET-запит на шляху "/SignUp". Він створює об'єкт UserDto, додає його до моделі та повертає рядок "SignUp", що вказує на використання шаблону "SignUp" для відображення сторінки.

Частина 2 - Обробка POST-запиту на шляху "/SignUp":

```
@PostMapping ("/SignUp")
    public String registration(Model model, UserDto userDto){
        EmailValidator emailValidator = EmailValidator.getInstance();
        if (!emailValidator.isValid(userDto.getEmail())){
            return "redirect:/SignUp";
        }
        User user = new User();
        user.setName(userDto.getEmail());
        user.setPassword(passEncoder.encode(userDto.getPassword()));
        user.setActive(true);
        user = userRepo.save(user);
        Patient patient = new Patient();
        patient.setName(userDto.getFirstName());
        patient.setPhone(userDto.getPhone());
        patient.setUser(user);
        patientRepo.save(patient);
        Authority authority = new Authority();
        authority.setUsername(userDto.getEmail());
        authority.setAuthority("ROLE_CLIENT");
        authorityRepo.save(authority);
        return "login";
    }
```

У цій частині метод registration обробляє POST-запит на шляху "/SignUp".

- Спочатку виконується перевірка валідності електронної пошти з використанням EmailValidator. Якщо електронна пошта не є валідною, відбувається перенаправлення на шлях "/SignUp".
- Створюється об'єкт User на основі введених даних з UserDto. Пароль шифрується з використанням passEncoder.
- Зберігається об'єкт User у репозиторії userRepo.
- Створюється об'єкт Patient на основі введених даних з UserDto. Пов'язується з об'єктом User та зберігається у репозиторії patientRepo.
- Створюється об'єкт Authority та зберігається у репозиторії authorityRepo.
- Після успішного виконання збереження повертається рядок "login", що вказує на перенаправлення користувача на сторінку логіну.

Цей код включає в себе два методи: signUp() та registration(), які відповідають за реєстрацію нового користувача.

У методі signUp(), який обробляє GET-запит на шлях "/SignUp", створюється об'єкт UserDto та додається до моделі. Це дозволяє відобразити форму реєстрації на сторінці "SignUp".

У методі registration(), який обробляє POST-запит на шлях "/SignUp", виконується процес реєстрації користувача.

Спочатку перевіряється валідність введеної електронної пошти за допомогою EmailValidator. Якщо пошта не валідна, користувач перенаправляється на сторінку "/SignUp" знову.

Потім створюється об'єкт User, в який зберігається інформація про нового користувача, така як ім'я, пароль (закодований з використанням passEncoder) та активність облікового запису. Об'єкт User зберігається в репозиторії userRepo.

Далі створюється об'єкт Patient, в який зберігається інформація про нового пацієнта, така як ім'я, телефон та пов'язаний з ним обліковий запис User. Об'єкт Patient зберігається в репозиторії patientRepo.

Також створюється об'єкт Authority, в якому зберігається інформація про роль користувача. У цьому випадку, роль встановлюється як "ROLE_CLIENT". Об'єкт Authority зберігається в репозиторії authorityRepo.

Нарешті, метод повертає рядок "login", що вказує на перенаправлення користувача на сторінку входу після успішної реєстрації [10 – 20].

Висновки до розділу 3

У даному розділі надано детальний огляд процесу розробки проекту, де розглянуто структуру проекту та наведено докладний опис його функцій та кодових частин. Крім того, проілюстровано реалізацію структури та вигляду сторінки за допомогою рисунків, що демонструють використання відповідних елементів та компонентів. В даному розділі також розглянуто основне робоче середовище, яке використовувалося під час розробки проекту, зокрема засоби програмування та інші технології, що сприяли успішному втіленню задуманого проекту.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи створено web-застосунок для стоматологічного сайту, який надає зручну можливість онлайн запису до стоматолога. Цей інноваційний інструмент дозволяє клієнтам з легкістю запланувати свої візити, обираючи зручний час та дату без додаткових зусиль. За допомогою вбудованої форми запису, пацієнти можуть вибрати бажаного стоматолога, переглянути його розклад на сайті та негайно забронювати зустріч. Цей інтерактивний функціонал простий у використанні та забезпечує збільшення задоволення клієнтів, швидше реагування на їхні потреби та оптимізацію роботи стоматологічного закладу. Більше не потрібно тратити час на дзвінки та узгодження графіків, оскільки онлайн записи до стоматолога роблять процес максимально зручним та ефективним.

