

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

КВАЛІФІКАЦІЙНА РОБОТА
на тему: «Розробка web-платформеної CRM-системи для
малого бізнесу»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

*Кваліфікаційна робота містить результати власних досліджень. Використання
ідей, результатів і текстів інших авторів мають посилання
на відповідне джерело*

(підпис) Ярослав СВИРИДЮК

Виконав: здобувач вищої освіти групи ПД-42

Ярослав СВИРИДЮК

Керівник: Юрій ЗАДОНЦЕВ
канд. техн. наук

Рецензент: _____

Київ 2026

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Свиридюку Ярославу Сергійовичу

1. Тема кваліфікаційної роботи: «Розробка web-платформеної CRM-системи для малого бізнесу»

керівник кваліфікаційної роботи канд техн. наук Юрій ЗАДОНЦЕВ,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості щодо управління відносинами з клієнтами (CRM) в умовах малого бізнесу; науково-технічна література з проєктування хмарних мультитенантних SaaS-архітектур; офіційна документація екосистеми Java Spring Boot, бібліотеки React та СКБД PostgreSQL; специфікації протоколів Stateless-автентифікації (JWT) та документація для інтеграції сторонніх сервісів (хмарного API Llama-3 та Slack Webhooks).

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Системний аналіз предметної області та технологій розробки хмарних CRM-систем.

2. Проєктування мультитенантної архітектури та бази даних системи.

3. Програмна реалізація веб-платформи та інтеграція інтелектуальних інструментів.
4. Тестування, оцінка якості та економічне обґрунтування програмного продукту.
5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів.
 2. Вимоги до програмного забезпечення.
 3. Програмні засоби реалізації.
 4. Вибір стек технологій.
 5. Алгоритм роботи застосунку.
 6. Діаграма класів.
 7. Екранні форми.
 8. Апробація результатів дослідження
6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Огляд сучасних технологій, архітектурних підходів та методів побудови CRM-систем для малого бізнесу	08.03-17.03.2026	
4	Проектування архітектури веб-платформної CRM-системи та структури бази даних	18.03-27.03.2026	
5	Програмна реалізація застосунку	28.03-10.04.2026	
6	Тестування застосунку	11.04-21.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	22.04-29.04.2026	
8	Розробка демонстраційних матеріалів	30.04-10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти _____
(підпис)

Ярослав СВИРИДЮК

Керівник
кваліфікаційної роботи _____
(підпис)

Юрій ЗАДОНЦЕВ

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 100 стор., 9 табл., 13 рис., 25 джерел.

Мета роботи – підтримка у побудові стабільного, прибуткового, клієнт-орієнтованого малого бізнесу за рахунок автоматизації процесів продаж через впровадження CRM-системи.

Об'єкт дослідження – процеси автоматизації управління продажами та взаємодією з клієнтами в умовах малого та мікробізнесу.

Предмет дослідження – технології, інструменти, методи, архітектурні рішення та алгоритми побудови мультитенантної веб-платформної CRM-системи.

Короткий зміст роботи. У кваліфікаційній роботі вирішено актуальне завдання цифровізації малого бізнесу шляхом створення сучасного програмного продукту «SmartBiz CRM». Проведено ґрунтовний аналіз предметної області та існуючих на ринку рішень, виявлено їх недоліки (висока вартість, надлишковість або відсутність ізоляції даних). Обґрунтовано вибір сучасного стека веб-технологій (Java Spring Boot 3, React, PostgreSQL).

Спроектовано клієнт-серверну архітектуру за моделлю SaaS із застосуванням мультитенантного підходу (Shared Database, Shared Schema) для надійної ізоляції даних орендарів. Розроблено бекенд із використанням принципів REST API та захистом кінцевих точок через Stateless JWT-автентифікацію та Role-Based Access Control (RBAC). Реалізовано інтерактивний фронтенд (Single Page Application) з Канбан-дошкою. Здійснено глибоку інтеграцію сторонніх сервісів: налаштовано подієво-орієнтовану відправку асинхронних сповіщень у корпоративний месенджер Slack через Webhooks, а також впроваджено хмарне API великої мовної моделі (Llama-3) для генерації III-аналітики клієнтського портфеля на основі знеособлених даних. Проведено модульне та функціональне тестування, оцінено економічну ефективність впровадження.

Сфера використання – автоматизація комерційної діяльності, оптимізація воронки продажів та управління лідами на підприємствах малого та середнього бізнесу (B2B та B2C сегменти).

КЛЮЧОВІ СЛОВА: CRM-СИСТЕМА, МАЛИЙ БІЗНЕС, ВЕБ-ПЛАТФОРМА, SAAS, МУЛЬТИТЕНАНТНІСТЬ, SPRING BOOT, REACT, ШТУЧНИЙ ІНТЕЛЕКТ, REST API, SLACK, JWT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	11
ВСТУП.....	12
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ РОЗРОБКИ CRM-СИСТЕМ	14
1.1 Концепція CRM-систем та їх роль у цифровій трансформації малого бізнесу	14
1.2 Порівняльний аналіз існуючих програмних рішень для управління продажами.....	15
1.2.1. Базові офісні інструменти (MS Excel, Google Sheets)	15
1.2.2. Універсальні хмарні SaaS-рішення (HubSpot, Pipedrive, KeyCRM)	16
1.2.3. Важкі ERP-системи та модульні платформи (Odoo, Salesforce)	17
1.2.4. Системне порівняння.....	18
1.3. Обґрунтування вибору технологічного стека та архітектурних патернів	19
1.3.1. Вибір архітектурного шаблону: Модульний моноліт проти Мікросервісів	19
1.3.2. Серверна екосистема: Java 21 та фреймворк Spring Boot 3	21
1.3.3. Клієнтська архітектура: React, Vite та Tailwind CSS.....	23
1.3.4. Система управління реляційними базами даних: PostgreSQL	24
1.4. Аналіз методів інтелектуальної обробки даних та сучасних підходів до корпоративних комунікацій.....	25
1.4.1. Впровадження великих мовних моделей (LLM) у процеси продажу	25
1.4.2. Асинхронні комунікації та подієво-орієнтована архітектура сповіщень	26
1.5. Аналіз технологій фронтенд-розробки та інтерфейсних рішень	27
1.5.1. Бібліотека React та компонентно-орієнтований підхід.....	28
1.5.2. Інструмент збірки Vite та оптимізація розробки	28
1.5.3. CSS-фреймворку Tailwind та Mobile-first дизайн	29
1.6. Інтелектуальні модулі: AI-аналітика (Llama-3) та асинхронні Slack-сповіщення	29
1.6.1. Програмна інженерія підказок (Prompt Engineering) для моделі Llama-3	29
1.6.2. Подієво-орієнтована архітектура та Slack-сповіщення	31
2. ПРОЄКТУВАННЯ МУЛЬТИТЕНАНТНОЇ АРХІТЕКТУРИ ТА БАЗИ ДАНИХ СИСТЕМИ	33

2.1. Обґрунтування вибору архітектурної моделі SaaS та стратегії мультитенантності	33
2.2. Функціональне моделювання системи засобами уніфікованої мови моделювання UML.....	36
2.2.1. Моделювання функціональних вимог: діаграма прецедентів.....	36
2.2.2. Моделювання динаміки автентифікації: діаграма послідовностей.....	38
2.2.3. Моделювання бізнес-процесу продажу: діаграма активностей.....	39
2.2.4. Моделювання архітектури компонентів: діаграма класів	39
2.3. Концептуальне та логічне моделювання бази даних	40
2.3.1. Побудова інфологічної моделі (ER-діаграма).....	41
2.3.2. Нормалізація бази даних та забезпечення цілісності.....	41
2.3.3. Специфікація фізичної моделі даних	42
2.3.4. Стратегія мультитенантності на рівні схеми	43
2.4. Проектування підсистеми безпеки та архітектури JWT-автентифікації.....	44
2.4.1. Обґрунтування вибору Stateless-підходу та JWT	44
2.4.2. Структурне проектування JSON Web Token.....	44
2.4.3. Механізм автентифікації та життєвий цикл токена	45
2.4.4. Рольова модель доступу (RBAC) та безпека API	46
2.5. Проектування інтеграцій із зовнішніми сервісами та механізмів асинхронної взаємодії	47
2.5.1. Подієво-орієнтована архітектура модуля сповіщень (Slack Webhooks) .	47
2.5.2. Проектування взаємодії з хмарним API штучного інтелекту (Llama-3) .	50
2.5.3. Моделювання асинхронності на базі Virtual Threads (Java 21 Project Loom).....	50
2.5.4. Забезпечення відмовостійкості (Fault Tolerance) інтеграцій.....	51
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ПЛАТФОРМИ SMARTBIZ CRM.....	53
3.1. Вибір та налаштування інструментального середовища розробки	53
3.2. Розробка серверної частини на базі Spring Boot 3 та Java 21	54
3.2.1. Структура пакетів серверного застосунку	54
3.2.2. Програмна реалізація об'єктно-реляційної моделі (JPA Entities)	55
3.2.3. Реалізація шару доступу до даних (Repository Layer).....	56
3.3. Програмна реалізація підсистеми безпеки та JWT-автентифікації	57
3.3.1. Генерація та валідація JSON Web Token.....	57

3.3.2. Перехоплення запитів: Фільтр автентифікації.....	58
3.3.3. Конфігурація ланцюга безпеки (SecurityFilterChain)	59
3.4. Програмна реалізація модуля інтеграції з месенджером Slack на базі віртуальних потоків Java 21	60
3.4.1. Конфігурація асинхронного виконання та Project Loom	61
3.4.2. Розробка сервісу відправки Slack-сповіщень та форматування повідомлень	62
3.4.3. Інтеграція сповіщень на рівні сервісного шару та транзакційність	63
3.5. Програмна реалізація модуля інтелектуальної бізнес-аналітики на базі LLM (Llama-3).....	64
3.5.1. Програмна реалізація об'єктів передачі даних для III (DTO)	64
3.5.2. Реалізація сервісу інтелектуального аналізу на базі Spring AI	65
3.5.3. Архітектурні особливості реалізації REST-клієнта	66
3.6. Програмна реалізація клієнтської частини та інтерактивного інтерфейсу (React & Tailwind CSS)	67
3.6.1. Архітектура фронтенд-застосунку та захищена маршрутизація	67
3.6.2. Мережева взаємодія та перехоплення JWT-токенів (Axios Interceptors) ...	68
3.6.3. Управління станом (State Management) та інтерактивна Канбан-дошка	69
3.6.4. Адаптивна стилізація за допомогою Tailwind CSS	70
4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПРОГРАМНОГО ПРОДУКТУ	71
4.1. Стратегія забезпечення якості та модульне тестування компонентів	71
4.2. Функціональне тестування та перевірка безпеки API.....	72
4.3. Навантажувальне тестування та перевірка ефективності віртуальних потоків	74
4.4. Оцінка економічної ефективності впровадження системи.....	75
4.4.1. Розрахунок економії робочого часу:.....	75
4.4.2. Розрахунок прибутку від утримання клієнтів:.....	76
4.4.3. Розрахунок окупності (ROI):	76
ВИСНОВКИ.....	77
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	80
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛ.....	82
ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ.....	89

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

CI/CD – Continuous Integration / Continuous Deployment

CRM – Customer Relationship Management

CRUD – Create, Read, Update, Delete

DTO – Data Transfer Object

ERP – Enterprise Resource Planning

GDPR – General Data Protection Regulation

HTTP – Hypertext Transfer Protocol

JPA – Java Persistence API

JSON – JavaScript Object Notation

JVM – Java Virtual Machine

JWT – JSON Web Token

LLM – Large Language Model

ORM – Object-Relational Mapping

RBAC – Role-Based Access Control

REST – Representational State Transfer

ROI – Return on Investment

SaaS – Software as a Service

SPA – Single Page Application

SQL – Structured Query Language

TCO – Total Cost of Ownership

URL – Uniform Resource Locator

UX/UI – User Experience / User Interface

БД – База Даних

MVP – Minimum Viable Product (Мінімально життєздатний продукт)

СКБД – Система Керування Базами Даних

ІІІ – Штучний Інтелект

ВСТУП

Актуальність. У сучасних умовах цифровізації економіки підприємства малого та мікробізнесу часто втрачають клієнтів і прибуток через використання застарілих інструментів обліку продажів (паперові нотатки, Excel-таблиці) та відсутність оперативної аналітики. Впровадження легкого хмарного програмного забезпечення із автоматизованою ШІ-аналітикою клієнтського портфеля та миттєвими сповіщеннями дозволяє оптимізувати воронку продажів, скоротити час обробки заявок і підвищити ефективність командної роботи. Тема є високоактуальною в умовах зростання попиту на доступні інтелектуальні SaaS-рішення для автоматизації комерційної діяльності.

Об'єктом дослідження є процеси автоматизації управління продажами та взаємодією з клієнтами в умовах малого та мікробізнесу.

Предметом дослідження є програмне забезпечення для мультитенантної веб-платформної CRM-системи.

Метою роботи є підвищення ефективності управління продажами у сегменті малого бізнесу шляхом проектування та програмної реалізації хмарної CRM-системи з інтегрованими смарт-функціями.

Методи дослідження: першим кроком було проаналізовано існуючі аналоги та рішення для автоматизації продажів (HubSpot, Odoo, Google Sheets) для формування технічних та бізнес-вимог до власного програмного забезпечення.

Далі було проведено огляд технологічного стека, здатного забезпечити ефективну реалізацію системи. Для розробки серверної частини було обрано платформу Java Spring Boot, для клієнтського інтерфейсу — бібліотеку React, а для збереження даних — реляційну СУБД PostgreSQL. Управління міграціями бази даних здійснювалося через Flyway. GitHub та Git використовувалися як системи контролю версій, а Postman — для інтерактивного тестування REST API. Дослідження архітектури, процесів інтеграції сторонніх сервісів (Llama-3 через REST API та Slack через Webhooks) здійснювалося безпосередньо під час етапів проектування, розробки та емпіричного тестування системи.

Наукова новизна роботи визначається наступними функціональними та архітектурними складовими: імплементація гібридної мультитенантної фільтрації даних орендарів на рівні Spring Security Context без інфраструктурного перевантаження сервера; розробка модуля ІІІ-аналітики для автоматизованої генерації текстових бізнес-звітів і рекомендацій менеджерам на основі агрегованого клієнтського портфеля; реалізація алгоритму препроцесингу та знеособлення комерційних даних перед відправкою до стороннього LLM API для захисту конфіденційності; автоматизоване фонове формування асинхронних сповіщень у Slack за допомогою віртуальних потоків Java 21 (Project Loom). Застосунок виступає як ефективний інструмент проактивної підтримки прийняття рішень менеджерами.

Практична значущість результатів полягає у можливості безпосереднього використання розробленого прототипу хмарної системи «SmartBiz CRM» у реальному секторі малого бізнесу для автоматизації комерційної діяльності, миттєвої координації менеджерів та зниження витрат на аналітику.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ РОЗРОБКИ CRM-СИСТЕМ

1.1 Концепція CRM-систем та їх роль у цифровій трансформації малого бізнесу

Цифрова трансформація малого бізнесу в Україні сьогодні відбувається в умовах безпрецедентних викликів: глобальної діджиталізації, потреби у віддаленій роботі та необхідності оперативного прийняття рішень. Для малих підприємств, які зазвичай мають обмежені людські та фінансові ресурси, CRM-система стає інтелектуальним ядром, що консолідує всі бізнес-процеси в єдиному цифровому просторі.

Роль CRM у процесах цифрової трансформації. Впровадження платформи «SmartBiz CRM» дозволяє малому бізнесу вирішити низку стратегічних завдань:

Централізація інформаційних активів: усунення проблеми «інформаційних островів», коли дані про клієнтів розпорошені між паперовими носіями, особистими месенджерами менеджерів та електронними таблицями.

Оптимізація операційної діяльності: автоматизація рутинних завдань (створення карток угод, нагадування про дзвінки) звільняє до 30-40% робочого часу персоналу для безпосередніх комунікацій.

Підвищення прозорості бізнесу: керівник отримує об'єктивну картину воронки продажів у реальному часі, що дозволяє виявляти «вузькі місця» на ранніх етапах.

Забезпечення безперервності процесів: завдяки хмарній архітектурі та мобільній адаптивності, менеджери можуть ефективно працювати з будь-якої точки світу, що є критичним для сучасного українського контексту.

