МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА CRM-СИСТЕМИ ДЛЯ МЕДИЧНОГО ЗАКЛАДУ НА БАЗІ ASP.NET CORE ІЗ ЗАСТОСУВАННЯМ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Б.І. Москвичов
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Лимаренко Ю.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент професор кафедри комп'ютерних наук, професор, д.т.н. Шило Г.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Москвичову Богдану Ігоровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка CRM-системи для медичного закладу на базі ASP.NET Core із застосуванням мікросервісної архітектури

керівник роботи Лимаренко Юлія Олексіївна, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та реалізація CRM-системи для медичного закладу.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	01.04.2024	
2.	Збір вихідних даних.	30.04.2024	
3.	Обробка методичних та теоретичних	05.06.2024	
	джерел.		
4.	Розробка першого та другого розділу.	02.09.2024	
5.	Розробка третього розділу.	24.10.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	12.12.2024	

Студент _____ Б.І. Москвичов
(підпис) (ініціали та прізвище)

Керівник роботи _____ Ю.О. Лимаренко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка CRM-системи для медичного закладу на базі ASP.NET Core із застосуванням мікросервісної архітектури»: 66 с., 38 рис., 20 джерел.

БАЗИ ДАНИХ, ІНТЕГРАЦІЯ, МЕДИЧНИЙ ЗАКЛАД, МІКРОСЕРВІСНА АРХІТЕКТУРА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, CRM-СИСТЕМА, .NET, ASP.NET CORE, ENTITY FRAMEWORK, MS SQL SERVER, REST API.

Об’єкт дослідження – технології розробки CRM-систем для медичних закладів.

Предмет дослідження – процес розробки та впровадження CRM-системи для медичного закладу (Sfit Studio), яка базується на мікросервісній архітектурі та використовує технології ASP.NET Core.

Мета роботи: розробка CRM-системи для медичного закладу, яка забезпечить ефективне управління даними пацієнтів, медичними послугами, персоналом та іншими аспектами функціонування медичного закладу, з використанням сучасних технологій мікросервісної архітектури на платформі ASP.NET Core.

Метод дослідження – моделювання, проєктування, програмний, аналітичний.

У даній роботі здійснено розробку CRM-системи для медичного закладу, що дозволяє автоматизувати процеси управління пацієнтами, лікарями, медичними записами та іншими важливими аспектами діяльності медичного закладу. Основною особливістю системи є її побудова на основі мікросервісної архітектури, що забезпечує масштабованість, гнучкість та надійність. У роботі детально розглянуто вибір технологій, процеси інтеграції та забезпечення безпеки даних, а також проведено тестування та оцінку ефективності розробленої системи.

SUMMARY

Master’s qualifying paper “Development of a CRM System for a Medical Institution Based on ASP.NET Core Using a Microservices Architecture”: 66 pages, 38 figures, 20 references.

ASP.NET CORE, DATABASES, ENTITY FRAMEWORK, INTEGRATION, MEDICAL FACILITY, MICROSERVICE ARCHITECTURE, MS SQL SERVER, PROGRAM SOFTWARE, REST API, CRM SYSTEM, .NET.

The object of the study is the technologies for developing CRM systems for healthcare institutions.

The subject of the study is the process of developing and implementing a CRM system for a healthcare institution (Sfit Studio), based on microservice architecture and utilizing ASP.NET Core technologies.

The aim of the study is to develop a CRM system for a healthcare facility that ensures effective management of patient data, medical services, staff, and other aspects of the institution’s operations, using modern microservice architecture technologies on the ASP.NET Core platform.

The methods of research are modeling, design, software, and analytical methods.

This paper presents the development of a CRM system for a healthcare institution, which automates the management of patients, doctors, medical records, and other essential processes within the institution. The main feature of the system is its construction based on microservice architecture, which ensures scalability, flexibility, and reliability. The work also thoroughly discusses the choice of technologies, integration processes, data security measures, and includes testing and evaluation of the developed system’s effectiveness.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз потреб медичних закладів та існуючих CRM-систем..... 10
1.1 Аналіз ведення обліку в медичному закладі «SfitStudio» на поточний момент..... 10
1.2 Аналіз існуючих CRM-систем..... 11
1.2.1 BloknotApp..... 11
1.2.2 EasyWeek..... 12
1.2.3 CleverBox 14
1.2.4 Обґрунтування написання власної CRM-системи..... 15
1.3 Визначення вимог до CRM-системи для медичного закладу..... 17
2 Проєктування..... 19
2.1 Обґрунтування вибору інструментів розробки та проєктування..... 19
2.1.1 Огляд C# та ASP.NET Core..... 19
2.1.2 Огляд СУБД Microsoft SQL Server та ORM Entity Framework 22
2.1.3 Огляд інструментів для розробки клієнтської частини 24
2.1.4 Використання UML під час проєктування системи..... 27
2.2 Архітектурні аспекти застосунку 27
2.2.1 Структура frontend-проєкту 27
2.2.2 Структура backend-проєкту 39
3 Реалізація..... 53
Висновки 63
Перелік посилань..... 65

ВСТУП

У сучасному світі ефективне управління медичними закладами стає все більш критичним завданням. Зростання кількості пацієнтів, ускладнення медичних послуг та збільшення обсягу даних вимагають від медичних установ впровадження сучасних інформаційних систем для оптимізації роботи та підвищення якості обслуговування. Одним із найефективніших інструментів для досягнення цих цілей є CRM-системи (Customer Relationship Management), які дозволяють автоматизувати та покращити процеси взаємодії з пацієнтами, управління персоналом та ресурсами закладу.

Розробка CRM-системи для медичного закладу (Sfit Studio) на базі ASP.NET Core із застосуванням мікросервісної архітектури є актуальною з кількох причин. По-перше, ASP.NET Core є сучасним, високопродуктивним та кросплатформенним фреймворком для розробки вебдодатків, який підтримує широкий спектр можливостей для створення надійних і масштабованих рішень. По-друге, мікросервісна архітектура дозволяє розробляти гнучкі системи, які можуть легко адаптуватися до змінних вимог бізнесу, забезпечувати високу доступність та легкість інтеграції з іншими системами.

Впровадження CRM-системи на базі таких технологій сприятиме підвищенню ефективності роботи медичного закладу, покращенню якості медичних послуг, зменшенню часу на обробку даних та підвищенню задоволеності пацієнтів.

Метою дослідження є розробка CRM-системи для медичного закладу, яка забезпечить ефективне управління даними пацієнтів, медичними послугами, персоналом та іншими аспектами функціонування медичного закладу, з використанням сучасних технологій мікросервісної архітектури.

Досягнення поставленої мети передбачає реалізацію наступних завдань:

– проаналізувати існуючі рішення для управління медичними даними та CRM-системи;

– дослідити принципи мікросервісної архітектури та її застосування у медичних установах;
– розробити архітектуру CRM-системи для медичного закладу;
– спроєктувати базу даних та систему взаємодії між мікросервісами;
– протестувати реалізовану систему з метою оцінки її ефективності та безпеки.

Об’єкт дослідження – технології розробки CRM-систем для медичних закладів.

Предмет дослідження – процес розробки та впровадження CRM-системи для медичного закладу, яка базується на мікросервісній архітектурі та використовує технології ASP.NET Core.

Для досягнення результату було обрано мову програмування C#, оскільки вона є потужним інструментом для розробки на платформі .NET, забезпечуючи високу продуктивність та інтеграцію з іншими технологіями. Для створення вебзастосунку використано ASP.NET Core, який дозволяє будувати масштабовані, високопродуктивні додатки на основі мікросервісної архітектури. Для фронтенду застосовуються HTML, CSS та JavaScript, що дає можливість створити інтерактивний інтерфейс користувача.

Для зберігання даних було вибрано реляційну базу даних MS SQL Server, оскільки вона забезпечує надійність, швидкість обробки запитів та підтримку складних запитів для великих обсягів даних. Для роботи з базою даних використано Entity Framework, що дозволяє зручно працювати з даними через об’єктно-орієнтовану модель, знижуючи рівень складності взаємодії з базою даних.

Розроблено систему CRM для конкретного медичного закладу – Sfit Studio, який вже впровадив її в своїй діяльності. Система автоматизує управління даними пацієнтів, медичними записами, послугами та персоналом, що сприяє покращенню ефективності роботи клініки.

Отримані результати дозволяють оптимізувати процеси взаємодії з пацієнтами та медичним персоналом, знижуючи час на обробку даних та

полегшуючи доступ до важливої інформації. Теоретичне значення роботи полягає в дослідженні і впровадженні мікросервісної архітектури в медичну сферу, а практичне – в створенні готової, масштабованої CRM-системи, що може бути застосована не тільки в Sfit Studio, а й в інших медичних установах для автоматизації внутрішніх процесів.

Основні положення і результати кваліфікаційної роботи доповідались, обговорювались та знайшли схвалення на науково-практичній конференції молодих дослідників «Актуальні проблеми математики та інформатики» (Запоріжжя, 25-26 квітня 2024 р.), за результатами якої опубліковано тези.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та переліку посилань. Загальний обсяг роботи становить 66 сторінок. У переліку посилань використано 20 джерел.

Перший розділ присвячений теоретичному аналізу існуючих підходів до розробки CRM-систем для медичних закладів.

У другому розділі розглянуто процес проєктування архітектури та інтеграції мікросервісів у рамках CRM-системи.

В третьому розділі описано реалізацію розробленої CRM-системи на платформі ASP.NET Core.

1 АНАЛІЗ ПОТРЕБ МЕДИЧНИХ ЗАКЛАДІВ ТА ІСНУЮЧИХ CRM-СИСТЕМ

1.1 Аналіз ведення обліку в медичному закладі «SfitStudio» на поточний момент

Власниця медичного закладу «SFitStudio», розташованого у м. Запоріжжі, який спеціалізується на косметологічних процедурах, масажах та інших пов’язаних послугах, звернулася із проханням розробити CRM-систему для покращення обліку та управління даними. Заклад функціонує вже кілька років і має стабільну, що постійно розширюється, базу клієнтів. Зараз всі записи про клієнтів, прийоми та інші адміністративні функції ведуться в Excel, але через зростання кількості клієнтів та даних, це стає дедалі більш незручним та обтяжливим.

Основні проблеми ведення обліку в Excel, з якими стикається даний медичний заклад:

– незручність та обмеження функціоналу: Excel не забезпечує достатньої функціональності для комплексного управління даними про клієнтів, історією їхніх записів та іншими важливими аспектами роботи закладу;
– ризики втрати даних: відсутність надійних механізмів захисту та резервного копіювання даних призводить до ризиків їх втрати або пошкодження;
– відсутність інтеграції: Excel не дозволяє легко інтегруватися з іншими системами та сервісами, такими як онлайн-записи, електронна пошта або SMS-розсилки;
– трудомісткість оновлення даних: усі зміни потрібно вносити вручну, що забирає багато часу та може призводити до помилок.

Через вищенаведені пункти власниця хоче замовити невелику CRM-систему для покращення ефективності роботи.

1.2 Аналіз існуючих CRM-систем

На українському ринку CRM-систем є кілька рішень, які можуть бути корисними для таких медичних закладів. Розглянемо найпопулярніші з них.

1.2.1 BloknotApp

Почнемо розглядати одну з популярних CRM-систем BloknotApp (див. рис. 1.1).

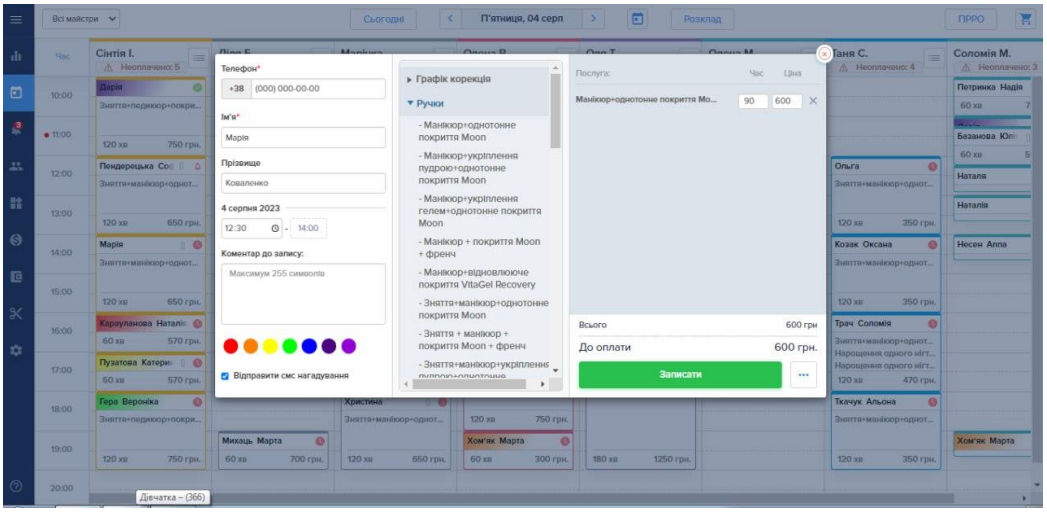


Рисунок 1.1 – Інтерфейс BloknotApp

BloknotApp широко використовується в косметологічних салонах. Ця система має кілька ключових переваг. Перш за все, BloknotApp пропонує простий та інтуїтивний інтерфейс, який легко освоїти навіть користувачам без технічної підготовки. Це робить систему доступною для всіх працівників салону, незалежно від їхнього рівня технічних знань. Крім того, BloknotApp забезпечує можливість клієнтам записуватись на прийом через вебсайт або мобільний додаток, що значно спрощує процес запису та підвищує зручність для клієнтів.

Однією з важливих функцій BloknotApp є управління розкладом працівників. Система дозволяє зручно планувати розклад, автоматично

оновлювати записи та надсилати нагадування про прийоми, що мінімізує можливість помилок і пропущених зустрічей. Додатково, BloknotApp надає інструменти для аналізу діяльності салону, включаючи звіти про доходи та ефективність. Це дозволяє власникам салонів отримувати цінні аналітичні дані для прийняття обґрунтованих рішень щодо управління бізнесом.

BloknotApp – корисний інструмент для багатьох, але у великих салонах краси його функціональність може бути обмежена [1]. У цьому програмному забезпеченні присутні не всі функції, необхідні для великого бізнесу: детальний аналіз продажів, Гнучка настройка маркетингових кампаній, інтеграція з різними системами (бухгалтерські програми, системи лояльності і т.д.). Відсутність можливості масштабуватися та адаптуватися до потреб зростаючого бізнесу може означати, що компанії, можливо, доведеться перейти на іншу платформу, що передбачає додаткові витрати часу та ресурсів. Крім того, обмежений набір інструментів для управління персоналом може ускладнити організацію роботи великих команд і відстеження їх ефективності.

Таким чином, BloknotApp є ефективним рішенням для невеликих та середніх косметологічних салонів, що шукають просту у використанні систему з базовими функціями для управління записами та розкладом. Завдяки інтуїтивному інтерфейсу та мінімальним вимогам до впровадження, програма може швидко стати незамінним інструментом для оптимізації щоденних процесів. Однак, великі салони з більш складними потребами можуть зіткнутися з певними обмеженнями функціональності та інтеграції, що може вимагати пошуку більш масштабованих альтернатив.

1.2.2 EasyWeek

Розглянемо наступну популярну CRM-систему – EasyWeek, яка також знайшла широке застосування у косметологічних салонах (див. рис. 1.2).

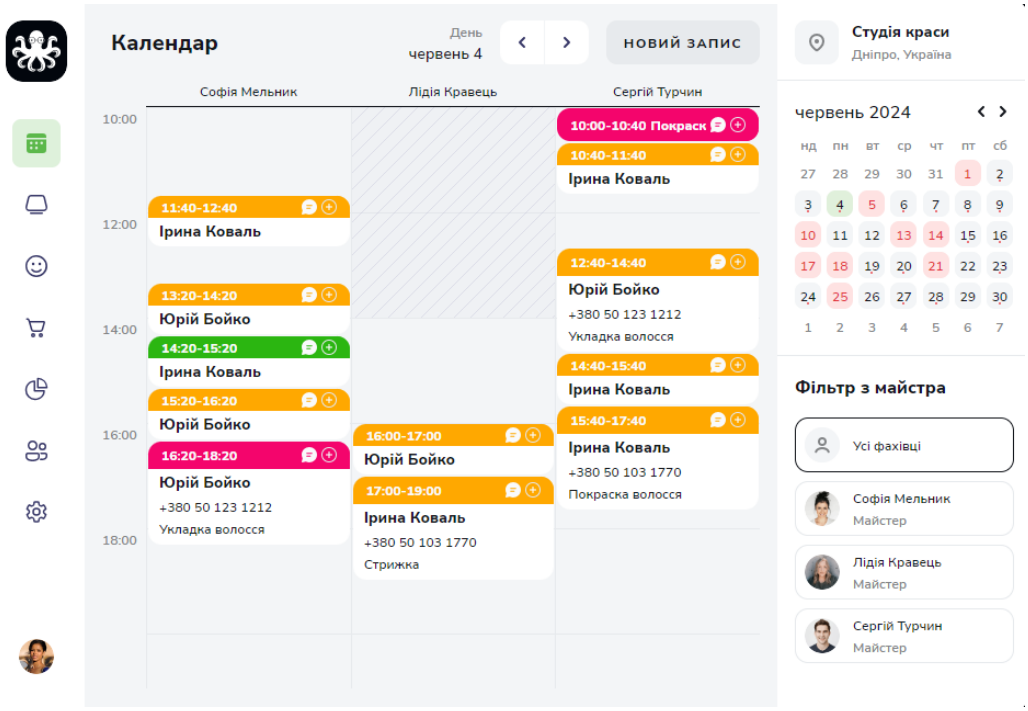


Рисунок 1.2 – Інтерфейс CRM-системи EasyWeek

EasyWeek має кілька ключових переваг. По-перше, EasyWeek відзначається багатофункціональністю, пропонуючи широкий спектр можливостей. Вона включає онлайн-запис, управління розкладом, базу клієнтів та фінансовий облік, що робить її комплексним рішенням для ефективного управління салоном [2].

Інтерфейс EasyWeek є інтуїтивним та зручним для користувачів, що значно полегшує його освоєння та використання у повсякденній роботі. Крім того, система забезпечує широкий спектр маркетингових інструментів, таких як розсилка повідомлень, проведення акцій та програми лояльності, що допомагає залучати нових клієнтів та утримувати існуючих. EasyWeek також підтримує мобільний додаток, який надає зручний доступ до CRM-системи з будь-якого місця, що підвищує мобільність та ефективність роботи співробітників салону.

