МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
Катерина НЕСТЕРЕНКО
“____”_____2024 р.

ДИПЛОМНИЙ ПРОЕКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСНИКА ОСВІТНЬОГО СТУПЕНЯ
“БАКАЛАВР”

Тема: “Програмна система онлайн-запису до лікарів стоматологічної клініки ”

Виконавець: Лисик Богдан Сергійович

Керівник: Гололобов Дмитро Олександрович

Нормоконтролер: Варнавський В’ячеслав Володимирович

Київ 2024

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення
Освітній ступінь бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Програмне забезпечення систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Катерина НЕСТЕРЕНКО
"___" _____ 2024 р.

ЗАВДАННЯ
на виконання дипломного проекту студента
Лисика Богдана Сергійовича

1. Тема проекту: «Програмна система онлайн-запису до лікарів стоматологічної клініки»
затверджена наказом ректора від 8.12.2023 р. № 2483/ст

2. Термін виконання проекту: з 8.12.2023 р. до 29.02.2024 р.

3. Вихідні данні до проекту: програмний продукт розробити у вигляді застосунку на базі мови програмування Python.

4. Зміст пояснювальної записки:
   1. Аналіз існуючого стану питання.
   2. Проектування системи і вимоги до програмного застосунку.
   3. Розробка структури і тестування застосунку.
   4. Результати роботи програми.

5. Календарний план-графік

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Складання та затвердження графіку роботи дипломного проектування Написання 1 розділу, представлення керівнику	08.12.23– 09.01.24	
2.	Попередній друк 1 розділу та допоміжних сторінок (чорновик) - титульної, завдання, графіка, реферат, список скорочень, зміст, вступ, список джерел, 1-й нормо-контроль.	10.01.24 – 17.01.24	
3.	Написання 2 розділу, представлення керівнику	18.01.24 – 22.01.24	
4.	Написання 3 розділу, представлення керівнику	23.01.24 – 26.01.24	
5.	Написання 4 розділу, представлення керівнику	27.01.24 – 31.01.24	
6.	Загальне редагування та друк пояснювальної записки, графічного матеріалу	01.02.24 – 09.02.24	
7.	Проходження нормо-контролю, перевірка на антиплагіат, перепліт пояснювальної записки.	01.02.24 – 05.02.24	
8.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	05.02.24 – 09.02.24	
9.	Отримання відгуку керівника, рецензії.	10.02.24 – 22.02.24	
10.	Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, CD-R з електронними копіями ПЗ, презентації, відгук керівника, рецензія) в папці	23.02.24 – 29.02.24	

7. Дата видачі завдання 08.12.2023
Керівник: Дмитро ГОЛОЛОБОВ
Завдання прийняв до виконання: Богдан ЛИСИК
Дата

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Програмна система онлайн-запису до лікарів стоматологічної клініки»: 60 с., 16 рис., 3 табл., 23 інформаційних джерел.

ОНЛАЙН-ЗАПИС, CRM, FLASK, PYTHON, WEB-DEVELOPMENT

Об’єкт розробки – процес автоматизації записів до стоматологічної клініки.

Мета роботи – підвищити ступінь автоматизації в роботі стоматологічної клініки.

ABSTRACT

Explanatory note to the diploma project "Software system of online appointment to doctors of a dental clinic": 60 pages, 16 figures, 3 tables, 23 information sources.

ONLINE REGISTRATION, CRM, FLASK, PYTHON, WEB DEVELOPMENT

The object of development is a dental clinic records system.

The purpose of the work is to increase the degree of automation in the work of the dental clinic.

ЗМІСТ

ВСТУП

Сучасний світ відзначається стрімким розвитком інформаційних технологій та комп'ютерних систем, що перетворили практично всі сфери життя та діяльності людини. Однією з важливих складових цього процесу є автоматизація різноманітних операцій, пов'язаних з обробкою даних, зокрема, у сфері управління медичними закладами.

Ця дипломна робота присвячена розробці системи управління медичними закладами, яка спрямована на покращення процесів прийому пацієнтів, призначення медичних консультацій та управління лікарським персоналом.

Мета роботи полягає в створенні ефективного інструменту, який сприятиме підвищенню якості надання медичних послуг та зручності для пацієнтів.

Об'єкт дослідження: процес автоматизації керування медичними закладами.

Предмет дослідження: розробка, реалізація та впровадження системи управління медичними закладами.

Очікувані результати включають в себе поліпшення організації роботи медичних закладів, зменшення часу очікування пацієнтів, зручну систему призначення консультацій та оптимізацію управління лікарським персоналом.

РОЗДІЛ 1.
АНАЛІЗ ІСНУЮЧОГО СТАНУ ПИТАННЯ

1.1.Огляд предметної області

Поліклініка (або амбулаторія, або амбулаторія) — це заклад охорони здоров'я, який в основному орієнтований на надання амбулаторної допомоги хворим. Клініки можуть бути приватними або державними, керованими та фінансованими. Зазвичай вони покривають потреби населення у первинній медичній допомозі в місцевих громадах, на відміну від більших лікарень, які пропонують більш спеціалізоване лікування та приймають пацієнтів на нічне перебування.

Найчастіше англійське слово clinic відноситься до загальної практики, якою керують один або більше лікарів загальної практики, які пропонують невеликі терапевтичні процедури, але воно також може означати спеціалізовану клініку. Деякі клініки зберігають назву «клініка», навіть коли вони перетворюються на такі великі установи, як великі лікарні, або асоціюються з лікарнями чи медичними школами.

Етимологія

Слово клініка походить від давньогрецького κλίνειν klinein, що означає нахилитися, нахилитися або відкинутися. Отже, κλίνη klinē – це кушетка або ліжко, а κλινικός klinikos – це лікар, який відвідує своїх пацієнтів у їхніх ліжках [1]. На латині це стало clīnicus [2][3].

Перше використання слова клініка було «той, хто приймає хрещення на ліжку хворого» [4].

Клініки часто асоціюються із загальною медичною практикою, якою керують один або декілька лікарів загальної практики. Інші типи клінік керуються типом спеціаліста, пов’язаного з цим типом: клініками фізичної терапії – фізіотерапевтами, а психологічними клініками – клінічними психологами, і так далі для кожної медичної професії. (Це може стосуватися навіть певних послуг поза медичною галуззю: наприклад, юридичними клініками керують юристи.)

Деякими клініками керують роботодавці, урядові організації чи лікарні, а деякі клінічні послуги надаються приватним корпораціям, які спеціалізуються на наданні медичних послуг. У Китаї, наприклад, власники таких клінік не мають формальної медичної освіти. У 2011 році в Китаї було 659 596 сільських амбулаторій [5].

Медичне обслуговування в Індії, Китаї та Африці надається величезним сільським районам цих регіонів мобільними медичними клініками або придорожніми диспансерами, деякі з яких інтегрують традиційну медицину. В Індії ці традиційні клініки пропонують аюрведичну медицину та медичну практику унані. У кожній із цих країн традиційна медицина, як правило, є спадковою практикою.

Функції клінік відрізняються від країни до країни. Наприклад, місцева амбулаторія загальної практики, якою керує один лікар загальної практики, надає первинну медичну допомогу і зазвичай є комерційною компанією власника, тоді як державна спеціалізована клініка може надавати субсидовану або спеціалізовану медичну допомогу. .

Деякі клініки слугують місцем для людей з травмами або захворюваннями, які можуть бути оглянуті медсестрою або іншим медичним працівником. У цих клініках травма чи хвороба можуть бути недостатньо

серйозними, щоб вимагати відвідування відділення невідкладної допомоги (ER), але за потреби людину можна перевести туди.

Лікування в цих клініках часто дешевше, ніж у відділенні невідкладної допомоги. Крім того, на відміну від швидкої допомоги, ці клініки часто не працюють 24/7/365. Іноді вони мають доступ до діагностичного обладнання, такого як рентгенівські апарати, особливо якщо клініка є частиною більшої установи. Лікарі таких клінік часто можуть направляти пацієнтів до спеціалістів, якщо виникає потреба [6].

У типових великих амбулаторних клініках працюють лікарі загальної практики (GP), такі як лікарі та медсестри, які надають амбулаторну допомогу та деякі послуги невідкладної допомоги, але їм не вистачає основних хірургічних, до- та післяопераційних закладів, які зазвичай пов’язані з лікарнями.

Крім лікарів загальної практики, якщо поліклініка є поліклінікою, в ній можуть розміщуватися амбулаторні відділення деяких медичних спеціальностей, таких як гінекологія, дерматологія, офтальмологія, отоларингологія, неврологія, пульмонологія, кардіологія, ендокринологія. У деяких університетських містах поліклініки містять амбулаторні відділення для всієї навчальної лікарні в одній будівлі.

1.2. Розробка програмного забезпечення

Розробка програмного забезпечення — це процес, який використовується для розробки, визначення, проектування, програмування, документування, тестування та виправлення помилок з метою створення та підтримки додатків, фреймворків або інших компонентів програмного забезпечення. Розробка програмного забезпечення передбачає написання та підтримку вихідного коду, але в ширшому сенсі вона включає всі процеси від концепції бажаного програмного забезпечення до остаточного прояву, як правило, у спланованому та структурованому процесі, який часто збігається з розробкою програмного забезпечення. Розробка програмного забезпечення також включає дослідження, нові розробки, створення прототипів, модифікацію, повторне використання, переробку, технічне обслуговування або будь-яку іншу діяльність, яка призводить до створення програмних продуктів [1].

Методики

Одна методологія розробки системи не обов’язково підходить для використання в усіх проектах.

Кожна з доступних методологій найкраще підходить для конкретних видів проектів, виходячи з різних технічних, організаційних, проектних і командних міркувань [2].

Діяльності

Виявлення потреби

Джерел ідей для програмних продуктів досить багато. Ці ідеї можуть виникнути в результаті дослідження ринку, включаючи демографічні дані потенційних нових клієнтів, існуючих клієнтів, потенційних клієнтів, які відмовилися від продукту, іншого внутрішнього персоналу з розробки програмного забезпечення або креативної третьої сторони. Ідеї для програмних продуктів, як правило, спочатку оцінюються маркетинговим персоналом на

предмет економічної доцільності, відповідності існуючим каналам розповсюдження, можливого впливу на існуючі лінійки продуктів, необхідних функцій і відповідності маркетинговим цілям компанії. На етапі маркетингової оцінки оцінюються припущення щодо витрат і часу. На початку першої фази приймається рішення про те, чи слід продовжувати проект на основі більш детальної інформації, отриманої спеціалістами з маркетингу та розробки [3].

Процес планування

Важливим завданням при створенні ПЗ є аналіз вимог. Клієнти зазвичай мають абстрактне уявлення про те, що вони хочуть як кінцевий результат, але не знають, що програмне забезпечення повинно робити. Кваліфіковані та досвідчені інженери програмного забезпечення визнають неповні, неоднозначні або навіть суперечливі вимоги на цьому етапі. Часта демонстрація коду в реальному часі може допомогти зменшити ризик того, що вимоги будуть неправильними.