Стратегічна цінність для малого бізнесу. На відміну від великих корпорацій, малий бізнес не може дозволити собі втрату навіть одного ліда (потенційного клієнта). CRM-система забезпечує так званий «ефект пам'яті» організації. Кожна

взаємодія — чи то дзвінок, чи зустріч, чи повідомлення у Slack — фіксується в системі. Це дозволяє персоналізувати кожне наступне звернення, підвищуючи лояльність (LTV — Lifetime Value) та зменшуючи вартість залучення клієнта (CAC — Customer Acquisition Cost).

Таким чином, цифрова трансформація через впровадження CRM є не лише технологічним оновленням, а й переходом до науково обґрунтованого управління даними. Це закладає фундамент для масштабування бізнесу та підвищення його конкурентоспроможності в умовах глобального цифрового ринку.

1.2 Порівняльний аналіз існуючих програмних рішень для управління продажами

Ринок програмного забезпечення для управління взаємовідносинами з клієнтами є одним із найбільш динамічних сегментів світової ІТ-індустрії. Проте, незважаючи на величезну кількість пропозицій, підприємства малого та мікробізнесу в Україні часто стикаються з проблемою вибору: існуючі рішення є або занадто примітивними і не відповідають вимогам безпеки, або занадто складними та економічно недоцільними.

Для обґрунтування технічних вимог до проєктуваної веб-платформи «SmartBiz CRM» було проведено системний аналіз трьох основних груп інструментів, які найчастіше використовуються малим бізнесом.

1.2.1. Базові офісні інструменти (MS Excel, Google Sheets)

На початкових етапах розвитку більшість дрібних підприємств використовують електронні таблиці як імпровізовану базу даних. Цей феномен часто відносять до категорії «Тіньових ІТ» (Shadow IT), оскільки такі інструменти не контролюються політиками корпоративної безпеки.

З інженерної точки зору використання електронних таблиць замість реляційних баз даних має ряд критичних архітектурних вад:

Відсутність контролю конкурентного доступу (Concurrency Control): При одночасній роботі кількох менеджерів виникають конфлікти блокування або так звані «втрачені оновлення» (Lost Updates).

Порушення цілісності даних (Data Integrity): Відсутність зовнішніх ключів (Foreign Keys) та строгих типів даних призводить до дублювання записів про клієнтів та неможливості побудови складних зв'язків типу «Багато-до-багатьох».

Вразливість інформаційної безпеки: У таблицях неможливо реалізувати гранульовану рольову модель доступу (RBAC). Будь-який менеджер, що має доступ до файлу, може скопіювати всю клієнтську базу компанії.

Статичність та відсутність тригерів: Таблиці не здатні самостійно ініціювати фонові процеси. Наприклад, неможливо автоматично відправити POST-запит (Webhook) у корпоративний месенджер при зміні статусу комірки.

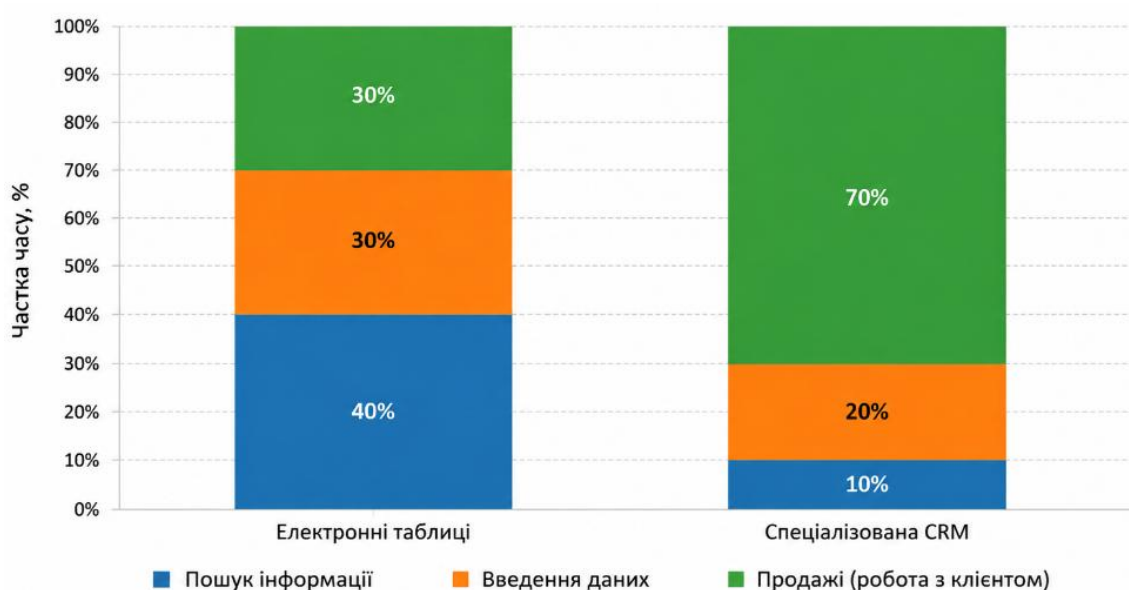


Рис. 1.1 Порівняльна діаграма витрат часу менеджера на обробку ліда)

1.2.2. Універсальні хмарні SaaS-рішення (HubSpot, Pipedrive, KeyCRM)

Ця категорія є найпопулярнішою серед бізнесу, що пройшов етап електронних таблиць. Ці системи пропонують зручний веб-інтерфейс, Канбан-дошки та базову аналітику. Проте для українського малого бізнесу вони створюють низку бар'єрів:

Економічна недоцільність при масштабуванні: Більшість таких сервісів використовують модель тарифікації «Pay-per-user» (оплата за кожного користувача). Зі зростанням команди вартість підписки зростає експоненціально, досягаючи сотень доларів на місяць.

Проблема «Vendor Lock-in» (Залежність від постачальника): Системи мають закритий вихідний код та пропрієтарні формати зберігання даних, що робить міграцію на інші платформи вкрай складною.

Обмеження інновацій: Передові функції, такі як інтелектуальна аналітика на базі генеративних нейромереж (LLM), зазвичай доступні лише у найдорожчих тарифах (Enterprise), що робить їх недоступними для мікробізнесу.

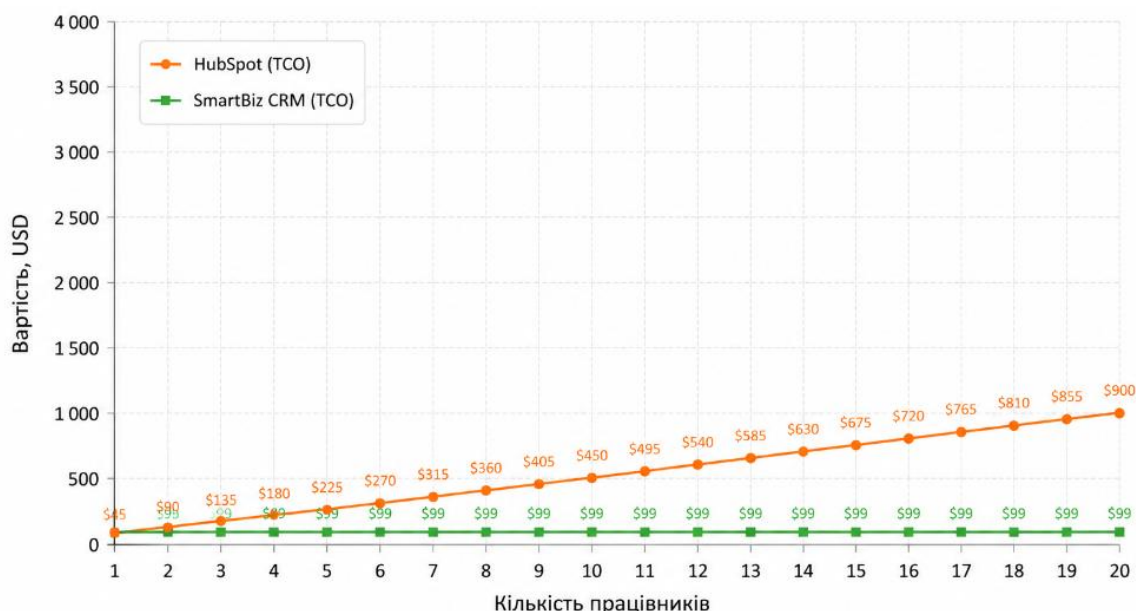


Рис. 1.2 Графік зростання вартості володіння (TCO) SaaS-системами залежно від кількості користувачів

1.2.3. Важкі ERP-системи та модульні платформи (Odoo, Salesforce)

Системи класу ERP (Enterprise Resource Planning) забезпечують тотальну автоматизацію всіх процесів: від виробництва до бухгалтерії та CRM. Яскравим представником є система Odoo.

Архітектурна надлишковість: Odoo базується на мові Python та СУБД PostgreSQL, використовуючи архітектуру з виділеною базою даних для кожного

клієнта (Database-per-tenant). Для підприємства з 3-5 співробітниками розгортання такої інфраструктури є не виправданим споживанням апаратних ресурсів сервера.

Складність інтеграції та супроводу: Для налаштування бізнес-логіки та адаптації під локальне законодавство вимагається залучення вузькопрофільних програмістів, послуги яких значно перевищують бюджет малого бізнесу.

1.2.4. Системне порівняння

Для формалізації результатів дослідження було складено порівняльну матрицю за ключовими технічними критеріями, які є визначальними для розробки сучасної, безпечної та ефективної системи.

Таблиця 1.1

Порівняльний аналіз архітектурних та функціональних характеристик CRM-систем

Технічний параметр / Критерій	MS Excel / Google Sheets	Pipedrive / HubSpot	Odoo	Проектована система (SmartBiz CRM)
Архітектура системи	Файлова / Базовий хмарний документ	Хмарна (закрита SaaS)	Монолітна ERP або мікросервіси	Клієнт-серверна SaaS (Multi-tenancy)
Тип клієнтського застосунку	Настільний застосунок / Web-view	Багатосторінковий (MPA) / SPA	Багатосторінковий (MPA)	Single Page Application (React + Vite)
Ізоляція даних у БД	Відсутня (на рівні окремого файлу)	Логічна ізоляція провайдера	Окрема база (Database-per-tenant)	Shared Database, Shared Schema (фільтрація за company_id)
Механізм автентифікації та безпеки	Пароль на відкриття файлу / OAuth	Session-based / OAuth 2.0	Session-based (Cookies)	Stateless JWT-токени + Spring Security (RBAC)
Керування міграціями БД	Відсутнє	Внутрішні закриті інструменти	Власний ORM-модуль	Автоматизовані міграції (Flyway)

Порівняльний аналіз архітектурних та функціональних характеристик CRM-систем

Технічний параметр / Критерій	MS Excel / Google Sheets	Pipedrive / HubSpot	Odoo	Проектована система (SmartBiz CRM)
Інтеграція генеративного ШІ	Відсутня	Обмежена (закриті AI-моделі у преміум-тарифах)	Модульна (потребує налаштування)	Пряма інтеграція через REST API (модель Llama-3 / Groq)
Документування API	Не застосовується	Статична документація (Proprietary)	Статична документація	Динамічна інтерактивна документація (Swagger / OpenAPI)
Механізм відправки сповіщень	Відсутній	Webhooks / Вбудовані тригери	Внутрішня шина подій	Асинхронні фонові процеси (@Async) + Telegram Bot API

1.3. Обґрунтування вибору технологічного стека та архітектурних патернів

Успіх будь-якого програмного продукту на етапі його життєвого циклу значною мірою залежить від рішень, прийнятих на етапі проєктування базової архітектури та вибору технологічного стека. Для SaaS-платформи «SmartBiz CRM» ключовими критеріями вибору були: надійність збереження даних, швидкість розробки (Time-to-Market), можливість горизонтального масштабування та оптимізація витрат на інфраструктуру.

1.3.1. Вибір архітектурного шаблону: Модульний моноліт проти Мікросервісів

Сучасний тренд інженерії програмного забезпечення часто диктує використання мікросервісної архітектури за замовчуванням. Однак глибокий

аналіз предметної області малого бізнесу показав, що для стартової версії платформи (MVP) такий підхід є антипатерном (Overengineering).

Мікросервіси вимагають розв'язання складних проблем розподілених систем: мережевих затримок (Network Latency), узгодженості даних (Distributed Transactions, патерн SAGA), складного моніторингу та значних витрат на оркестрацію (Kubernetes). Зважаючи на обмежені ресурси та потребу у високій швидкості розробки, для «SmartBiz CRM» було обрано архітектуру Модульного моноліту (Modular Monolith).

У модульному моноліті весь застосунок розгортається як єдиний виконуваний файл, проте всередині код жорстко розділений на незалежні бізнес-домени (клієнти, угоди, інтеграції). Це забезпечує:

Високу продуктивність: Взаємодія між модулями відбувається через прямі виклики методів у пам'яті (In-memory calls), а не через повільні мережеві HTTP-запити.

Простоту транзакцій: Гарантується строга узгодженість бази даних (ACID) без необхідності застосування двофазних комітів (2PC).

Легкість рефакторингу: У майбутньому, при зростанні навантаження, чітко ізольовані модулі можна буде легко винести в окремі мікросервіси.

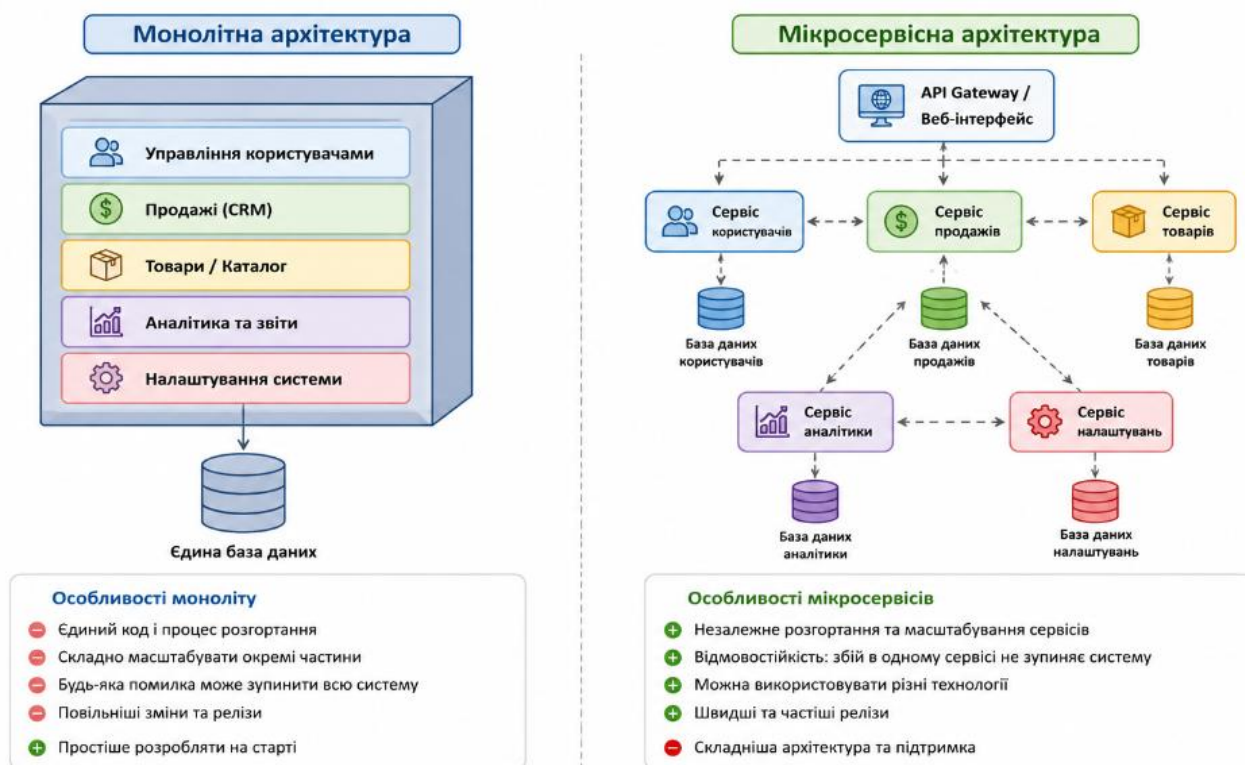


Рис. 1.3 Порівняльна схема архітектур: моноліт проти мікросервісів

1.3.2. Серверна екосистема: Java 21 та фреймворк Spring Boot 3

У якості основної мови програмування бекенду було обрано Java 21, що є стандартом де-факто у сфері розробки корпоративного (Enterprise) програмного забезпечення завдяки своїй строгій типізації, надійній віртуальній машині (JVM) та зворотній сумісності.