Проте, незважаючи на численні переваги, EasyWeek має і певні недоліки. Для деяких невеликих салонів вартість підписки на систему може виявитися високою, що може стати фінансовим обтяженням. Крім того, можливості інтеграції EasyWeek з деякими зовнішніми системами можуть

бути обмеженими, що може створювати труднощі для салонів, які використовують інші програмні продукти та потребують їхньої взаємодії з CRM-системою.

Отже, EasyWeek є потужним та функціональним рішенням для косметологічних салонів, що шукають комплексну CRM-систему з багатими можливостями управління та маркетингу. Однак, висока вартість та обмежені можливості інтеграції можуть бути вагомими факторами при виборі цієї системи для невеликих салонів.

1.2.3 CleverBox

Також розглянемо ще одну CRM-систему, яка є популярним рішенням для інтеграції в медичних закладах формату «SfitStudio» (див. рис. 1.3).

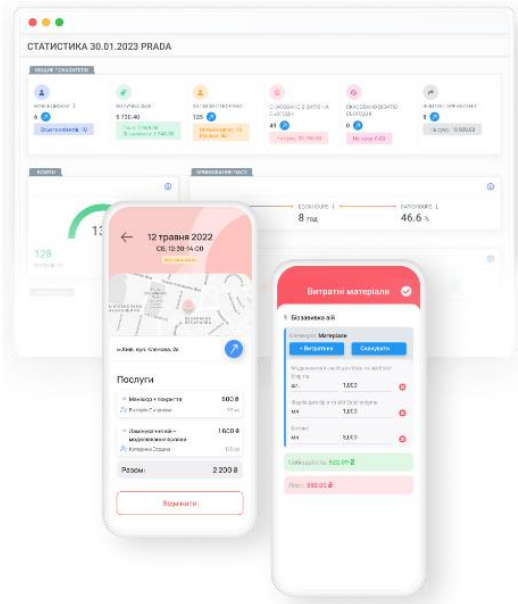


Рисунок 1.3 – Інтерфейс CRM-системи CleverBox

CleverBox має низку суттєвих переваг. Перш за все, ця система відзначається високою гнучкістю та можливістю налаштування. CleverBox дозволяє користувачам адаптувати функціонал під конкретні потреби їхнього бізнесу, що робить її універсальною для різних типів салонів [3].

CleverBox не тільки автоматизує роботу салонів краси, але і піднімає рівень обслуговування клієнтів на нову висоту. Завдяки докладній історії кожного відвідувача, співробітники салону можуть персоналізувати свою пропозицію і запропонувати кожному клієнту індивідуальний підхід. Потужні інструменти аналітики допомагають виявляти тенденції та уподобання клієнтів, що дозволяє розробляти більш ефективні маркетингові кампанії та професійні послуги. Крім того, завдяки можливості проведення інформаційних розсилок та рекламних акцій салон завжди може залишатися на зв’язку з клієнтами та інформувати їх про нові продукти та спеціальні пропозиції.

Проте, CleverBox має і певні недоліки. Для деяких користувачів система може здатися складною у налаштуванні та освоєнні, що може вимагати додаткового часу та ресурсів на навчання персоналу. Також висока вартість CleverBox може бути обмежувальним фактором для невеликих салонів, які прагнуть мінімізувати свої витрати.

Як результат, CleverBox – це гнучкий та потужний інструмент управління салонами краси, який надає розширені можливості налаштування, управління клієнтами та аналітики. Однак складність і висока вартість настройки можуть стати серйозною перешкодою для деяких салонів при виборі цієї системи.

1.2.4 Обґрунтування написання власної CRM-системи

Після детального огляду існуючих CRM-систем, таких як BloknotApp, EasyWeek, CleverBox можна зробити висновок про доцільність розробки власної CRM-системи на основі ASP.NET Core та мікросервісної архітектури.

Це рішення краще відповідає потребам «SFitStudio» з наступних причин.

Висока вартість існуючих систем. Багато з існуючих CRM-систем мають високу вартість підписки, що може бути надмірним фінансовим тягарем для невеликого косметологічного салону. Наприклад, EasyWeek та CleverBox

пропонують широкий функціонал, але їхня вартість може бути занадто високою для «SFitStudio», який прагне оптимізувати свої витрати. Розробка власної системи дозволить уникнути регулярних витрат на підписку та забезпечити повний контроль над бюджетом.

Надмірна функціональність. Багато існуючих CRM-систем, таких як EasyWeek та CleverBox, пропонують дуже широкий спектр функцій, які можуть виявитися надмірними для потреб невеликих перукарень. Використання такої системи може ускладнити роботу персоналу, оскільки існує занадто багато налаштувань і функцій, які не використовуються в повсякденній роботі салону. За допомогою нашої фірмової CRM-системи ви можете створити простий і інтуїтивно зрозумілий інтерфейс, який відповідає специфічним потребам SFitStudio без надмірної функціональності.

Гнучкість та масштабованість. Розробка власної CRM-системи на базі мікросервісної архітектури забезпечує високу гнучкість та масштабованість. Мікросервіси дозволяють легко додавати нові функції та модулі без значних змін у системі. Це особливо важливо для «SFitStudio», який постійно розширює базу клієнтів та обсяг послуг. Власна система дозволить швидко адаптуватися до змінних вимог бізнесу.

Інтеграція з існуючими системами. Багато існуючих CRM-систем мають обмежені можливості інтеграції з іншими системами та сервісами, які вже використовуються у салоні, такими як бухгалтерські програми, системи для управління запасами та інші інструменти. Власна CRM-система, розроблена на базі ASP.NET Core, може бути легко інтегрована з будь-якими існуючими системами через API або інші методи інтеграції, забезпечуючи єдину екосистему для всіх бізнес-процесів салону.

Високий рівень захисту даних. Збереження конфіденційності даних клієнтів є критично важливим для косметологічного салону. Власна CRM-система дозволяє впроваджувати необхідні заходи безпеки та захисту даних відповідно до вимог салону. Використання ASP.NET Core, який підтримує сучасні стандарти безпеки, забезпечить високий рівень захисту даних від несанкціонованого доступу та втрати.

Локалізація та відповідність місцевим потребам. Багато з існуючих CRM-систем можуть не повністю відповідати місцевим особливостям та вимогам ринку України. Власна CRM-система дозволить врахувати специфіку ведення бізнесу в Україні, такі як мовні налаштування, локальні закони та регуляції, що зробить її більш зручною для використання.

Отже, розробка власної CRM-системи на базі ASP.NET Core та мікросервісної архітектури для косметологічного салону «SFitStudio» є оптимальним рішенням, яке забезпечить необхідний баланс між вартістю, функціональністю та гнучкістю. Це дозволить салону ефективно керувати базою клієнтів, покращити якість обслуговування та оперативно адаптуватися до змінних вимог бізнесу, забезпечуючи високий рівень захисту даних та відповідність місцевим потребам.

1.3 Визначення вимог до CRM-системи для медичного закладу

У сучасному світі, особливо в галузі медицини та косметології, ефективне управління клієнтською базою визначає конкурентоспроможність закладу. Саме тут набуває великого значення використання Customer Relationship Management (CRM) систем, які спрямовані на підтримку і оптимізацію взаємодії з клієнтами. Розробка CRM-системи для медичного закладу, зокрема для косметологічного салону, має бути націленою на забезпечення широкого спектру функціональних можливостей, які відповідають специфіці даного сегменту.

Першочерговим завданням у визначенні вимог до CRM-системи є ретельний аналіз потреб медичного закладу, зокрема косметологічного салону. До цього входить збір відомостей про специфіку його роботи, обсяги та характер послуг, які надаються, особливості взаємодії з клієнтами, процеси адміністрування, а також проблемні аспекти, які виникають у повсякденній роботі.

Тож розглянемо основні вимоги до застосунку, які, до того ж, були узгоджені з власницею медичного закладу.

Важливим аспектом є управління клієнтською базою. CRM-система повинна забезпечувати можливість ефективного зберігання і обробки інформації про клієнтів, включаючи їх контактні дані, історію відвідувань, переваги і т.д. Це допоможе персоналізованим співробітникам салону надавати послуги і покращувати взаємодію з клієнтами.

Розклад та призначення. Функціонал системи має забезпечити можливість легкого створення та керування розкладом роботи співробітників, а також призначення процедур клієнтам. Крім того, система повинна контролювати доступність послуг та уникати перекриттів у часі, щоб забезпечити ефективну роботу салону.

Звітність та аналітика. Один з ключових аспектів – це можливість генерації звітів та аналізу даних щодо обсягів роботи, фінансових показників та ефективності маркетингових кампаній. Аналіз цих даних дозволить керівництву салону приймати обґрунтовані рішення щодо розвитку бізнесу та покращення обслуговування клієнтів.

Підтримка розсилки нагадувань клієнтам про найближчі візити або заплановані процедури також є важливою функціональністю для CRM-системи косметологічного салону. Це дозволить забезпечити зручність для клієнтів та допоможе уникнути пропуску запланованих візитів, що в свою чергу підвищить рівень задоволеності клієнтів та ефективність роботи салону.

Налаштування системи як вебзастосунку також є важливою вимогою. Наявність вебінтерфейсу забезпечить зручність у використанні та підвищить доступність системи для персоналу салону, незалежно від їхнього місця роботи або пристрою, яким вони користуються.

Підсумовуючи, можемо сказати, що функціонал CRM-системи є критичним елементом для ефективного управління клієнтськими відносинами та оптимізації бізнес-процесів в медичних закладах, забезпечуючи зручність у веденні даних, організації робочого часу та аналізу даних для прийняття обґрунтованих стратегічних рішень.

2 ПРОЄКТУВАННЯ

2.1 Обґрунтування вибору інструментів розробки та проєктування

Вибір технологій для будь-якого проєкту є важливим етапом, оскільки від цього залежить ефективність та успішність розробки, а також майбутнє розширення та підтримка системи. У випадку CRM-системи для медичного закладу, рішення було прийняте на користь розробки із використанням C#, ASP.NET Core, HTML, CSS, JavaScript, MS SQL Server та Entity Framework Core. Такий стек технологій забезпечує повний спектр можливостей для створення високопродуктивного, масштабованого та зручного у використанні додатка.

2.1.1 Огляд C# та ASP.NET Core

Для розробки серверної частини CRM-системи обрано мову програмування C# та вебфреймворк ASP.NET Core, що забезпечить стабільність, продуктивність і масштабованість.

C# є сучасною об’єктно-орієнтованою мовою програмування, розробленою компанією Microsoft, яка активно використовується для створення вебдодатків, корпоративних систем, мобільних застосунків та ігор. Її обирають через численні переваги [4].

Однією з основних особливостей C# є підтримка об’єктно-орієнтованого програмування. Це означає, що мова дозволяє розробникам використовувати принципи інкапсуляції, успадкування та поліморфізму, що сприяє створенню модульного та структурованого коду. Завдяки цьому розробка стає більш ефективною, а підтримка та розширення коду – зручними. Для CRM-системи, яка має працювати з різними типами об’єктів, такими як пацієнти, послуги, записи на прийом та історії процедур, ООП дозволить реалізувати логіку

таким чином, що кожен об’єкт матиме свої властивості та методи, зменшуючи дублювання коду та підвищуючи читабельність.

C# створена з урахуванням вимог до продуктивності сучасних додатків. Завдяки підтримці асинхронного програмування, реалізованого через ключові слова async та await, мова дозволяє обробляти численні запити від користувачів без блокування основного потоку. Це особливо важливо для CRM-систем, де одночасно можуть відбуватися запити на перегляд розкладу, створення нових записів, оновлення даних пацієнтів тощо. Асинхронне програмування забезпечує швидкий відгук системи, підвищуючи її продуктивність і зменшуючи час очікування для користувачів.

Крім того, C# має розвинені механізми обробки помилок, реалізовані через конструкції try-catch-finally [5]. Це дозволяє програмістам передбачати можливі винятки та обробляти їх належним чином, що підвищує стабільність роботи додатка. Також C# підтримує статичну типізацію, яка зменшує ймовірність помилок у коді на етапі компіляції. Мова має вбудовані функції управління пам’яттю за допомогою механізму збирання сміття (Garbage Collection), що знижує ризик виникнення витоків пам’яті, які можуть призвести до збою додатка [6].

Важливо зазначити, що у контексті CRM-систем, де обробляється конфіденційна інформація пацієнтів, безпека є ключовим фактором. C# забезпечує високу безпеку завдяки підтримці типово безпечного коду та захисту від атак, таких як SQL-ін’єкції та XSS [7].

Для безпосереденьо написання backend-частини обрано ASP.NET Core. Це сучасний вебфреймворк, розроблений Microsoft, який дозволяє створювати швидкі, безпечні та кросплатформені вебдодатки. Його вибір для серверної частини CRM-системи обґрунтований рядом переваг [8].

Нова версія ASP.NET Core (.NET 8) є кросплатформеною, тобто вона підтримує розробку додатків для операційних систем Windows, Linux та macOS [9]. Це відкриває більше можливостей для розгортання додатка в різних середовищах, таких як хмарні сервіси (AWS, Azure, Google Cloud) чи локальні

сервери. Завдяки цьому можна вибирати оптимальне середовище розгортання залежно від потреб бізнесу та наявних ресурсів.

ASP.NET Core дозволяє розробляти додатки з використанням мікросервісної архітектури, яка передбачає поділ додатка на невеликі незалежні сервіси. Це означає, що функціональність CRM-системи може бути розділена на окремі сервіси, такі як сервіс управління пацієнтами, сервіс розкладу прийомів, сервіс обробки платежів. Кожен з цих сервісів може розроблятися, тестуватися та розгортатися незалежно один від одного, що знижує ризик виникнення проблем при оновленні системи та спрощує її підтримку.

ASP.NET Core підтримує інтеграцію з сучасними протоколами автентифікації та авторизації, такими як OAuth 2.0, JWT (JSON Web Token) та OpenID Connect [10]. Це дозволяє легко реалізувати безпечний механізм доступу до ресурсів системи, що є особливо важливим для CRM-системи медичного центру, яка зберігає конфіденційні дані пацієнтів. Використання JWT, наприклад, дозволяє створювати токени доступу, які можуть бути легко перевірені на сервері, що забезпечує захищений обмін даними між клієнтом та сервером.

ASP.NET Core відомий своєю високою продуктивністю та оптимізацією, що робить його одним з найшвидших вебфреймворків для розробки серверної частини додатків. Використання Kestrel, вбудованого вебсервера, дозволяє значно зменшити час відповіді сервера та підвищити загальну швидкість роботи додатка. Крім того, підтримка асинхронних методів дозволяє ефективно обробляти одночасні запити від багатьох користувачів, що особливо важливо для великої CRM-системи з високим навантаженням.

Завдяки активній підтримці з боку Microsoft та великій спільноті розробників, ASP.NET Core має розвинений набір інструментів для розробки, тестування та розгортання додатків. Використання Entity Framework Core, ORM-бібліотеки для роботи з базою даних, спрощує доступ до даних і дозволяє працювати з ними через об’єктно-орієнтовані конструкції C#, зменшуючи кількість коду SQL.

Таким чином, вибір мови програмування C# та фреймворку ASP.NET Core забезпечить стабільну, продуктивну та безпечну серверну частину CRM-системи для медичного центру, що відповідає вимогам сучасного бізнесу та дозволяє ефективно масштабувати систему з ростом кількості користувачів та даних.

2.1.2 Огляд СУБД Microsoft SQL Server та ORM Entity Framework

Вибір системи управління базами даних (СУБД) є одним з ключових етапів при розробці будь-якої інформаційної системи. У випадку CRM-системи для медичного закладу було обрано Microsoft SQL Server як основну СУБД, а також Entity Framework як ORM (Object-Relational Mapping) для взаємодії з цією базою даних. Цей вибір є обґрунтованим і забезпечує високий рівень надійності, ефективності та зручності у розробці. Більш детально розглянемо переваги кожної з цих технологій та їхню інтеграцію в рамках розробки CRM-системи.

Microsoft SQL Server є однією з найпопулярніших реляційних систем управління базами даних, що використовується у багатьох підприємствах по всьому світу, завдяки своїй стабільності, високій продуктивності та зручності в адмініструванні. Ця СУБД підтримує широкий набір функціональних можливостей, що робить її ідеальним вибором для розробки комплексних систем, таких як CRM [11].

Однією з основних переваг SQL Server є його висока надійність і масштабованість. Вона може обробляти великі обсяги даних, що є критичним для медичних закладів, де необхідно зберігати та обробляти інформацію про пацієнтів, лікарів, призначення, розклади прийому тощо. SQL Server дозволяє ефективно управляти транзакціями та забезпечує цілісність даних завдяки підтримці ACID-принципів (атомарність, консистентність, ізольованість, довговічність).

Особливу увагу слід приділити підтримці складних запитів і аналітичним можливостям. SQL Server надає потужні інструменти для виконання складних запитів, агрегування та фільтрації даних і може ефективно обробляти великі обсяги медичних даних. Зокрема, використання індексації та оптимізації запитів забезпечує швидкий доступ до даних навіть у високонавантажених великомасштабних системах.

Ще однією важливою перевагою SQL Server є вбудовані засоби для резервного копіювання та відновлення даних, що дозволяє захистити критичну інформацію в разі несподіваних ситуацій. Враховуючи конфіденційність медичних даних, це є важливою частиною системи безпеки.

Для того, щоб прискорити розробку та спростити взаємодію з базою даних, у проєкті використовується ORM (Object-Relational Mapping). ORM – це технологія, яка дозволяє відображати об’єктно-орієнтовані структури програмного коду на реляційну базу даних, забезпечуючи зручний спосіб маніпулювання даними без необхідності написання складних SQL-запитів [12].

Вибір Entity Framework (EF) як ORM для роботи з Microsoft SQL Server обумовлений його потужними можливостями, що значно спрощують взаємодію між програмним кодом і базою даних [13]. EF дозволяє працювати з базою даних через об’єкти класів, що у свою чергу забезпечує більшу зручність, гнучкість і безпеку в процесі розробки. Відмова від написання низькорівневих SQL-запитів дозволяє зменшити обсяг коду, зробити його більш читабельним і зручним для підтримки, одночасно зберігаючи високий рівень контролю над структурами та взаємодією з даними.

EF дозволяє здійснювати відображення об’єктів класів на таблиці бази даних через механізм Code First або Database First, що дає розробникам можливість вибору найбільш зручного підходу залежно від потреб проєкту. У випадку CRM-системи для медичного закладу, використання Code First підходу дозволяє розробникам створювати моделі даних у вигляді класів C# і автоматично генерувати базу даних з цих класів, що спрощує інтеграцію між бізнес-логікою та базою даних.

Entity Framework також підтримує асинхронні операції, що дозволяє значно підвищити продуктивність при роботі з великою кількістю запитів або при обробці даних на стороні сервера. Це важливо для CRM-системи, оскільки медичні заклади часто мають велику кількість пацієнтів та запитів, які потребують обробки в режимі реального часу.