«Хоча на етапі розробки вимог докладається багато зусиль, щоб гарантувати, що вимоги є повними та узгодженими, рідко це трапляється; залишаючи етап проектування програмного забезпечення найвпливовішим, коли йдеться про мінімізацію впливу нових або змінених вимог. Нестабільність вимог є складним, оскільки вони впливають на майбутні або вже реалізовані зусилля» [5].

Робоча сила

Розробник програмного забезпечення — це особа або компанія, яка займається процесом розробки програмного забезпечення, включаючи дослідження, проектування, програмування, тестування та інші аспекти створення комп’ютерного програмного забезпечення. Інші назви посад для людей із подібним значенням включають програміста, програмного аналітика або програмного інженера. Компанії, що спеціалізуються на програмному

забезпеченні, можна назвати будинками програмного забезпечення. У великій компанії можуть бути співробітники, одноосібна відповідальність яких складається тільки з однієї з дисциплін. У невеликих середовищах розробки кілька людей або одна людина можуть керувати повним процесом. Середовища спільної роботи, такі як програмне забезпечення з відкритим кодом, можуть об’єднати багато розробників.

Переглянути модель

Модель представлення — це структура, яка надає точки зору на систему та її середовище, які будуть використовуватися в процесі розробки програмного забезпечення. Це графічне представлення основної семантики перегляду.

Мета точок зору та поглядів полягає в тому, щоб дозволити інженерам-людинам зрозуміти дуже складні системи та організувати елементи проблеми навколо областей знань. У розробці фізично інтенсивних систем точки зору часто відповідають можливостям і обов'язкам в інженерній організації.[6]

1.3. Огляд існуючих рішень

ACE Dental – це програма, яка виділяється своїм користувацьким інтерфейсом та легкістю використання при плануванні зустрічей. Однак її обмежена можливість налаштування та відсутність мобільного додатку можуть бути недоліками для деяких клінік.

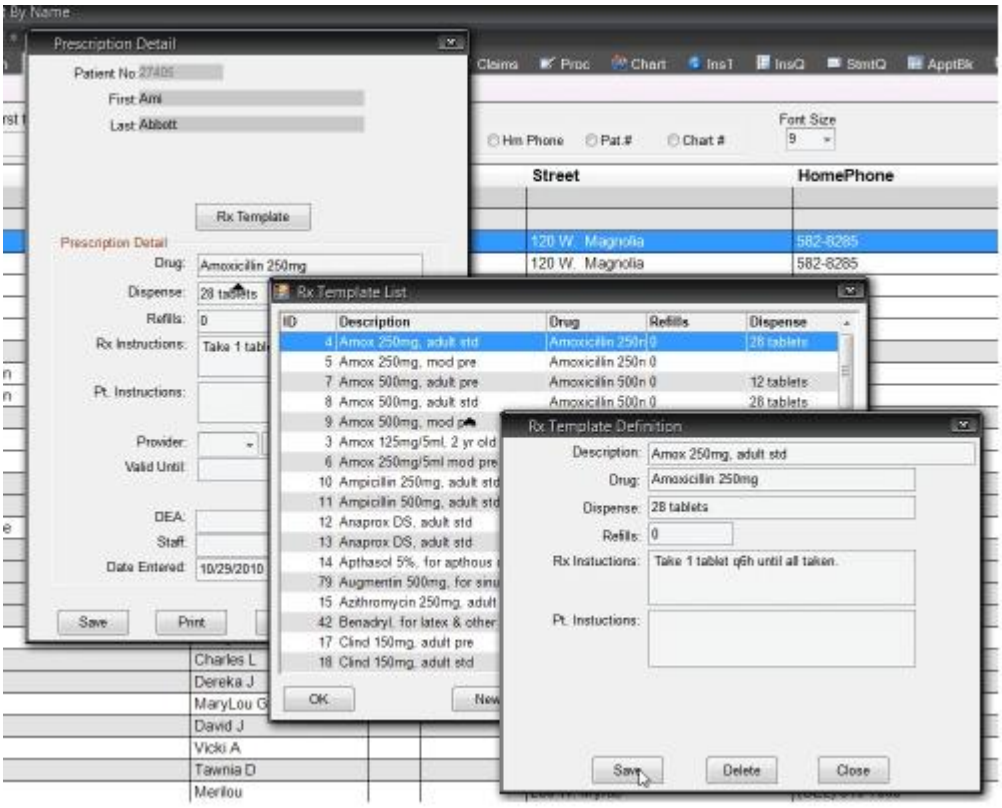

Рис. 1.1. ACE Dental

Oryx Dental Software забезпечує ефективне спілкування та залучення пацієнтів у реальному часі. Водночас, висока вартість та обмежені можливості налаштування можуть бути перешкодою для малих клінік.

Рис. 1.2. Oryx Dental Software

Clixlo відрізняється своєю гнучкістю у налаштуваннях та зосередженням на пацієнтському досвіді. Вона пропонує доступну ціну, але потребує певного навчання для повного використання можливостей кастомізації.

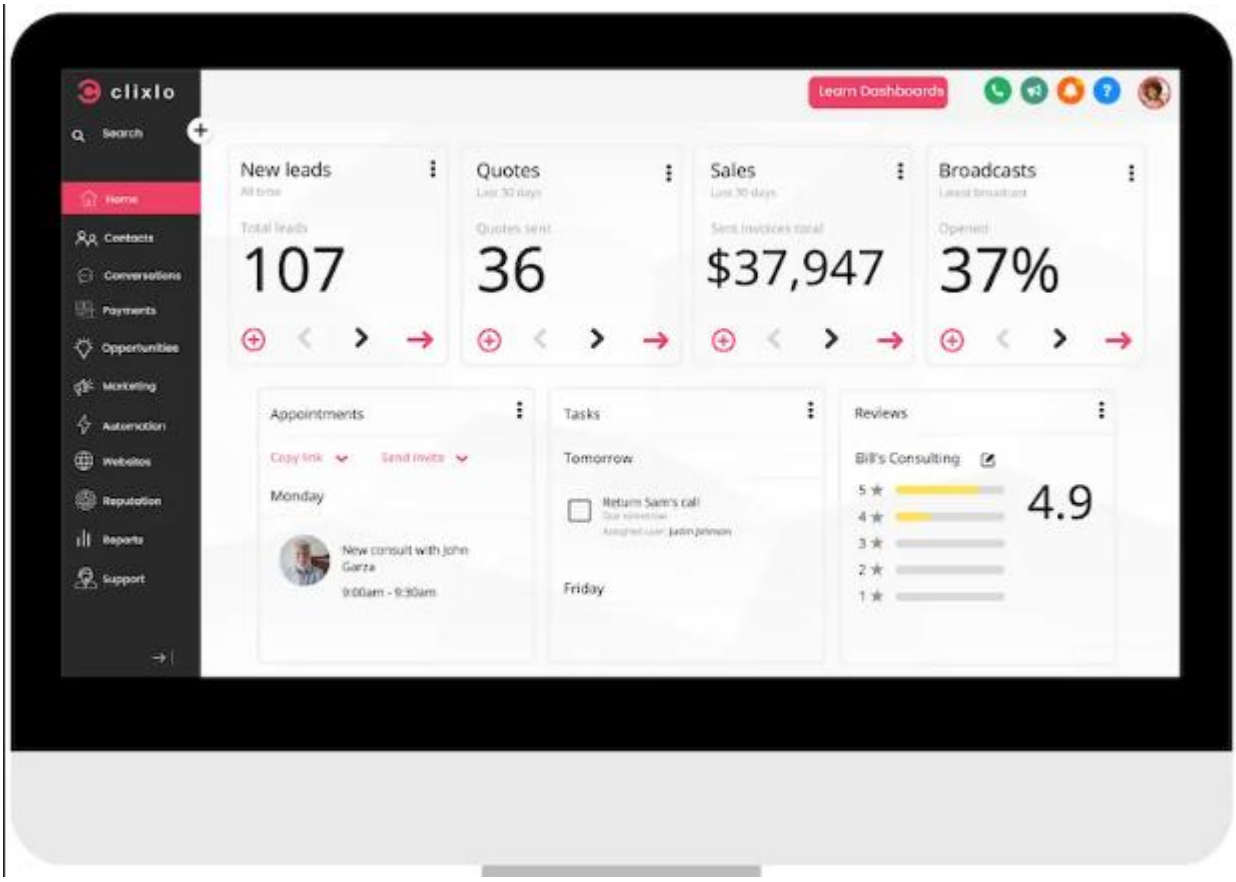

Рис. 1.3. Clixlo

NexHealth фокусується на автоматизації взаємодій з пацієнтами і нагадуваннях про зустрічі. Її висока ціна може бути недосяжною для малих клінік.

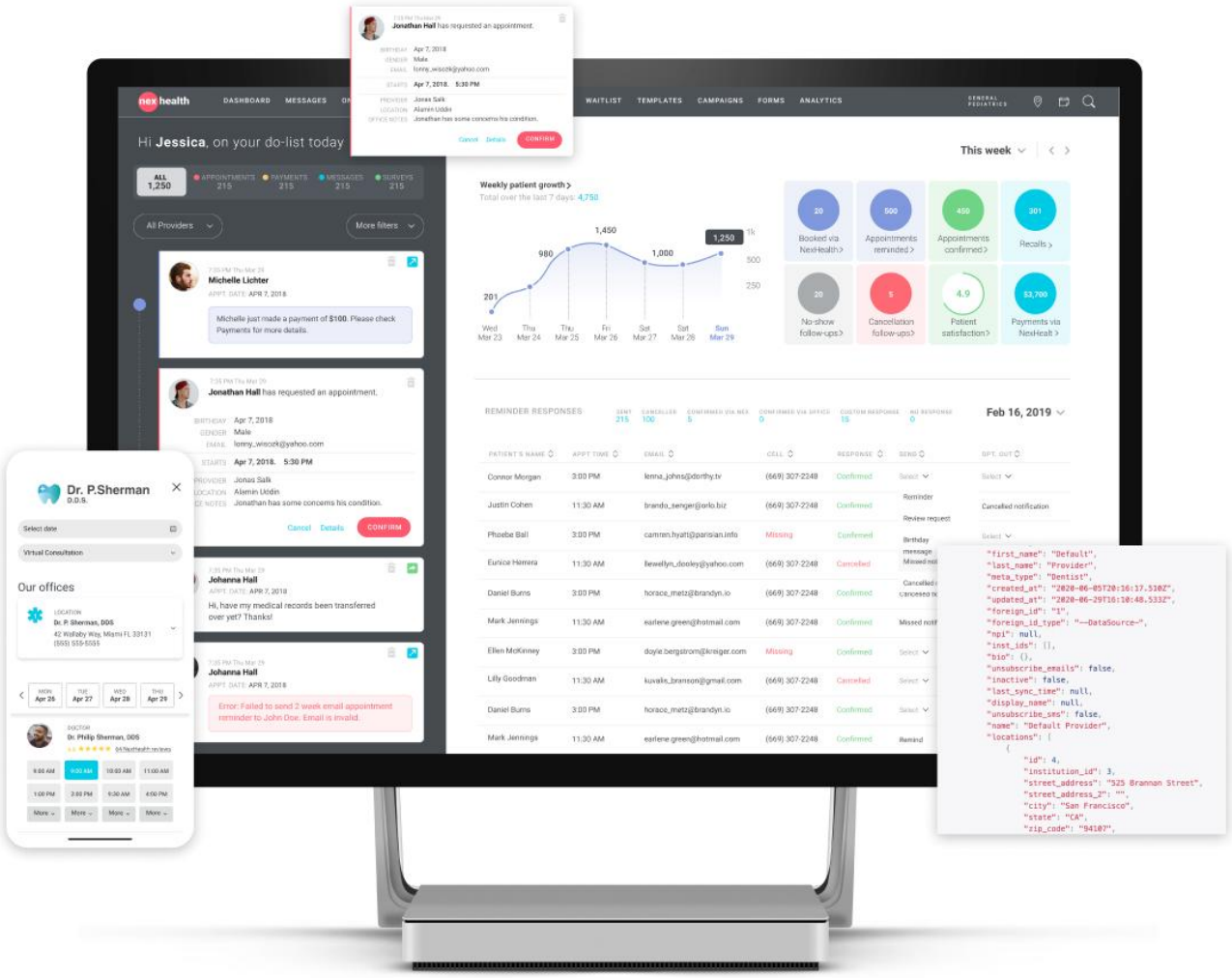

Рис. 1.4. NexHealth

Цей аналіз показує, що існуючі рішення мають свої переваги та недоліки, а ваша розробка може зосередитися на усуненні існуючих недоліків, пропонуючи рішення, яке буде більш доступним і гнучким для різних типів клінік.