Револьюційним фактором на користь Java 21 стало впровадження Віртуальних потоків (Project Loom). Традиційні системи керування потоками створювали обгортку над потоками операційної системи (OS Threads). Вони є "важкими", вимагають близько 1 МБ пам'яті та значного часу на перемикання контексту (Context Switching).

Це створювало проблеми при великій кількості одночасних I/O операцій, таких як звернення до API Slack або Llama-3. Віртуальні потоки керуються безпосередньо JVM. Їх можна створювати мільйонами без ризику переповнення

пам'яті (OutOfMemoryError), що дозволяє серверу обробляти тисячі паралельних вебхуків асинхронно, не блокуючи роботу інших користувачів системи.

Основним інфраструктурним каркасом виступає фреймворк Spring Boot 3.

Inversion of Control (IoC) та Dependency Injection (DI): Spring бере на себе життєвий цикл об'єктів (Bean), що знижує зв'язність коду та полегшує модульне тестування (Unit Testing) за допомогою мок-об'єктів.

Spring Security: Використовується для побудови фільтрів автентифікації на базі JWT-токенів та забезпечення рольової моделі (RBAC) доступу до REST API.

Spring Data JPA: Виступає абстракцією над ORM-провайдером Hibernate. Це дозволяє розробнику працювати з базою даних у термінах Java-об'єктів, автоматично генеруючи параметризовані SQL-запити (Prepared Statements), що є абсолютним захистом від SQL-ін'єкцій.

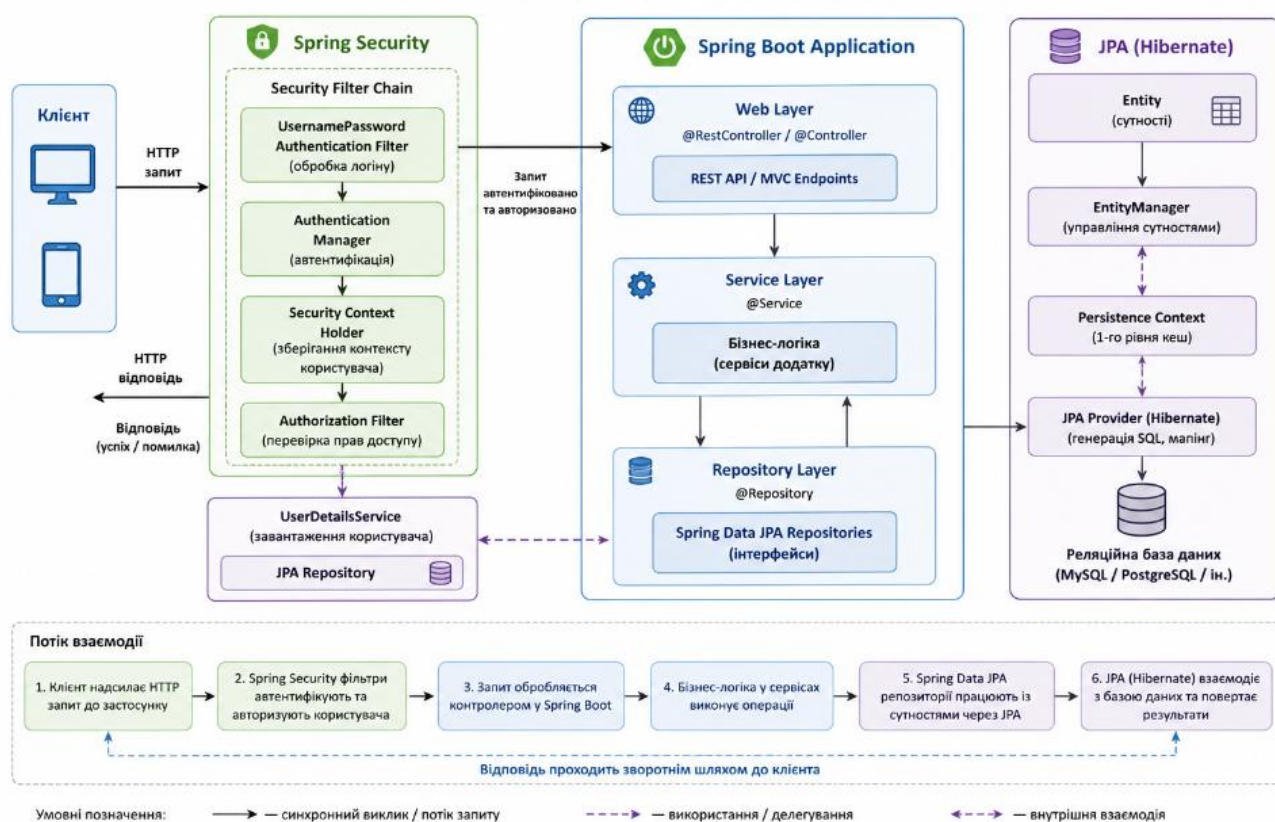


Рис. 1.4 Порівняльна схема архітектур: моноліт проти мікросервісів

1.3.3. Клієнтська архітектура: React, Vite та Tailwind CSS

Для забезпечення користувацького досвіду (UX) рівня десктопних застосунків, фронтенд реалізовано за архітектурою Single Page Application (SPA) за допомогою бібліотеки React.

На відміну від класичних багатосторінкових застосунків (MPA), SPA завантажує базовий HTML-каркас та JavaScript-бандл лише один раз. Подальша комунікація із сервером відбувається у фоновому режимі (AJAX/Fetch) шляхом обміну легковагими JSON-структурами.

Ключові технології фронтенд-шару:

Virtual DOM (React): Оновлення реального дерева DOM у браузері є ресурсомісткою операцією. React використовує віртуальне представлення DOM-дерева в пам'яті, обчислює різницю (diffing) при зміні стану (State) і точково оновлює лише ті елементи інтерфейсу (наприклад, картку на Канбан-дошці), які дійсно змінилися. Це забезпечує плавний рендеринг при 60 кадрах на секунду.

Інструмент збірки Vite: Замінив традиційний Webpack. Vite використовує нативні ES-модулі браузера (ESM), що скорочує час запуску середовища розробки з хвилин до мілісекунд (швидке HMR — Hot Module Replacement).

Tailwind CSS: Замість написання кастомних CSS-правил або використання важких компонентних бібліотек (як Bootstrap), використано методологію Utility-first. Це дозволяє створювати складні, повністю адаптивні (Responsive) інтерфейси прямо в JSX-розмітці, при цьому фінальний CSS-файл мінімізується, видаляючи всі невикористані стилі (Tree shaking).

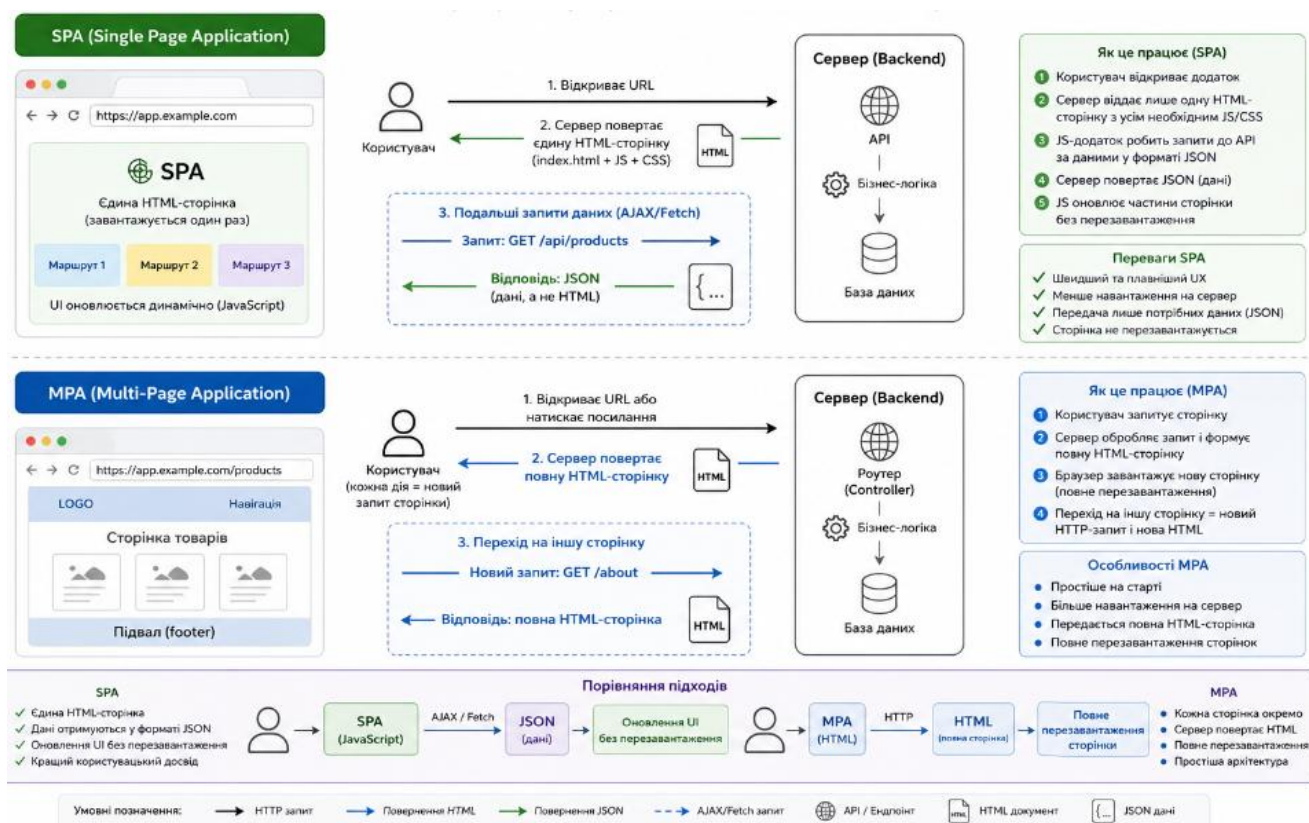


Рис. 1.5 Порівняльна схема архітектур: моноліт проти мікросервісів

1.3.4. Система управління реляційними базами даних: PostgreSQL

Враховуючи, що CRM-система оперує фінансовими даними (суми угод, контракти), вибір NoSQL рішень (наприклад, MongoDB) був відкинутий через слабку підтримку транзакційності. Обробка бізнес-даних вимагає суворого дотримання принципів ACID (Atomicity, Consistency, Isolation, Durability).

Вибір зроблено на користь PostgreSQL — найпотужнішої у світі об'єктно-реляційної СУБД з відкритим вихідним кодом.

Ізоляція даних: Надійна підтримка зовнішніх ключів (Foreign Keys) з каскадним видаленням гарантує, що у базі не залишиться «осиротілих» записів.

Гнучкість: PostgreSQL підтримує тип даних JSONB, що ідеально підходить для збереження неструктурованих відповідей від ШІ-моделі Llama-3, поєднуючи переваги реляційних та NoSQL баз даних в одній платформі.

Керування міграціями (Flyway): Для автоматизованого створення таблиць та контролю версій схеми БД під час CI/CD розгортання інтегровано утиліту Flyway.

1.4. Аналіз методів інтелектуальної обробки даних та сучасних підходів до корпоративних комунікацій

Еволюція CRM-систем у 2025–2026 роках характеризується переходом від статичних систем фіксації даних (Systems of Record) до інтелектуальних систем підтримки прийняття рішень (Systems of Intelligence). Для малого бізнесу, де керівник часто поєднує ролі маркетолога, аналітика та головного сейлз-менеджера, автоматизація аналітики та комунікацій є критичною необхідністю.

1.4.1. Впровадження великих мовних моделей (LLM) у процеси продажу

Традиційна бізнес-аналітика в CRM-системах базується на статичних дашбордах та заздалегідь запрограмованих звітах (наприклад, графік конверсії чи лійка продажів). Проте такий підхід вимагає від користувача самостійного трактування цих графіків.

Сучасний підхід полягає в інтеграції генеративного штучного інтелекту, зокрема Великих мовних моделей (Large Language Models, LLM). У проєкті «SmartBiz CRM» обґрунтовано використання моделі Llama-3 (розробки компанії Meta), доступ до якої здійснюється через швидкісне хмарне API (наприклад, Groq).

Вибір моделі з відкритими вагами (Open-weights), такої як Llama-3, замість пропрієтарних закритих моделей, має дві вагомі переваги:

1. Економічна ефективність: Вартість генерації токенів через API для Llama-3 є значно нижчою, що дозволяє підтримувати низьку вартість підписки на саму CRM-систему.

2. Керованість та конфіденційність: Алгоритми препроцесингу дозволяють розробнику чітко контролювати контекст. Перед відправкою запиту до LLM-провайдера система агрегує дані (кількість лідів, суми угод, етапи) та знеособлює їх (видаляє імена клієнтів, контактні дані). Модель працює лише з метаданими та числовими показниками, що відповідає європейським стандартам захисту персональних даних (GDPR) та українському законодавству.

ІІІ виступає у ролі "віртуального бізнес-аналітика": він аналізує тривалість перебування угод на певних етапах Канбан-дошки, виявляє аномалії (наприклад, зависання великих угод на етапі "В процесі") та генерує текстові рекомендації природною мовою щодо наступних кроків менеджера.



Рис. 1.6 Принципи взаємодії CRM та хмарної LLM-моделі

1.4.2. Асинхронні комунікації та подієво-орієнтована архітектура сповіщень

Швидкість реакції на нову заявку (Lead Response Time) є одним із найважливіших KPI у продажах. Згідно з дослідженнями, ймовірність успішної кваліфікації ліда падає у кілька разів, якщо менеджер зв'язується з клієнтом пізніше ніж через годину після звернення.

Традиційні методи сповіщень, такі як Email (електронна пошта) або SMS, втрачають свою ефективність. Email-скриньки перевантажені спамом, а SMS є фінансово затратними. На початкових етапах проєктування платформи розглядався варіант інтеграції з Telegram Bot API. Проте, зважаючи на корпоративні стандарти B2B-сегмента та політику інформаційної безпеки, було прийнято стратегічне рішення перевести підсистему сповіщень на корпоративний месенджер Slack.

Переваги використання Slack Incoming Webhooks:

Розмежування контекстів: Робоча комунікація відбувається у спеціалізованому просторі (Workspace), не перетинаючись з особистими листуваннями менеджерів (як це буває у Telegram чи Viber).

Канальна маршрутизація: Система дозволяє налаштовувати різні вебхуки для різних подій (наприклад, канал #sales-new для нових лідів та #sales-won для успішно закритих угод).

Багате форматування: Використання формату Slack Block Kit дозволяє відправляти структуровані повідомлення з кнопками, кольоровими індикаторами статусу та прямими посиланнями на картку клієнта в CRM.

Реалізація такого механізму на рівні бекенду вимагає застосування подієво-орієнтованої архітектури (Event-Driven Architecture), де збереження даних в основній БД ініціює асинхронну подію, яка обробляється у фоновому віртуальному потоці (Project Loom), щоб не уповільнювати HTTP-відповідь для користувача веб-інтерфейсу.



Рис. 1.7 Маршрутизація сповіщень через Slack Webhooks

1.5. Аналіз технологій фронтенд-розробки та інтерфейсних рішень

Сучасні користувачі веб-застосунків очікують від інтерфейсу миттєвої реакції, що за рівнем комфорту не поступається десктопним програмам. Для реалізації клієнтської частини веб-платформи «SmartBiz CRM» було обрано стек технологій, що базується на бібліотеці React, інструменті збірки Vite та CSS-фреймворку Tailwind.

1.5.1. Бібліотека React та компонентно-орієнтований підхід

Вибір React (розробка компанії Meta) обумовлений його декларативною природою та ефективним механізмом оновлення інтерфейсу. Головною особливістю React є використання Virtual DOM (віртуального об'єктного моделювання документа).

Процес роботи Virtual DOM включає три етапи:

1. При зміні стану (State) будь-якого компонента, React будує нове віртуальне дерево інтерфейсу.
2. Алгоритм порівняння (Diffing algorithm) обчислює мінімальну різницю між старим та новим станом.
3. React виконує «патчинг» лише тих елементів реального DOM у браузері, які дійсно змінилися.

Для CRM-системи, де центральним елементом є Канбан-дошка, це має вирішальне значення. Коли менеджер перетягує картку угоди з одного етапу на інший, React оновлює лише стан конкретної колонки та картки, що забезпечує плавність анімації та відсутність затримок навіть при великій кількості активних угод.