Однією з основних переваг EF є безпека та продуктивність. ORM-підхід забезпечує захист від SQL-ін’єкцій, оскільки усі запити автоматично параметризуються, що знижує ризик помилок або зловживань з боку зловмисників. Крім того, Entity Framework дозволяє легко виконувати міграції (наприклад, при зміні схеми бази даних) без необхідності вручну змінювати таблиці, що зменшує ймовірність виникнення помилок.

Інтеграція Microsoft SQL Server і Entity Framework дозволяє створити потужну, масштабовану і безпечну систему для управління медичними даними. Завдяки високому рівню узгодженості даних ORM спрощує розробку і надає потужні можливості для виконання складних запитів і аналізу.

Використання Microsoft SQL Server разом із Entity Framework дозволяє створити гнучку структуру даних для CRM-системи, яка обробляє записи пацієнтів, медичні дані, розклад лікарів, платежі тощо. Ці технології забезпечують високу продуктивність, доступність і спрощують підтримку та розширення системи, відповідаючи потребам сучасних медичних установ.

2.1.3 Огляд інструментів для розробки клієнтської частини

У процесі розробки клієнтської частини вебдодатка важливо вибрати правильні інструменти, які допоможуть створити інтерактивний, зручний і адаптивний інтерфейс користувача. Для цієї мети в проєкті використовуються такі інструменти, як HTML, CSS, JavaScript, препроцесор SCSS, а також фреймворк Bootstrap. Кожен з цих інструментів має важливу роль у створенні сучасного вебдодатка, що відповідає вимогам користувачів і бізнесу.

HTML (Hypertext Markup Language) є основним інструментом для створення структури вебсторінок. Це стандартна мова розмітки, яка дозволяє визначити, які елементи будуть присутні на сторінці, як вони будуть взаємодіяти між собою та з користувачем. HTML надає семантичні елементи, такі як <header>, <footer>, <article>, <section>, що полегшує створення доступних і зручних інтерфейсів, а також покращує SEO-оптимізацію та доступність вебресурсів [14].

Використання HTML в поєднанні з CSS і JavaScript дозволяє створювати багатофункціональні і зручні інтерфейси. Він слугує основою для структурування контенту, забезпечує правильну навігацію і взаємодію з іншими елементами на сторінці.

CSS (Cascading Style Sheets) – це мова стилів, яка дозволяє описати візуальний вигляд елементів на вебсторінці. Використовуючи CSS, можна задавати кольори, шрифти, відступи, розміри елементів, а також застосовувати анімації та трансформації, що підвищують інтуїтивність і привабливість інтерфейсу користувача. У сучасному вебдизайні CSS є незамінним інструментом, оскільки він дозволяє створювати адаптивні і естетичні інтерфейси, які однаково добре виглядають на різних пристроях.

CSS також є потужним інструментом для створення адаптивних вебсторінок за допомогою медіа-запитів (media queries). Це дозволяє забезпечити правильне відображення контенту на різних розмірах екранів – від мобільних пристроїв до десктопних моніторів, що особливо важливо в умовах поширення мобільних пристроїв для доступу до інтернет-ресурсів.

Для більш складних проєктів, де необхідно працювати з великими обсягами стилів і організувати їх більш ефективно, використовуються препроцесори CSS, зокрема SCSS (Sassy CSS). SCSS дозволяє використовувати змінні, вкладені правила, міксини та інші можливості, які значно полегшують підтримку та розширення стилів.

Однією з головних переваг SCSS є можливість зручного управління великими файлами стилів, оскільки код стає більш структурованим і зрозумілим. За допомогою SCSS можна створювати змінні для кольорів,

шрифтів або розмірів, що спрощує зміну стилів по всьому проєкту, а також дозволяє створювати повторно використовувані компоненти (мікси) та функції для складніших стилістичних рішень [15].

JavaScript є мовою програмування, що використовується для додавання динамічних елементів на вебсторінки. Без JavaScript більшість вебдодатків були б статичними та неприязними до користувача. Це дозволяє здійснювати операції з даними в реальному часі, реагувати на події користувача, обробляти форми, здійснювати запити до серверів і створювати інтерактивні елементи, такі як модальні вікна, вкладки, слайдери та багато іншого [16].

З JavaScript можна реалізувати різні функціональності, від простих перевірок введених даних до складних динамічних взаємодій з сервером, таких як AJAX-запити [17]. Завдяки асинхронності JavaScript дозволяє забезпечити високий рівень взаємодії користувача з вебдодатком без необхідності перезавантажувати сторінки.

Також для більш швидкої розробки використовуємо у проєкті Bootstrap. Bootstrap є популярним фреймворком CSS, який допомагає швидко створювати адаптивні і мобільні вебсторінки. Він включає в себе набір готових стилів, компонентів і утиліт, які можна використовувати для створення інтерфейсів без необхідності писати великий обсяг коду. Bootstrap надає вже готові компоненти для кнопок, навігаційних панелей, форм, карток, модальних вікон, таблиць та інших елементів інтерфейсу [18].

Однією з головних переваг Bootstrap є його адаптивність – усі компоненти автоматично підлаштовуються під розміри екранів різних пристроїв. Це дозволяє значно скоротити час на розробку та забезпечити кросбраузерну сумісність.

Всі ці інструменти працюють в тандемі, забезпечуючи ефективне створення клієнтської частини вебдодатка. HTML надає структуру, CSS і SCSS відповідають за візуальну частину, JavaScript забезпечує інтерактивність, а Bootstrap дозволяє швидко розгорнути адаптивний інтерфейс. Разом вони дають можливість розробити потужну, зручну та ефективну клієнтську частину, яка задовольнятиме вимоги користувачів.

Підводячи підсумок, можна сказати, що для створення клієнтської частини використовуються сучасні інструменти і технології, що допомагають забезпечити високий рівень продуктивності, простоту підтримки і можливості швидкого розширення функціоналу. Це робить інтерфейс доступним і привабливим для користувача.

2.1.4 Використання UML під час проєктування системи

В процесі проєктування системи медичного закладу була застосована нотація UML (Unified Modeling Language), яка дозволила ефективно візуалізувати, планувати і аналізувати різні аспекти програмної архітектури і функціонування системи. Використовуючи UML-діаграми, ми змогли чітко визначити структуру і взаємодію компонентів і забезпечити зручну комунікацію між усіма учасниками проєкту, включаючи розробників, аналітиків, тестувальників і замовників [19].

Основними діаграмами, які були використані під час проєктування, стали діаграми компонентів, діаграми випадків використання, діаграми класів та діаграми активності. Кожен тип діаграми виконує свою специфічну роль, що дозволяє детально розглянути різні аспекти системи [20].

2.2 Архітектурні аспекти застосунку

Розглянемо основну структуру frontend- та backend проєктів.

2.2.1 Структура frontend-проєкту

Фронтенд частина застосунку розроблена за монолітним підходом, що означає, що весь код користувацького інтерфейсу зосереджений у єдиній

структурі проєкту. Цей підхід є традиційним для вебдодатків, які не потребують високого рівня масштабованості на стороні клієнта. Замість розділення фронтенду на окремі модулі або мікросервіси, вся логіка, стилі, компоненти та скрипти об’єднані в одну систему, що дозволяє забезпечити цілісність та стабільність додатку.

Монолітний підхід до розробки фронтенду має низку переваг, які роблять його привабливим вибором для створення вебдодатків. Перш за все, цей підхід дозволяє спростити процес розробки, оскільки розробники можуть зосередитись на створенні єдиного додатку, не розділяючи код на окремі модулі або сервіси. Логіка додатку зосереджена в одній структурі коду, що значно полегшує розуміння системи, а також спрощує її підтримку та внесення змін. Завдяки меншій кількості абстракцій та простої структури коду, складність проєкту залишається нижчою, особливо на початкових етапах розробки, коли важливо швидко створити основний функціонал.

Розгортання монолітного фронтенду також значно простіше, оскільки весь додаток представлений єдиним пакетом файлів, що містить HTML, CSS та JavaScript. Це дозволяє легко розмістити додаток на сервері або CDN без необхідності координувати розгортання різних фронтенд-сервісів, що знижує ризики і прискорює процес оновлення. Важливою перевагою є також спрощення забезпечення кросбраузерної підтримки, оскільки тестувати потрібно єдиний додаток, а не кілька окремих модулів, що значно знижує ймовірність виникнення помилок.

Цілісність інтерфейсу є ще одним важливим аспектом монолітного підходу. Використовуючи спільні стилі та компоненти, розробники можуть забезпечити єдиний дизайн та користувацький досвід на всіх сторінках додатку. Це не тільки покращує взаємодію користувача з системою, але й значно спрощує внесення змін та доповнень до інтерфейсу, оскільки весь код зосереджений в одному місці, і не потрібно синхронізувати зміни між різними модулями.

Монолітний фронтенд напряму взаємодіє з API Gateway, що значно спрощує маршрутизацію запитів та знижує складність інтеграції з бекендом.

Відсутність потреби у створенні окремих клієнтських мікросервісів зменшує накладні витрати на передачу даних та координацію запитів, що позитивно впливає на загальну продуктивність системи. Інтеграція з API стає більш передбачуваною та простою, а це, своєю чергою, знижує час розробки і дозволяє зосередитися на створенні корисних функцій.

Монолітний підхід до розробки фронтенду також дозволяє знизити витрати на розробку та підтримку. Оскільки відсутня необхідність створювати складні клієнтські архітектури з численними абстракціями, це зменшує потребу у великих командах розробників та використанні спеціалізованих інструментів для управління залежностями. Нова команда може швидше включитися в процес розробки, оскільки структура коду проста і зрозуміла, а це зменшує витрати на навчання та адаптацію.

Нарешті, завдяки сучасним інструментам зборки, таким як Webpack або Vite, можна оптимізувати код, зменшивши розмір файлів та прискоривши завантаження сторінок. Відсутність зайвих клієнтських мікросервісів додатково знижує накладні витрати на обробку запитів, що забезпечує високу продуктивність додатку навіть під час значного навантаження.

Отже, проєкт побудований таким чином, щоб забезпечити зручну навігацію, організацію коду та легкість у підтримці. Він складається з трьох основних компонентів:

– html (html-сторінки);
– папка assets (ресурси);
– файл gulpfile.js (автоматизація).

Фронт проєкту містить багато сторінок, кожна з яких виконує певну функцію в межах системи. Адміністратори (основні користувачі) мають доступ до різних сторінок, що дозволяє їм управляти записами на прийом, даними пацієнтів, лікарським розкладом, оплатами та іншими аспектами роботи медичного центру.

Головна сторінка та сторінки аутентифікації.

Index.html. Це основна сторінка додатку, з якої починається робота користувачів після входу в систему. Вона зазвичай відображає оглядову

панель (дашборд), яка містить статистичні дані, основні показники діяльності центру та посилання на основні розділи системи. Може включати такі елементи: графіки, список останніх записів на прийом, сповіщення про майбутні події та інші індикатори стану системи.

Login.html. Сторінка для входу в систему. Включає форму з полями для введення логіна та пароля. Після успішного входу перенаправляє користувача на відповідний дашборд.

Error-page.html. Використовується для відображення помилок, таких як 404 (сторінка не знайдена) або 500 (внутрішня помилка сервера). Включає посилання для повернення на головну сторінку або сторінку входу.

Сторінки з відображенням інформації про лікарів.

Doctor-dashboard.html. Головна сторінка для лікаря, яка містить інформацію про розклад прийомів, список пацієнтів, новини медичного центру та інші індикатори. Вона слугує швидким доступом до основних функцій лікаря, таких як перегляд графіку, нових записів та історії прийомів. Може містити інтегровані графіки з даними про активність лікаря (кількість прийомів, нових пацієнтів тощо).

Doctor-schedule.html. Сторінка для управління розкладом лікаря. Включає інтерактивний календар, де лікар може переглядати свої прийоми та додавати нові записи. Дозволяє легко керувати своїм графіком, відзначати вільні та зайняті слоти.

Add-doctor.html. Форма для додавання нового лікаря в систему. Використовується адміністраторами або менеджерами медичного центру для внесення даних про нового співробітника, включаючи ім’я, спеціалізацію, контактні дані та графік роботи.

Edit-doctor.html. Сторінка для редагування даних про лікаря. Дозволяє змінити інформацію про спеціалізацію, досвід роботи, графік прийомів та контактні дані.

All-doctors.html. Сторінка зі списком усіх лікарів медичного центру. Включає таблицю з даними про лікарів, фільтри за спеціалізацією та функцію

пошуку. Дозволяє адміністраторам легко знайти потрібного лікаря та переглянути його дані.

Сторінки з відображенням інформації про пацієнтів.

Patient-dashboard.html. Панель управління для пацієнта. Включає інформацію про записані прийоми, медичні рахунки та історію відвідувань. Пацієнт може переглядати свої дані та оновлювати особисту інформацію. Може містити сповіщення про майбутні прийоми, рахунки до оплати та нові повідомлення від лікаря.

Add-patient.html. Сторінка для додавання нового пацієнта в систему. Включає форму для введення особистих даних, контактної інформації та медичних записів.

Edit-patient.html. Сторінка для редагування даних про пацієнта. Адміністратори або лікарі можуть змінювати контактні дані, додавати нові записи до медичної картки.

All-patients.html. Список усіх пацієнтів, зареєстрованих у системі. Включає функції пошуку, фільтрації та сортування за різними параметрами (ім’я, дата народження, номер телефону).

Сторінки для управління записами на прийом.

Appointment-list.html. Сторінка зі списком усіх записів на прийом. Включає таблицю з інформацією про дату, час, лікаря та пацієнта. Дозволяє переглядати історію прийомів та фільтрувати дані за різними параметрами.

Add-appointment.html. Форма для додавання нового запису на прийом. Користувач може вибрати лікаря, пацієнта, дату та час прийому.

Edit-appointment.html. Сторінка для редагування існуючого запису на прийом. Дозволяє змінити час, дату або лікаря.

Сторінки для управління оплатами.

Payment-list.html. Сторінка зі списком всіх транзакцій. Містить таблицю з даними про платежі (номер рахунку, пацієнт, сума, дата оплати). Дозволяє адміністраторам відстежувати статуси платежів та шукати рахунки за різними критеріями.

Add-payment.html. Сторінка для додавання нового платежу. Використовується адміністраторами для реєстрації нових транзакцій.

Edit-payment.html. Форма для редагування даних про платіж, з можливістю змінити суму, статус або дату оплати.

Patient-invoice.html. Сторінка для перегляду рахунків пацієнта. Включає список оплачених та невиплачених рахунків, з можливістю роздрукувати або завантажити рахунок.

Інші сторінки: event-management.html. Використовується для управління подіями та записами, такими як перенесення прийомів або створення нових заходів.

Детально структуру папки html відображено на рисунку 2.1.

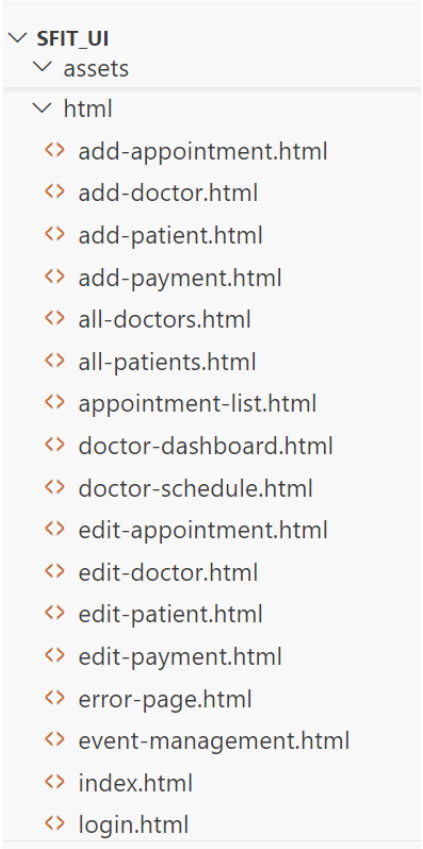


Рисунок 2.1 – Структура папки html

Також одною з ключових в структурі проєкту є папка **assets**. Вона містить усі ресурси, необхідні для створення, стилізації та забезпечення інтерактивності користувацького інтерфейсу CRM системи. Вона організована

таким чином, щоб спростити доступ до різних стилів, зображень, шрифтів та скриптів, що використовуються в додатку.

Основні підпапки наступні.

css/. Ця папка містить основні стилі додатку, які використовуються на всіх сторінках (див. рис. 2.2).

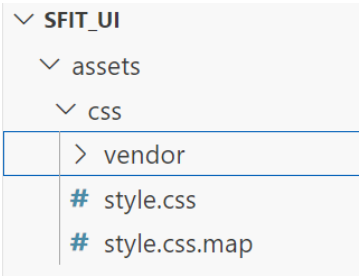


Рисунок 2.2 – Структура папки css

Папка **css** містить в собі **style.css** – основний файл стилів, що містить загальні правила оформлення для всього додатку. Він включає в себе базові стилі, налаштування теми, кольорову палітру та оформлення компонентів. А також **style.css.map** – карту стилів (source map), яка допомагає розробникам відстежувати початковий код CSS у браузері під час налагодження. Це полегшує процес виявлення та виправлення помилок у стилях.

В папці **css** присутня папка **vendor/.** Ця папка містить сторонні бібліотеки та плагіни, які додають додатковий функціонал до інтерфейсу. Вона розбита на підпапки відповідно до різних компонентів (див. рис. 2.3).

В підпапці **vendor** присутня папка **calendar**. Вона містить файли стилів для компонентів календаря. Тут присутній основний файл **calendar.css** – стилі для інтерактивних календарів, що використовуються на сторінках розкладу лікарів та планування зустрічей. Вони включають оформлення днів тижня, подій, навігаційних кнопок та інших елементів.

В підпапці **vendor** також присутня папка **chart**. Тут знаходяться стилі для відображення графіків і діаграм. Містить **chart.css** – файл, який визначає зовнішній вигляд різних типів графіків (лінійні, стовпчикові, кругові). Використовується на дешбордах для візуалізації даних.