1.4. Висновки до розділу

Програмне забезпечення для управління поліклінікою має велике значення для оптимізації роботи медичних закладів та покращення взаємодії з пацієнтами.

Аналіз існуючих рішень показує, що кожна з програм має свої переваги та обмеження. Наприклад, деякі програми відрізняються користувацьким інтерфейсом та ефективністю взаємодії з пацієнтами, але мають обмежені можливості налаштування або високу вартість.

З метою покращення роботи клінік та підвищення їхньої конкурентоспроможності, необхідно створити програмне рішення, яке буде поєднувати переваги існуючих систем і усувати їхні недоліки, забезпечуючи при цьому гнучкість, ефективність та доступність для різних клінічних потреб.

РОЗДІЛ 2.
ПРОЕКТУВАННЯ СИСТЕМИ

2.1. Проектування бази даних

База даних призначена для підтримки системи онлайн-запису до лікарів у стоматологічній клініці. Система дозволяє лікарям реєструвати свої дані, пацієнтам реєструватися, записуватися на прийоми та відслідковувати історію своїх записів.

Структура бази даних

База даних містить три сутності: "Лікар" (Doctor), "Пацієнт" (Patient) та "Прийом" (Appointment).

Сутність "Лікар" (Doctor)

- doctor_id (INT): Унікальний ідентифікатор лікаря.
- name (VARCHAR): Ім'я лікаря.
- specialty (VARCHAR): Спеціалізація лікаря.

Сутність "Пацієнт" (Patient)

- patient_id (INT): Унікальний ідентифікатор пацієнта.
- name (VARCHAR): Ім'я пацієнта.
- email (VARCHAR): Електронна пошта пацієнта (для реєстрації та сповіщень).

Сутність "Прийом" (Appointment)

- appointment_id (INT): Унікальний ідентифікатор прийому.
- date_time (DATETIME): Дата та час прийому.
- doctor_id (INT): Посилання на лікаря, до якого призначено прийом.
- patient_id (INT): Посилання на пацієнта, який зареєстрував прийом.

Ця база даних дозволяє зберігати та взаємодіяти з інформацією про лікарів, пацієнтів та прийоми для забезпечення функціональності системи онлайн-запису до лікарів стоматологічної клініки. Нижче наведені таблиці з детальним описом полів для кожної сутності.

Таблиця 2.1.

Сутність "Doctor" (Лікар)

Поле	Тип Даних	Призначення
doctor_id	INT	Унікальний ідентифікатор
name	VARCHAR	Ім'я лікаря
specialty	VARCHAR	Спеціалізація лікаря

Таблиця 2.2.

Сутність "Patient" (Пацієнт)

Поле	Тип Даних	Призначення
patient_id	INT	Унікальний ідентифікатор
name	VARCHAR	Ім'я пацієнта
email	VARCHAR	Електронна пошта пацієнта

Таблиця 2.3.

Сутність "Appointment" (Прийом)

Поле	Тип Даних	Призначення
appointment_id	INT	Унікальний ідентифікатор прийому
date_time	DATETIME	Дата та час прийому
doctor_id	INT	Посилання на лікаря
patient_id	INT	Посилання на пацієнта

На рис. 2.1. представлено ER-діаграму бази даних.

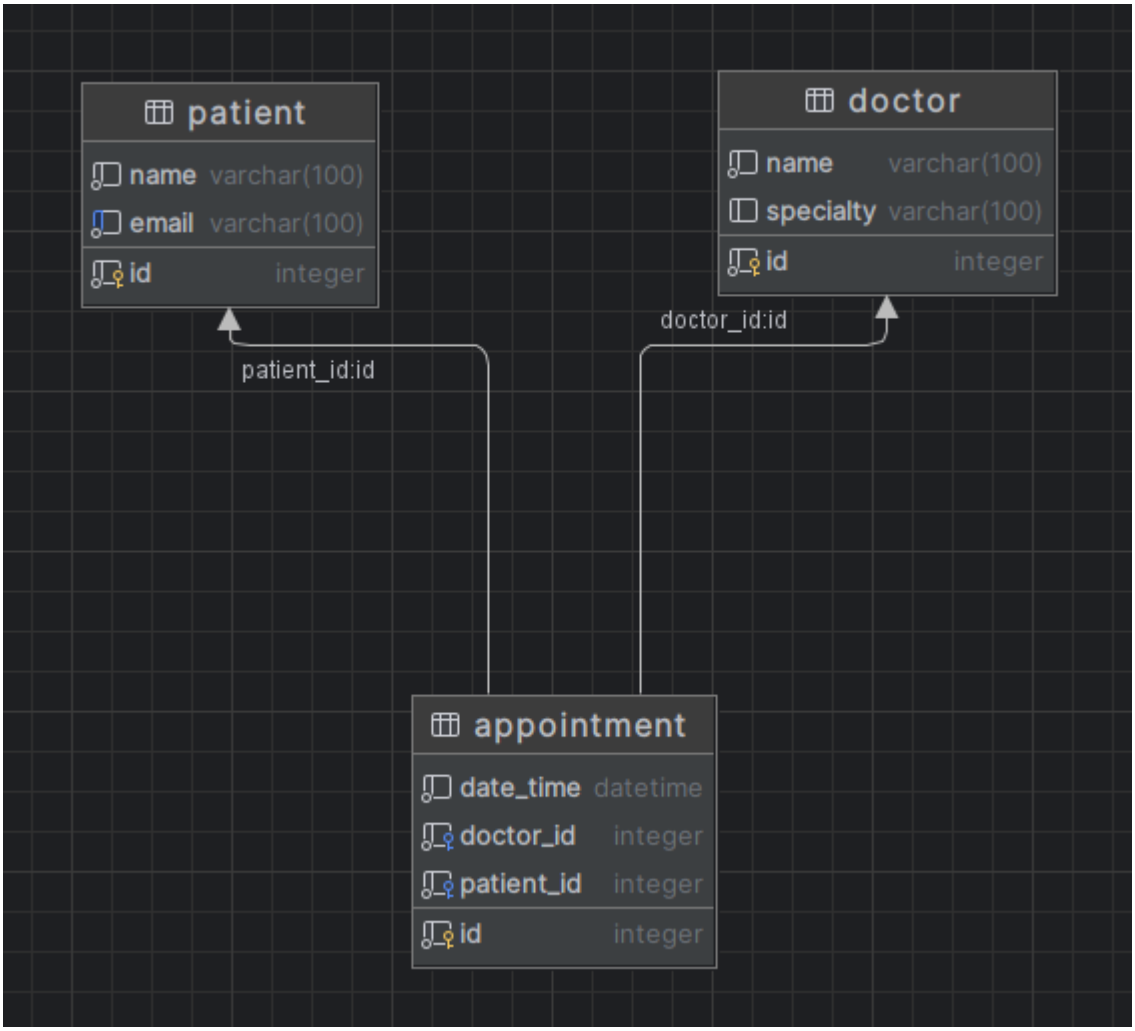

Рис. 2.1. ER-модель бази даних

2.2. Проектування функціоналу системи

Підрозділ присвячений проектуванню функціоналу системи онлайн-запису до лікарів стоматологічної клініки. Він включає в себе опис функцій та можливостей системи, що розробляється, а також детальний аналіз варіантів використання цієї системи.

Діаграма варіантів використання (Use Case Diagram) - це графічний інструмент, що використовується для моделювання функціональності системи та інтеракції користувачів з нею. Вона надає можливість ідентифікувати основні сценарії використання системи та визначити ролі, які беруть участь у цих сценаріях.

Система онлайн-запису до лікарів стоматологічної клініки має наступний функціонал:

1. Реєстрація пацієнта: Можливість реєстрації пацієнта в системі, включаючи особисті дані та контактну інформацію.
2. Авторизація лікаря: Можливість авторизації лікаря в системі за допомогою облікового запису.
3. Авторизація пацієнта: Можливість авторизації пацієнта в системі за допомогою облікового запису.
4. Запис на прийом: Можливість пацієнта записатися на прийом до лікаря на доступний час.
5. Перегляд історії записів: Можливість пацієнта переглядати історію своїх записів на прийоми.
6. Вихід з облікового запису: Можливість користувача (лікаря або пацієнта) вийти зі свого облікового запису.
7. Додавання лікаря (тільки для адміністратора): Можливість адміністратора системи додавати нових лікарів.

8. Додавання прийому (тільки для адміністратора): Можливість адміністратора створювати записи на прийом для лікарів.

Цей функціонал системи надає зручний та ефективний інструмент для онлайн-запису до лікарів стоматологічної клініки, відповідаючи потребам користувачів та вимогам сучасного суспільства.

На рис. 2.2. представлено діаграму варіантів використання, яка ілюструє даний функціонал.