1.5.2. Інструмент збірки Vite та оптимізація розробки

Традиційні інструменти збірки (наприклад, Webpack) при кожній зміні в коді змушені перезбирати значну частину застосунку, що на великих проєктах займає багато часу. Vite використовує принципово інший підхід — він обслуговує вихідний код через нативні ES-модулі браузера (ESM).

Завдяки механізму Hot Module Replacement (HMR), зміни в коді відображаються в браузері миттєво без перезавантаження всієї сторінки і втрати поточного стану інтерфейсу. Це дозволяє розробнику CRM-системи зосередитися на тонкому налаштуванні логіки взаємодії, значно прискорюючи цикл розробки та тестування.

1.5.3. CSS-фреймворку Tailwind та Mobile-first дизайн

Для стилізації інтерфейсу обрано Tailwind CSS, який базується на методології Utility-first. Замість написання громіздких кастомних класів, розробник використовує атомарні класи безпосередньо в JSX-розмітці.

Ключові переваги Tailwind для «SmartBiz CRM»:

Адаптивність (Responsive Design): Використання префіксів (наприклад, md:, lg:) дозволяє легко реалізувати стратегію Mobile-first. Оскільки менеджери малого бізнесу часто працюють «у полях» (на зустрічах або складах), інтерфейс автоматично перегрупується для зручного керування пальцями на смартфоні.

JIT (Just-In-Time) компілятор: Tailwind генерує CSS «на льоту», включаючи у фінальний файл лише ті стилі, які реально використані в проєкті. Це зменшує розмір бандла до мінімуму, що критично для швидкого завантаження системи через мобільний інтернет.

1.6. Інтелектуальні модулі: AI-аналітика (Llama-3) та асинхронні Slack-сповіщення

Еволюція корпоративного програмного забезпечення у 2025–2026 роках характеризується переходом від статичних систем фіксації даних (Systems of Record) до проактивних інтелектуальних систем підтримки прийняття рішень (Systems of Intelligence). Сучасна CRM-система повинна не лише зберігати інформацію, а й самостійно генерувати інсайти та керувати увагою користувача. Для вирішення цих завдань у платформу «SmartBiz CRM» інтегровано генеративний штучний інтелект та подієво-орієнтовану систему корпоративних сповіщень.

1.6.1. Програмна інженерія підказок (Prompt Engineering) для моделі Llama-3

Як було обґрунтовано в підрозділі 1.4.1, система використовує велику мовну модель у якості віртуального аналітика. Процес взаємодії з LLM не зводиться до

простого відправлення сирих даних. Він вимагає складної програмної підготовки контексту (Context Window). Архітектура ІІІ-модуля системи реалізує наступні етапи:

1. Призначення системної ролі (System Prompt): Моделі задається жорсткий контекст поведінки. Наприклад: «Ти є головним аналітиком відділу продажів B2B компанії. Твоє завдання — проаналізувати надану статистику та надати три конкретні, короткі (дієві) поради щодо підвищення конверсії».

2. Агрегація метрик: Серверна частина (Spring Boot) виконує складні SQL-запити до бази даних PostgreSQL, збираючи ключові показники ефективності (KPI) компанії за останній місяць: загальний дохід, середня тривалість угоди на кожному етапі Канбан-дошки, відсоток відмов (Churn Rate).

3. Форматування виводу (Structured Output): Для коректного відображення на фронтенді (React), промпт містить інструкцію повернути відповідь у строгому форматі Markdown або JSON, що гарантує цілісність інтерфейсу.

Забезпечення конфіденційності (Data Privacy): Критичним викликом при інтеграції хмарних LLM є захист комерційної таємниці та персональних даних клієнтів відповідно до стандартів GDPR. Проєктування системи передбачає строгий шар знеособлення (Anonymization Layer).

Таблиця 1.2

Приклад роботи алгоритму знеособлення даних перед відправкою до LLM API

Параметр	Вихідні дані (в базі)	Знеособлені дані (для ІІІ)
Ім'я клієнта	Олександр Іваненко	Клієнт_A
Контакт	3,80501E+11	[ВИДАЛЕНО]
Email	oleksandr.ivanenko@example.com	[ВИДАЛЕНО]
Адреса	м. Київ, вул. Хрещатик, 10, кв. 25	[ВИДАЛЕНО]
Сума угоди	150 000 грн	150 000 грн

Продовження таблиці 1.2

Приклад роботи алгоритму знеособлення даних перед відправкою до LLM API

Назва компанії	ТОВ «Інноваційні рішення»	Компанія_1
Дата створення угоди	12.05.2025 14:35	12.05.2025 14:35
Опис угоди	Постачання CRM системи для автоматизації відділу продажів	Постачання CRM системи для автоматизації відділу продажів
Менеджер	Марія Петренко	Менеджер_Б
Статус угоди	В роботі	В роботі

Таким чином, Llama-3 аналізує виключно математичні тенденції та структуру воронки продажів, не маючи жодного доступу до чутливої інформації, що робить систему юридично безпечною для впровадження.

1.6.2. Подієво-орієнтована архітектура та Slack-сповіщення

Традиційні канали сповіщень, такі як електронна пошта (Email), потерпають від проблеми перевантаженості (Alert Fatigue) та низького рівня відкритості. Використання особистих месенджерів (Viber, Telegram) розмиває межі між особистим і робочим простором, що суперечить корпоративним стандартам. Тому у фінальній реалізації платформи «SmartBiz CRM» було обрано інтеграцію з провідним корпоративним месенджером Slack.

Архітектура механізму Incoming Webhooks: Інтеграція реалізована за допомогою механізму вхідних вебхуків (Incoming Webhooks). Це дозволяє зовнішньому застосунку (нашій CRM) відправляти інформацію в канали Slack у режимі реального часу за допомогою простих HTTP POST запитів.

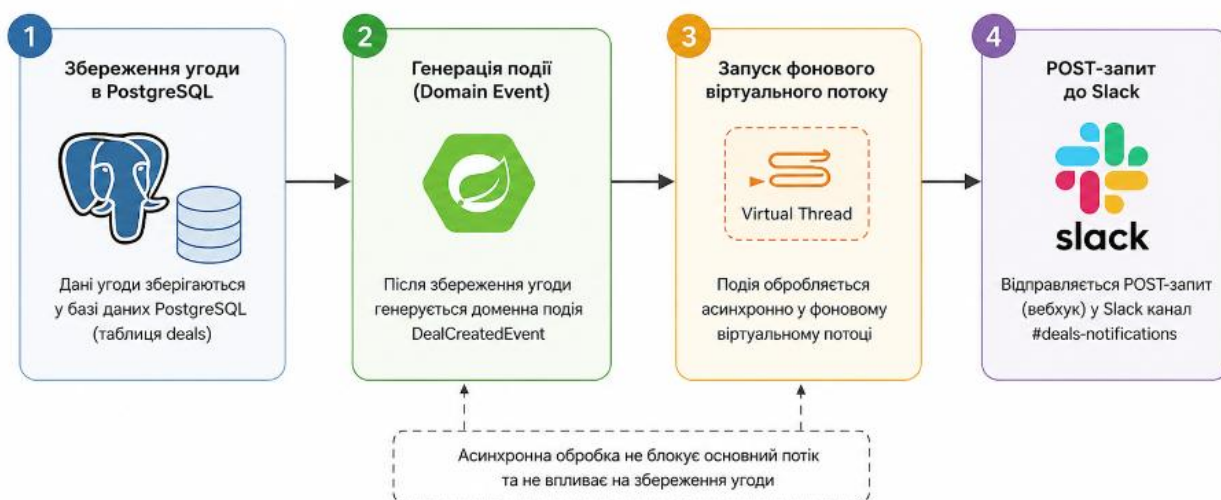


Рис. 1.8 Архітектура потоку даних при генерації події та відправці вебхука

Структурування повідомлень (Slack Block Kit): Сучасні сповіщення не можуть бути просто "сухим" текстом. Система генерує JSON-навантаження (Payload) з використанням фреймворку Block Kit. Це дозволяє створювати багаті візуальні інтерфейси безпосередньо у вікні месенджера. Повідомлення містить:

Блок заголовка (Header) з колірним індикатором (наприклад, зелений для успішної угоди, жовтий для нового ліда).

Блок контексту (Context Block) із зазначенням імені менеджера та суми.

Кнопку швидкої дії (Action Button), яка дозволяє в один клік перейти зі Slack прямо в картку клієнта у CRM-системі.

Надійність та асинхронність обробки: Враховуючи, що API Slack є зовнішнім мережевим ресурсом, воно може мати затримки (Latency) або тимчасово не відповідати. Щоб гарантувати, що процес збереження даних менеджером не буде заблокований в очікуванні відповіді від Slack, інтеграція побудована на принципах асинхронності.

Використання віртуальних потоків (Project Loom) на платформі Java 21 дозволяє бекенду виділяти окремий легковагий потік для кожного сповіщення. Якщо Slack повертає помилку мережі (наприклад, HTTP 429 Too Many Requests), система має можливість реалізувати патерн повторних спроб (Retry Pattern) у фоновому режимі, абсолютно непомітно для кінцевого користувача.

2. ПРОЄКТУВАННЯ МУЛЬТИТЕНАНТНОЇ АРХІТЕКТУРИ ТА БАЗИ ДАНИХ СИСТЕМИ

2.1. Обґрунтування вибору архітектурної моделі SaaS та стратегії мультитенантності

Процес проєктування будь-якої інформаційної системи, що орієнтована на комерційне використання багатьма незалежними організаціями, починається з вибору моделі постачання програмного забезпечення. Для веб-платформи «SmartBiz CRM» за основу було обрано парадигму SaaS (Software as a Service — програмне забезпечення як сервіс).

Порівняльний аналіз SaaS та On-premise моделей. Традиційна модель On-premise передбачає інсталяцію та підтримку програмного продукту на власних серверних потужностях замовника. Для малого бізнесу такий підхід є критично обтяжливим, оскільки вимагає:

1. Значних капітальних інвестицій у закупівлю серверного обладнання та ліцензій на системне ПЗ.
2. Наявності штатного системного адміністратора для забезпечення працездатності та безпеки системи.
3. Складного процесу оновлення, що часто потребує зупинки бізнес-процесів.

Натомість модель SaaS, обрана для даного проєкту, дозволяє перенести всі інфраструктурні складнощі на сторону провайдера (розробника). Користувач отримує доступ до функціоналу через браузер, сплачуючи лише за фактичне використання.

Таблиця 2.1

Порівняння моделей постачання ПЗ для сегмента малого бізнесу

Критерій порівняння	Модель On-premise	Модель SaaS (SmartBiz CRM)
Капітальні витрати	Високі	Відсутні
Витрати на обслуговування	Значні	Мінімальні
Оновлення системи	Ручне / Складне	Автоматичне / Безшовне
Доступ до системи	Лише в локальній мережі	Через браузер з будь-якого місця
Масштабованість	Обмежена ресурсами сервера	Гнучка та швидка
Швидкість впровадження	Від кількох тижнів	Протягом декількох годин
Необхідність власного сервера	Обов'язкова	Не потрібна
Резервне копіювання	Виконується вручну	Автоматизоване
Безпека даних	Залежить від компанії	Забезпечується провайдером
Технічна підтримка	Внутрішній IT-відділ	Централізована підтримка
Інтеграція з зовнішніми сервісами	Складна	Через API та webhooks
Оплата	Одноразова ліцензія	Підписка (щомісячна/річна)
Оновлення функціоналу	Потребує перевстановлення	Доступне автоматично
Мобільність користувачів	Обмежена	Висока
Оптимальність для малого бізнесу	Часто економічно не вигідна	Оптимальна модель

Стратегії реалізації мультитенантності. Оскільки SaaS-система передбачає обслуговування багатьох незалежних клієнтів (тенантів) у межах одного розгорнутого екземпляра ПЗ, ключовим архітектурним питанням є вибір стратегії мультитенантності. У сучасній інженерії виділяють три основні підходи до ізоляції даних:

1. Database-per-tenant (Окрема база даних для кожного клієнта):

Особливості: Кожна компанія має власну фізичну базу даних, індивідуальне резервне копіювання.

Недоліки: Висока вартість утримання інфраструктури; складність оновлення схеми даних для тисяч клієнтів одночасно.

2. Schema-per-tenant (Окрема схема для кожного клієнта):

Особливості: Клієнти ділять одну СУБД, але мають власні логічні схеми (namespaces).

Переваги: Компроміс між ізоляцією та складністю управління.

Недоліки: Обмеження масштабованості на рівні конкретної СУБД.

3. Shared Database, Shared Schema (Спільна база даних та спільна схема):

Особливості: Дані всіх клієнтів зберігаються в одних і тих самих таблицях. Розподіл здійснюється за допомогою ідентифікатора компанії (Tenant ID / Company ID).

Переваги: Максимальна економічна ефективність; найпростіший процес оновлення та масштабування; мінімальне споживання ресурсів сервера.

Недоліки: Потреба у надзвичайно ретельній реалізації фільтрації даних на рівні програмного коду для запобігання витоку інформації.

Обґрунтування вибору для «SmartBiz CRM». Для реалізації проєкту було обрано третю стратегію — Shared Database, Shared Schema. Даний вибір детермінований специфікою цільової аудиторії системи (малий та мікробізнес). Малі підприємства зазвичай мають невеликі обсяги даних, тому виділення окремої бази даних для кожного ФОП було б технічно надлишковим та фінансово нерентабельним.

Для нівелювання ризиків безпеки, притаманних цій моделі, у проєкті спроектовано багаторівневу систему фільтрації:

На рівні автентифікації: `company_id` вшивається у криптографічно підписаний JWT-токен, який неможливо підробити на стороні клієнта.

На рівні бізнес-логіки: Впроваджено механізм автоматичної фільтрації запитів за допомогою Hibernate Filters та аспектно-орієнтованого програмування (AOP). Кожен запит до PostgreSQL автоматично доповнюється умовою `WHERE company_id = ?`.

Таким чином, обрана архітектура дозволяє забезпечити Enterprise-рівень ізоляції даних при збереженні низької вартості хостингу, що є ключовою конкурентною перевагою для українського ринку малого бізнесу.

2.2. Функціональне моделювання системи засобами уніфікованої мови моделювання UML

Проектування архітектури та функціональних можливостей веб-платформи «SmartBiz CRM» вимагає використання стандартизованих засобів візуального моделювання. Вибір мови UML (Unified Modeling Language) обумовлений її універсальністю та здатністю описувати систему з різних точок зору: від зовнішньої поведінки до внутрішньої структури об'єктів та їхньої взаємодії в часі.

У межах даного проєкту було побудовано комплекс моделей, що охоплюють функціональні вимоги, динаміку процесів та статичну структуру програмних компонентів.

2.2.1. Моделювання функціональних вимог: діаграма прецедентів

Першим етапом проектування є визначення меж системи та ролей користувачів. Діаграма прецедентів (Use Case Diagram) дозволяє зафіксувати, які саме функції платформа надає різним категоріям користувачів.

У системі «SmartBiz CRM» виділено два основні актори:

1. Менеджер з продажу: основний операційний користувач, який працює з клієнтською базою та угодами.

2. Адміністратор компанії: користувач із розширеними правами, який відповідає за налаштування мультитенантного середовища та аналіз стратегічних показників.

Таблиця 2.2

Специфікація головних прецедентів системи

Назва прецеденту	Актор	Опис дії	Передмова
Управління угодою	Менеджер	Створення, редагування та переміщення картки по Канбан-дошці	Користувач автентифікований
Реєстрація користувача	Адміністратор	Створення нового облікового запису користувача	Адміністратор має права доступу
Авторизація в системі	Користувач	Вхід до CRM за email та паролем	Користувач зареєстрований
Перегляд Канбан-дошки	Менеджер	Перегляд активних угод за статусами	Користувач увійшов у систему
Генерація AI-рекомендацій	Менеджер	Отримання рекомендацій від LLM-моделі щодо угоди	Угода існує в системі
Надсилання Slack-сповіщення	Система	Автоматичне формування та надсилання webhook-повідомлення	Подія створення угоди відбулася
Редагування профілю	Користувач	Оновлення персональних даних та налаштувань	Користувач автентифікований
Пошук угоди	Менеджер	Пошук угод за назвою, статусом або клієнтом	У базі даних є записи
Видалення угоди	Менеджер	Видалення непотрібної або помилкової угоди	Користувач має відповідні права
Перегляд аналітики	Керівник	Перегляд статистики продажів та активності менеджерів	Дані накопичені у системі

Продовження таблиці 2.2

Специфікація головних прецедентів системи

Експорт даних	Адміністратор	Вивантаження інформації у CSV/Excel формат	Користувач має права адміністратора
Інтеграція з API	Система	Обмін даними із зовнішніми сервісами через REST API	API-ключ налаштований

Особливістю функціональної моделі є прецеденти, пов'язані зі смарт-функціями. Наприклад, прецедент «Перегляд ШІ-аналітики» доступний лише адміністратору та передбачає включення (*include*) прецеденту «Агрегація даних тенанта», що підкреслює мультитенантну ізоляцію системи.