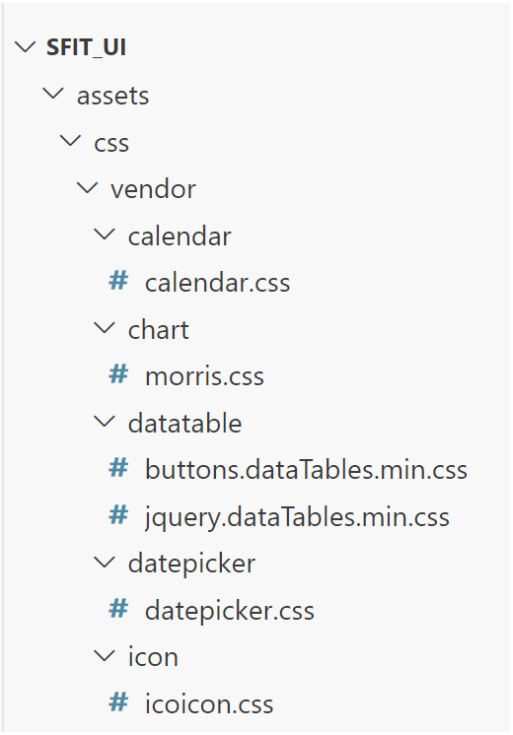


Рисунок 2.3 – Частковий зміст папки підпапки vendor

В підпапці **vendor** є папка **datatable**. Вона містить стилі для інтерактивних таблиць з можливістю фільтрації, сортування та пагінації. Містить **buttons.dataTables.min.css** – стилі для кнопок таблиць, включаючи експорт, друк і копіювання даних. Файл jquery.dataTables.min.css – основні стилі для таблиць з розширеними можливостями, такими як фільтрація та сортування.

В підпапці **vendor** є папка **datepicker**. Тут знаходяться стилі для плагіна вибору дати. Файл **datepicker.css** – включає оформлення календаря для вибору дати, зокрема поля введення дати, кнопок навігації та підсвічування вибраної дати. Використовується на сторінках додавання та редагування записів.

В підпапці **vendor** є папка icon. Вона містить стилі для різних наборів іконок. Містить **icoicon.css** – файл з основними стилями для кастомних іконок, що використовуються в додатку. Також **themify-icons.css** – стилі для набору іконок від Themify, які які застосовуються для різних елементів інтерфейсу (кнопки, меню, статуси).

В підпапці **vendor** також є папка slider. Вона містить стилі для плагіна слайдера тощо.

Також безпосередньо у папці **vendor** розташовані такі файли:

– **bootstrap.css** – основні стилі фреймворку Bootstrap, які забезпечують базове оформлення та компоненти (кнопки, форми, сітку, навігацію тощо);
– **font-awesome.css –** стилі для набору іконок Font Awesome, що використовуються для відображення іконок у різних елементах інтерфейсу;
– **iconico.css** – стилі для кастомних іконок додатку;
– **simplebar.css** – стилі для кастомного скролбару, що покращує зовнішній вигляд скролів у браузері;
– **themify-icons.css** – стилі для іконок Themify, що використовуються у різних елементах навігації та управління.

В папці **assets** також є папка **fonts** – ця папка містить різні шрифти, що використовуються у додатку (див. рис. 2.4).

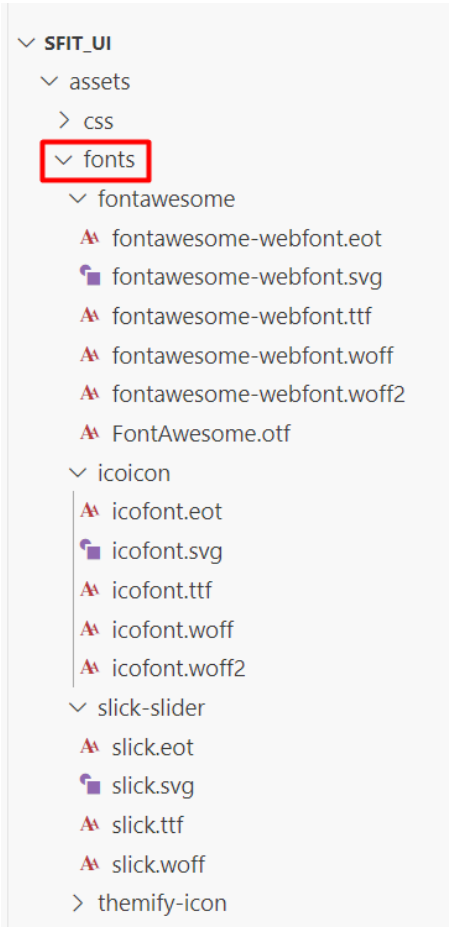


Рисунок 2.4 – Зміст папки fonts

В папці **assets** також є папка зображень **images** (див. рис. 2.5), організована за тематикою та містить підпапки:

– **auth** – зображення для сторінок автентифікації (фонові зображення, логотипи);
– **avatar** – аватарки користувачів (лікарі, пацієнти);
– **dashboard** – ілюстрація та графіка, що використовується на дашбордах;
– **logo** – логотипи компанії;
– **pages** – зображення, що використовуються на окремих сторінках додатку;
– **slick-slider** – зображення для компоненту слайдера.

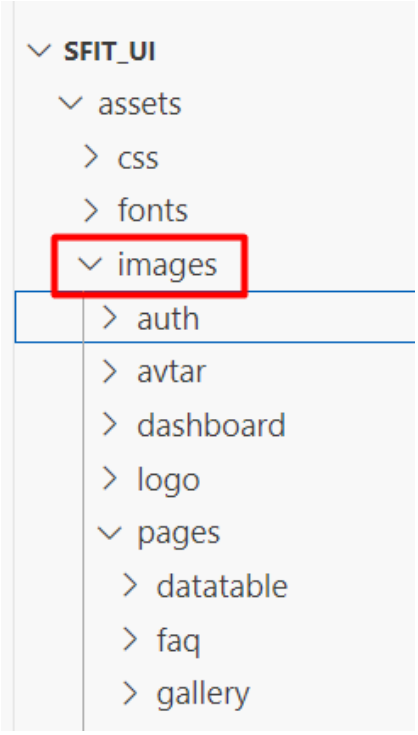


Рисунок 2.5 – Зміст папки images

В папці **assets** також є одна з ключових папок – **js**. Вона містить JavaScript-файли, які забезпечують динамічну поведінку та інтерактивність користувацького інтерфейсу. Вона організована таким чином, щоб відокремити логіку для різних модулів додатку, сторонні бібліотеки, а також загальні користувацькі скрипти.

Основні підпапки наступні.

Dashboard (ця підпапка містить скрипти, що відповідають за логіку роботи різних дешбордів для різних типів користувачів). Наприклад, в файлі **doctor-dashboard.js** міститься логіка для відображення та управління даними на дашборді лікаря. Включає функції для відображення графіків, статистики пацієнтів та розкладу лікаря. В файлі **hospital-dashboard.js** – скрипт для основного дашборду медичного центру, що використовується адміністрацією для відстеження загальних показників, таких як кількість пацієнтів, лікарів, запланованих зустрічей та фінансових даних. В файлі **patient-dashboard.js** – файл, що містить функціонал для дашборду пацієнта. Він забезпечує відображення особистих даних, історії зустрічей, медичних записів та рахунків пацієнта.

Icons (ця підпапка містить файли для роботи з іконками). Наприклад, є файл **feather.js** – скрипт для роботи з набором іконок Feather. Він використовується для відображення SVG-іконок на сторінках додатку, забезпечуючи їх динамічну зміну та налаштування.

Vendors (ця папка містить сторонні бібліотеки та плагіни, що додають додатковий функціонал до додатку. Вона також розбита на підпапки за типами плагінів). Наприклад, в підпапці **calendar** міститься **calendar.js** – скрипт для роботи з календарем, що використовується на сторінках розкладу лікарів та планування зустрічей. Папка chart включає javaScript-файли для відображення графіків (наприклад, **chart.min.js**), які використовуються для візуалізації даних на різних дашбордах тощо.

Services (ця папка містить код для взаємодії з API. Вона розбита на підпапки відповідно до функціональних зон додатку. Наприклад, у підпапці authservice міститься файл authService.js, який реалізує функції для роботи з API, що стосуються аутентифікації та авторизації користувачів, такі як вхід, вихід та перевірка токенів. У підпапці appointmentservice зберігається appointmentService.js, де описані запити для управління записами на прийоми, зокрема створення, оновлення, видалення та отримання списку записів).

Також безпосередньо в папці **js** містяться наступні файли:

– **bootrstrap.bundle.min.js** – основний файл JavaScript для Bootstrap, який містить всі необхідні компоненти фреймворку, включаючи модальні вікна, випадаючі меню, вкладки, каруселі та інші інтерфейсні елементи;
– **custom-script.js** – користувацькі скрипти додатку, що містять загальні функції, використовувані на різних сторінках: сюди входять кастомні обробники подій, функції для валідації форм, анімації тощо;
– **customizer.js** – логіка для налаштування інтерфейсу користувача: цей файл містить функції, які дозволяють змінювати теми, кольорові схеми, розміщення елементів на сторінці та інші параметри, що впливають на вигляд додатку тощо.

Загальна структура папки js зображена на рисунку 2.6.

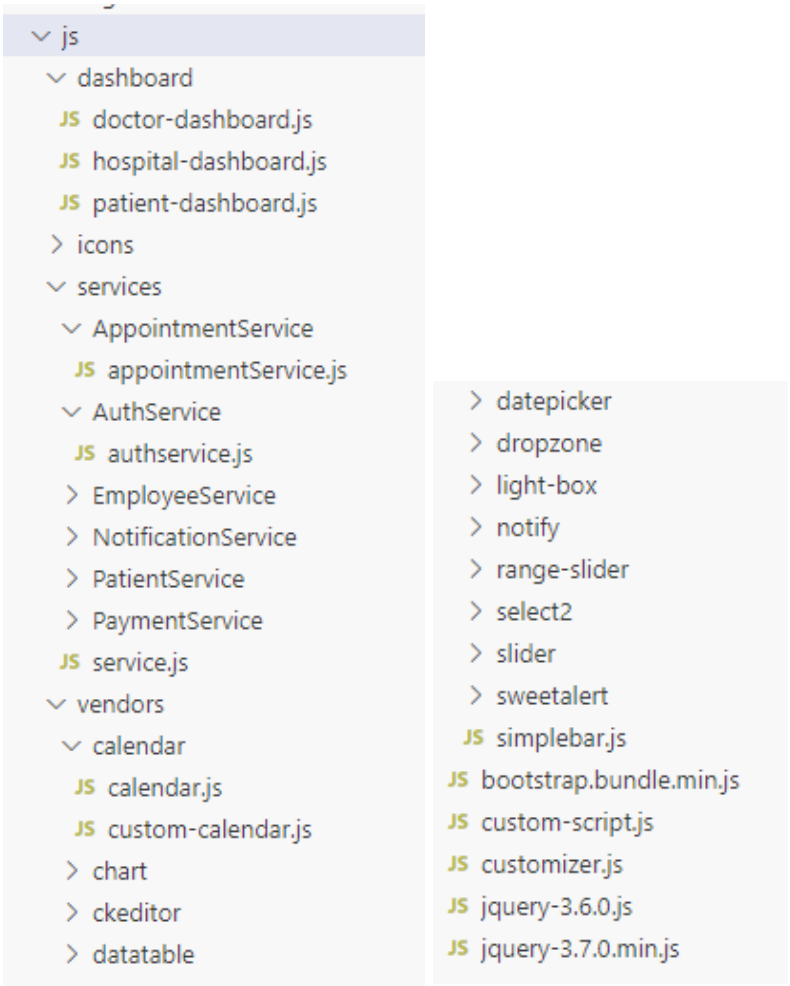


Рисунок 2.6 – Зміст папки js

Також проєкт містить папки pug і scss, які використовуються для генерації HTML-шаблонів та стилів. Pug містить компоненти та міксини для створення повторно використовуваних елементів інтерфейсу, спрощуючи розробку HTML-сторінок. Scss містить стилі, організовані за категоріями (базові стилі, компоненти, теми), з використанням змінних та міксинів для зручної кастомізації та підтримки проєкту (див. рис. 2.7).

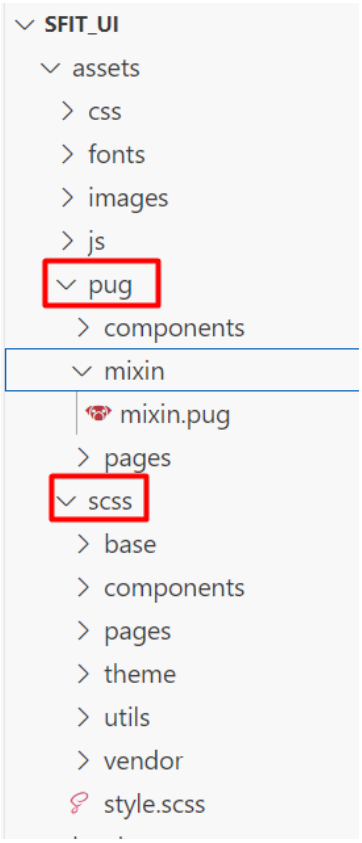


Рисунок 2.7 – Зміст папки pug, scss

2.2.2 Структура backend-проєкту

Бекенд проєкту побудований на основі мікросервісної архітектури, що забезпечує модульність, гнучкість та масштабованість системи. Кожен мікросервіс відповідає за конкретну бізнес-логіку та працює незалежно від інших, що дозволяє окремо розробляти, тестувати, розгортати та підтримувати сервіси без впливу на загальну функціональність системи.

Для кожного мікросервісу створено окрему базу даних, що дозволяє забезпечити ізоляцію даних, зменшити взаємозалежність між сервісами та спростити управління даними на рівні кожного окремого мікросервісу.

Загальну архітектуру системи зображено на діаграмі компонентів (рис. 2.8).

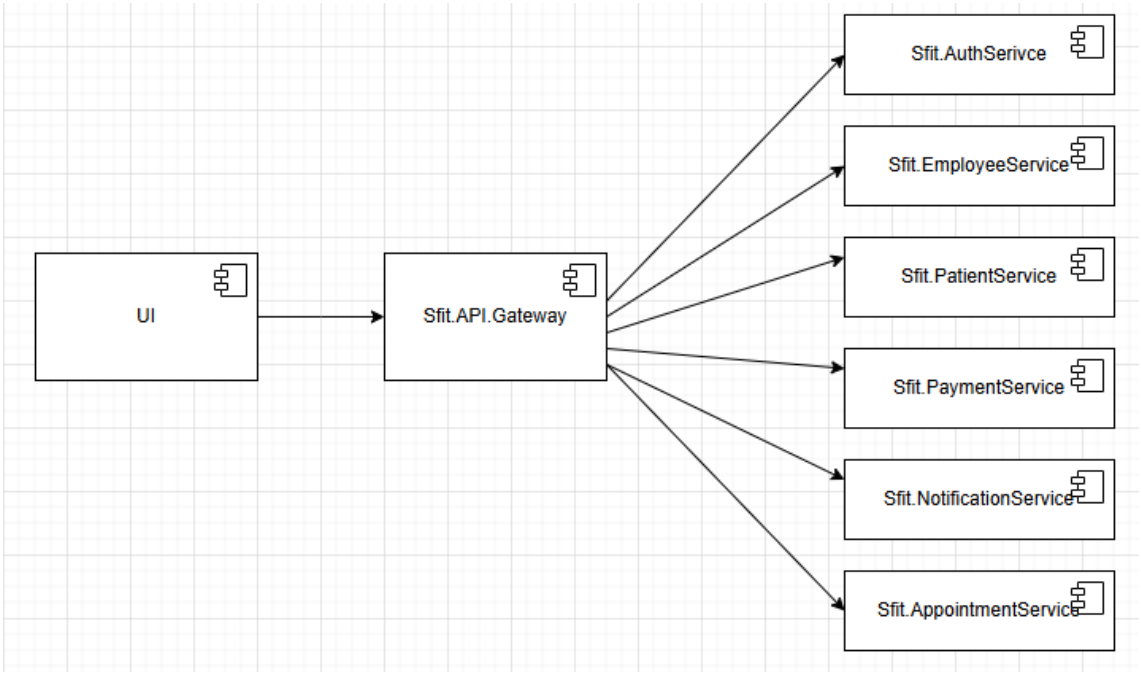


Рисунок 2.8 – Діаграма компонентів проєкту

Частину структуру бекенд-проєкту представлено на рисунку 2.9, де показано ключові компоненти та їх взаємодію. Ця схема допомагає краще зрозуміти організацію коду та логіку роботи системи.

В ролі центрального шлюзу, через який приходять всі запити від клієнтів (фронтенду) виступає **Sfit.API.Gateway**. Його основне призначення – спрощення взаємодії клієнта з різними мікросервісами, надаючи єдину точку доступу до всіх API.

Основними функціями SFit.API.Gateway є маршрутизація запитів, спрощений доступ до сервісів та оптимізація трафіку. Gateway приймає HTTP-запити від клієнтів, таких як фронтенд, і перенаправляє їх на відповідні мікросервіси, включаючи Sfit.AuthService, Sfit.EmployeeService, Sfit.PatientService, Sfit.AppointmentService, Sfit.PaymentService та

Sfit.NotificationService. Це дозволяє забезпечити єдиний інтерфейс для взаємодії з системою, приховуючи складну внутрішню архітектуру від клієнтів. Крім того, API Gateway спрощує доступ до сервісів, абстрагуючи логіку взаємодії з кожним мікросервісом, що дозволяє використовувати прості URL-запити для доступу до різних функцій системи. Важливою функцією Gateway є також оптимізація трафіку, оскільки він може виконувати балансування навантаження між сервісами, що забезпечує стабільну роботу системи навіть при великій кількості запитів.

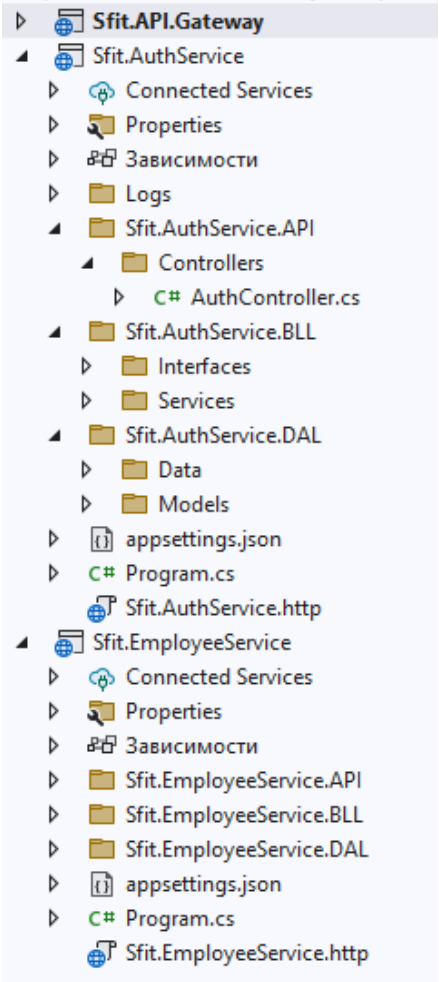


Рисунок 2.9 – Частина структури бекенд-проєкту

Sfit.AuthService – це сервіс, який відповідає за обробку всіх запитів, що стосуються аутентифікації та авторизації користувачів в системі. Його основні функції включають реєстрацію нових користувачів, вхід до системи, перевірку прав доступу та управління токенами, такими як JWT (JSON Web Token).