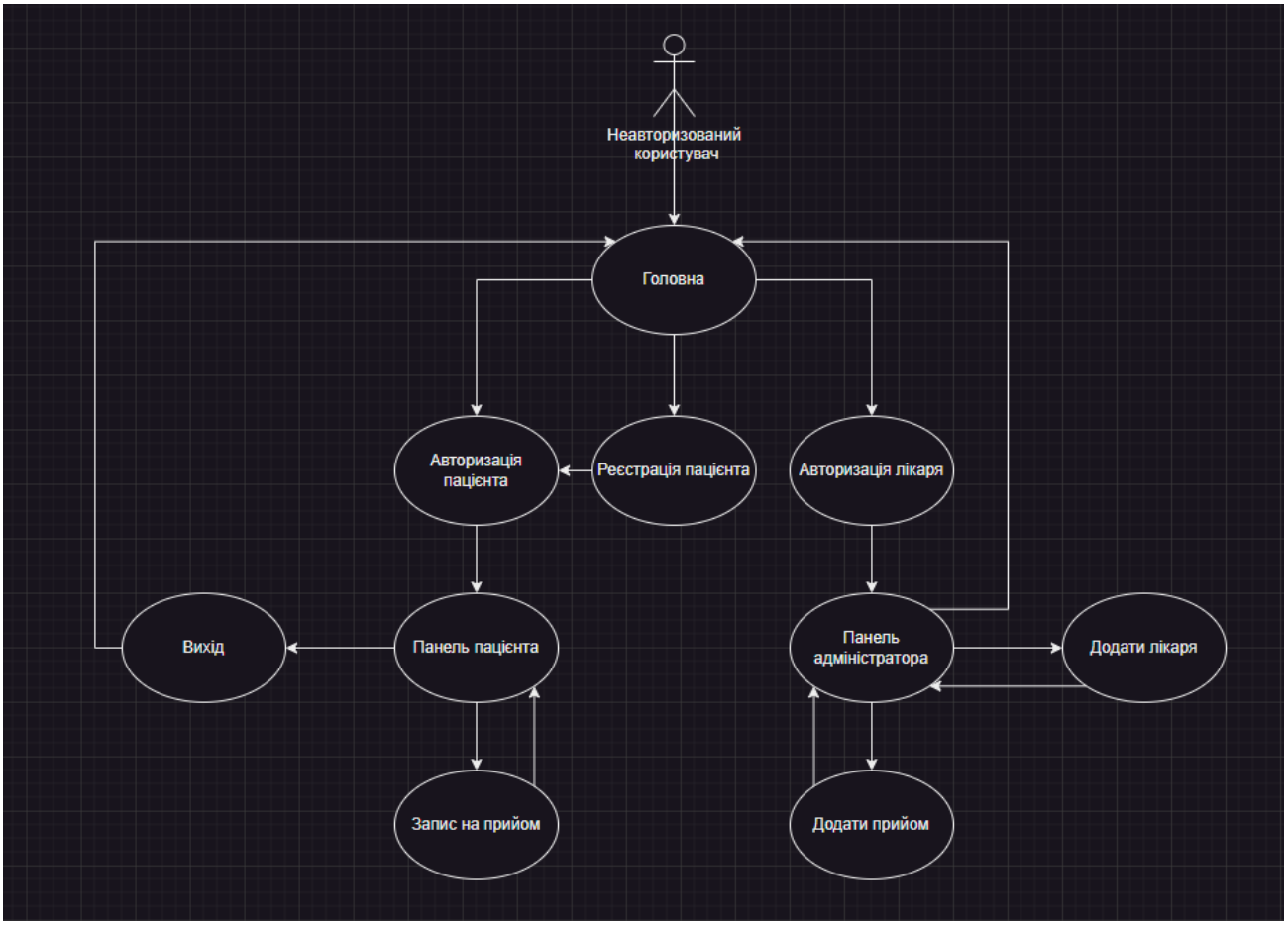

Рис. 2.2. Діаграма варіантів використання

2.3. Проектування внутрішньої будови

Під час розробки програмного забезпечення, зазвичай, зображення внутрішньої структури системи подається у формі діаграми класів, яка демонструє склад класів та модулів проекту та їх взаємодію.

У сфері програмної інженерії, діаграма класів, згідно з уніфікованою мовою моделювання (UML), є статичною структурною діаграмою, що надає огляд структури системи, включаючи системні класи, їх атрибути, операції (або методи) та зв'язки між об'єктами.

Діаграми класів виступають як ключові будівельні блоки об'єктно-орієнтованого моделювання. Вони використовуються для загального концептуального моделювання структури програми, а також для більш детального моделювання з метою перетворення моделі в програмний код. Також ці діаграми можуть служити інструментом моделювання даних. Кожен клас на діаграмі класів представляє основні елементи програми та класи, які будуть реалізовані.

На графічному зображенні класу зазвичай можна виділити три окремі частини: • Верхній блок, який містить назву класу, зазвичай надруковану жирним шрифтом та вирівняну за центром. Перша літера назви має бути великою. • Середній блок, де розташовані атрибути класу, які вирівнюються ліворуч, а перша літера їх назви повинна бути мала. • Нижній блок, який містить операції, які можуть бути виконані класом. Операції також вирівнюються ліворуч, і перша літера їх назви має бути мала.

При проектуванні системи, визначення багатьох класів та їх групування в схему класів допомагають зрозуміти статичні відносини між ними. Для більш детального моделювання, категорія концептуального проектування зазвичай поділяється на декілька підкатегорій.

Асоціація описує взаємозв'язок між елементами моделі. Якщо зміна одного елемента може призвести до зміни іншого, то між ними існує асоціація. Це відношення є одностороннім і представлене пунктирною лінією зі стрілкою, що вказує на напрямок замовника до постачальника.

Для подальшого опису системи, ці діаграми класів можуть бути доповнені діаграмами станів або становими автоматами UML.

Асоціація включає в себе різні типи: двосторонню, односторонню, агрегаційну (включаючи комбіновану агрегацію) та рефлексивну. Серед них найпоширеніші двосторонні та односторонні асоціації.

В моделюванні UML відношення реалізації вказують на те, що один елемент моделі (клієнт) втілює або реалізує функціональність, яка була задана іншим елементом моделі (постачальником). Це відношення графічно позначається порожнім трикутником, який з'єднаний інтерфейсом або деревом реалізаторів. Крім того, на діаграмі компонентів використовується графічна позначка "м'ячна розетка" для позначення реалізації. Важливо відзначити, що відношення реалізації зазвичай зображуються лише на діаграмах класів або компонентів, оскільки вони вказують на зв'язок між класами, інтерфейсами, компонентами і пакетами.

Залежність є слабкішою формою взаємодії, яка вказує на те, що один клас залежить від іншого і може використовувати його в певний момент часу. Відмінністю від асоціації є те, що залежність не передбачає наявність атрибутів одного класу, які були б екземплярами іншого класу. Відношення реалізації і асоціації графічно представлені як лінії, що з'єднують класи, з додатковими символами на кінцях ліній для вказівки додаткових деталей відношення.

Класи сутності в моделюванні представляють інформацію, яку система обробляє, а також поведінку, пов'язану з цією інформацією. Графічне представлення класів сутностей зазвичай має форму кола з короткою лінією,

прикріпленою до нижньої частини кола, або їх можна намалювати як звичайні класи зі стереотипом "сутність" над назвою класу.

Структура системи

Клас Doctor (Лікар):

Клас Doctor відображає лікарів у системі. Він має наступні поля:

- id (ціле число): Унікальний ідентифікатор лікаря.
- name (рядок): Ім'я лікаря.
- specialty (рядок): Спеціалізація лікаря.

Клас має конструктор __init__, який приймає параметри name і specialty, та створює новий об'єкт лікаря з цими полями.

Клас Patient (Пацієнт):

Клас Patient представляє пацієнтів у системі. Він має такі поля:

- id (ціле число): Унікальний ідентифікатор пацієнта.
- name (рядок): Ім'я пацієнта.
- email (рядок): Email пацієнта (унікальний).

Клас також має конструктор __init__, який приймає параметри name і email, і створює новий об'єкт пацієнта з цими полями.

Клас Appointment (Прийом):

Клас Appointment відображає інформацію про прийоми між лікарями та пацієнтами. Він має наступні поля:

- id (ціле число): Унікальний ідентифікатор прийому.
- date_time (дата і час): Дата і час прийому.
- doctor_id (ціле число): Зовнішній ключ, який посилається на ідентифікатор лікаря, з яким пов'язаний прийом.
- patient_id (ціле число): Зовнішній ключ, який посилається на ідентифікатор пацієнта, з яким пов'язаний прийом.

У класі Appointment є конструктор __init__, який приймає параметри doctor_id, patient_id і date_time, і створює новий об'єкт прийому з цими полями.

Методи маршрутизації:

1. / - Головна сторінка (маршрут home). Відображає загальну інформацію про систему.
2. /admin - Сторінка входу для адміністратора (маршрут admin_login). Адміністратору необхідно ввести ім'я користувача та пароль для входу в систему.
3. /admin/dashboard - Панель адміністратора (маршрут admin_dashboard). Відображає список лікарів та список прийомів, відсортованих за датою.
4. /patient - Сторінка входу для пацієнтів (маршрут patient_login). Пацієнту необхідно ввести свій email для входу в систему.
5. /patient/dashboard/<int:patient_id> - Панель пацієнта (маршрут patient_dashboard). Відображає інформацію про пацієнта та список його прийомів, відсортованих за датою.
6. /register - Сторінка реєстрації (маршрут register). Пацієнт може зареєструвати свій обліковий запис, вказавши своє ім'я та email.
7. /admin/add_doctor - Сторінка додавання нового лікаря (маршрут add_doctor). Адміністратор може додати нового лікаря, вказавши його ім'я та спеціалізацію.
8. /admin/add_appointment - Сторінка додавання нового прийому (маршрут add_appointment). Адміністратор може додати новий прийом, вказавши лікаря, пацієнта та дату та час прийому.
9. /logout - Вихід з облікового запису (маршрут logout). Пацієнт може вийти з системи.

10./patient/book_appointment - Сторінка бронювання прийому (маршрут book_appointment). Пацієнт може забронювати новий прийом, вказавши лікаря та дату і час.

На рис. 2.3 представлено діаграму класів системи.