2.2.2. Моделювання динаміки автентифікації: діаграма послідовностей

Для детального опису процесу взаємодії між клієнтською частиною (React), сервером (Spring Boot) та базою даних (PostgreSQL) було побудовано діаграму послідовностей (Sequence Diagram). Вона критично важлива для розуміння того, як працює stateless-автентифікація на базі JWT.

Процес, відображений на діаграмі, включає наступні етапи:

1. Клієнт відправляє POST-запит із логіном та паролем.
2. SecurityFilter перехоплює запит і передає його в AuthenticationManager.
3. Сервіс перевіряє хеш пароля в базі даних.
4. При успішній валідації JwtProvider генерує токен, у який «зашивається» company_id.
5. Токен повертається клієнту і зберігається в localStorage браузера для подальшої автентифікації кожного запиту.

2.2.3. Моделювання бізнес-процесу продажу: діаграма активностей

Для опису логіки переходу угоди між етапами та ініціації асинхронних сповіщень використано діаграму активностей (Activity Diagram). Вона демонструє розгалуження процесів та паралельне виконання завдань.

На діаграмі відображено ключовий розрив між синхронною та асинхронною логікою:

Коли менеджер перетягує картку на Канбан-дошці, система виконує синхронне оновлення статусу в БД.

Одночасно з цим ініціюється розгалуження (*Fork*), де у фоновому віртуальному потоці (Project Loom) запускається процес формування JSON-пакета для Slack та його відправка через Webhook. Це гарантує, що затримки мережі Slack не впливають на швидкість відгуку інтерфейсу для менеджера.

2.2.4. Моделювання архітектури компонентів: діаграма класів

Діаграма класів (Class Diagram) описує статичну структуру бекенду. Вона відображає застосування паттерну Layered Architecture (Багаторівнева архітектура), що є стандартом для Spring Boot застосунків.

Структурні рівні, відображені на моделі:

1. Controller Layer: класи, що відповідають за обробку REST-запитів та валідацію вхідних DTO.

2. Service Layer: містить основну бізнес-логіку. Саме тут реалізовано логіку перерахунку сум угод та взаємодію з ШІ-модулем.

3. Repository Layer (Spring Data JPA): інтерфейси для автоматизованої генерації SQL-запитів.

4. Entity Layer: об'єктно-реляційні моделі, що відображають структуру таблиць бази даних.

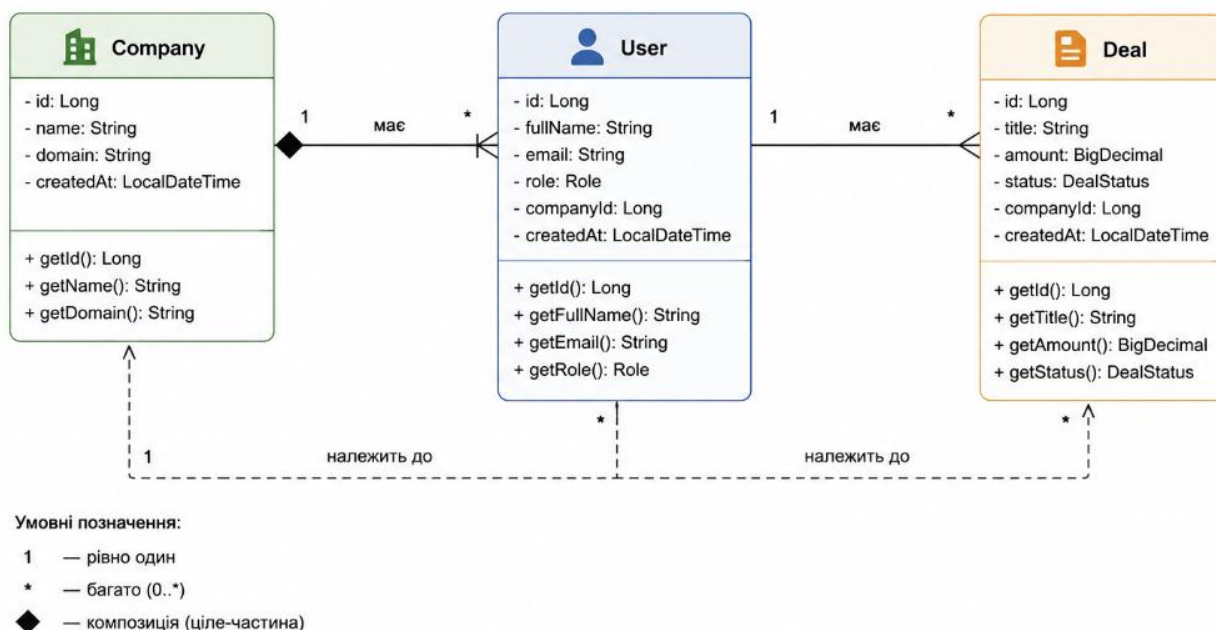


Рис. 2.1 Зв'язки між сутностями «Компанія», «Користувач» та «Угода» на діаграмі класів

Застосування такої моделі дозволяє досягти принципу Separation of Concerns (Розподіл відповідальностей), що робить систему легкою для тестування та подальшого розширення, наприклад, при додаванні нових модулів інтеграції з іншими месенджерами чи платіжними системами.

2.3. Концептуальне та логічне моделювання бази даних

База даних є фундаментом будь-якої CRM-системи, оскільки вона виступає єдиним джерелом істини (Single Source of Truth) для всіх бізнес-процесів організації. При проектуванні сховища даних для веб-платформи «SmartBiz CRM» першочергова увага приділялася забезпеченню високої продуктивності запитів, надійності збереження інформації та суворій логічній ізоляції даних різних компаній-орендарів.

2.3.1. Побудова інфологічної моделі (ER-діаграма)

Процес проєктування розпочався з побудови концептуальної моделі «Сутність–Зв’язок» (Entity-Relationship Diagram), яка дозволяє визначити ключові бізнес-об’єкти та характер їхніх взаємозв’язків без прив’язки до особливостей конкретної СУБД.

Основними сутностями системи «SmartBiz CRM» є:

1. Компанія (Company): коренева сутність тенанта.
2. Користувач (User): співробітник компанії (адміністратор або менеджер).
3. Клієнт (Client): фізична або юридична особа, з якою ведеться комунікація.
4. Угода (Deal): основна одиниця продажу, що має суму та статус.
5. Взаємодія (Interaction): лог подій (дзвінки, зустрічі, коментарі).

Зв’язки між сутностями спроектовано за наступними правилами:

Одна Компанія може мати багато Користувачів та Угод (відношення 1:N).

Один Користувач (менеджер) може вести багато Клієнтів та Угод (1:N).

Одна Угода обов’язково належить одному Клієнту (N:1).

Кожна сутність (крім самої Компанії) містить зовнішній ключ `company_id`, що є основою нашої мультитенантної стратегії.

2.3.2. Нормалізація бази даних та забезпечення цілісності

З метою усунення надлишковості даних та запобігання аномаліям оновлення (Update Anomaly), структуру бази даних було приведено до третьої нормальної форми (3NF).

Етапи нормалізації:

Перша нормальна форма (1NF): забезпечено атомарність усіх атрибутів (наприклад, ім’я та прізвище клієнта розділені на окремі поля).

Друга нормальна форма (2NF): усі неключові атрибути функціонально залежать від повного первинного ключа. Створено окремі таблиці для статусів та ролей.

Третя нормальна форма (3NF): усунуто транзитивні залежності. Дані про компанію (наприклад, Slack Webhook URL) винесені в окрему таблицю companies, а не дублюються в кожному записі користувача.

Для забезпечення цілісності даних на рівні СУБД PostgreSQL реалізовано механізм декларативної цілісності:

1. Primary Keys (PK): використання типу UUID для всіх ідентифікаторів. Це робить систему більш стійкою до перебору ID та спрощує майбутню синхронізацію між шардами бази.

2. Foreign Keys (FK): налаштовано обмеження ON DELETE CASCADE. Це означає, що при видаленні акаунту компанії всі пов'язані з нею користувачі, угоди та клієнти будуть видалені автоматично, що запобігає появі «сміттєвих» даних.

2.3.3. Специфікація фізичної моделі даних

На етапі даталогічного проєктування було визначено типи полів, індекси та обмеження для кожної таблиці. Нижче наведено детальні специфікації основних таблиць бази даних.

Таблиця 2.3

Специфікація таблиці companies (Реєстр тенантів)

Поле (Column)	Тип даних	Обмеження (Constraints)	Опис
id	UUID	Primary Key	Унікальний ідентифікатор компанії
name	VARCHAR(255)	NOT NULL	Офіційна назва організації
slack_webhook_url	TEXT	UNIQUE	URL для інтеграції зі Slack
created_at	TIMESTAMP	DEFAULT NOW()	Дата реєстрації в системі

Таблиця 2.4

Специфікація таблиці deals (Управління продажами)

Поле (Column)	Тип даних	Обмеження (Constraints)	Опис
id	UUID	Primary Key	Ідентифікатор угоди
title	VARCHAR(255)	NOT NULL	Назва угоди (напр. «Продаж ПЗ»)
amount	DECIMAL(15,2)	CHECK (amount >= 0)	Бюджет угоди
status	VARCHAR(50)	NOT NULL	Етап воронки (NEW, PROPOSAL, WON)
company_id	UUID	Foreign Key → companies	Маркер належності до тенанта
client_id	UUID	Foreign Key → clients	Зв'язок із контрагентом

2.3.4. Стратегія мультитенантності на рівні схеми

Особливістю проекту «SmartBiz CRM» є використання підходу Shared Schema. Це означає, що всі дані зберігаються в одному наборі таблиць. Для забезпечення безпеки на етапі проєктування БД передбачено створення індексів за полем company_id для кожної таблиці.

Переваги такого підходу:

Прискорення фільтрації: за наявності індексу PostgreSQL миттєво знаходить дані конкретної компанії, ігноруючи мільйони записів інших тенантів.

Масштабованість: ми можемо легко додавати нові компанії без необхідності змінювати структуру БД чи виконувати DDL-запити (CREATE TABLE).

Використання типу даних JSONB у PostgreSQL дозволяє зберігати результати аналітики від моделі Llama-3 у гнучкому форматі, не створюючи при цьому десятки статичних колонок. Це забезпечує системі необхідну адаптивність до нових типів звітів, які можуть з'явитися в майбутньому.

2.4. Проєктування підсистеми безпеки та архітектури JWT-автентифікації

Забезпечення конфіденційності, цілісності та доступності даних є пріоритетним завданням при розробці мультитенантних SaaS-платформ. У системі «SmartBiz CRM», де дані багатьох організацій зберігаються в єдиній інфраструктурі, архітектура безпеки повинна гарантувати неможливість перехресного доступу до інформації. Для вирішення цього завдання було спроектовано підсистему безпеки, що базується на протоколі Stateless-автентифікації із використанням JSON Web Tokens (JWT).

2.4.1. Обґрунтування вибору Stateless-підходу та JWT

Традиційні механізми автентифікації на основі сесій (Session-based) передбачають збереження стану користувача в оперативній пам'яті сервера або в спеціалізованому сховищі (наприклад, Redis). Для масштабованої хмарної системи такий підхід має ряд недоліків:

1. Проблеми масштабування: при горизонтальному розширенні (додаванні нових екземплярів сервера) виникає потреба в синхронізації сесій між ними.

2. Навантаження на ресурси: збереження тисяч активних сесій у пам'яті споживає значні ресурси RAM.

3. Вразливість до CSRF: сесійні куки автоматично відправляються браузером, що створює вектор для атак типу Cross-Site Request Forgery.

Вибір JWT дозволяє реалізувати Stateless-архітектуру. Сервер не зберігає стан користувача; вся необхідна інформація для авторизації запиту міститься в самому токени, який передається в заголовку Authorization: Bearer <token>.

2.4.2. Структурне проєктування JSON Web Token

У межах проєкту було розроблено специфічну структуру токена, яка оптимізована для мультитенантного середовища. JWT складається з трьох частин, розділених крапками:

1. Header (Заголовок): містить метадані про тип токена та алгоритм хешування. У нашому випадку обрано алгоритм HS256 (HMAC з використанням SHA-256), що забезпечує високу швидкість перевірки підпису на стороні сервера.

2. Payload (Корисне навантаження): містить набір тверджень (*Claims*). Для «SmartBiz CRM» було спроектовано використання наступних клеймів:

sub (Subject): ідентифікатор користувача (UUID).

company_id: критично важливе поле, що вказує на приналежність користувача до конкретного тенанта.

roles: список прав доступу (Admin, Manager).

iat та exp: час видачі та час закінчення дії токена (TTL — Time To Live).

3. Signature (Підпис): результат хешування заголовка та навантаження із використанням секретного ключа (*Secret Key*), що зберігається виключно на сервері.

Таблиця 2.5

Специфікація корисного навантаження (Payload) токена SmartBiz CRM.

Клейм (Claim)	Значення	Призначення
userId	UUID	Ідентифікація суб'єкта в системі
companyId	UUID	Фільтрація даних у межах мультитенантності
role	String	Авторизація доступу до методів API (RBAC)
exp	Timestamp	Захист від використання старих токенів (12 годин)

2.4.3. Механізм автентифікації та життєвий цикл токена

Процес автентифікації спроектовано як закритий цикл взаємодії між клієнтом та сервером.

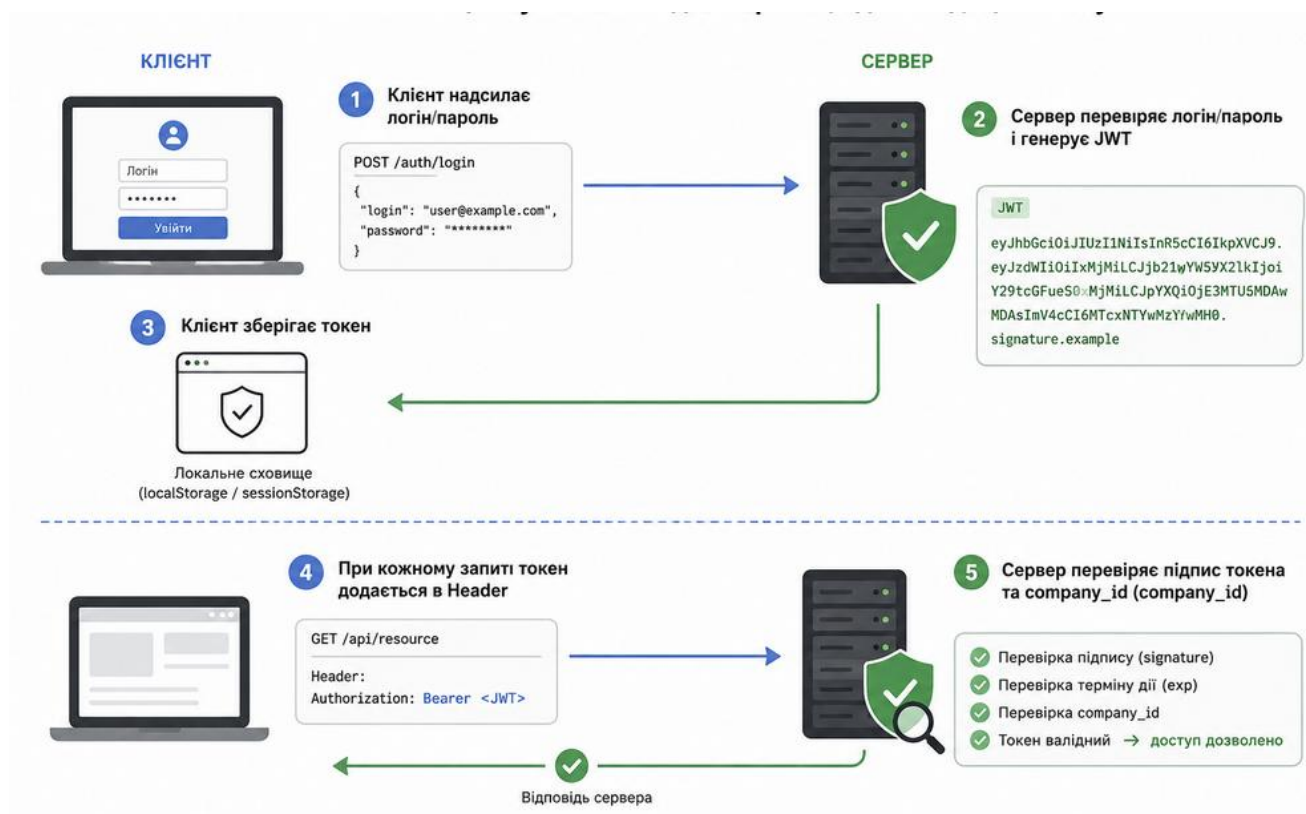


Рис. 2.2 Схема життєвого циклу токена: від авторизації до валідації запиту

Для підвищення рівня безпеки термін дії токена обмежений 12 годинами. Це нівелює ризики у разі компрометації токена, оскільки зловмисник матиме обмежений час на його використання.