У процесі розробки даного проекту було здійснено наступні кроки:

- Проведено детальний аналіз переваг та недоліків існуючих аналогічних веб-сайтів.
- Вибрано набір технологій для реалізації, зокрема Java, MySQL, Spring Boot, HTML і CSS.
- Розроблено сучасний та легко сприйнятний дизайн застосунку.
- Створено стоматологічний веб-сайт, що містить можливість онлайн запису до стоматолога.
- Впроваджено заходи з охорони праці для забезпечення безпеки під час роботи над проектом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Meet the Spring team this August at SpringOne *spring.io:* вебсайт. URL: https://spring.io/ (дата звернення: 02.03.2023)
2. Java *uk.wikipedia.org:* вебсайт. URL: https://uk.wikipedia.org/wiki/Java (дата звернення: 02.03.2023)
3. Документація MySQL *mysql.com:* вебсайт. URL: https://dev.mysql.com/doc/ (дата звернення: 03.03.2023)
4. Maven *maven.apache.org:* вебсайт. URL: https://maven.apache.org/ (дата звернення: 03.03.2023)
5. ООП *techtarget.com:* вебсайт. URL: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP (дата звернення: 03.03.2023)
6. Spring-boot *spring.io:* вебсайт. URL: https://spring.io/projects/spring-boot (дата звернення: 04.03.2023)
7. Spring Data JPA - Reference Documentation *docs.spring.io:* вебсайт. URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ (дата звернення: 04.03.2023)
8. Web MVC framework *docs.spring.io:* вебсайт. URL: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html (дата звернення: 03.03.2023)
9. Defining JPA Entities *baeldung.com:* вебсайт. URL: https://www.baeldung.com/jpa-entities (дата звернення: 06.03.2023)
10. Spring Security *spring.io:* вебсайт. URL: https://spring.io/projects/spring-security (дата звернення: 07.03.2023)
11. Building REST services with Spring *spring.io:* вебсайт. URL: https://spring.io/guides/tutorials/rest/ (дата звернення: 10.03.2023)
12. Spring Security and Angular *spring.io:* вебсайт. URL: https://spring.io/guides/tutorials/spring-security-and-angular-js/ (дата звернення: 10.03.2023)

13. React.js and Spring *spring.io:* вебсайт. URL: https://spring.io/guides/tutorials/react-and-spring-data-rest/ (дата звернення: 23.03.2023)

14. Spring Boot and OAuth2 *spring.io:* вебсайт. URL: https://spring.io/guides/tutorials/spring-boot-oauth2/ (дата звернення: 23.03.2023)

15. Building web applications with Spring Boot and Kotlin *spring.io:* вебсайт. URL: https://spring.io/guides/tutorials/spring-boot-kotlin/ (дата звернення: 01.04.2023)

16. Spring Boot with Kotlin Coroutines and RSocket *spring.io:* вебсайт. URL: https://spring.io/guides/tutorials/spring-webflux-kotlin-rsocket/ (дата звернення: 01.04.2023)

17. Metrics and Tracing with Spring *spring.io:* вебсайт. URL: https://spring.io/guides/tutorials/metrics-and-tracing/ (дата звернення: 09.04.2023)

18. Spring Security Architecture *spring.io:* вебсайт. URL: https://spring.io/guides/topicals/spring-security-architecture/ (дата звернення: 10.04.2023)

19. Spring Boot Docker *spring.io:* вебсайт. URL: https://spring.io/guides/topicals/spring-boot-docker/ (дата звернення: 10.04.2023)

20. Spring on Kubernetes *spring.io:* вебсайт. URL: https://spring.io/guides/topicals/spring-on-kubernetes/ (дата звернення: 14.04.2023)