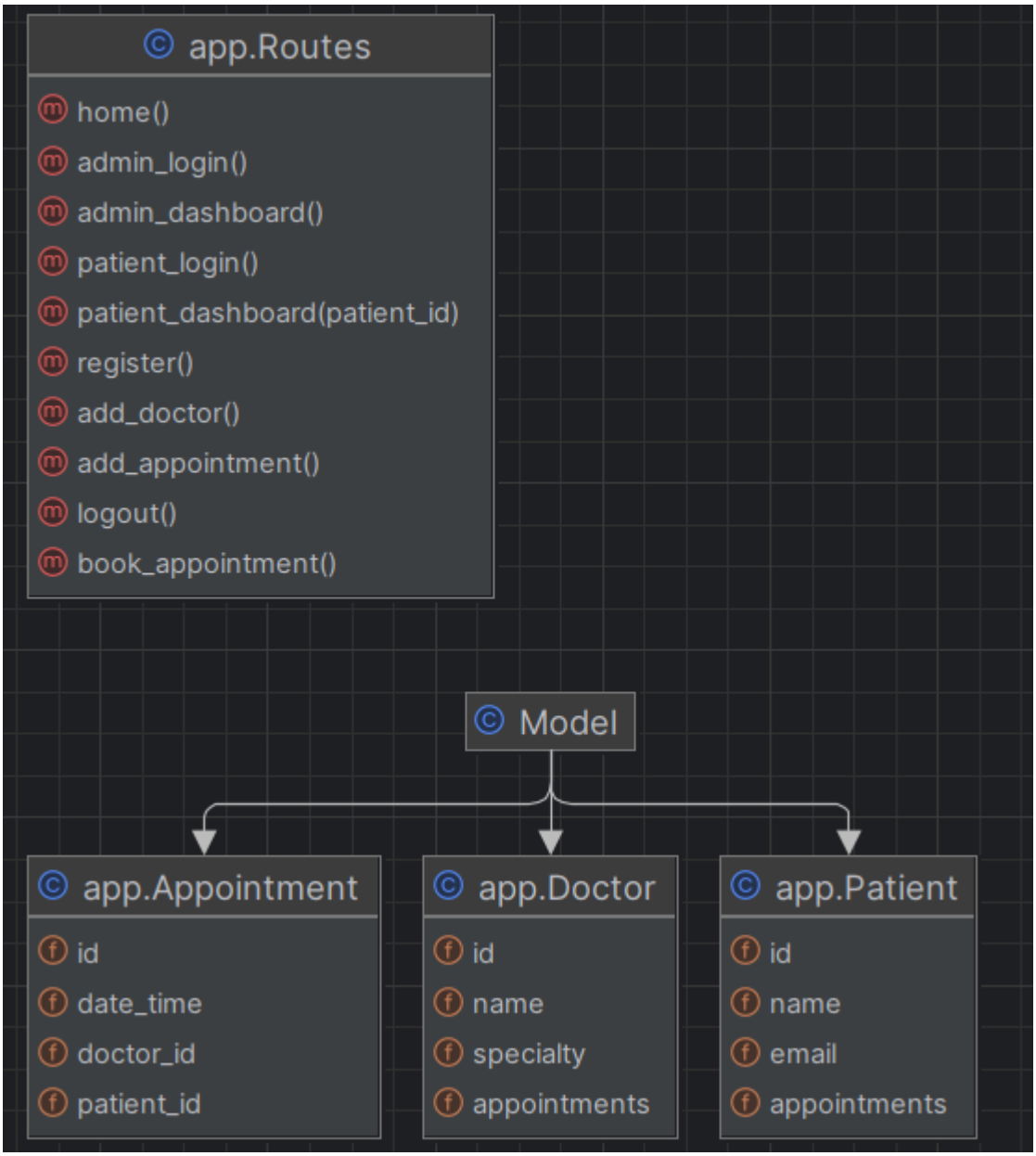

Рис. 2.3. Діаграма класів системи

2.4. Висновки до розділу

Завдяки проектуванню бази даних ми створили структуру, яка дозволяє зберігати та взаємодіяти з інформацією про лікарів, пацієнтів та прийоми, забезпечуючи необхідний функціонал системи. Проектування функціоналу системи дозволило нам визначити ключові можливості, які має надавати система, та описати варіанти використання.

Завдяки проектуванню внутрішньої будови ми розробили діаграму класів та методи маршрутизації для веб-додатка, що забезпечує логічну організацію та ефективну навігацію користувачів.

Усі ці кроки в проектуванні сприяли створенню системи, яка відповідає потребам користувачів та вимогам сучасного суспільства, та готової до подальшого розвитку та вдосконалення.

РОЗДІЛ 3.
РОЗРОБКА І ТЕСТУВАННЯ СИСТЕМИ

3.1. Вибір інструментів розробки

Python є мовою програмування високого рівня, що акцентує на читабельності коду за допомогою відступів. Ця мова підтримує кілька парадигм програмування, включаючи структурне, об'єктно-орієнтоване та функціональне програмування. Однією з ключових особливостей Python є його багата стандартна бібліотека, яка забезпечує різноманітні вбудовані можливості.

Python був розроблений Гвідо ван Россумом у кінці 1980-х років і вперше випущений у 1991 році. Значними етапами його розвитку стали випуск Python 2.0 у 2000 році, який включав нові функції, такі як розширене керування списками та удосконалену систему збору сміття, і Python 3.0 у 2008 році, який став основною версією мови. Незважаючи на те, що Python 3 не є повністю сумісним з Python 2, він визначає сучасний стандарт для розробки Python. Python 2 був офіційно припинений у 2020 році з випуском версії 2.7.18.

Python, розроблений Гвідо у кінці 1980-х років, відомий своєю високою популярністю серед мов програмування. Він був задуманий як наступник мови програмування ABC, впроваджуючи особливості для обробки винятків і взаємодії з операційною системою Amoeba. Розробка Python розпочалася у грудні 1989 року, і ван Россум керував проектом до липня 2018 року, коли він оголосив про свою "постійну відпустку" і отримав титул "Диктатора, дружнього до життя" Python.

Python 2.0 був випущений 16 жовтня 2000 року, додавши значну кількість нових функцій, включаючи систему збору сміття для ефективного управління пам'яттю і підтримку Unicode.

За ним прийшов Python 3.0, випущений 3 грудня 2008 року, і він був значною ревізією мови, яка не була повністю сумісною з попередньою версією. Багато ключових особливостей були поступово впроваджені в Python версіях 2.6.x і 2.7.x, і для полегшення переходу існує утиліта 2to3, яка допомагає автоматично перетворювати код з Python 2 на Python 3.

Початково планувалося припинити підтримку Python 2.7 в 2015 році, але через обширну кількість існуючого коду, який було б складно перенести на Python 3, ця дата була перенесена до 2020 року. Після закінчення терміну підтримки, Python 2.7 більше не отримуватиме оновлень безпеки та інших виправлень, і підтримуватимуться лише версії Python 3.6.x і новіші.

Python є багатопарадигмовою мовою програмування, повністю підтримує об'єктно-орієнтоване і структурне програмування, а також надає можливості функціонального і аспектно-орієнтованого програмування, включаючи метапрограмування та метаоб'єктний підхід. Python також включає динамічний збір сміття і підтримує динамічне розділення імен для зв'язування методів і змінних під час виконання програми.

Зокрема, Python розроблений з метою підтримки функціонального програмування, запозичуючи функції, такі як фільтрація, відображення, скорочення і ітератори. Стандартна бібліотека містить модулі itertools і functools, які реалізують функціональні інструменти, вдосконалені за запозиченням з мов Haskell і Standard ML.

Багато альфа-, бета- та реліз-кандидатів проходять попередній огляд і тестування перед остаточним релізом. Незважаючи на приблизний графік

виправлення проблем, якщо код не готовий, зазвичай випуск затримується. Під час розробки команда розробників Python активно виконує велику кількість модульних тестів для відстеження стану коду.

Головною академічною конференцією для Python є PyCon. Крім того, існують спеціалізовані навчальні програми для Python, такі як Pyladies.

Python 3.10 припиняє підтримку wstr (ця функція буде видалена в Python 3.12). Це означає, що розширення Python повинно бути адаптоване до цих змін, перш ніж вийде Python 3.12. Крім того, в майбутній версії планується додати підтримку відповідності шаблону до мови.

Python є однією з найпопулярніших мов програмування, яка з 2003 року постійно перебуває у топ-10 індексу TIOBE. Станом на лютий 2021 року Python займає третє місце за популярністю, поступаючись лише Java та C. Ця мова отримала звання "Мова програмування року" за версією TIOBE у 2007, 2010, 2018 та 2020 роках, ставши єдиною мовою, яка досягла такої честі чотири рази.

Python відомий своєю продуктивністю у програмуванні, особливо при роботі з рядками та словниками, часто перевершуючи традиційні мови, такі як C та Java, за ефективністю та використанням пам'яті. Ця мова широко використовується великими організаціями, включаючи Wikipedia, Google, Yahoo!, CERN, NASA, Facebook, Amazon, Instagram, Spotify, а також меншими компаніями, такими як ILM та ITA. Reddit, популярний сайт соціальних новин, також в основному написаний на Python.

Python є важливою частиною багатьох операційних систем, включаючи Linux, AmigaOS 4, FreeBSD, NetBSD, OpenBSD та macOS, і часто використовується в інсталяційних програмах для дистрибутивів Linux, таких як Ubuntu та Red Hat Linux, а також у системі керування пакетами Portage для Gentoo Linux.

З урахуванням широкого спектру застосувань та великих можливостей мови Python, яку було описано вище, можна визначити, що ця мова програмування відзначається універсальністю. Саме через це вона обрана для виконання цієї роботи.

PyCharm представляє собою інтегроване середовище розробки (IDE), спеціально призначене для програмування мовою Python. Це програмне забезпечення було розроблене чеською компанією JetBrains і володіє широким функціоналом, включаючи аналіз коду, графічний налагоджувач, вбудований модуль тестування та можливість інтеграції з системами контролю версій (VCS). PyCharm також підтримує використання фреймворку Django для розробки веб-додатків і платформи Anaconda для аналізу даних.

Це інтегроване середовище розробки доступна для користувачів Windows, macOS та Linux, завдяки чому вона є кросплатформовою. Крім безкоштовної версії PyCharm Community Edition з відкритим вихідним кодом, існує також професійна версія, яка надає додатковий функціонал і розповсюджується за власною ліцензією.

Релізи PyCharm виходять періодично, дозволяючи користувачам отримувати оновлення та нові можливості. Наприклад, бета-версія була запущена у липні 2010 року, і вже через три місяці з'явилася версія 1.0. Пізніше були випущені версії 2.0, 3.0 і 4.0 у різні роки.

Зокрема, PyCharm Community Edition, яка є вільно доступною для користувачів з відкритим кодом, була представлена 22 жовтня 2013 року

Фласк (Flask) - це мінімалістичний веб-фреймворк для розробки веб-додатків на мові програмування Python. Завдяки своїй простоті та гнучкості, Flask став популярним вибором серед розробників для створення веб-додатків різного роду, від невеликих проектів до більших і складніших застосунків.

Основні особливості Flask:

1. Мінімалізм: Flask відзначається мінімалізмом і не завантажує проект зайвим функціоналом. Розробники можуть вибирати, які розширення і функції вони хочуть використовувати в своєму проекті.
2. Рутинг: Flask надає простий та ефективний механізм рутингу для визначення, як обробляти HTTP-запити за певними URL-адресами. Це дозволяє легко створювати маршрути та обробляти їх відповідними функціями.
3. Шаблони: Flask підтримує використання шаблонів з використанням Jinja2, що спрощує генерацію HTML-сторінок і вивід динамічного контенту на сторінках.
4. Розширення: Велика кількість розширень доступних для Flask дозволяє легко додавати новий функціонал до вашого додатку. Наприклад, є розширення для автентифікації, роботи з базами даних, обробки форм, апі, і багато інших.
5. Підтримка RESTful API: Flask добре підходить для створення RESTful API, що робить його популярним вибором для розробки мікросервісів та веб-служб.
6. Активна спільнота: Flask має велику та активну спільноту розробників, яка підтримує фреймворк, надає документацію та вирішує проблеми, які виникають у користувачів.

Завдяки своїм перевагам, Flask ідеально підходить для розробки простих веб-сайтів, веб-додатків для стартапів та прототипування. Він дозволяє швидко створювати веб-додатки та залишає велику свободу вибору компонентів для реалізації вашого проекту.