2.4.4. Рольова модель доступу (RBAC) та безпека API

На етапі проєктування підсистеми безпеки було закладено принцип мінімальних привілеїв. Це реалізується через механізм Role-Based Access Control (RBAC).

Кожен REST-ендпоінт бекенду захищений анотаціями безпеки (наприклад, `@PreAuthorize`), які перевіряють ролі, вилучені з JWT-токена.

Менеджер має доступ лише до операційних модулів (Канбан, Клієнти).

Адміністратор додатково має доступ до критичних налаштувань (інтеграція зі Slack, генерація III-звітів).

Особлива увага приділена захисту від ID Enumeration (перебору ідентифікаторів). Навіть якщо менеджер компанії «А» спробує звернутися до угоди

компанії «Б», знаючи її UUID, серверний фільтр зіставить company_id з токена із company_id запитуваного об'єкта в базі даних. При їх неспівпадінні буде повернуто HTTP-статус 403 Forbidden.

Така архітектура дозволяє створити надійний периметр безпеки, де кожна компанія-орендар відчуває себе в повній ізоляції, попри використання спільних обчислювальних ресурсів.

2.5. Проєктування інтеграцій із зовнішніми сервісами та механізмів асинхронної взаємодії

У сучасній архітектурі корпоративного програмного забезпечення CRM-система не може функціонувати як ізольований програмний моноліт. Для забезпечення високої конкурентоспроможності платформи «SmartBiz CRM» на етапі проєктування було закладено гнучкий інтеграційний шар, що дозволяє системі взаємодіяти з потужними хмарними екосистемами. Основні вектори зовнішньої взаємодії — це корпоративний месенджер Slack (для оперативного управління подіями) та велика мовна модель Llama-3 (для глибокої бізнес-аналітики).

2.5.1. Подієво-орієнтована архітектура модуля сповіщень (Slack Webhooks)

На відміну від традиційних Email-сповіщень, які мають низький рівень оперативності, інтеграція з месенджером Slack дозволяє команді малого бізнесу реагувати на зміни у воронці продажів у режимі реального часу. Під час проєктування було проведено порівняльний аналіз між використанням повноцінного Slack Bot API (через WebSocket) та Incoming Webhooks (вхідні вебхуки). Вибір було зроблено на користь останнього через спрощену модель автентифікації та відсутність потреби у підтримці постійного з'єднання, що критично для економії ресурсів сервера у SaaS-моделі.

Логіка роботи модуля базується на архітектурному патерні «Спостерігач» (Observer) та принципах подієво-орієнтованої архітектури (Event-Driven Architecture).

Процес ініціації сповіщення:

1. Генерація події (Event Publication): При зміні стану сутності «Угода» (Deal) у базі даних (наприклад, перехід у статус «Успішно закрито»), транзакційний сервіс публікує внутрішню подію DealStatusChangedEvent у шину подій Spring (ApplicationEventPublisher).

2. Перехоплення (Event Listener): Спеціалізований слухач (SlackNotificationListener) перехоплює цю подію. Важливо, що слухач відокремлений від основної бізнес-логіки (принцип слабкої зв'язності — Loose Coupling).

3. Трансформація (Payload Mapping): Дані угоди перетворюються у складну JSON-структуру відповідно до специфікації Slack Block Kit. Цей фреймворк дозволяє формувати візуально багаті повідомлення з колірними маркерами, розділювачами та інтерактивними кнопками переходу в CRM.

Таблиця 2.6

Структура JSON-навантаження (Payload) для Slack API

Вузол JSON	Тип	Опис
blocks	Array	Масив візуальних блоків повідомлення Slack
type: "section"	Object	Текстовий блок з інформацією про угоду або повідомлення
text	Object	Об'єкт, що містить текстовий контент блоку
type: "mrkdwn"	String	Форматування тексту за допомогою Markdown

Структура JSON-навантаження (Payload) для Slack API

text.text	String	Основний текст повідомлення
accessory	Object	Додатковий елемент: кнопка, зображення або меню
type: "button"	String	Визначає елемент як кнопку
text.type	String	Тип тексту кнопки
text.text	String	Напис на кнопці
action_id	String	Унікальний ідентифікатор дії кнопки
value	String	Значення, яке передається після натискання кнопки
url	String	Посилання, що відкривається після натискання
image_url	String	URL-адреса зображення
alt_text	String	Альтернативний текст для зображення
divider	Object	Розділювач між блоками повідомлення
context	Object	Блок з додатковою службовою інформацією
elements	Array	Масив елементів контекстного блоку
type: "image"	Object	Відображення зображення у повідомленні
fields	Array	Набір коротких текстових полів у секції

2.5.2. Проєктування взаємодії з хмарним API штучного інтелекту (Llama-3)

Інтеграція з генеративним штучним інтелектом вимагає передачі значних обсягів текстових даних. Звернення до API Llama-3 проєктується за моделлю REST (Representational State Transfer) поверх протоколу HTTPS.

Головною архітектурною проблемою при роботі з LLM є недетермінований час відповіді (Latency). На відміну від звичайних запитів до БД, які виконуються за мілісекунди, генерація розгорнутого аналітичного звіту моделлю Llama-3 може займати від 3 до 10 секунд залежно від навантаження на сервери провайдера (Groq / Meta).

Для вирішення цієї проблеми спроектовано механізм опитування (Polling) або використання анімації очікування на клієнті:

Фронтенд (React) відправляє запит на генерацію звіту і відображає стан «Завантаження» (Skeleton loader).

Бекенд встановлює з'єднання з API ШІ. Для запобігання нескінченному зависанню потоків жорстко налаштовано параметри Connection Timeout (час на встановлення з'єднання) та Read Timeout (максимальний час очікування відповіді).

2.5.3. Моделювання асинхронності на базі Virtual Threads (Java 21 Project Loom)

Оскільки обидві ключові інтеграції (Slack та Llama-3) є I/O-bound операціями (залежними від пропускну здатності мережі та швидкості зовнішніх серверів), їх синхронне виконання є категорично неприпустимим. Якщо під час збереження угоди сервер буде чекати відповіді від Slack, інтерфейс користувача "зависне", а пул потоків веб-сервера Tomcat швидко вичерпається.

Як було теоретично обґрунтовано в п. 1.3.2, для вирішення цієї архітектурної проблеми у проєкті застосовано інноваційну технологію віртуальних потоків (Project Loom) платформи Java 21. На відміну від традиційної моделі, віртуальні потоки не блокують системні ресурси (Carrier Threads) під час очікування мережевої відповіді від зовнішніх API (рис. 2.6).

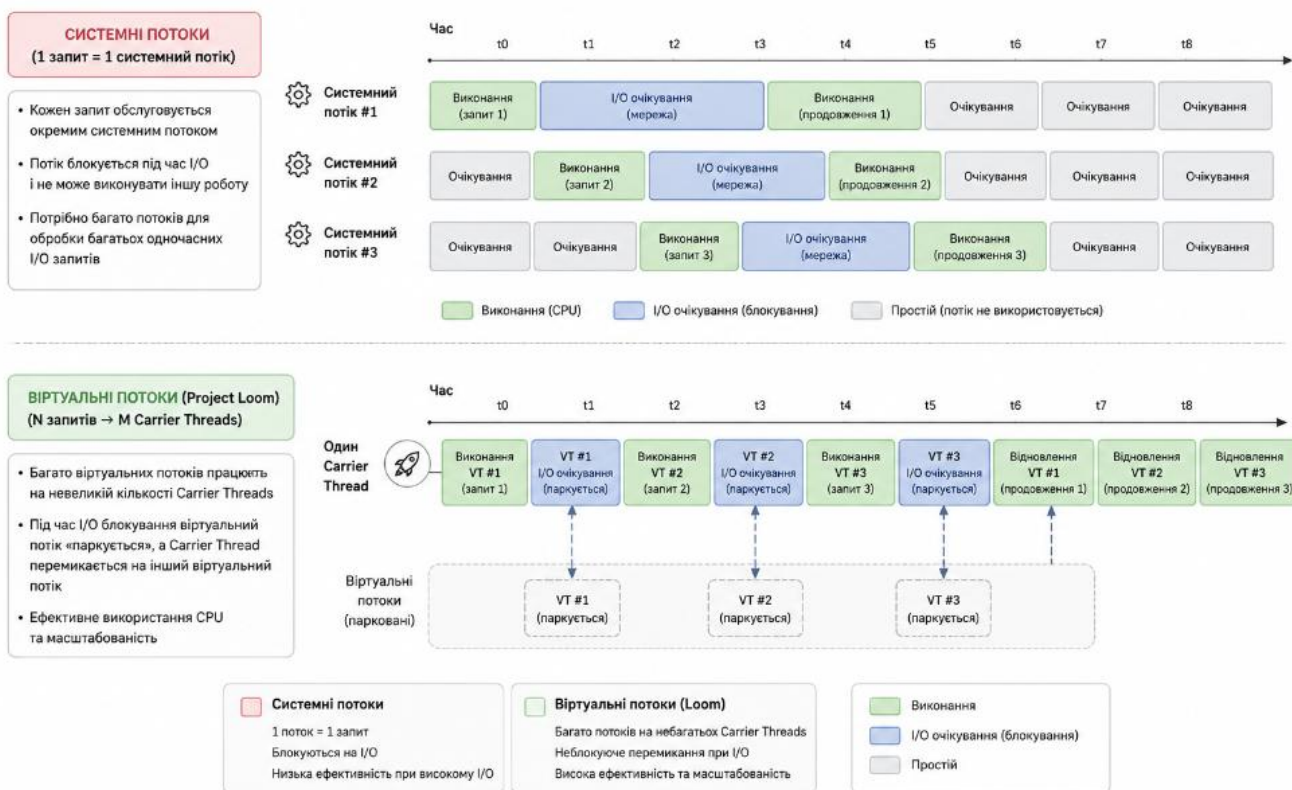


Рис. 2.3 Порівняльна схема роботи системних та віртуальних потоків при I/O блокуванні

Завдяки цій архітектурі, при використанні анотації `@Async` у Spring Boot, сервер створює новий легковагий віртуальний потік для кожного сповіщення, споживаючи всього кілька кілобайтів пам'яті. Це гарантує високу пропускну здатність (High Throughput) платформи навіть при масовому навантаженні.

2.5.4. Забезпечення відмовостійкості (Fault Tolerance) інтеграцій

При роботі із зовнішніми вебсервісами необхідно враховувати теорему Брюера (CAP-теорему) та той факт, що мережа ніколи не буває на 100% надійною. Якщо API Slack тимчасово недоступне (повертає помилки 502/503), CRM-система не повинна втрачати функціональність або викидати критичні винятки користувачу.

Для забезпечення відмовостійкості інтеграційного шару було спроектовано застосування патерну Повторних спроб (Retry Pattern):

1. Якщо REST-клієнт отримує помилку таймауту або статус 5xx від зовнішнього API, ініціюється механізм повторного запиту.

2. Використовується стратегія Exponential Backoff (експоненційне збільшення затримки): перша повторна спроба через 1 секунду, друга — через 3 секунди, третя — через 10 секунд. Це дозволяє уникнути DDOS-атаки на сервери, що відновлюються.

3. Якщо всі спроби вичерпано, подія записується у спеціальну таблицю логів БД (failed_notifications) (патерн *Dead Letter Queue*) для ручного або автоматичного розбору адміністратором.

Такий глибокий інженерний підхід до інтеграцій гарантує Enterprise-рівень надійності системи, забезпечуючи безперебійну роботу комерційних процесів малого бізнесу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ПЛАТФОРМИ SMARTBIZ CRM

3.1. Вибір та налаштування інструментального середовища розробки

Ефективність розробки складної мультитенантної системи безпосередньо залежить від правильно обраного інструментарію. Для реалізації «SmartBiz CRM» було сформовано комплексне інтегроване середовище розробки (IDE) та допоміжних засобів, що відповідають сучасним стандартам індустрії (DevOps та Agile).

Середовища розробки (IDE): Основним інструментом для написання серверної частини (Back-end) обрано IntelliJ IDEA Ultimate Edition від компанії JetBrains. Дане середовище надає глибоку підтримку фреймворку Spring Boot, забезпечуючи інтерактивне налагодження коду (Debugging), управління залежностями через Maven та візуалізацію профілювання пам'яті. Для фронтенд-розробки використовувався редактор Visual Studio Code з розширеннями для React (ESLint, Prettier) та Tailwind CSS (IntelliSense), що забезпечило високу швидкість написання клієнтського коду.

Контейнеризація та бази даних: Для забезпечення ідентичності середовищ розробки (Development) та майбутнього продакшену (Production) використовувалася платформа Docker. Замість локального встановлення PostgreSQL, база даних запускала у вигляді ізольованого Docker-контейнера. Це мінімізує ризики архітектурних розбіжностей (синдром "на моєму комп'ютері це працює").

Система контролю версій: Керування версіями вихідного коду здійснювалося за допомогою розподіленої системи Git із використанням хмарної платформи GitHub. Робота велася за методологією Feature Branch Workflow, де кожна нова функція (наприклад, інтеграція Slack або III) розроблялася в окремій гілці.

Інструменти тестування API: Для розробки та інтерактивного тестування RESTful API використовувався інструмент Postman. Він дозволив створити

колекції запитів для швидкої перевірки JWT-автентифікації та коректності відповідей бекенду без необхідності взаємодії з клієнтським інтерфейсом.

3.2. Розробка серверної частини на базі Spring Boot 3 та Java 21

Серверна частина системи спроектована за принципами «Чистої архітектури» (Clean Architecture) Роберта Мартіна та відповідає принципам SOLID. Це забезпечує низьку зв'язність (Loose Coupling) між модулями та високу здатність до тестування.

3.2.1. Структура пакетів серверного застосунку

Проект організовано за доменним принципом (Domain-Driven Design), де кожен пакет відповідає за певну функціональну зону:

`com.smartbiz.crm.config` — конфігураційні класи (Spring Security, CORS, Swagger);

`com.smartbiz.crm.entity` — об'єктно-реляційні моделі (JPA Entities), що відображають таблиці БД;

`com.smartbiz.crm.repository` — інтерфейси доступу до даних (Spring Data JPA);

`com.smartbiz.crm.service` — шар бізнес-логіки, де реалізовано алгоритми обробки даних;

`com.smartbiz.crm.controller` — REST-контролери для перехоплення HTTP-запитів;

`com.smartbiz.crm.security` — фільтри та провайдери JWT-автентифікації;

`com.smartbiz.crm.dto` — об'єкти передачі даних (Data Transfer Objects) для ізоляції внутрішніх сутностей від зовнішнього API.

3.2.2. Програмна реалізація об'єктно-реляційної моделі (JPA Entities)

Основою роботи з даними у серверній частині є Entity-класи. Для проєкту «SmartBiz CRM» було обрано підхід, де кожна сутність не лише відображає таблицю бази даних, а й містить механізми автоматичного аудиту та забезпечення мультитенантності.

Для мінімізації шаблонного коду (Boilerplate code) — гетерів, сетерів, конструкторів та патерну «Builder» — використано бібліотеку **Lombok**. Це дозволяє тримати класи лаконічними та зосередитися на структурі даних. Особливістю реалізації є використання типу UUID для первинних ключів, що підвищує безпеку та ускладнює несанкціонований перебір ідентифікаторів.