Користувачі можуть отримати доступ до різних ресурсів через цей сервіс, який відповідає за їх автентифікацію та перевірку дозволів на виконання конкретних дій у системі.

Проєкт **Sfit.AuthService** має чітко розділену структуру на кілька основних прошарків, кожен з яких відповідає за певні аспекти роботи з даними та бізнес-логікою. Це дозволяє забезпечити гнучкість, масштабованість та простоту обслуговування.

Sfit.AuthService.API містить усі контролери, які обробляють HTTP-запити від клієнтів. Контролери виконують роль шлюзу між користувачами і бізнес-логікою. Вони отримують запити від клієнтів, перевіряють дані на коректність та викликають відповідні сервіси, які реалізують основну логіку роботи системи. Контролери можуть обробляти запити для реєстрації користувачів, входу, оновлення паролів, перевірки прав доступу та інші функції, пов’язані з аутентифікацією. Важливу роль також відіграє обробка помилок, яка дозволяє надавати клієнтам зрозумілі повідомлення про проблеми, що виникли. Окрім цього, контролери забезпечують аудит і логування ключових подій, що є критично важливим для забезпечення безпеки системи. Вони також легко розширюються, дозволяючи додавати нові функції без істотних змін у коді. Приклад AuthController зображено на рисунках 2.10 – 2.12.

```
using Microsoft.AspNetCore.Mvc;
using Sfit.AuthService.Sfit.AuthService.BLL.Interfaces;
using Sfit.AuthService.Sfit.AuthService.DAL.Models.DTO;

namespace Sfit.AuthService.Sfit.AuthService.API.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    Ссылок: 3
    public class AuthController : ControllerBase
    {
        private readonly IUserService _userService;
        private readonly ILogger<AuthController> _logger;

        Ссылок: 0
        public AuthController(IUserService userService, ILogger<AuthController> logger)
        {
            _userService = userService;
            _logger = logger;
        }
```

Рисунок 2.10 – Фрагмент коду з контролеру AuthController

```
[HttpPost("register")]
Ссылок: 0
public async Task<IActionResult> Register(UserDTO userDTO)
{
    _logger.LogInformation($"API call /register for user {userDTO.UserName}");
    try
    {
        await _userService.Register(userDTO);
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { message = "An error occured during registration: " + ex.Message });
    }

    return Ok(new { message = "User registered succesfully" });
}
```

Рисунок 2.11 – Фрагмент action-методу реєстрації з контролеру AuthController

```
[HttpPost("login")]
Ссылок: 0
public async Task<IActionResult> Login(UserDTO userDTO)
{
    var ipAddress = HttpContext.Connection.RemoteIpAddress;
    string clientIp = ipAddress?.IsIPv4MappedToIPv6 == true
                    ? ipAddress.MapToIPv4().ToString()
                    : ipAddress?.ToString() ?? "";

    _logger.LogInformation($"API call /login for user {userDTO.UserName} from {clientIp}");
    var token = await _userService.Login(userDTO);
    return Ok(new { token });
}
```

Рисунок 2.12 – Фрагмент action-методу входу з контролеру AuthController

Sfit.AuthService.BLL (Business Logic Layer) є прошарком, що містить усю бізнес-логіку, необхідну для обробки аутентифікації та авторизації. Тут зберігаються сервіси, які виконують основні операції, такі як реєстрація користувачів, перевірка правильності введених даних, генерування та валідація JWT токенів, перевірка прав доступу до різних ресурсів. Цей прошарок забезпечує абстракцію для інших компонентів, дозволяючи зручно змінювати або додавати нові правила бізнес-логіки без впливу на інші частини системи (див. рис. 2.13).

Під час реєстрації користувача в методі Register, спочатку відбувається логування інформації про початок процесу реєстрації, де вказується ім’я користувача, що надається для подальшого відслідковування. Далі генерується випадкова сіль для пароля, яка створюється за допомогою класу

HMACSHA512. Ця сіль використовується для підвищення безпеки пароля, адже вона додається до нього перед хешуванням, що ускладнює можливі атаки, такі як використання таблиць з попередньо обчисленими хешами.

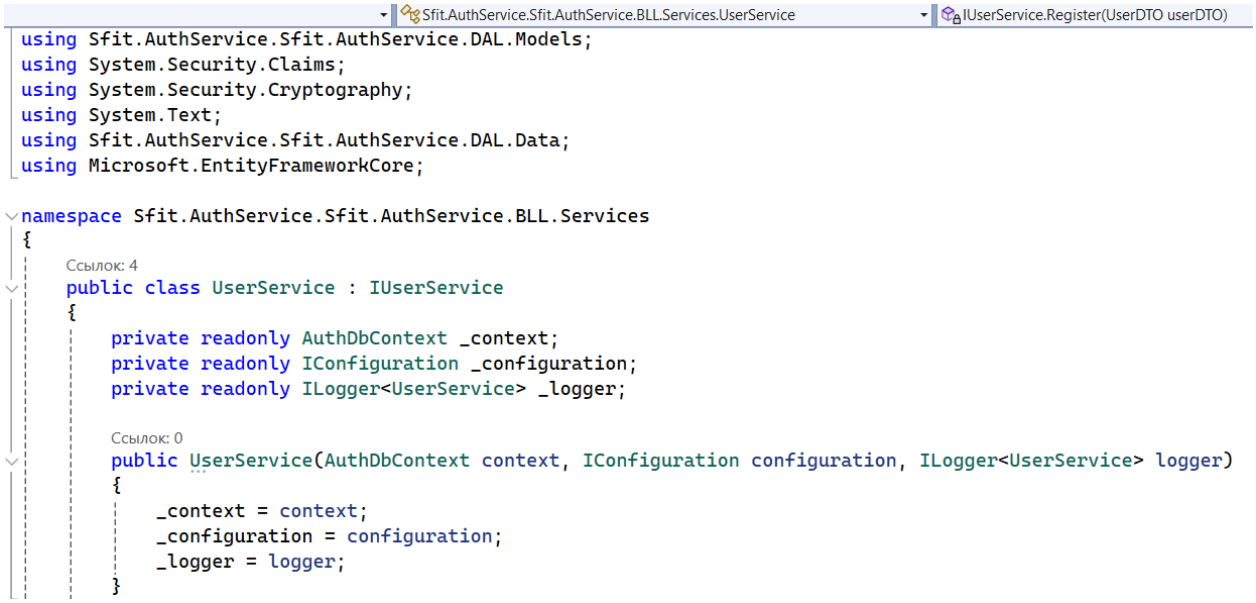


Рисунок 2.13 – Фрагмент сервісу UserService для авторизації та реєстрації

Згенерована сіль потім зберігається як частина хешу пароля, а сам пароль користувача перед хешуванням спершу перетворюється в масив байтів за допомогою кодування UTF-8. Хешування виконується за допомогою функції ComputeHash з використанням алгоритму SHA-512, де результатом є хешоване значення пароля. Після цього створений користувач додається в базу даних разом з його ім’ям, хешованим паролем і збереженою сіллю.

Після успішної реєстрації записується інформація про те, що користувач був успішно зареєстрований. У разі виникнення помилки під час реєстрації (наприклад, при взаємодії з базою даних), відповідна помилка фіксується в журналі і передається далі, що дозволяє системі обробити її належним чином.

У цьому процесі для хешування пароля застосовується алгоритм HMACSHA512 з хеш-функцією SHA-512, що забезпечує високу ступінь захисту. В результаті, в базі даних зберігається лише хеш пароля та сіль, а сам пароль користувача ніколи не зберігається у відкритому вигляді (див. рис. 2.14).

```
async Task IUserService.Register(UserDTO userDTO)
{
    _logger.LogInformation($"Start process of registration user: {userDTO.UserName}");

    try
    {
        using var hmac = new HMACSHA512();

        var user = new User
        {
            Username = userDTO.UserName,
            PasswordSalt = hmac.Key,
            PasswordHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(userDTO.Password))
        };

        _context.Users.Add(user);

        await _context.SaveChangesAsync();

        _logger.LogInformation($"User {userDTO.UserName} successfully registered!");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"Error while registration of user {userDTO.UserName}");
        throw;
    }
}
```

Рисунок 2.14 – Фрагмент сервісу UserService, метод реєстрації

Метод Login (див. рис. 2.15) обробляє спробу входу користувача в систему. Спочатку він шукає користувача в базі даних за вказаним ім’ям користувача. Якщо користувача не знайдено або пароль не збігається з хешом, метод записує попередження в журнал та викидає виняток UnauthorizedAccessException.

```
Ссылок: 2
async Task<string> IUserService.Login(UserDTO userDTO)
{
    _logger.LogInformation($"Attempting to log in user: {userDTO.UserName}");
    var user = await _context.Users.FirstOrDefaultAsync(x => x.Username == userDTO.UserName);

    if (user == null || !VerifyPasswordHash(userDTO.Password, user.PasswordHash, user.PasswordSalt))
    {
        _logger.LogWarning($"Invalid login attempt for user: {userDTO.UserName}");
        throw new UnauthorizedAccessException("Invalid credentials");
    }

    var token = GenerateJwtToken(user);
    _logger.LogInformation($"User {userDTO.UserName} successfully logged in");

    return token;
}
```

Рисунок 2.15 – Фрагмент сервісу UserService, метод авторизації

Якщо введені дані правильні, метод генерує JWT токен (див. рис. 2.16), використовуючи дані користувача, і повертає цей токен. Під час перевірки пароля метод викликає допоміжний метод VerifyPasswordHash (див. рис. 2.17), який обчислює хеш введеного пароля та порівнює його з збереженим хешем, використовуючи сіль, що зберігається разом із паролем користувача.

```
Ссылок: 1
private string GenerateJwtToken(User user)
{
    var tokenHandler = new System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler();

    var keySection = _configuration.GetSection("Jwt:Key").Value;
    var issuer = _configuration.GetSection("Jwt:Issuer").Value;
    var audience = _configuration.GetSection("Jwt:Audience").Value;

    if (string.IsNullOrEmpty(keySection))
    {
        throw new InvalidOperationException("JWT Key is not configured");
    }

    if (string.IsNullOrEmpty(issuer))
    {
        throw new InvalidOperationException("JWT Issuer is not configured");
    }

    if (string.IsNullOrEmpty(audience))
    {
        throw new InvalidOperationException("JWT Audience is not configured");
    }

    var key = Encoding.ASCII.GetBytes(keySection);

    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new System.Security.Claims.ClaimsIdentity(new[] { new Claim(ClaimTypes.Name, user.Username) }),
        Expires = DateTime.UtcNow.AddDays(1),
        SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature),
        Issuer = issuer,
        Audience = audience
    };

    var token = tokenHandler.CreateToken(tokenDescriptor);
    return tokenHandler.WriteToken(token);
}
```

Рисунок 2.16 – Фрагмент сервісу UserService, метод генерації JWT-токена

```
Ссылок: 1
private bool VerifyPasswordHash(string password, byte[] storedHash, byte[] storedSalt)
{
    using var hmac = new HMACSHA512(storedSalt);
    var computedHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
    return computedHash.SequenceEqual(storedHash);
}
```

Рисунок 2.17 – Фрагмент сервісу UserService, метод перевірки хешу, який порівнює обчислений хеш пароля з тим, що збережено в базі даних, використовуючи сіль для хешування

Sfit.AuthService.DAL (Data Access Layer) – містить компоненти для роботи з базою даних. У цьому прошарку знаходяться моделі даних, контексти бази даних, репозиторії для роботи з користувачами та їх ролями.

Sfit.AuthService.DAL.Models. Тут містяться класи моделей, які описують структуру таблиць у базі даних (наприклад, користувач, роль користувача).

Sfit.AuthService.DAL.Data. Містить контексти бази даних, що взаємодіють з репозиторіями для виконання CRUD-операцій.

Також реалізовано логування за допомогою бібліотеки Serilog, що дозволяє зручно та ефективно записувати інформацію про роботу системи як у файли, так і безпосередньо в базу даних. Serilog є потужним та гнучким інструментом для реєстрації логів, підтримуючи різноманітні формати виведення, рівні логування та можливості для фільтрації даних (див. рис. 2.18), що робить його універсальним рішенням для потреб різних проєктів. Завдяки інтеграції з платформами, такими як Elasticsearch, Seq чи SQL Server, бібліотека забезпечує гнучкість і масштабованість для складних систем.

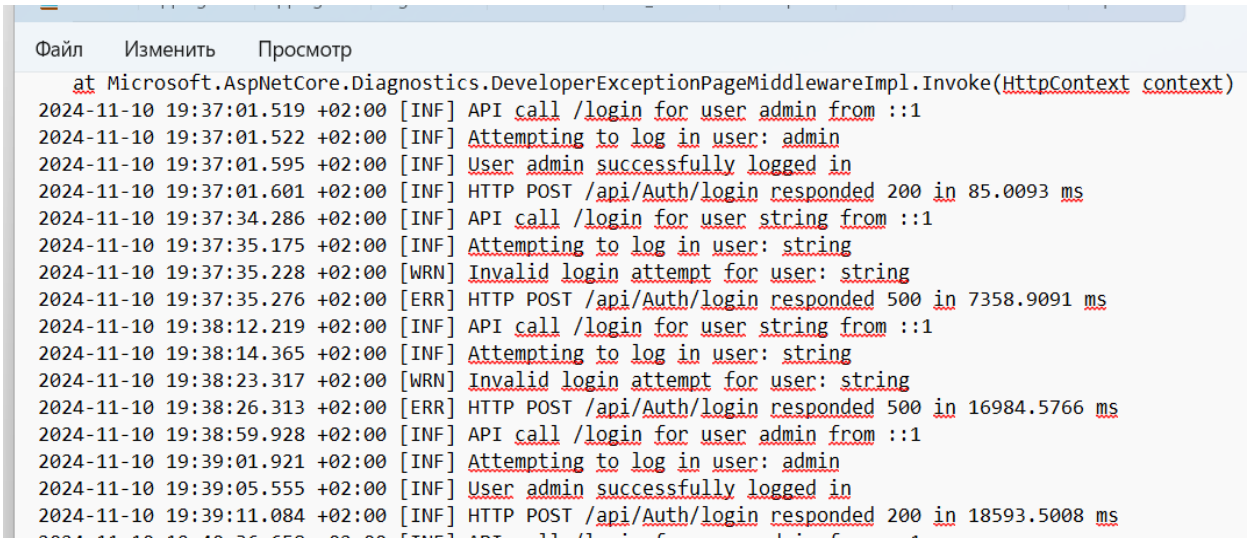


Рисунок 2.18 – Приклад лог-файлу за поточний день в сервісі Sfit.AuthService

Безпосередньо ER-діаграму БД даного сервісу можна побачити на рисунку 2.19.

Приклад зберігання даних у таблиці Users, яка містить інформацію про користувачів системи, наведено на рисунку 2.20. У цій таблиці представлені ключові атрибути, такі як ідентифікатор користувача, хеш пароля та безпосередньо соль.

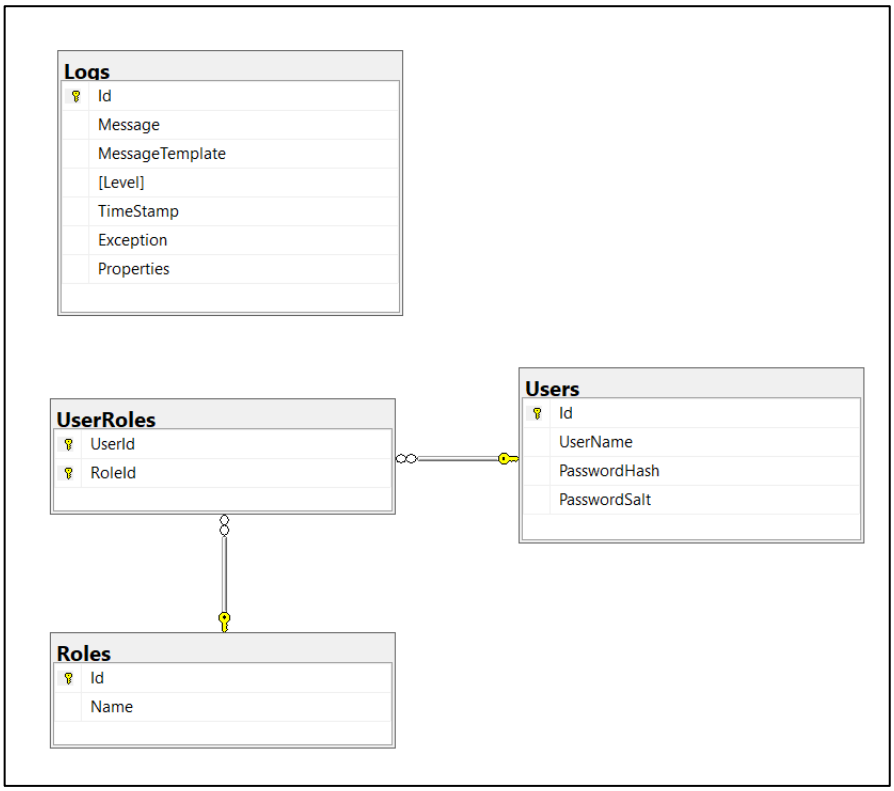


Рисунок 2.19 – ER-діаграма сервісу SFit.AuthService

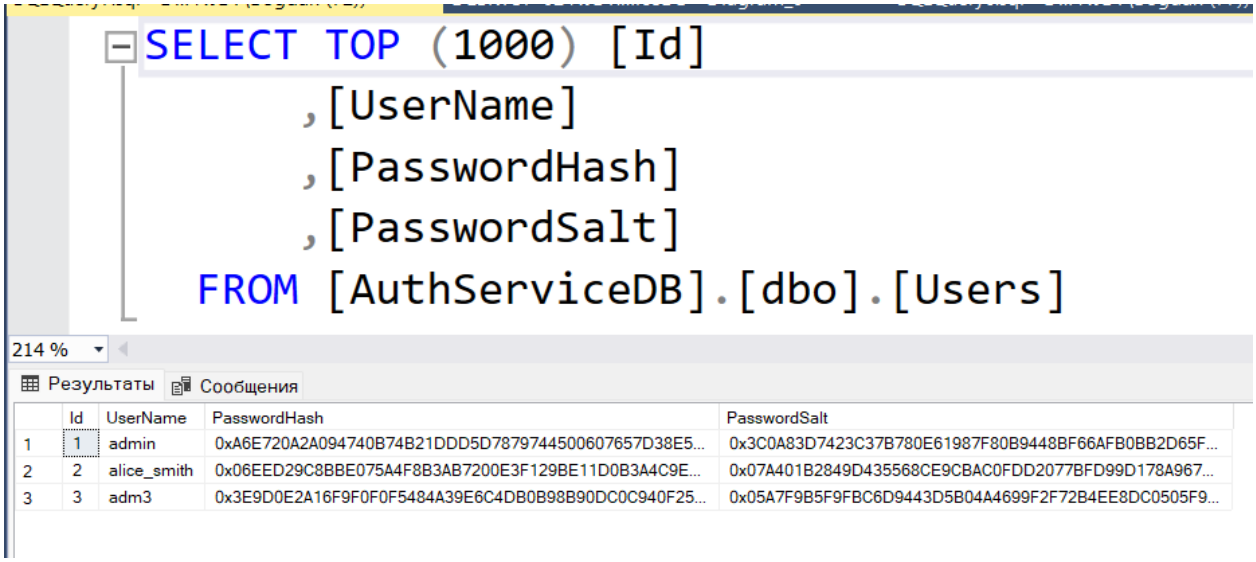


Рисунок 2.20 – Вибірка даних з таблиці Users в AuthServiceDb

За аналогією побудовано архітектуру в інших мікросервісах.