SQLite - це легковагова вбудована система керування базами даних (СКБД), яка дозволяє зберігати, організовувати та оптимізувати дані. Вона є

популярним вибором для великої кількості додатків та прокладання шляху для взаємодії з базами даних. SQLite має такі основні характеристики:

1. Вбудована: База даних SQLite може бути вбудованою в додаток, що робить її ідеальним вибором для невеликих або однокористувацьких додатків.
2. Крос-платформеність: SQLite підтримується на багатьох платформах, включаючи Windows, macOS, Linux і більше.
3. Невеликий розмір: Файли баз даних SQLite мають невеликий розмір, що полегшує їхнє розповсюдження та зберігання.
4. SQL-сумісність: SQLite використовує мову запитів SQL, що робить його сумісним з більшістю систем управління базами даних.
5. Автономна робота: SQLite не потребує постійного підключення до сервера бази даних, що робить його ідеальним для локальних або офлайн додатків.
6. Транзакції: SQLite підтримує механізм транзакцій, який дозволяє забезпечити цілісність та надійність даних.
7. Широкий спектр типів даних: SQLite підтримує різні типи даних, включаючи цілі числа, рядки, дати та інші.
8. Спільнота та підтримка: SQLite має активну спільноту розробників та регулярно оновлюється, що забезпечує безпеку та стабільність.
9. Висока продуктивність: SQLite відомий своєю високою продуктивністю та швидкими запитами до бази даних.
10. Використання в різних областях: SQLite використовується у багатьох галузях, включаючи розробку мобільних додатків, веб-серверів, настільних програм, IoT та інше.

SQLite - це потужна та універсальна вбудована система керування базами даних, яка може бути використана в різних областях розробки. Її легкість

використання, крос-платформеність, невеликий розмір та висока продуктивність роблять її популярним вибором серед розробників. Відчутна активність спільноти та підтримка забезпечують стабільність та безпеку даних. SQLite ідеально підходить для невеликих та середніх проектів, а також для мобільних додатків, веб-серверів та інтернету речей (IoT). Ця система баз даних дозволяє розробникам легко зберігати та керувати даними без складних налаштувань і надмірної складності, що робить її важливим інструментом для будь-якого проекту, де потрібно зберігати та обробляти інформацію.

HTML (Hypertext Markup Language) - це стандартна мова розмітки для створення та відображення веб-сторінок у браузерах. HTML є основою для створення веб-контенту і визначає структуру та вигляд веб-сторінок.

HTML використовується для створення розмітки веб-сторінок, що визначає структуру та організацію контенту. Він використовує теги, що оточують текст та медіаелементи для визначення їх ролі на сторінці. Наприклад, <p> використовується для визначення абзаців, <h1> - для заголовків, <img> - для зображень і так далі.

Однією з важливих особливостей HTML є можливість створення гіпертекстових посилань (hyperlinks), які дозволяють переходити з однієї веб-сторінки на іншу або до інших ресурсів в Інтернеті. Посилання визначаються за допомогою тегу <a>.

HTML також дозволяє вставляти мультимедійний вміст, такий як зображення, відео та аудіо, за допомогою спеціальних тегів, наприклад, <img> для зображень і <video> для відео.

HTML - це стандартний мовний інструмент для розробників веб-сайтів та веб-додатків. Він забезпечує спосіб створення структурованого інтерфейсу користувача, який браузери можуть інтерпретувати і відображати на екрані. Разом з CSS (Cascading Style Sheets) і JavaScript, HTML стає основним інструментом для створення динамічних та взаємодіючих веб-додатків.

CSS (Cascading Style Sheets) - це мова стилізації, яка використовується для оформлення та вигляду веб-сторінок та додатків. CSS дозволяє розробникам контролювати вигляд HTML-елементів, встановлюючи стилі, кольори, розміри, розташування та інші візуальні аспекти веб-сторінок.

CSS працює в парі з HTML, де HTML визначає структуру контенту, а CSS визначає його вигляд. Ось декілька ключових характеристик CSS:

1. Селектори: CSS використовує селектори для вибору HTML-елементів, які потребують стилізації. Наприклад, селектор "p" визначає всі абзаци на сторінці.
2. Властивості: CSS дозволяє встановлювати різні властивості для обраних елементів, такі як колір тексту, фон, шрифт, розмір, відступи та багато інших.
3. Значення: Для кожної властивості CSS встановлює значення. Наприклад, властивість "color" може мати значення "red", "blue", "#FFA500" (кольоровий код) і т.д.
4. Каскадність: CSS використовує каскадний механізм для визначення, яка зміна стилю має переважити інші. Це дозволяє зберігати стилізацію легкою та керованою.
5. Зовнішні файли: CSS може бути включено безпосередньо в HTML-файл, або розміщено у зовнішньому файлі з розширенням ".css". Використання зовнішніх CSS-файлів дозволяє розділити структуру та вигляд, що полегшує обслуговування та зміни.

6. Респонсивний дизайн: CSS дозволяє створювати респонсивні веб-сайти, які адаптуються до різних розмірів екранів і пристроїв.
7. Анімація та переходи: CSS дозволяє створювати анімацію та плавні переходи між станами елементів на сторінці за допомогою ключових фреймів (keyframes) та інших властивостей.

CSS є необхідною складовою веб-розробки, оскільки вона дозволяє створювати привабливі та користувацьки-орієнтовані веб-сайти та додатки. Разом з HTML і JavaScript воно утворює основну трійку технологій веб-розробки.

Bootstrap - це відкрите програмне забезпечення для розробки веб-сайтів та веб-додатків. Це набір готових стилів, компонентів та інструментів, який допомагає розробникам швидко створювати сучасні та респонсивні інтерфейси. Ось короткий огляд Bootstrap та його ключових характеристик:

1. Респонсивний дизайн: Bootstrap надає гнучкий респонсивний сітковий систему, яка дозволяє легко адаптувати веб-сторінки до різних розмірів екранів, включаючи мобільні пристрої, планшети та настільні комп'ютери.
2. Готові компоненти: Bootstrap постачається з великим набором готових компонентів, таких як кнопки, форми, навігація, каруселі, модальні вікна, таблиці і багато інших. Ці компоненти можна легко використовувати в проектах без необхідності вручну створювати їх з нуля.
3. Стилі та теми: Bootstrap надає однорідний та сучасний дизайн, який може бути легко налаштований або змінений за допомогою стилів та тем. Розробники можуть змінювати кольори, шрифти, відступи і інші параметри за допомогою Sass або Less.
4. Компоненти JavaScript: Bootstrap включає багато компонентів JavaScript, таких як модальні вікна, вибір дати, валідація форм та інші. Це спрощує

реалізацію функціональності на сторінці без необхідності писати власний код.

5. Спільнота та підтримка: Bootstrap має велику та активну спільноту розробників, яка надає документацію, приклади, розширення та розробляє оновлення для фреймворку.
6. Сумісність з браузерами: Bootstrap підтримує багато сучасних браузерів, що дозволяє створювати веб-додатки, які працюють в різних середовищах.

Bootstrap є ідеальним вибором для розробників, які шукають швидкий і ефективний спосіб створити веб-інтерфейси з сучасним дизайном і респонсивним виглядом. Він пропонує велику кількість інструментів та ресурсів, які значно спрощують процес розробки та дозволяють швидко отримати результати.

SQLAlchemy - це високорівнева бібліотека для роботи з базами даних у мові програмування Python. Вона надає розширений інтерфейс для взаємодії з реляційними базами даних, такими як PostgreSQL, MySQL, SQLite та іншими. SQLAlchemy дозволяє розробникам створювати та керувати моделями даних, виконувати запити до бази даних та забезпечує гнучкість та контроль над роботою з даними. Ось короткий огляд SQLAlchemy та його ключових характеристик:

1. ORM (Об'єктно-реляційне відображення): SQLAlchemy використовує підхід ORM, що дозволяє розробникам працювати з об'єктами Python, як з таблицями бази даних. Це спрощує створення, зчитування, оновлення та видалення даних без прямої роботи з SQL.
2. Сесії та транзакції: SQLAlchemy дозволяє керувати транзакціями та сесіями з базою даних, що робить роботу з даними безпечною та надійною.

3. Підтримка багатьох СКБД: SQLAlchemy підтримує роботу з різними системами керування базами даних, включаючи PostgreSQL, MySQL, SQLite, Oracle та інші.
4. Модульність: SQLAlchemy розділено на кілька компонентів, які можна використовувати окремо або в поєднанні. Наприклад, ви можете використовувати лише ORM-частину або використовувати SQLAlchemy Core для генерації SQL-запитів.
5. Міграції: SQLAlchemy також надає інструменти для керування міграціями схеми бази даних, що дозволяє легко оновлювати структуру бази даних з часом.
6. Спільнота та підтримка: SQLAlchemy має активну спільноту розробників та підтримується підтримкою, що забезпечує стабільність та надійність бібліотеки.

SQLAlchemy є потужним інструментом для роботи з базами даних у Python і дозволяє розробникам створювати додатки, які легко взаємодіють з реляційними базами даних, забезпечуючи високий рівень абстракції та контролю. Він часто використовується для розробки веб-додатків, програмного забезпечення для обробки даних та інших проектів, де потрібна робота з базами даних.

3.2. Розробка графічного інтерфейсу

Графічний інтерфейс користувача (GUI) - це засіб взаємодії людини з комп'ютерною програмою або системою за допомогою графічних елементів, таких як вікна, кнопки, меню, іконки та миша. GUI забезпечує зручний та інтуїтивно зрозумілий спосіб керування програмами та взаємодії з комп'ютерною системою. Ось короткий огляд поняття GUI та його ключових характеристик:

1. Графічні елементи: GUI використовує графічні об'єкти, такі як вікна, кнопки, текстові поля, чекбокси, радіокнопки та інші, для створення інтерфейсу користувача.
2. Мультимедіа: GUI дозволяє включати мультимедійні елементи, такі як зображення, аудіо та відео, що робить інтерфейс більш зрозумілим та привабливим.
3. Взаємодія: Користувач може взаємодіяти з програмою за допомогою миші, клавіатури або сенсорного екрана, обираючи графічні елементи та виконуючи дії.
4. Меню: GUI може містити різні види меню, які дозволяють користувачам вибирати опції та виконувати дії за допомогою відповідних команд.
5. Розташування: Елементи інтерфейсу можуть розташовуватися у вікні програми або на робочому столі, що робить їх зручно доступними для користувача.
6. Стилі та дизайн: GUI може мати різні стилі та дизайни, які надають програмі або системі власний вигляд та індивідуальність.
7. Підтримка мов: GUI може підтримувати різні мови та регіональні налаштування, що дозволяє користувачам вибирати мову інтерфейсу, яка для них найбільш зручна.

GUI широко використовується у багатьох видомах програмного забезпечення, включаючи операційні системи, офісні пакети, веб-браузери, графічні редактори, ігри та багато інших додатків. Він робить комп'ютери більш доступними для користувачів, надаючи їм зручний спосіб взаємодії з технологією.

Система складається з 9 основних сторінок, представлених на рисунках 3.1-3.9.