Програмну реалізацію центральної сутності системи — «Угода» (Deal), яка інтегрує в собі зв'язки з клієнтами та механізм логічного розділення даних між компаніями-орендарями, приклад коду наведено у додатку Б.

Аналіз технічних особливостей реалізації сутності Deal підтверджує орієнтацію на принципи безпеки даних, оптимізації ресурсів та високої точності обчислень. Насамперед, інтеграція поля company через реляційний зв'язок @ManyToOne виступає фундаментальним елементом реалізації мультитенантної архітектури (Multi-tenancy), забезпечуючи жорстку ізоляцію комерційних даних та унеможливлюючи несанкціонований міжтендантний доступ на рівні окремих підприємств. Автоматизація системного аудиту реалізована шляхом імплементації анотацій @CreationTimestamp та @UpdateTimestamp, що дозволяє декларативно фіксувати часові мітки створення та модифікації записів у фоновому режимі, повністю ізолюючи цю логіку від основного контуру бізнес-сервісів. З метою оптимізації взаємодії із базою даних PostgreSQL та запобігання виникненню класичної проблеми надлишкових запитів «N+1», для об'єктних зв'язків із сутностями Client та Company прецизійно встановлено стратегію лінивого завантаження (FetchType.LAZY). Це дозволяє динамічно підтягувати асоційовані граф-залежності лише в момент безпосереднього звернення до відповідних геттерів, суттєво знижуючи витрати оперативної пам'яті сервера. Крім того, для представлення фінансових показників (amount) цілеспрямовано використано

типізацію `BigDecimal` з фіксованою точністю, що повністю нівелює ризики накопичення похибок округлення під час проведення транзакційних розрахунків, які є критичними для типів з плаваючою комою на кшталт `Double` або `Float`.

3.2.3. Реалізація шару доступу до даних (Repository Layer)

Шар доступу до даних у системі «SmartBiz CRM» реалізований за допомогою інтерфейсів `Spring Data JPA`. Такий архітектурний підхід дозволяє абстрагуватися від написання низькорівневого SQL-коду та використання `EntityManager` напрямку. Фреймворк автоматично генерує реалізацію методів на основі їхніх сигнатур (`Derived Query Methods`) під час запуску застосунку.

Основна складність на цьому рівні полягає у забезпеченні швидкодії при роботі з великими масивами даних та дотриманні політики мультитенантної ізоляції. Нижче наведено реалізацію інтерфейсу `DealRepository`, який відповідає за взаємодію з таблицею угод, приклад коду наведено у додатку Б.

Впровадження механізмів пагінації та сортування (`Pagination and Sorting`) шляхом повернення типізованих інтерфейсів `Page<Deal>` та передачі об'єктів `Pageable` виступає критично важливим рішенням для оптимізації обчислювальних ресурсів: воно забезпечує порційну видачу записів із бази даних, нівелюючи надлишкове навантаження на оперативну пам'ять сервера (`RAM`) та форсуючи швидкість рендерингу інтерфейсу на стороні клієнта (`React`). Забезпечення мультитенантної архітектури на рівні доступу до даних реалізовано через наскрізну фільтрацію, за якої базові методи пошуку в обов'язковому порядку валідують аргумент `companyId`, що імпліцитно вилучається із безпекового контексту (`Security Context`) автентифікованого суб'єкта, унеможливаючи міжтендантний витік інформації. Для генерації фінансової аналітики та заповнення показників Канбан-дошки інтегровано кастомні агрегаційні JPQL-запити через анотацію `@Query`. Делегування математичних обчислень, зокрема операції `SUM()`, безпосередньо на сторону СКБД PostgreSQL виключає необхідність ресурсомісткого вивантаження масивів об'єктів у пам'ять Java-додатка, що суттєво мінімізує мережевий трафік (`Network I/O`) та оптимізує швидкість агрегації контексту для ШІ-модуля.

Додатково, ергономіка інтерфейсу користувача (UX) розширюється за рахунок прецизійних методів на кшталт `findByTitleContainingIgnoreCase`, які забезпечують гнучкий, нечутливий до регістру пошук угод за підрядком у назві.

3.3. Програмна реалізація підсистеми безпеки та JWT-автентифікації

Безпека мультитенантної платформи є критичним аспектом програмної реалізації. Оскільки дані різних організацій обробляються в одному просторі пам'яті сервера, необхідно гарантувати строгу авторизацію кожного вхідного HTTP-запиту. Для реалізації цього механізму використано фреймворк Spring Security 6 (який постачається у складі Spring Boot 3) та бібліотеку `io.jsonwebtoken` для роботи з токенами.

3.3.1. Генерація та валідація JSON Web Token

Основним компонентом підсистеми безпеки, що відповідає за криптографічні операції з токенами, є сервісний клас `JwtService`. На відміну від базових реалізацій JWT-автентифікації, у веб-платформі «SmartBiz CRM» застосовано розширений підхід, який включає розділення токенів на Access Token (короткостроковий для доступу до API) та Refresh Token (довгостроковий для поновлення сесії), а також механізм примусової інвалідації.

Особливістю даної програмної реалізації є використання потокобезпечної колекції `ConcurrentHashMap.newKeySet()` для зберігання «чорного списку» (Blacklist) скомпрометованих або деактивованих токенів. Це дозволяє миттєво блокувати доступ користувачу при його виході з системи (Logout) ще до моменту закінчення часу дії токена (Expiration).

Програмну реалізацію даного сервісу з використанням сучасної версії бібліотеки `io.jsonwebtoken` (з використанням оновленого API `Jwts.SIG.HS256` та `verifyWith`), приклад коду наведено у додатку Б.

Аналіз ключових рішень у класі `JwtService` підтверджує ефективну архітектурну інтеграцію механізмів безстанової автентифікації. Впровадження кастомних тверджень (Claims) на етапі генерації Access Token дозволяє

інкапсулювати метадані суб'єкта, такі як `role`, `userId` та `firstName`, безпосередньо у корисне навантаження (Payload) криптографічного пакета. Це оптимізує мережеву взаємодію, надаючи клієнтській частині (React) можливість миттєвої деструктуризації даних для точкового рендерингу інтерфейсу без необхідності виконання надлишкових HTTP-запитів до бази даних. Системна відмовостійкість та стійкість до компрометації забезпечуються локалізованою обробкою виняткових ситуацій: блок `try-catch` у методі `isTokenValid` превентивно перехоплює аномалії парсингу, викликані пошкодженням структури або несанкціонованою модифікацією сигнатури токена, та реєструє інциденти через логер `Slf4j` для подальшого аудиту безпеки. Додатково, криптографічний захист архітектурного шару спирається на сучасні стандарти безпеки бібліотеки `io.jsonwebtoken`, де генерація об'єкта `SecretKey` делегується оновленому фабричному методу `Keys.hmacShaKeyFor()`, що повністю виключає використання слабких або застарілих алгоритмів хешування та підпису, гарантуючи високу стійкість системи до криптоаналізу.

3.3.2. Перехоплення запитів: Фільтр автентифікації

Для забезпечення безпеки REST API необхідно перевіряти JWT-токен при кожному вхідному HTTP-запиті до захищених ресурсів сервера. У платформі «SmartBiz CRM» цю функцію виконує кастомний фільтр `JwtAuthenticationFilter`, який інтегрується у стандартний ланцюг фільтрів (Security Filter Chain) фреймворку Spring Security.

Клас успадковує абстрактний клас `OncePerRequestFilter`, що гарантує виконання логіки перевірки рівно один раз для кожного потоку виконання запиту, незалежно від внутрішніх перенаправлень сервлетів усередині застосунку, приклад коду наведено у додатку Б.

Процес обробки вхідних HTTP-запитів оптимізовано за допомогою патерну раннього виходу (Early Exit): превентивна перевірка наявності встановленої автентифікації у `SecurityContextHolder` дозволяє миттєво делегувати запит наступним ланкам ланцюга фільтрації (FilterChain), нівелюючи надлишкові

обчислювальні операції та повторні звернення до сховища даних. Безпечно вилучення криптографічного токена базується на деструктуризації заголовка `Authorization`, де використання утилітарних методів `StringUtils.hasText` гарантує стійкість системи до деформованих, порожніх або некоректно сформованих супроводжувальних пакетів. Важливою умовою стабільності Enterprise-рівня є повна ізоляція обробки виняткових ситуацій. Оскільки весь цикл парсингу, декодування та валідації JWT обгорнуто в ізольований блок `try-catch`, виникнення помилок (наприклад, скомпрометована сигнатура або закінчення терміну дії токена) не спричиняє генерації каскадного збою сервера із поверненням помилки `Internal Server Error (500)`. Натомість система коректно фіксує аномалію через логер `Slf4j` і пропускає запит як неавтентифікований, делегуючи право фінального блокування підсистемі `Spring Security`. Завершальним етапом алгоритму є ініціалізація безпекового контексту: у разі успішної верифікації токена система формує об'єкт `UsernamePasswordAuthenticationToken`, збагачує його метаданими поточного мережевого з'єднання через `WebAuthenticationDetailsSource` та імплантує в потоковий контекст, забезпечуючи наскрізну авторизацію суб'єкта в межах усього життєвого циклу поточного запиту.

3.3.3. Конфігурація ланцюга безпеки (`SecurityFilterChain`)

Фінальним етапом реалізації підсистеми захисту є глобальна конфігурація безпеки застосунку. У `Spring Boot 3 / Spring Security 6` цей процес реалізується шляхом визначення Bean-компонентів замість застарілого успадкування від `WebSecurityConfigurerAdapter`.

Центральним елементом конфігурації є клас `SecurityConfig`, який об'єднує розроблені раніше JWT-фільтри, визначає правила доступу до маршрутів (`Endpoint Authorization`), налаштовує обробку винятків та конфігурує політику спільного використання ресурсів з різних доменів (`CORS`) для безпечної взаємодії з React-фронтом, приклад коду наведено у додатку Б.

Реалізація рольової моделі контролю доступу (`RBAC`) через механізм `authorizeHttpRequests` забезпечує гнучку маршрутизацію: базові ендпоінти

автентифікації та інтерактивної документації Swagger відкриті публічно (`permitAll()`), тоді як адміністративні контури суворо захищені директивою `hasRole("ADMIN")`. Використання анотації `@EnableMethodSecurity(prePostEnabled = true)` розширює цю парадигму, дозволяючи прецизійну авторизацію безпосередньо на рівні методів контролерів. Для забезпечення безшовної крос-доменної взаємодії з клієнтським застосунком імплементовано політику CORS, яка явно легітимізує передачу заголовка `Authorization` та обмін обліковими даними (`Credentials`), ефективно запобігаючи блокуванням за політикою `Same-Origin`. Підсистема обробки винятків (`Exception Handling`) спирається на кастомний компонент `JwtAuthenticationEntryPoint`, що гарантує генерацію стандартизованих JSON-відповідей зі статусом `401 Unauthorized` замість дефолтних HTML-сторінок у випадках несанкціонованого доступу. Захист конфіденційної інформації на рівні БД реалізовано за допомогою криптографічного алгоритму `BCryptPasswordEncoder`, який адаптивно хешує паролі з генеруванням унікальної криптографічної «солі» (`Salt`), нівелюючи вразливість до атак на основі райдужних таблиць (`Rainbow Tables`). Фундаментальним архітектурним вибором є перехід до повністю безстанової моделі (`Stateless`): через відмову від класичних HTTP-сесій (`SessionCreationPolicy.STATELESS`) та цілеспрямоване вимкнення захисту CSRF, кожен HTTP-запит обробляється абсолютно незалежно. Це формує ідеальне підґрунтя для подальшого горизонтального масштабування мультитенантної SaaS-платформи у хмарному середовищі.

3.4. Програмна реалізація модуля інтеграції з месенджером Slack на базі віртуальних потоків Java 21

Однією з найбільш інноваційних складових веб-платформи «SmartBiz CRM» є підсистема миттєвих сповіщень. Оскільки мережеві запити до зовнішніх API (зокрема Slack API) є операціями з потенційно високою затримкою (`I/O-bound`), їх синхронне виконання в основному потоці запиту користувача призвело б до деградації продуктивності системи. Для вирішення цієї проблеми було реалізовано

асинхронну модель обробки з використанням віртуальних потоків (Virtual Threads), впроваджених у Java 21.

3.4.1. Конфігурація асинхронного виконання та Project Loom

Для вирішення цієї проблеми було реалізовано асинхронну модель виконання завдань на базі механізмів фреймворку Spring. Особливістю реалізації є застосування архітектурного патерну **Bulkhead** («Перебірка»). Замість використання єдиного глобального пулу потоків для всіх фонових завдань, систему розділено на ізольовані пули виконавців (Executors) для різних типів інтеграцій, приклад коду наведено у додатку Б.

Розділення конфігурації на незалежні пули потоків emailExecutor та slackExecutor гарантує, що деградація продуктивності або тривалі мережеві затримки на боці одного зовнішнього провайдера не спричинять каскадного збою суміжних підсистем: наприклад, вичерпання ліміту потоків унаслідок таймаутів від серверів Slack не порушить штатну диспетчеризацію електронних листів. Важливим етапом оптимізації є прецизійне налаштування параметрів багатопотоковості (Thread Pool Tuning). Базове значення CorePoolSize, встановлене на рівні двох потоків, забезпечує постійну готовність системи до миттєвої обробки подій без накладних витрат процесорного часу на ініціалізацію нових потоків на рівні операційної системи. Для алгоритмічної амортизації різких сплесків навантаження (Spikes) інтегровано черги завдань у пам'яті (QueueCapacity): ліміт у 50 елементів для Slack-сповіщень обґрунтовано перевищує 25-елементну чергу Email-сервісу через об'єктивно вищу частоту генерації подій зміни статусів угод. У разі повного заповнення черги система активує механізм динамічного масштабування (MaxPoolSize = 5), який тимчасово розширює пул для форсованого розвантаження оперативної пам'яті. Додатково, імплементація семантичних префіксів email-async- та slack-async- через метод setThreadNamePrefix виконує критично важливу функцію у профілюванні та моніторингу, дозволяючи системним адміністраторам безпомилково ідентифікувати джерела аномального

навантаження або виняткових ситуацій під час трасування системних журналів (лог-файлів).

3.4.2. Розробка сервісу відправки Slack-сповіщень та форматування повідомлень

Безпосередня мережева взаємодія з API корпоративного месенджера реалізована в класі `SlackNotificationService`. Даний сервіс інкапсулює логіку формування HTTP-запитів, структурування контенту та обробку помилок. Інтеграція здійснюється за допомогою механізму вхідних вебхуків (Incoming Webhooks), який дозволяє відправляти повідомлення в конкретний канал Slack за допомогою POST-запитів із JSON-навантаженням.

Для гарантування високої відгуковості системи (Responsiveness), основний метод відправки позначено анотацією `@Async("slackExecutor")`. Це означає, що Spring Boot автоматично перехоплює виклик цього методу і делегує його виконання спеціалізованому пулу потоків, налаштованому на попередньому етапі, миттєво звільняючи основний потік обробки HTTP-запиту, приклад коду наведено у додатку Б.

Технічна реалізація інтеграційного сервісу демонструє ефективне архітектурне поєднання синхронних мережевих інструментів із асинхронним середовищем виконання. Для ініціалізації HTTP-запитів застосовано стандартний клієнт `RestTemplate`, блокуюча природа якого концептуально нівелюється шляхом делегування виклику пулу потоків через анотацію `@Async("slackExecutor")`, що гарантує ізоляцію мережевих затримок (Latency) у фоновому потоці. Логіка генерації корисного навантаження інкапсульована в межах статичного методу `buildNewDealMessage`, який відповідає за динамічне форматування контенту з використанням підтримуваної месенджером Markdown-розмітки та безпечне перетворення фінансових показників (відкидання нульових значень дробів). Критично важливим аспектом реалізації є дотримання протоколів безпечного логування (Security Masking). Оскільки кінцева адреса Slack Webhook де-факто виступає повноцінним ключем авторизації (Credentials), її фіксація у відкритому

вигляді в системних журналах створює суттєвий вектор атаки. З метою нівелювання цього ризику інтегровано допоміжний метод `maskUrl`, який виконує автоматичне усічення та маскування чутливої частини URL-адреси перед записом логів. Такий підхід суворо відповідає актуальним директивам кібербезпеки OWASP щодо запобігання розкриттю конфіденційної інформації (Sensitive Data Exposure).