Наприклад, є **Sfit.EmployeeService.** Цей сервіс управляє всіма аспектами, пов’язаними з інформацією про співробітників медичного центру, включаючи реєстрацію нових співробітників, оновлення їх даних та отримання інформації про існуючих співробітників.

Структура проєкту:

- **Sfit.EmployeeService.API**: тут знаходяться контролери, які обробляють запити на додавання, оновлення та отримання співробітників;
- **Sfit.EmployeeService.BLL**: тут логіка бізнес-процесів для роботи зі співробітниками, такі як перевірка даних, логіка для оновлення записів у базі даних;
- **Sfit.EmployeeService.DAL**: це прошарок для взаємодії з базою даних – містить моделі для співробітників, контексти бази даних, репозиторії для збереження та отримання інформації про співробітників.

Також присутній **Sfit.PatientService.** Цей мікросервіс охоплює всі аспекти, що стосуються пацієнтів медичного центру: реєстрація пацієнтів, оновлення їх даних, а також доступ до медичних записів і історії хвороб.

Структура проєкту:

- **Sfit.PatientService.API**: тут знаходяться контролери для обробки запитів, пов’язаних з пацієнтами, такими як реєстрація, оновлення даних пацієнтів, отримання медичних карт;
- **Sfit.EmployeeService.BLL**: тут логіка для обробки запитів, валідації даних та забезпечення процесів, пов’язаних з пацієнтами;
- **Sfit.EmployeeService.DAL:** це прошарок містить доступ до бази даних для роботи з пацієнтами: моделі, контексти та репозиторії.

За допомогою сервісу **Sfit.AppointmentService** можна керувати записами на прийом до лікарів, включаючи запис, зміну, скасування та перегляд запланованих прийомів.

Структура проєкту:

- **Sfit.AppointmentService.API:** тут знаходяться контролери для обробки запитів на запис до лікаря, перегляд наявних записів, скасування запису тощо;
- **Sfit.AppointmentService.BLL:** тут знаходиться бізнес-логіка для обробки запитів, валідації записів і управління часом прийому;

– **Sfit.AppointmentService.DAL:** це шар доступу до даних для збереження та отримання інформації про записи на прийом.

Сервіс **Sfit.PatientService** охоплює всі аспекти, що стосуються пацієнтів медичного центру: реєстрація пацієнтів, оновлення їх даних, а також доступ до медичних записів і історії хвороб.

Структура проєкту:

– **Sfit.PatientService.API:** контролери для обробки запитів, пов’язаних з пацієнтами, такими як додавання пацієнтів, оновлення даних пацієнтів, отримання медичних карт тощо;
– **Sfit.PatientService.BLL:** логіка для обробки запитів, валідації даних та забезпечення процесів, пов’язаних з пацієнтами;
– **Sfit.PatientService.DAL:** містить доступ до бази даних для роботи з пацієнтами: моделі, контексти та репозиторії.

Сервіс **Sfit.AppointmentService** керує записами на прийом до лікарів, включаючи запис, зміну, скасування та перегляд запланованих прийомів.

Структура проєкту:

– **Sfit.AppointmentService.API:** контролери для обробки запитів на запис до лікаря, перегляд наявних записів, скасування запису тощо;
– **Sfit.AppointmentService.BLL:** бізнес-логіка для обробки запитів, валідації записів і управління часом прийому;
– **Sfit.AppointmentService.DAL:** шар доступу до даних для збереження та отримання інформації про записи на прийом.

Сервіс **Sfit.PaymentService** обробляє платіжні транзакції, включаючи здійснення оплат за медичні послуги, перевірку статусу платежів та інтеграцію з платіжними системами.

Структура проєкту:

– **Sfit.PaymentService.API:** контролери для обробки запитів на платіжні операції;
– **Sfit.PaymentService.BLL:** логіка для обробки платежів та інтеграція з платіжними системами;

– **Sfit.PaymentService.DAL:** містить моделі, репозиторії та контексти для роботи з базою даних, що зберігає платіжні транзакції.

Сервіс **Sfit.NotificationService** відповідає за відправку повідомлень пацієнтам і співробітникам, таких як підтвердження запису, нагадування про прийом та інші повідомлення.

Структура проєкту:

– **Sfit.NotificationService.API:** контролери для обробки запитів на відправку сповіщень;
– **Sfit.NotificationService.BLL:** логіка для формування та відправки сповіщень;
– **Sfit.NotificationService.DAL:** тут зберігаються моделі для історії сповіщень і відповідні контексти бази даних.

Отже, кожен мікросервіс має чітко розподілену структуру з окремими прошарками для контролерів (API), бізнес-логіки (BLL) і доступу до даних (DAL). Це дозволяє легко розширювати функціональність, підтримувати масштабованість та керувати різними частинами системи незалежно. Кожен сервіс працює з власною базою даних, що забезпечує гнучкість і дозволяє уникнути взаємозалежностей між сервісами.

Загальну діаграму послідовності зображено на рисунку 2.21.

Загальний сценарій роботи запису на прийом наступний:

– **Клієнт -> API.Gateway.** Клієнт надсилає запит на створення запису (ID пацієнта, ID співробітника, дата і час);
– **API Gateway -> AppointmentService.** API Gateway переадресовує запит до AppointmentService;
– **AppointmentService -> EmployeeService.** AppointmentService перевіряє доступність співробітника, надсилаючи запит до EmployeeService;
– **EmployeeService -> AppointmentService.** EmployeeService повертає відповідь про доступність співробітника (доступний/недоступний);
– **AppointmentService -> PatientService.** AppointmentService перевіряє дані пацієнта, надсилаючи запит до PatientService;

– **PatientService -> AppointmentService.** PatientService повертає відповідь про коректність даних пацієнта (дані коректні/некоректні);
– **AppointmentService.** Якщо всі дані коректні, AppointmentService створює запис у базі даних;
– **AppointmentService -> NotificationService**. AppointmentService надсилає запит до NotificationService для відправки повідомлень;
– **NotificationService.** NotificationService відправляє повідомлення пацієнту та співробітнику;
– **AppointmentService -> API Gateway.** AppointmentService повертає відповідь клієнту через API Gateway (успішно/помилка).

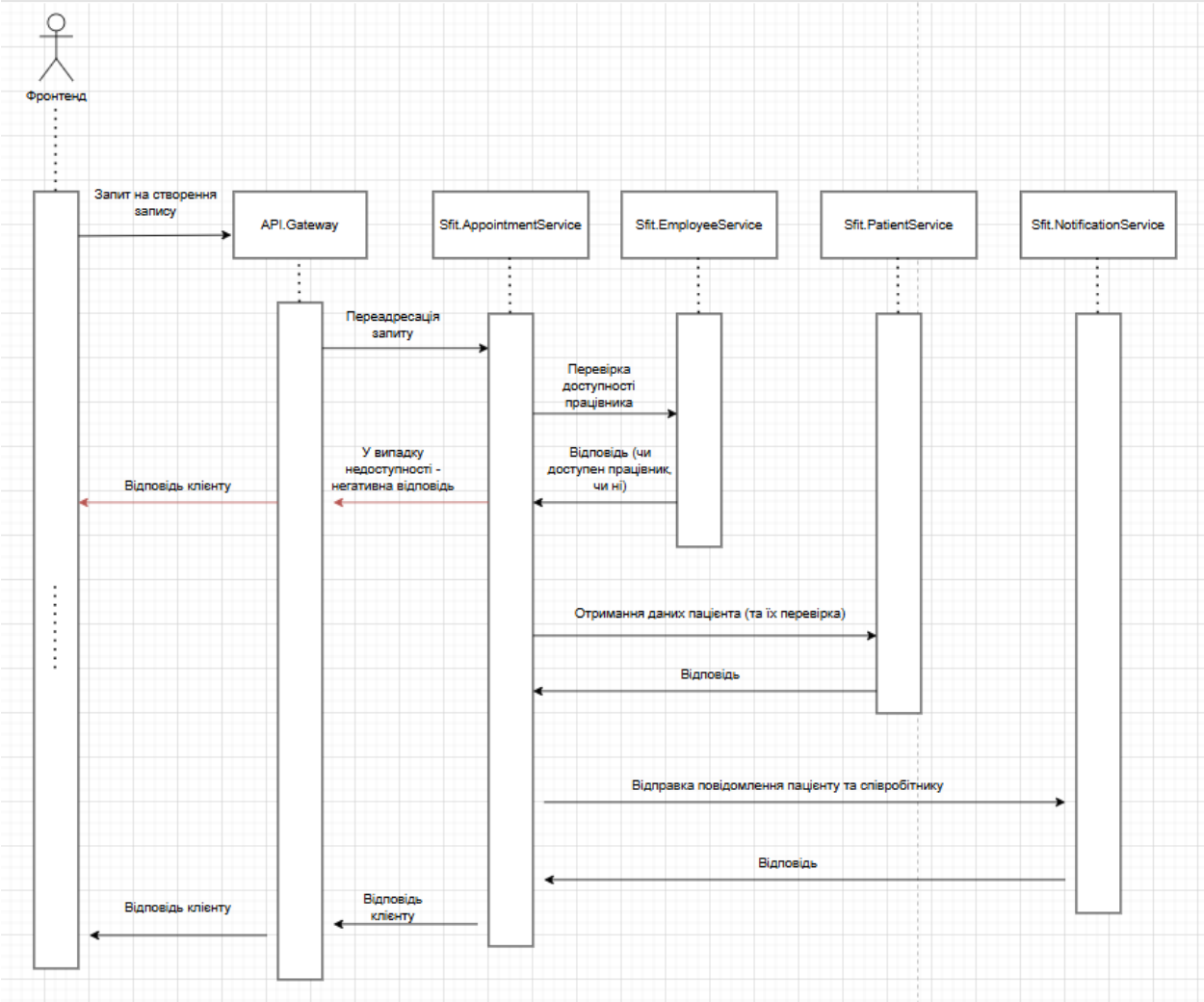


Рисунок 2.21 – Діаграма послідовності стосовно сценарію запису на прийом

3 РЕАЛІЗАЦІЯ

При відкритті вебсайту в браузері, користувач бачить вікно для авторизації. У цьому вікні є два поля: одне для введення логіна, а інше – для пароля.

Користувач повинен ввести ім’я користувача та пароль, зареєстровані в системі. Після введення даних і натискання кнопки «Увійти» система перевірить точність введених даних. Якщо ім’я користувача і пароль вказані правильно, користувач може перейти на головну сторінку сайту і продовжити роботу там з іншими функціями. Якщо введені дані неправильні, користувач отримає повідомлення про помилку, що інформує його про неправильні дані для автентифікації, і він матиме можливість спробувати ще раз. Це дозволяє вам перевірити коректність механізмів авторизації на сайті та надати відповідний зворотний зв’язок користувачам (див. рис. 3.1).

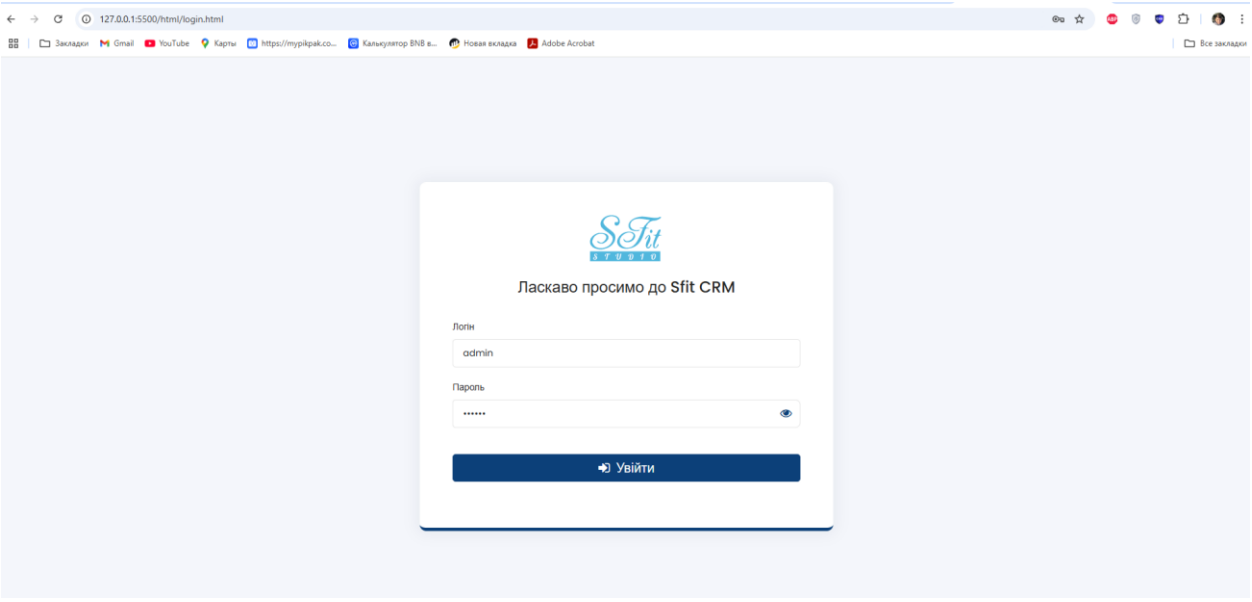


Рисунок 3.1 – Сторінка авторизації

До того ж, є можливість зробити пароль «видимим» при натисканні на відповідну іконку (рис. 3.2), що буде зручним для користувачів, які хочуть перевірити правильність введеного пароля перед його підтвердженням.

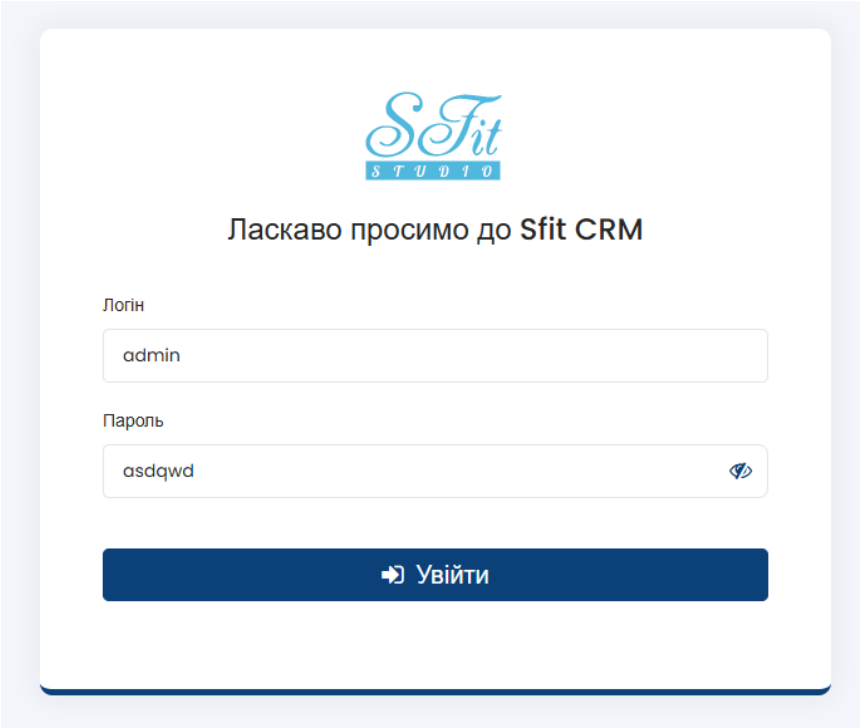


Рисунок 3.2 – Вікно після натискання на іконку для відображення пароля

Після успішного входу в систему користувач потрапляє на дашборд, який містить основний функціонал. На цьому екрані відображаються ключові показники: кількість прийнятих пацієнтів, кількість кваліфікованих спеціалістів, кількість запланованих візитів на тиждень та кількість візитів за минулий місяць.

Також на дашборді представлений інтерактивний графік, який дозволяє порівнювати кількість запланованих прийомів з фактично відвіданими.

Крім того, користувач може переглядати найближчі прийоми в інтерактивному режимі, а також має можливість натискання кнопки «Подивись усі прийоми», щоб отримати повний список прийомів (див. рис. 3.3).

Також при натисканні відповідної іконки є можливість перейти в повноекранний режим роботи (рис. 3.4).

Сама реалізація в програмному коді наступна.

На дашборді, що відображає інформацію про всіх співробітників, користувач має доступ до низки функцій, які допомагають ефективно працювати з даними.