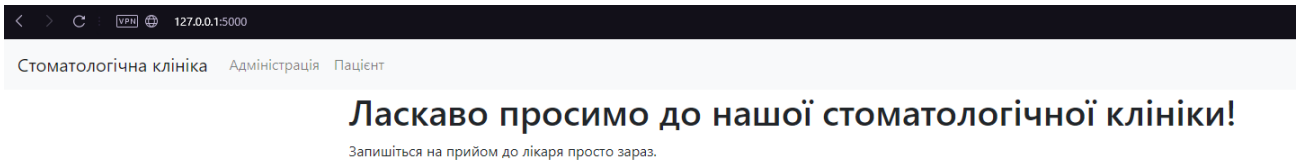

Рис. 3.1. Головна сторінка

Рис. 3.2. Сторінка авторизації адміністратора

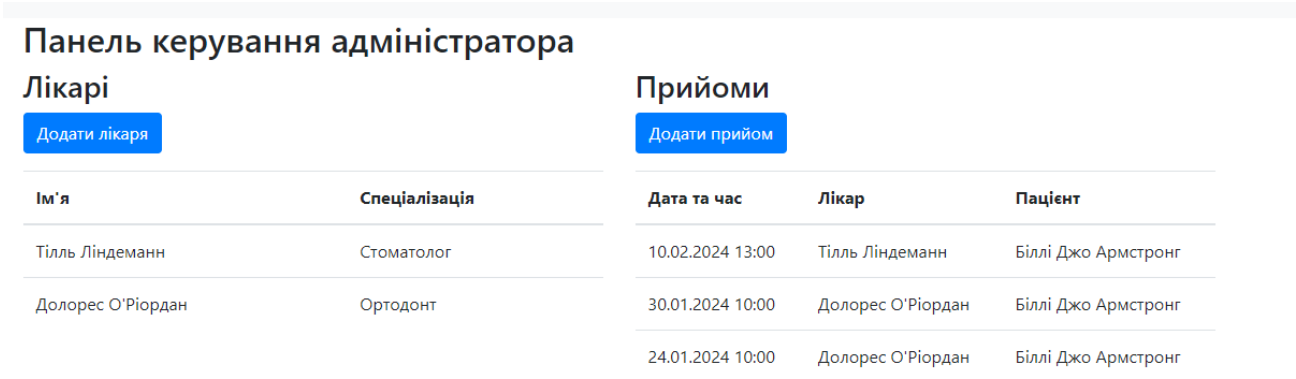

Рис. 3.3. Сторінка панелі керування адміністратора

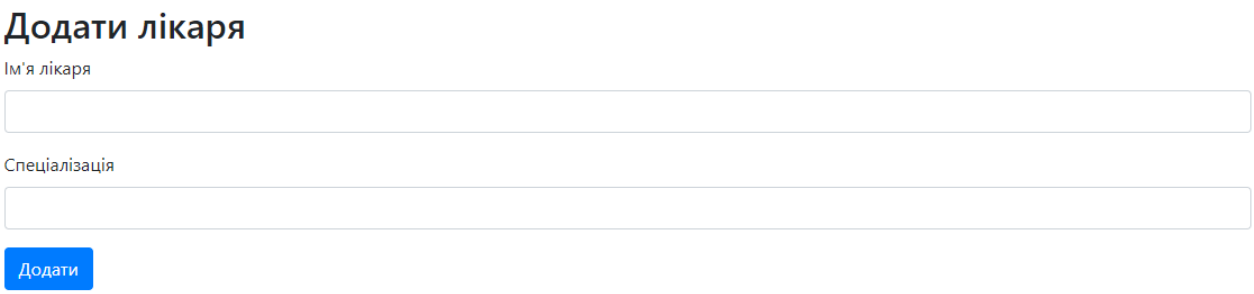

Рис. 3.4. Сторінка додавання лікаря

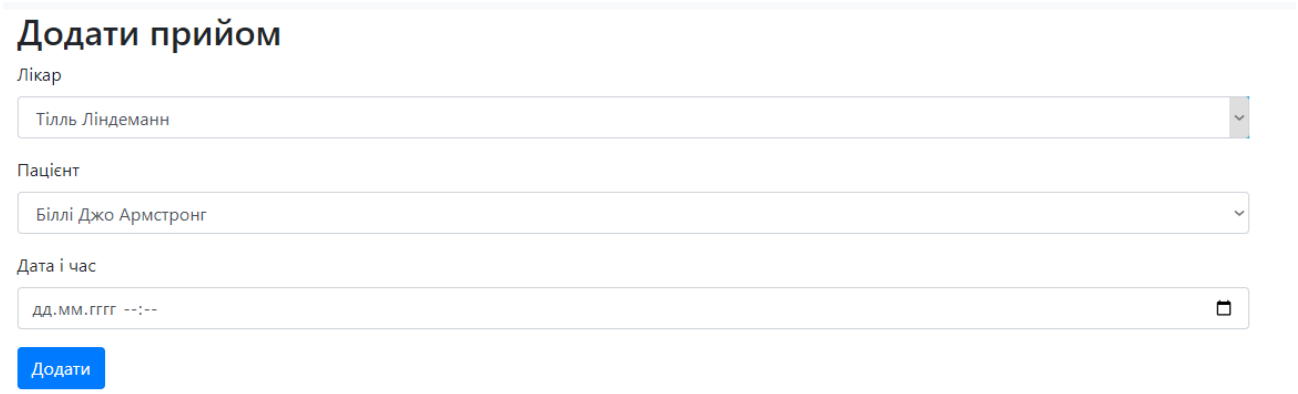

Рис. 3.5. Сторінка додавання прийому (адмін)

Рис. 3.6. Сторінка авторизації пацієнта

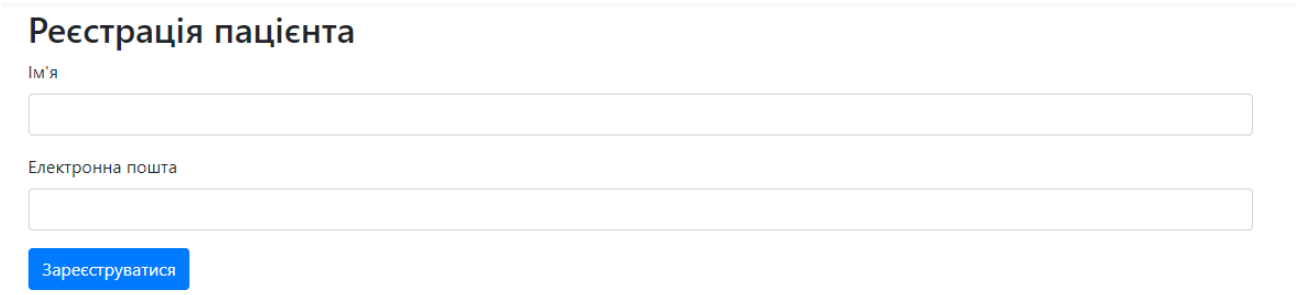

Рис. 3.7. Сторінка реєстрації пацієнта

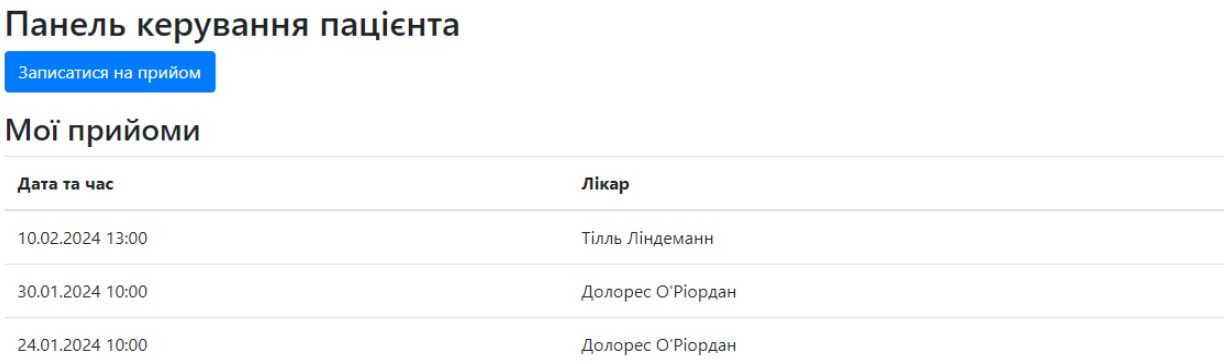

Рис. 3.8. Панель керування пацієнта

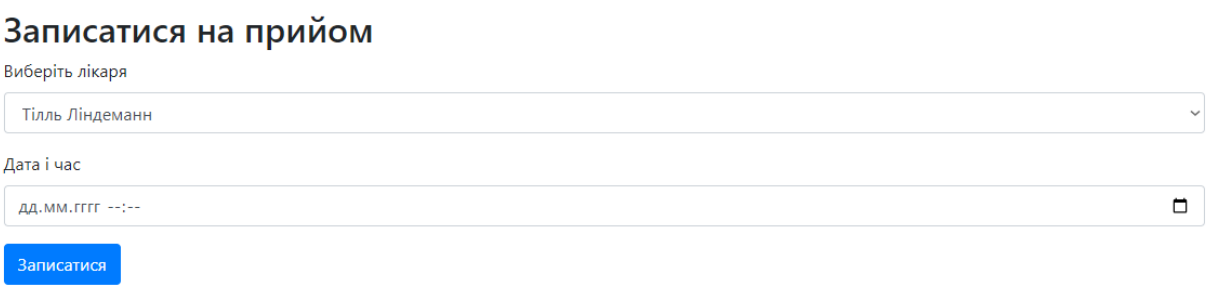

Рис. 3.9. Сторінка запису на прийом (пацієнт)

3.3. Розробка основного коду системи

Імпортування та налаштування Flask та SQLAlchemy:

```
from flask import Flask, render_template, request, redirect, url_for, session
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime

app = Flask(__name__)
app.secret_key = 'very_secret_key'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///clinic.db'
db = SQLAlchemy(app)
```

В цьому фрагменті коду імпортуються необхідні бібліотеки Flask та SQLAlchemy. Також створюється об'єкт Flask і налаштовується база даних SQLAlchemy, яка використовує SQLite для збереження даних.

Визначення моделей бази даних:

```
class Doctor(db.Model):
    # Визначення моделі для лікаря з полями id, name та specialty.
    # Також встановлені відносини з моделлю Appointment.

class Patient(db.Model):
    # Визначення моделі для пацієнта з полями id, name та email.
    # Також встановлені відносини з моделлю Appointment.

class Appointment(db.Model):
    # Визначення моделі для прийому з полями id, date_time, doctor_id та patient_id.
```

У цьому фрагменті коду створюються моделі бази даних для лікарів, пацієнтів та прийомів, використовуючи SQLAlchemy.

Створення таблиць бази даних:

```
with app.app_context():
    db.create_all()
```

Цей код створює таблиці в базі даних на основі описаних моделей. Це робиться в контексті додатку Flask.

Визначення шаблонного фільтру format_datetime:

```
@app.template_filter('format_datetime')
def format_datetime(value, format='%d.%m.%Y %H:%M'):
    # Фільтр для форматування дати та часу у заданому форматі.
```

Цей фільтр дозволяє форматувати дату та час у вигляді, заданому за замовчуванням.

Визначення маршрутів (routes):

/ - Головна сторінка (home).

/admin - Сторінка для входу адміністратора.

/admin/dashboard - Панель управління адміністратора.

/patient - Сторінка для входу пацієнта.

/patient/dashboard/<int:patient_id> - Панель управління пацієнта з параметром patient_id.

/register - Сторінка реєстрації нового пацієнта.

/admin/add_doctor - Сторінка для додавання нового лікаря адміністратором.

/admin/add_appointment - Сторінка для додавання нового прийому адміністратором.

/logout - Вихід з системи.

/patient/book_appointment - Сторінка для запису на прийом пацієнтом.

Кожен маршрут відповідає певному URL і може приймати різні HTTP-запити (GET, POST).