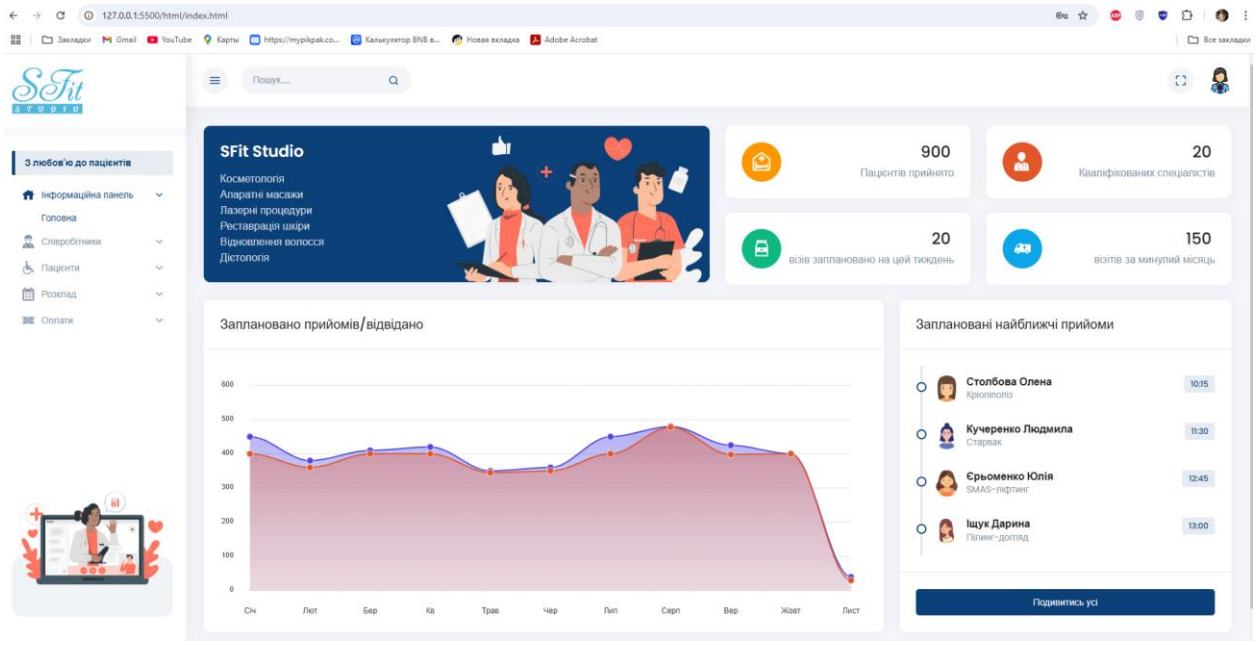


Рисунок 3.3 – Головний дашборд

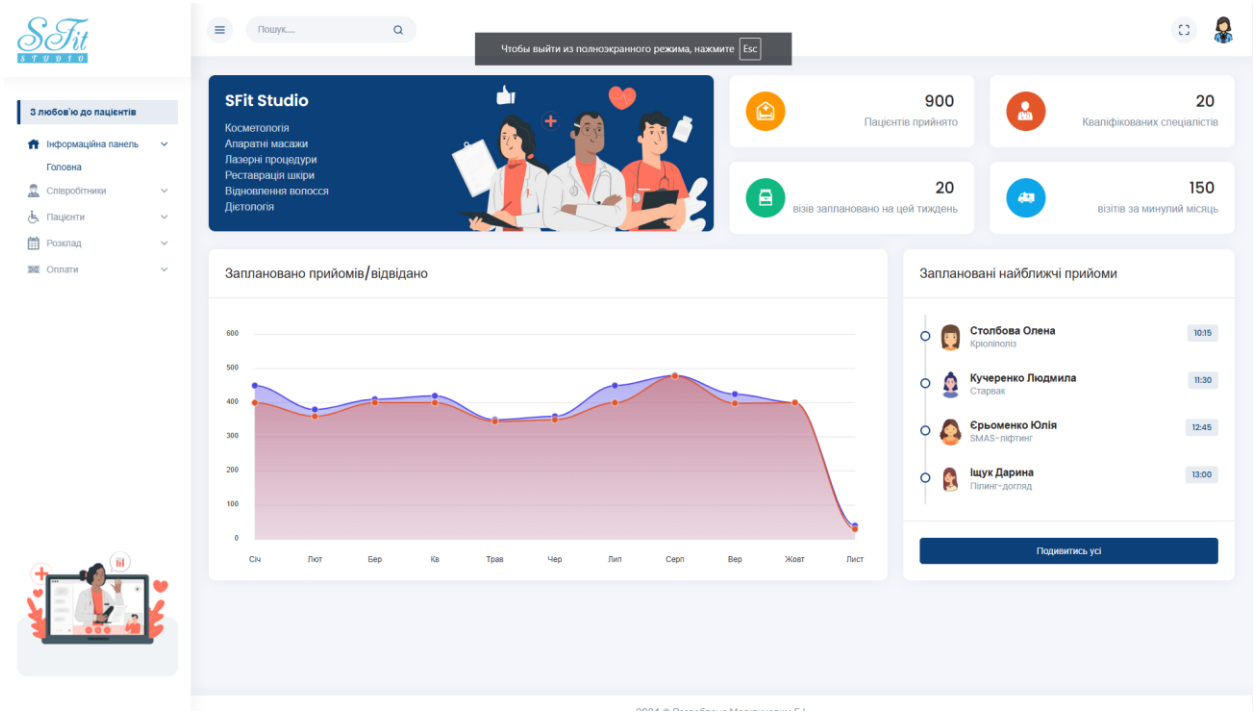


Рисунок 3.4 – Повноекранний режим роботу застосунку

По-перше, доступна функція пошуку, яка дозволяє швидко знайти конкретного співробітника за різними критеріями: імені, прізвища, посади та іншими параметрами. Пошук реалізований таким чином, що користувач може ввести деяку інформацію, і система автоматично видає потрібні результати, значно прискорюючи процес пошуку.

По-друге, є можливість сортування даних за різними параметрами. Наприклад, можна сортувати співробітників за ім’ям, датою прийому на роботу, посадою чи іншими характеристиками. Це дозволяє швидко змінювати вигляд списку співробітників і переглядати їх у бажаному порядку.

Крім того, передбачена функція редагування, яка дає можливість змінювати інформацію про співробітника безпосередньо з дашборду. Це може бути необхідно для оновлення контактних даних, посадових обов’язків або інших важливих відомостей.

Також є можливість видаляти записи про співробітників, що дозволяє оперативно оновлювати список співробітників у разі їх звільнення або з інших причин (див. рис. 3.5, 3.6).

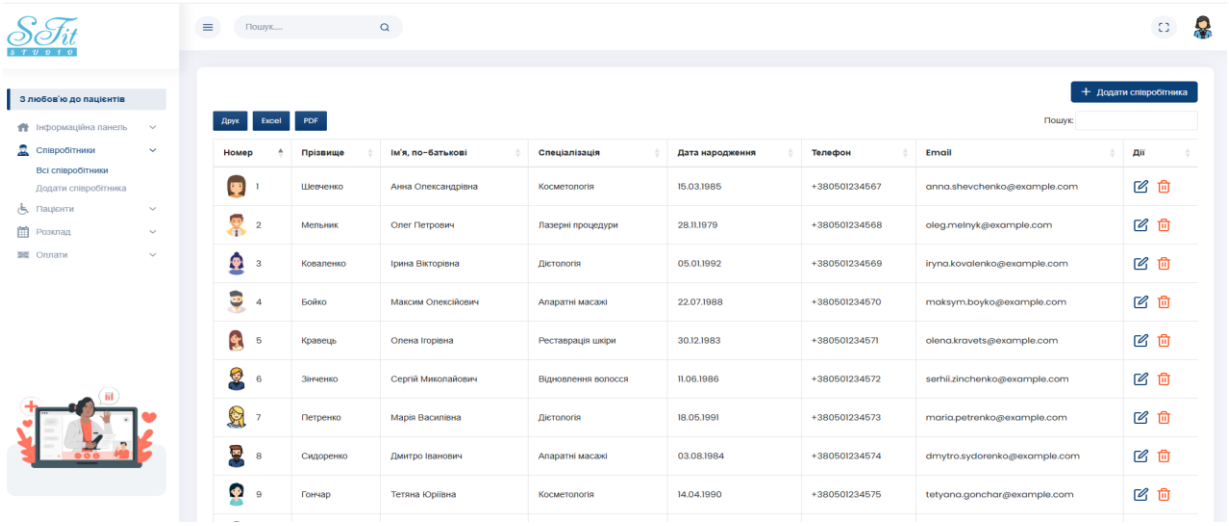


Рисунок 3.5 – Сторінка «Всі співробітники»

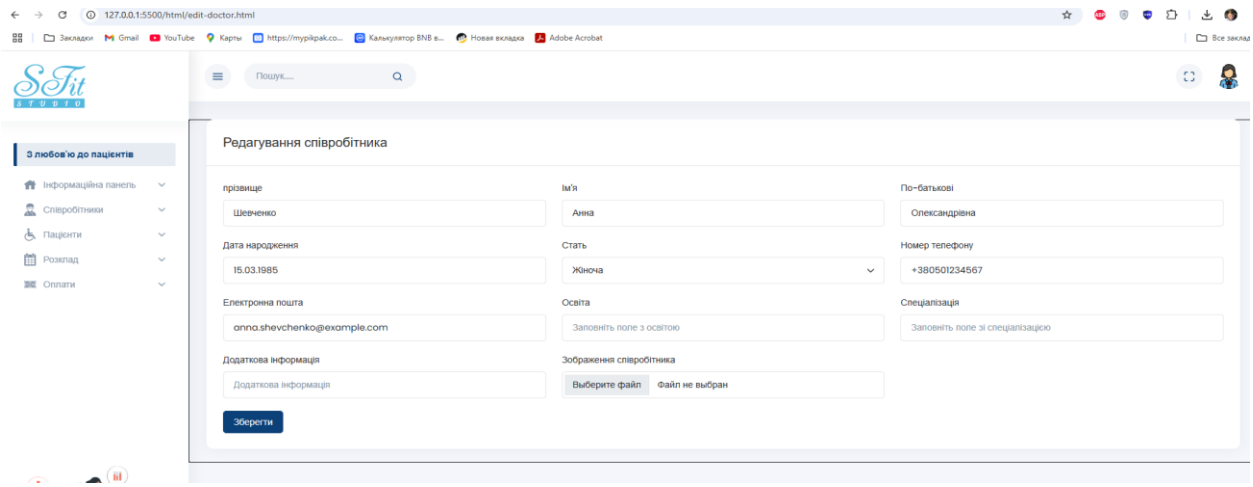


Рисунок 3.6 – Сторінка «Редагування співробітника»

Для зручності користувачів, є опція друку списку всіх співробітників. Це може бути корисно для фізичного збереження або передачі інформації у вигляді паперових документів.

Ви також можете завантажувати дані співробітників у популярні формати, такі як Excel та PDF (див. рис. 3.7 – 3.9).

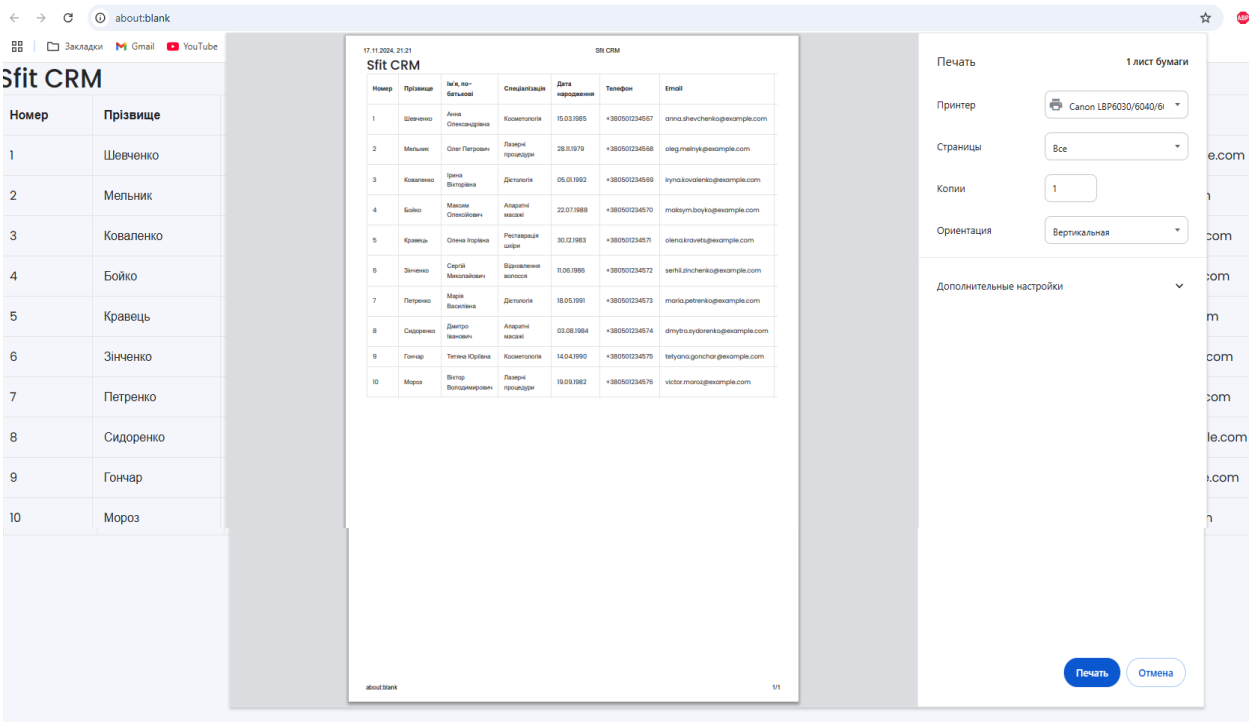


Рисунок 3.7 – Друк співробітників

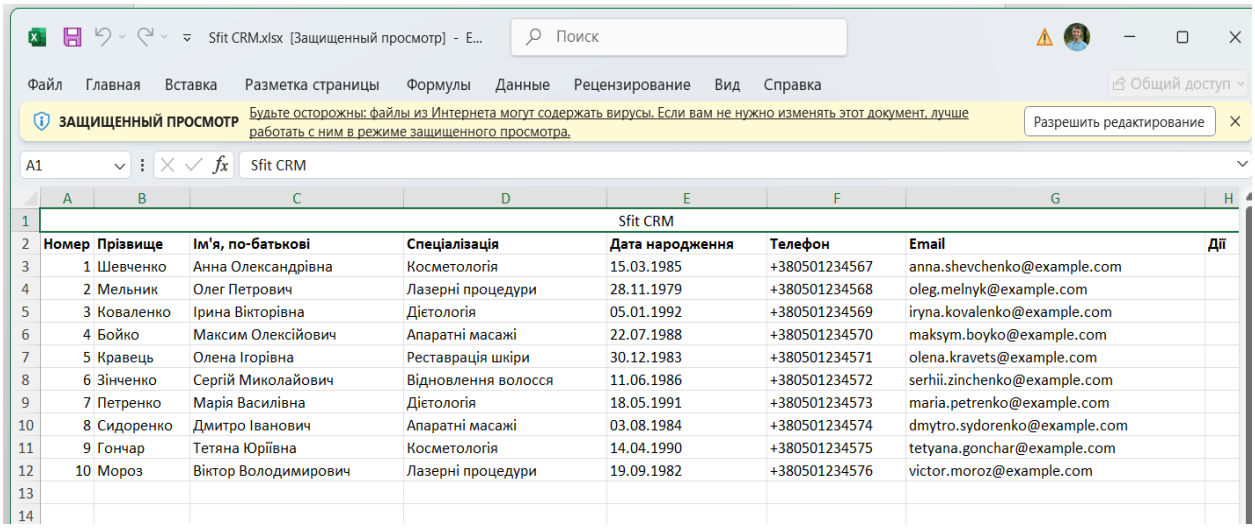


Рисунок 3.8 – Після натискання на кнопку вивантаження в Excel, завантажується відповідний файл

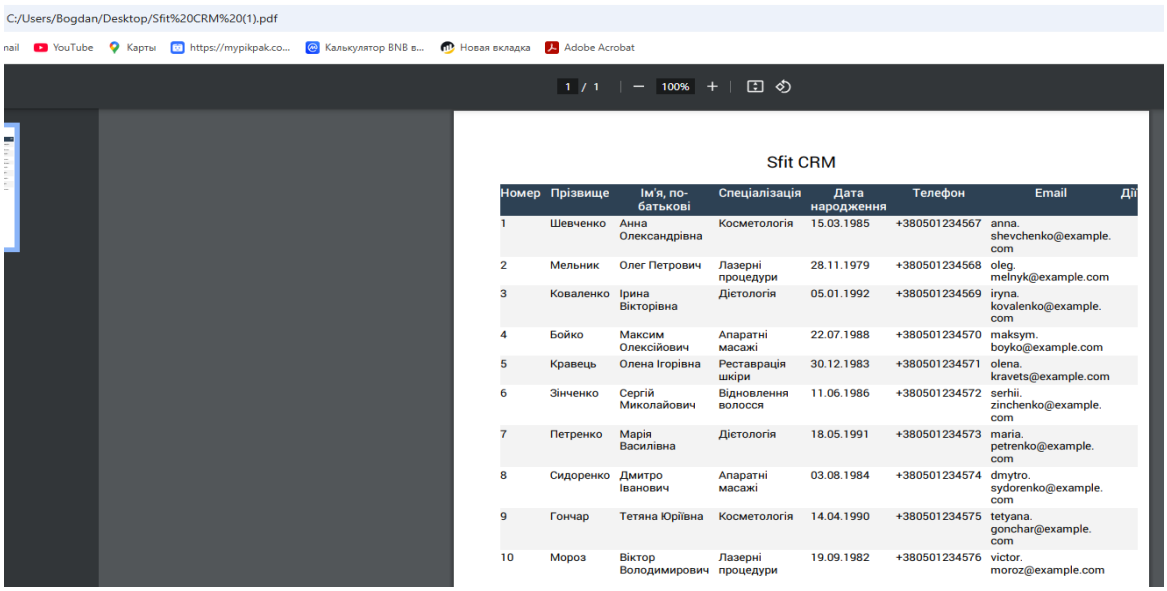


Рисунок 3.9 – Вивантаження списку співробітників в pdf

Це дозволяє зберігати дані для подальшої обробки та аналізу або зручно передавати іншим користувачам і підрозділам. Функція завантаження підтримує безліч налаштувань для вибору необхідних полів і форматів документів, що робить її універсальною для різних випадків використання.

Є можливість додати нового співробітника як на сторінці з усіма співробітниками, так і в окремому пункті меню (рис. 3.10, 3.11).