Головний блок:

```
if __name__ == '__main__':
    app.run(debug=True)
```

Цей блок перевіряє, чи файл запускається безпосередньо (а не імпортується в інший файл), і, у разі позитивної відповіді, запускає веб-застосунок на локальному сервері з включеним режимом налагодження (debug mode).

Цей код створює веб-застосунок для управління прийомами у лікаря та пацієнтами у клініці, включаючи функціональність авторизації для адміністратора та пацієнта, додавання лікарів, запис на прийоми та відображення інформації про прийоми.

3.4. Висновки до розділу

В ході виконання даної роботи я здійснив розробку веб-додатка для запису до стоматолога. Робота включала вибір необхідних інструментів розробки, розробку графічного інтерфейсу та написання основного коду системи. Під час виконання роботи я зосередився на створенні зручного та інтуїтивно зрозумілого інтерфейсу для користувачів, а також на забезпеченні ефективної функціональності додатка. Процес розробки включав у себе вибір оптимальних технологій та засобів для досягнення поставленої мети, а також ретельне тестування для забезпечення надійності та стабільності системи. В результаті роботи було створено веб-додаток, який дозволяє ефективно та зручно здійснювати запис до стоматолога, полегшуючи процес як для користувачів, так і для медичного персоналу.

ВИСНОВКИ

У ході виконання дипломної роботи був проведений аналіз існуючого стану питання, що пов'язане з управлінням медичними закладами. Оглянута предметна область та проведено дослідження щодо існуючих рішень у цій галузі. Виявлені недоліки та можливості для вдосконалення організації роботи медичних закладів.

На основі аналізу було розроблено систему управління медичними закладами, яка включає в себе проектування бази даних, функціоналу системи та внутрішньої будови. Розроблені рішення спрямовані на поліпшення організації роботи закладів та зручність обслуговування пацієнтів.

Для реалізації системи обрані необхідні інструменти розробки, розроблено графічний інтерфейс та основний код системи.

Після завершення розробки було проведено тестування системи з метою виявлення та усунення помилок та недоліків. В результаті тестування система продемонструвала свою працездатність та здатність вирішувати поставлені завдання.

Отже, виконана дипломна робота дозволила розробити та впровадити систему управління медичними закладами, яка сприятиме підвищенню якості медичного обслуговування та ефективності управління, відповідаючи сучасним вимогам та технологіям.

Мета дослідження була досягнута, а отримані результати можуть бути використані для подальших досліджень та вдосконалення систем управління медичними закладами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. DRM Associates. "New Product Development Glossary". 2002. URL: https://www.npd-solutions.com/glossary.html (дата звернення: 15.01.2024).
2. Knight Linda V., Steinbach Theresa A., Kellen Vince. "System Development Methodologies for Web-Enabled E-Business: A Customization Framework". DePaul University, USA; Blue Wolf, USA.
3. Morris Joseph M. "Software Industry Accounting". 2001. С. 1.10.
4. Davis Alan M. "Great Software Debates". Wiley-IEEE Computer Society Press, 2004. Сс. 125-128.
5. Otero Carlos. "Software Design Challenges". IT Performance Improvement. Taylor & Francis LLC. URL: https://www.ndsu.edu/fileadmin/cs/course_info/716.pdf (дата звернення: 17.01.2024).
6. Barkmeyer Edward J. та ін. "Concepts for Automating Systems Integration". NIST, 2003. URL: https://www.researchgate.net/publication/242281654_Concepts_for_Automating_Systems_Integration (дата звернення: 20.12.2023).
7. Smith Paul R., Sarfaty Richard. "Creating a strategic plan for configuration management using Computer Aided Software Engineering (CASE) tools". Paper For 1993 National DOE/Contractors and Facilities CAD/CAE User's Group, 1993.
8. Kuhn D.L. "Selecting and effectively using a computer-aided software engineering tool". Annual Westinghouse computer symposium; 6-7 Nov 1989; Pittsburgh, PA (USA); DOE Project, 1989.
9. Loucopoulos P., Karakostas V. "System Requirements Engineering". McGraw-Hill, 1995.

10. "CASE". In: Telecom Glossary 2000.
11. Robinson K. "Putting the Software Engineering into CASE". John Wiley and Sons Inc., New York, 1992.
12. Xiao He. "A metamodel for the notation of graphical modeling languages". In: Computer Software and Applications Conference, COMPSAC 2007 – Vol. 1. 31st Annual International, Volume 1, Issue, 24–27 July 2007, Cc. 219-224.
13. Merx Georges G., Norman Ronald J. "Unified Software Engineering with Java". Prentice-Hall, Inc., 2006. C. 201. ISBN 0130473766.
14. Kit Edward. "Software Testing in The Real World". Addison-Wesley Professional, 1992. ISBN 0201877562.
15. McCarthy Jim. "Dynamics of Software Development". Microsoft Press, 1995. ISBN 1556158238.
16. Conde Dan. "Software Product Management: Managing Software Development from Idea to Product to Marketing to Sales". Aspatore Books, 2002. ISBN 1587622025.
17. Davis A. M. "Just enough requirements management: Where software development meets marketing". Dorset House Publishing Company, Incorporated, 2005. ISBN 0932633641.
18. Hasted Edward. "Software That Sells: A Practical Guide to Developing and Marketing Your Software Project". Wiley Publishing, 2005. ISBN 0764597833.
19. Hohmann Luke. "Beyond Software Architecture: Creating and Sustaining Winning Solutions". Addison-Wesley Professional, 2003. ISBN 0201775948.
20. Horch John W. "Two Orientations On How To Work With Objects". In: IEEE Software. vol. 12, no. 2, c. 117–118, Mar., 1995.

21. Rittinghouse John. "Managing Software Deliverables: A Software Development Management Methodology". Digital Press, 2003. ISBN 155558313X.
22. Wiegers Karl E. "More About Software Requirements: Thorny Issues and Practical Advice". Microsoft Press, 2005. ISBN 0735622671.
23. Wysocki Robert K. "Effective Software Project Management". Wiley, 2006. ISBN 0764596365.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМНОГО КОДУ (БЕКЕНД)

```
    from flask import Flask, render_template, request, redirect,
url_for, session
    from flask_sqlalchemy import SQLAlchemy
    from datetime import datetime

    app = Flask(__name__)
    app.secret_key = 'very_secret_key'
    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///clinic.db'
    db = SQLAlchemy(app)

    # Моделі
    class Doctor(db.Model):
        id = db.Column(db.Integer, primary_key=True)
        name = db.Column(db.String(100), nullable=False)
        specialty = db.Column(db.String(100))

        appointments = db.relationship('Appointment', backref='doctor',
lazy=True)

    class Patient(db.Model):
        id = db.Column(db.Integer, primary_key=True)
        name = db.Column(db.String(100), nullable=False)
        email = db.Column(db.String(100), unique=True, nullable=False)

        appointments = db.relationship('Appointment', backref='patient',
lazy=True)

    class Appointment(db.Model):
        id = db.Column(db.Integer, primary_key=True)
        date_time = db.Column(db.DateTime, nullable=False)
        doctor_id = db.Column(db.Integer, db.ForeignKey('doctor.id'),
nullable=False)
        patient_id = db.Column(db.Integer, db.ForeignKey('patient.id'),
nullable=False)


    with app.app_context():
        db.create_all()


    @app.template_filter('format_datetime')
    def format_datetime(value, format='%d.%m.%Y %H:%M'):
        if value is None:
            return ""
        return value.strftime(format)
```

```
# Маршрути
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/admin', methods=['GET', 'POST'])
def admin_login():
    if request.method == 'POST':
        if request.form['username'] == 'admin' and
request.form['password'] == 'admin':
            return redirect(url_for('admin_dashboard'))
    return render_template('admin_login.html')

@app.route('/admin/dashboard')
def admin_dashboard():
    doctors = Doctor.query.all()
    # Сортування прийомів за датою від найновіших до найстаріших
    appointments =
Appointment.query.order_by(Appointment.date_time.desc()).all()
    return render_template('admin_dashboard.html', doctors=doctors,
appointments=appointments)



@app.route('/patient', methods=['GET', 'POST'])
def patient_login():
    if request.method == 'POST':
        email = request.form.get('email')
        patient = Patient.query.filter_by(email=email).first()
        if patient:
            session['patient_id'] = patient.id
            return redirect(url_for('patient_dashboard',
patient_id=patient.id))
    return render_template('patient_login.html')

@app.route('/patient/dashboard/<int:patient_id>')
def patient_dashboard(patient_id):
    patient = Patient.query.get(patient_id)
    # Сортування прийомів пацієнта за датою
    appointments =
Appointment.query.filter_by(patient_id=patient_id).order_by(Appointment.d
ate_time.desc()).all()
    return render_template('patient_dashboard.html',
patient=patient, appointments=appointments)


@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        name = request.form.get('name')
        email = request.form.get('email')
        new_patient = Patient(name=name, email=email)
        db.session.add(new_patient)
```

```
            db.session.commit()
            return redirect(url_for('patient_login'))
        return render_template('register.html')

    @app.route('/admin/add_doctor', methods=['GET', 'POST'])
    def add_doctor():
        if request.method == 'POST':
            name = request.form.get('name')
            specialty = request.form.get('specialty')
            new_doctor = Doctor(name=name, specialty=specialty)
            db.session.add(new_doctor)
            db.session.commit()
            return redirect(url_for('admin_dashboard'))
        return render_template('add_doctor.html')

    @app.route('/admin/add_appointment', methods=['GET', 'POST'])
    def add_appointment():
        if request.method == 'POST':
            doctor_id = request.form.get('doctor_id')
            patient_id = request.form.get('patient_id')
            date_time_str = request.form.get('date_time')
            date_time   =   datetime.strptime(date_time_str,   '%Y-%m-%d
%H:%M')
            new_appointment      =      Appointment(doctor_id=doctor_id,
patient_id=patient_id, date_time=date_time)
            db.session.add(new_appointment)
            db.session.commit()
            return redirect(url_for('admin_dashboard'))
        doctors = Doctor.query.all()
        patients = Patient.query.all()
        return  render_template('add_appointment.html',  doctors=doctors,
patients=patients)

    @app.route('/logout')
    def logout():
        session.pop('patient_id', None)
        return redirect(url_for('home'))

    @app.route('/patient/book_appointment', methods=['GET', 'POST'])
    def book_appointment():
        if 'patient_id' not in session:
            return redirect(url_for('patient_login'))

        patient_id = session['patient_id']
        if request.method == 'POST':
            doctor_id = request.form.get('doctor_id')
            date_time_str = request.form.get('date_time')
            date_time    =    datetime.strptime(date_time_str,    '%Y-%m-
%dT%H:%M')
            if    not    Appointment.query.filter_by(doctor_id=doctor_id,
date_time=date_time).first():
                new_appointment    =    Appointment(doctor_id=doctor_id,
patient_id=patient_id, date_time=date_time)
```

```
                db.session.add(new_appointment)
                db.session.commit()
            return                redirect(url_for('patient_dashboard',
patient_id=patient_id))

        doctors = Doctor.query.all()
        return render_template('book_appointment.html', doctors=doctors)


    if __name__ == '__main__':
        app.run(debug=True)
```