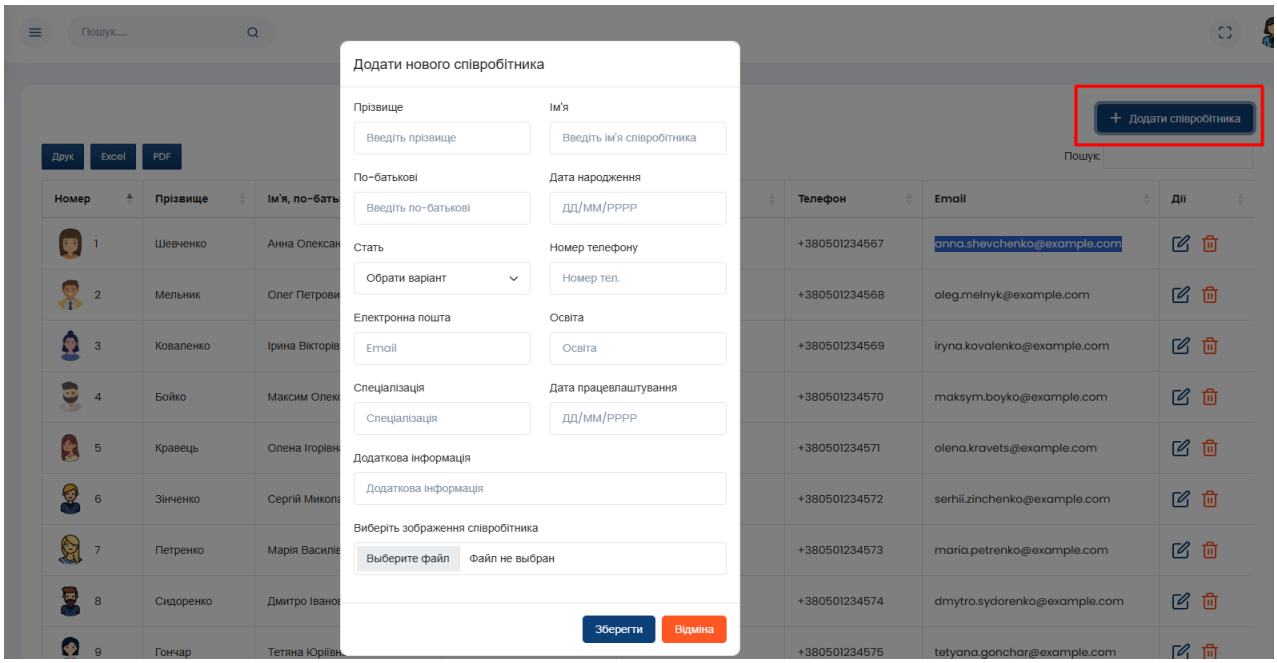


Рисунок 3.10 – Відкриття модульного вікна після натискання на кнопку додавання нового співробітника

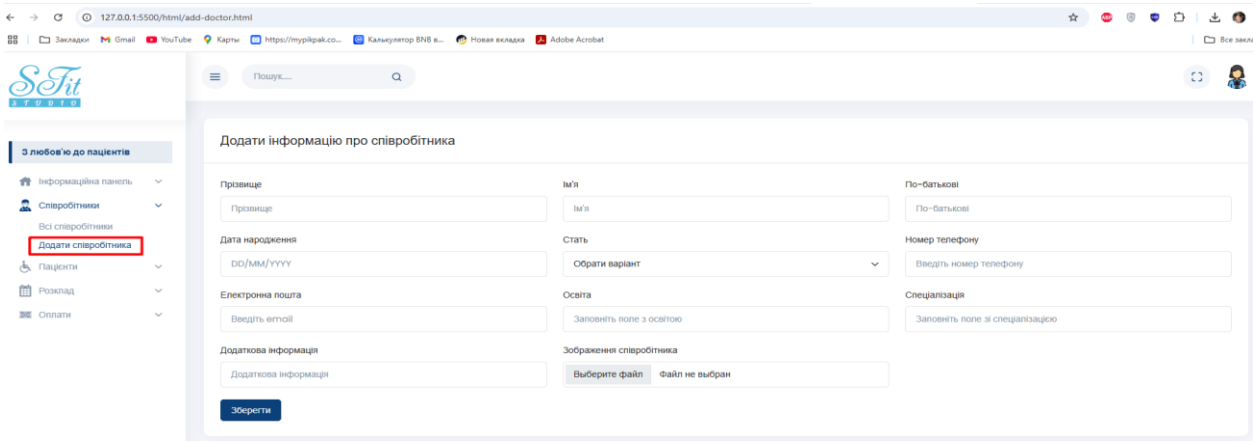


Рисунок 3.11 – Окремий розділ додавання інформації про співробітника

Окрім розділу для співробітників, у системі передбачений окремий розділ для управління інформацією про пацієнтів. Цей розділ має кілька функцій, що дозволяють ефективно працювати з даними пацієнтів і організувати процес їхнього обслуговування.

По-перше, в розділі для пацієнтів є можливість вести детальну базу даних про кожного пацієнта. Кожен запис містить основну інформацію, таку як ім’я, прізвище, дата народження, контактні дані, історія захворювань та інші важливі медичні відомості. Ця інформація зберігається в структурованому вигляді, що дозволяє швидко знаходити потрібні дані для подальшого використання.

Один з важливих аспектів – це можливість пошуку пацієнтів за різними критеріями, такими як ім’я, прізвище, номер телефону, медичний номер чи дата останнього візиту. Пошук виконується швидко та ефективно, що дозволяє знайти потрібного пацієнта за кілька секунд.

У цьому розділі також є можливість сортувати дані пацієнтів за різними параметрами, наприклад, за датою останнього прийому, медичним статусом чи іншими характеристиками. Це дозволяє зручно переглядати інформацію в різних форматах, що покращує навігацію по базі даних.

Крім того, є можливість редагувати інформацію про пацієнта, що дозволяє актуалізувати дані в разі зміни контактної інформації, змін у медичній картці або при оновленні історії хвороби. Функція редагування дозволяє безпосередньо вносити зміни до запису пацієнта на дашборді.

Також передбачена функція видалення запису про пацієнта, що може бути корисно, наприклад, при відсутності пацієнта в установі чи у разі його смерті. Видалення записів відбувається після підтвердження користувача, щоб уникнути випадкових помилок.

Як і в розділі для співробітників, для пацієнтів є можливість вивантажити всі дані в форматах Excel або PDF. Це дозволяє медичному персоналу зберігати дані для аналізу або подальшої обробки, а також передавати їх у відповідні відділи чи медичні установи. Формат вивантаження можна налаштовувати, вибираючи необхідні поля та тип документа.

Загалом, цей розділ дозволяє медичним працівникам ефективно працювати з інформацією про пацієнтів, що покращує якість обслуговування та дозволяє зручно і швидко отримувати необхідні відомості (рис. 3.12, 3.13).

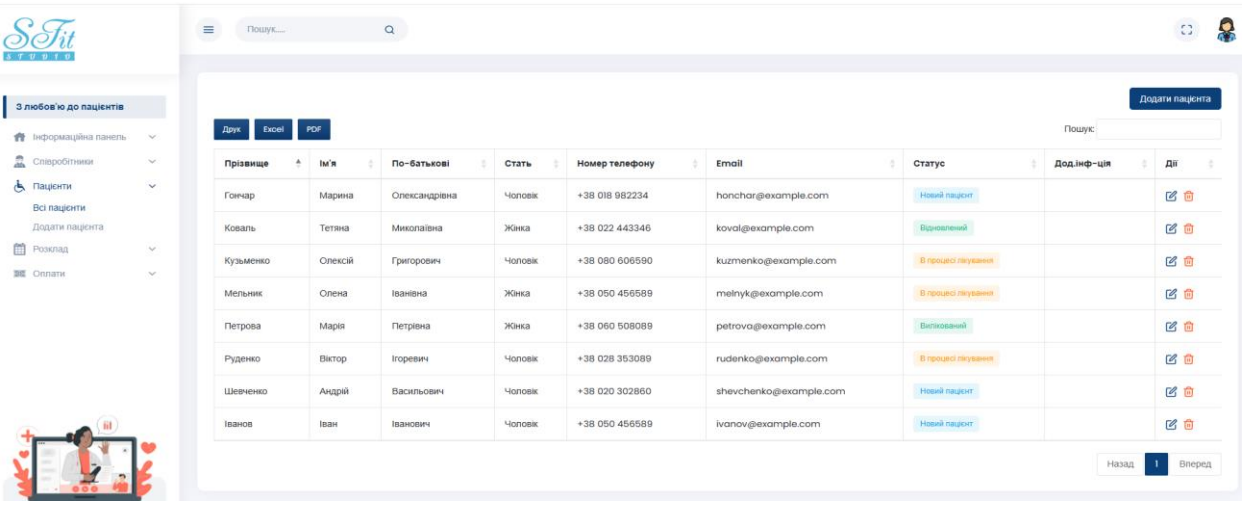


Рисунок 3.12 – Розділ «Всі пацієнти»

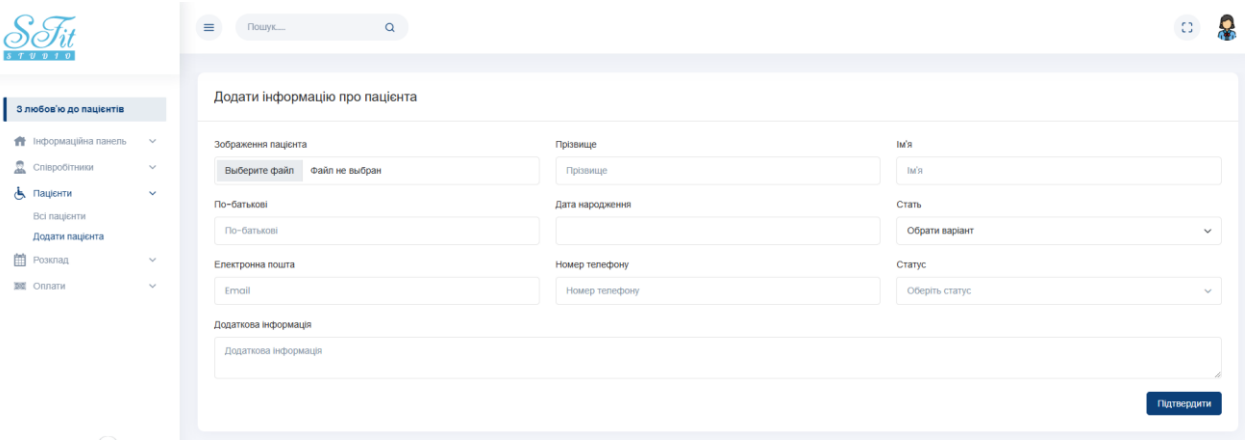


Рисунок 3.13 – Вікно для додавання інформації про пацієнта

Система також містить зручний функціонал для запису пацієнта на прийом через інтерфейс адміністратора, що дозволяє ефективно планувати візити та оптимізувати розклад лікарів. Адміністратор може вибрати пацієнта, лікаря та вказати бажану дату й час прийому, а також додати коментарі чи примітки, якщо це необхідно.

Для перегляду створених записів на прийом користувачеві доступні два варіанти.

По-перше, це перегляд через інтерактивний календар, де візуально відображаються всі заплановані прийоми у вигляді кольорових блоків із зазначенням пацієнта, лікаря та часу візиту. Календар дозволяє швидко побачити вільні слоти для нових записів або ж перенести існуючий візит, якщо це необхідно. Такий формат перегляду зручний для швидкої оцінки завантаженості лікарів і планування нових прийомів (рис. 3.14).

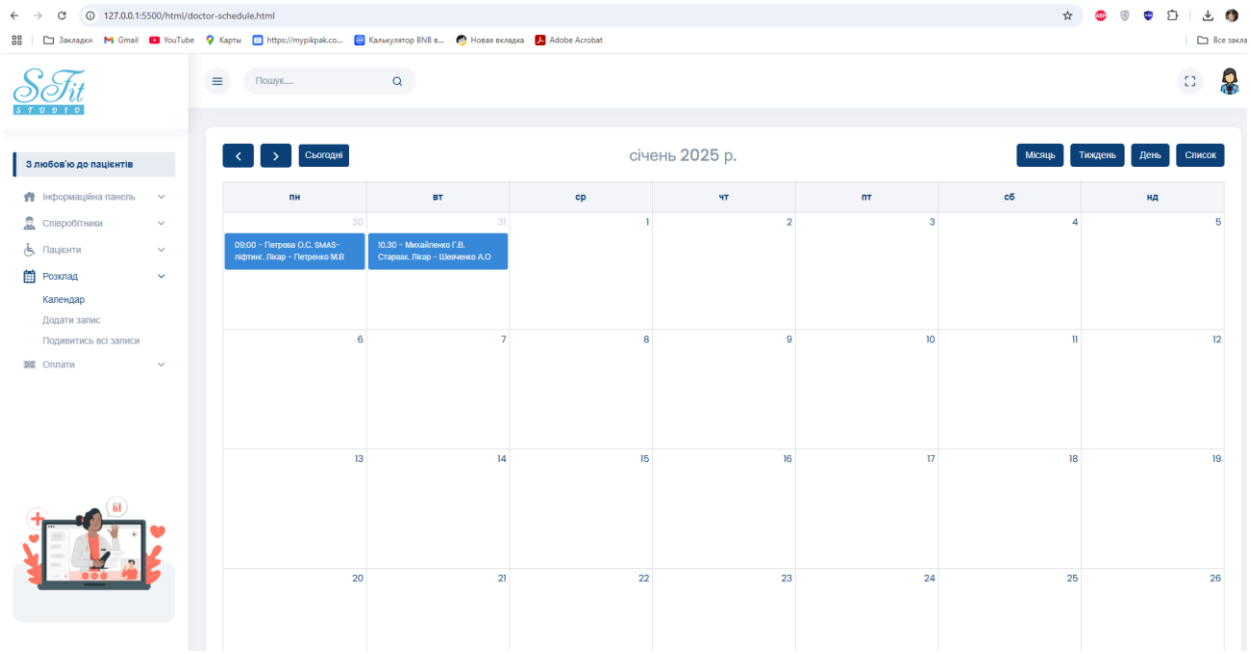


Рисунок 3.14 – Календар для перегляду візитів

По-друге, у системі є окремий розділ для роботи з записами на прийом у вигляді списку. Тут адміністратор може бачити детальну таблицю всіх візитів, де відображаються пацієнт, лікар, дата та час прийому, а також статус запису. У цьому розділі також можна додати новий запис, відредагувати існуючий або

видалити його за потреби. Це дозволяє зручно працювати з великою кількістю записів, швидко знаходити потрібні візити та вносити зміни.

Завдяки цим двом способам адміністратор може обирати найбільш зручний інструмент для перегляду та управління записами, що забезпечує гнучкість та ефективність у роботі з пацієнтами.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було розроблено CRM-систему для медичного закладу Sfit Studio з використанням мікросервісної архітектури на платформі ASP.NET Core. На початковому етапі було проведено аналіз існуючих рішень у сфері управління медичними даними та CRM-систем, що дозволило визначити ключові функціональні вимоги до майбутньої системи. Також було досліджено особливості мікросервісної архітектури та її переваги, такі як гнучкість, масштабованість та легкість інтеграції з іншими системами, що стало основою для вибору цього підходу.

У рамках роботи було спроєктовано архітектуру CRM-системи, яка включає окремі мікросервіси для управління пацієнтами, записами, послугами, графіками роботи персоналу та іншими аспектами функціонування медичного закладу. Крім того, було створено базу даних на основі MS SQL Server, яка забезпечує надійне зберігання великих обсягів даних. Для взаємодії з базою даних використовувалася технологія Entity Framework, що значно спростило реалізацію функціоналу обробки даних.

Програмна частина CRM-системи була реалізована за допомогою ASP.NET Core. У процесі розробки було створено вебінтерфейс із використанням HTML, CSS та JavaScript, що забезпечило зручність та інтерактивність для користувачів. Особливу увагу було приділено тестуванню системи, щоб гарантувати її відповідність технічним вимогам, стабільність роботи та безпеку даних.

Впровадження розробленої CRM-системи в медичному закладі Sfit Studio дозволило автоматизувати низку важливих процесів. Зокрема, система оптимізувала обробку даних пацієнтів, спростила управління записами на послуги та полегшила доступ медичного персоналу до необхідної інформації. Це сприяло підвищенню ефективності роботи закладу, скороченню часу на обробку інформації та покращенню якості надання медичних послуг.

Таким чином, розроблена система не лише вирішує актуальні задачі управління медичними закладами, а й має значний потенціал для подальшого розвитку. Її теоретична значимість полягає в аналізі та практичному застосуванні мікросервісної архітектури в медичній сфері, а практичне значення підтверджується успішним впровадженням у Sfit Studio. Отриманий досвід та результати можуть бути використані для створення подібних систем в інших медичних закладах, які прагнуть автоматизувати свої процеси та покращити взаємодію з клієнтами.

ПЕРЕЛІК ПОСИЛАНЬ

1. BloknotApp – CRM система для салонів краси. URL: https://bloknotapp.com/ (дата звернення: 10.07.2024).

2. EasyWeek – програма онлайн-запису та CRM управління бізнесом. URL: https://easyweek.com.ua/ (дата звернення: 12.07.2024).

3. CleverBox:CRM – автоматизація салонів, клінік та СПА. URL: https://cleverbox-crm.com/ (дата звернення: 14.07.2024).

4. Документація по C#. URL: https://learn.microsoft.com/uk-ua/dotnet/csharp/tour-of-csharp/ (дата звернення: 17.07.2024).

5. Exception-handling statements – throw and try, catch, finally – C#. URL: https://learn.microsoft.com/uk-ua/dotnet/csharp/language-reference/statements/exception-handling-statements/ (дата звернення: 14.08.2024).

6. Принципи роботи Garbage Collection. URL: https://dou.ua/lenta/articles/principles-of-garbage-collection/ (дата звернення: 18.08.2024).

7. SQL Injection. URL: https://www.w3schools.com/sql/sql_injection.asp (дата звернення: 19.08.2024).

8. ASP.NET Core | Open-Source web framework for .NET. URL: https://www.w3schools.com/sql/sql_injection.asp/ (дата звернення: 21.08.2024).

9. Mark J. Price. C# 12 and .NET 8 – Modern Cross-Platform Development Fundamentals: Start building websites and services with ASP.NET Core 8, Blazor, and EF Core 8 8th ed. Edition., 2024. 826 p.

10. JSON Web Tokens. URL: https://jwt.io/ (дата звернення: 21.09.2024).

11. Microsoft SQL documentation – SQL Server. URL: https://learn.microsoft.com/uk-ua/sql/?view=sql-server-ver16/ (дата звернення: 25.08.2024).

12. What is an ORM – The Meaning of Object Relational Mapping Database Tools. URL: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/ (дата звернення: 21.08.2024).

13. Огляд Entity Framework Core. URL: https://learn.microsoft.com/uk-ua/ef/core/ (дата звернення: 25.08.2024).

14. HTML і CSS довідник. URL: https://html-css.co.ua/ (дата звернення: 26.08.2024).

15. Sass:Documentation. URL: https://sass-lang.com/documentation/ (дата звернення: 28.08.2024).

16. JavaScipt | MDN. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/ (дата звернення: 02.09.2024).

17. AJAX: прискорення веброзробки. URL: https://foxminded.ua/ajax/ (дата звернення: 15.09.2024).

18. Bootstrap – The most popular HTML, CSS, and JS library in the world. URL: https://getbootstrap.com/ (дата звернення: 17.09.2024).

19. What is a UML Diagram. URL: https://miro.com/diagramming/what-is-a-uml-diagram/ (дата звернення: 20.09.2024).

20. Діаграми UML для моделювання процесів і архітектури проєкту. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html/ (дата звернення: 25.09.2024).