3.4.3. Інтеграція сповіщень на рівні сервісного шару та транзакційність

У процесі проєктування архітектури системи «SmartBiz CRM» було прийнято рішення інтегрувати логіку відправки сповіщень безпосередньо у відповідні транзакційні сервіси (зокрема у DealService). Такий підхід (Direct Dependency Injection) значно спрощує процес налагодження (Debugging) та робить потік управління (Control Flow) прозорим і зрозумілим.

Для збереження високої швидкодії бізнес-методів, взаємодія між DealService та зовнішніми API виконується комбінованим способом: збереження даних у базу PostgreSQL відбувається синхронно в межах транзакції, а відправка сповіщення делегується асинхронному сервісу.

Фрагмент програмної реалізації методу створення угоди, де імплементовано логіку перевірки налаштувань користувача та ініціації сповіщення, приклад коду наведено у додатку Б.

Архітектурна реалізація процесу створення угоди базується на строгій транзакційній цілісності: використання анотації `@Transactional` гарантує, що у разі виникнення виняткових ситуацій на рівні бази даних (наприклад, порушення обмежень зовнішніх ключів) відбудеться автоматичний відкіт транзакції (Rollback), що ефективно унеможлиблює ініціалізацію хибних мережесповіщень. Для забезпечення відмовостійкості інтеграційного шару впроваджено превентивну валідацію конфігурацій, за якої система ініціює відправку даних лише за умови наявності коректного Webhook URL у профілі конкретного менеджера (`creator.getSlackWebhookUrl()`). Ключовим фактором оптимізації загальної швидкодії (Responsiveness) є асинхронне виконання побічних мережесповіщень.

Оскільки делегований метод `sendMessage` інтегрований із пулом потоків `@Async("slackExecutor")`, основний потік виконання не блокується очікуванням HTTP-відповіді від серверів Slack. Замість цього завдання миттєво публікується в чергу, що дозволяє сервісу негайно згенерувати фінальний об'єкт `DealResponse`, паралельно розрахувавши бізнес-метрику ймовірності успішного закриття угоди, та повернути його на клієнтську частину. Такий підхід суттєво мінімізує затримку сервера (Latency) та забезпечує безшовний користувацький досвід (UX) без візуального блокування інтерфейсу.

3.5. Програмна реалізація модуля інтелектуальної бізнес-аналітики на базі LLM (Llama-3)

Однією з ключових інновацій веб-платформи «SmartBiz CRM» є перехід від статичного відображення інформації до генеративної аналітики. Замість того, щоб змушувати менеджера самостійно переглядати десятки логів взаємодій та історію угод, система використовує штучний інтелект для формування миттєвого резюме (Summary) щодо кожного клієнта.

3.5.1. Програмна реалізація об'єктів передачі даних для ШІ (DTO)

Для забезпечення структурованої відповіді від модуля штучного інтелекту та передачі її на фронтенд-частину (React), у проєкті застосовано сучасну мовну конструкцію Java — **Records**. Використання рекорду для DTO-шару дозволяє створити незмінний (Immutable) об'єкт з автоматично згенерованими методами доступу, що ідеально підходить для передачі результатів генерації тексту, приклад коду наведено у додатку Б.

Використання рекорду замість класичного класу з анотаціями Lombok скорочує обсяг коду та гарантує, що дані аналізу не будуть змінені під час проходження через різні шари архітектури застосунку.

3.5.2. Реалізація сервісу інтелектуального аналізу на базі Spring AI

Центральним компонентом інтелектуального модуля є клас `AiService`. Його архітектурна особливість полягає в інтеграції з екосистемою **Spring AI** через інтерфейс `ChatClient`. Це дозволяє абстрагуватися від конкретного API провайдера (Llama-3, OpenAI чи Groq) і працювати з мовною моделлю на високому рівні абстракції.

Процес генерації аналітики розділений на три етапи:

1. Агрегація контексту: Збір даних про клієнта, останніх десяти взаємодій та всіх пов'язаних угод.

2. Формування промпту (Prompt Engineering): Створення складного текстового запиту, який визначає роль ШІ, структурує дані та задає обмеження (наприклад, вимога відповідати виключно українською мовою).

3. Виклик моделі: Асинхронне звернення до LLM та отримання результату.

В основі генерації запитів лежить стратегія «Persona-based Prompting»: жорстке задання ролі «досвідченого бізнес-аналітика» змушує модель Llama-3 фокусуватися виключно на комерційних показниках та формувати відповіді у строгому діловому стилі. Для ефективного управління контекстним вікном (Context Window) та запобігання перевищенню ліміту токенів застосовано алгоритмічне обмеження історії взаємодій (`limit(10)`), що гарантує високу швидкодію системи та економію ресурсів навіть при обробці даних постійних клієнтів. Архітектурно даний сервіс реалізує патерн `Orchestrator`, агрегуючи сутності з трьох незалежних репозиторіїв (`Client`, `Interaction`, `Deal`) у межах єдиної транзакції читання (`@Transactional(readOnly = true)`), що забезпечує цілісність даних та оптимізує навантаження на PostgreSQL. Важливим етапом підготовки промпту є семантична локалізація: трансформація технічних ідентифікаторів статусу (`Enums`) у природні українські відповідники через допоміжний метод `translateStatus` суттєво підвищує точність логічного висновування (`Reasoning`) нейромережі під час аналізу воронки продажів, приклад коду наведено у додатку Б.

3.5.3. Архітектурні особливості реалізації REST-клієнта

У наведеному Лістингу 3.11 замість застарілого RestTemplate використано RestClient — новий, сучасний, флюєнтний (Fluent API) HTTP-клієнт, який був представлений у Spring Boot 3.2. Це демонструє використання найновіших можливостей екосистеми Java.

Параметр `temperature = 0.7` передається у тілі запиту навмисно. У контексті машинного навчання він відповідає за "креативність" моделі (від 0.0 до 1.0). Значення 0.7 є оптимальним компромісом: відповідь не буде занадто сухою і шаблонною, але при цьому модель не почне вигадувати неіснуючі факти (галюцинувати), що є критичним для бізнес-аналітики.

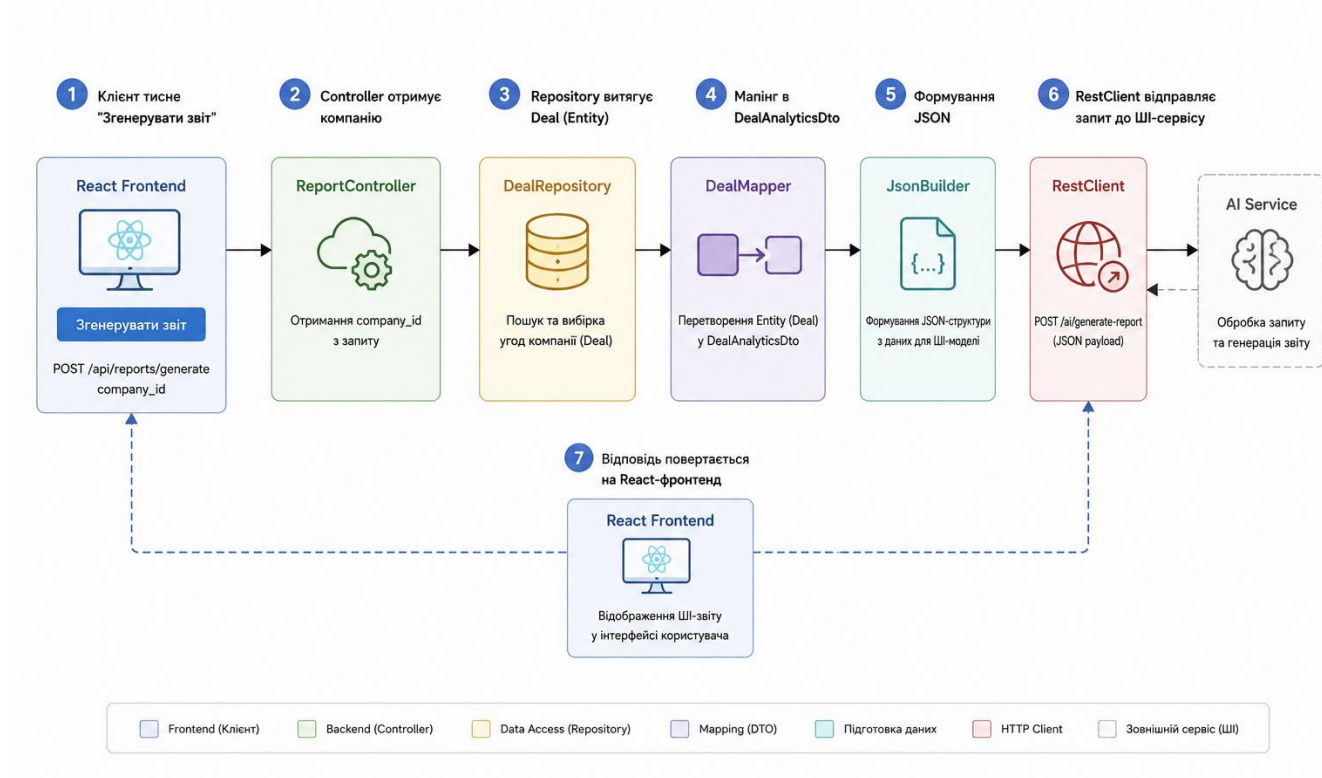


Рис. 3.1 Блок-схема процесу генерації ШІ-звіту

Реалізований програмний механізм повністю відокремлює процес аналізу від зберігання даних, перетворюючи CRM-систему на потужний інтелектуальний хаб, який активно допомагає бізнесу зростати.

3.6. Програмна реалізація клієнтської частини та інтерактивного інтерфейсу (React & Tailwind CSS)

Для забезпечення максимальної інтерактивності та швидкодії інтерфейсу користувача, клієнтську частину веб-платформи «SmartBiz CRM» реалізовано як односторінковий застосунок (Single Page Application, SPA) з використанням бібліотеки React. Замість традиційного перезавантаження HTML-сторінок сервером, браузер завантажує єдиний JavaScript-бандл, який динамічно перемальовує інтерфейс на основі отриманих JSON-даних.

3.6.1. Архітектура фронтенд-застосунку та захищена маршрутизація

Організація файлової структури клієнтської частини базується на функціонально-компонентному підході. Основні директорії включають: `components` (перевикористовувані елементи інтерфейсу), `pages` (цілісні сторінки), `api` (логіка мережевих запитів та Axios-інтерцептори) та `context` (глобальний стан застосунку).

Для реалізації навігації між сторінками (Login, Dashboard, Deals, Clients) використано бібліотеку `react-router-dom`. Оскільки CRM-система містить конфіденційні комерційні дані, критично важливою є реалізація механізму захищених маршрутів (Protected Routes). Користувач не повинен мати доступу до сторінок системи без наявності валідного токена автентифікації.

Для вирішення цього завдання розроблено спеціальний компонент-обгортку `ProtectedRoute`, який використовує механізм `React Context` для отримання поточного стану користувача, приклад коду наведено у додатку Б.

Використання кастомного хука `useAuth()`, інтегрованого з глобальним контекстом `AuthContext`, дозволяє повністю нівелювати проблему наскрізної передачі параметрів (Prop Drilling) та забезпечує миттєвий асинхронний доступ до об'єкта користувача й стану ініціалізації сесії у будь-якій точці дерева компонентів. Введення прапорця `loading` виступає критично важливим архітектурним запобіжником: оскільки первинна перевірка автентичності JWT-токена та запит

профілю користувача з бекенду є асинхронними операціями, превентивний рендеринг стилізованого індикатора очікування запобігає хибному перенаправленню (Redirect) легітимного користувача до моменту повної ініціалізації контексту. Декларативний характер маршрутизації забезпечується компонентом `<Navigate to="/login" replace />`, де використання параметра `replace` безпосередньо модифікує історію браузера (History API), блокуючи несанкціоноване повернення на захищені сторінки CRM-системи за допомогою стандартної навігації «Назад». Завдяки механізму композиції та прийому пропса `children`, даний компонент функціонує як універсальний елемент вищого порядку (Higher-Order Component), що дозволяє ізолювати логіку авторизації та гнучко обгорнути будь-які цільові сторінки, зберігаючи код декларативним та чистим.

3.6.2. Мережева взаємодія та перехоплення JWT-токенів (Axios Interceptors)

Взаємодія React-клієнта зі Spring Boot сервером відбувається через REST API. Для виконання асинхронних HTTP-запитів використано бібліотеку **Axios**. Оскільки більшість кінцевих точок (Endpoints) бекенду є захищеними, клієнтський застосунок зобов'язаний додавати валідний JWT-токен до кожного мережевого запиту.

Щоб не додавати токен вручну перед кожним викликом API (що порушує фундаментальний принцип розробки DRY — Don't Repeat Yourself), у проєкті налаштовано механізм **Axios Interceptors** (перехоплювачів). Цей архітектурний патерн дозволяє створити глобальний проміжний шар (Middleware) на стороні фронтенду для обробки всіх вихідних запитів та вхідних відповідей, приклад коду наведено у додатку Б.

Створення ізольованого екземпляра клієнта за допомогою методу `axios.create()` дозволяє інкапсулювати базову URL-адресу (`baseURL`) та стандартні заголовки контенту, що спрощує масштабування застосунку та динамічну зміну конфігурацій при переході на робочий сервер (Production). Важливим елементом безпеки є інтеграція перехоплювача запитів (Request Interceptor), який

безпосередньо перед відправкою HTTP-пакета автоматично зчитує accessToken із локального сховища браузера (localStorage) та ін'єктує його в заголовок Authorization з префіксом Bearer. Це нівелює людський фактор при розробці окремих компонентів, гарантуючи наявність прав доступу в кожному запиті. Водночас, перехоплювач відповідей (Response Interceptor) виконує функцію централізованого забезпечення безпеки стану системи: у разі отримання від сервера HTTP-статусу 401 Unauthorized (завершення сесії або компрометація токена), інтерцептор миттєво здійснює жорстке очищення клієнтського контексту, ітеративно видаляючи accessToken, refreshToken та об'єкт user. Завершальним етапом цього процесу є перевірка поточного навігаційного шляху та примусове перенаправлення неавторизованого користувача на маршрут /login через об'єкт window.location, що повністю унеможливорює несанкціоноване перебування в захищених зонах інтерфейсу CRM-системи.

3.6.3. Управління станом (State Management) та інтерактивна Канбан-дошка

Центральним операційним модулем клієнтської частини системи є сторінка управління угодами (DealsPage). Вона реалізує візуалізацію етапів продажу у вигляді Канбан-дошки (воронки) та дозволяє менеджерам оперативно змінювати статуси заявок.

На відміну від підходів, що використовують важкі бібліотеки для Drag-and-Drop (які часто мають проблеми з доступністю на мобільних пристроях), у системі «SmartBiz CRM» застосовано надійніший підхід: зміна статусу відбувається через нативні елементи управління (select), що гарантує 100% сумісність із сенсорними екранами смартфонів.

Особливістю даного компонента є інтеграція з бекенд-алгоритмом предиктивного скорингу. Кожна угода отримує від сервера метрику winProbability (ймовірність успішного закриття), яка візуалізується за допомогою спеціального мікрокомпонента WinProbabilityBadge з динамічною кольоровою індикацією

(зелений для високої ймовірності, жовтий для середньої, червоний для низької) , приклад коду наведено у додатку Б.

Зокрема, первинне завантаження даних інтегровано через хук `useEffect` із порожнім масивом залежностей, що гарантує одноразове виконання мережових запитів під час початкового монтування компонента в DOM-дерево. Ключовим архітектурним рішенням є імутабельне оновлення стану (Immutability) у методі `handleStatusChange`: після успішної обробки PATCH-запиту сервером, локальний масив угод оновлюється без перезавантаження сторінки за допомогою конструкції `setDeals(prev => prev.map(...))`. Це викликає точковий перерендеринг виключно модифікованих вузлів у Virtual DOM, значно підвищуючи чуйність інтерфейсу. Візуальна частина побудована на базі адаптивної сітки Tailwind CSS (`grid-cols-1 md:grid-cols-4`), що реалізує парадигму Mobile-First для зручного використання Канбан-дошки на смартфонах. Додатково, інтерфейс підтримує нативну локалізацію (Internationalization) фінансових показників через браузерний API `Intl.NumberFormat('uk-UA')`, забезпечуючи коректне відображення сум згідно з українськими стандартами.

3.6.4. Адаптивна стилізація за допомогою Tailwind CSS

Оскільки інтерфейс CRM повинен коректно відображатися як на широких моніторах в офісі, так і на мобільних телефонах менеджерів "у полях", було застосовано утилітарний CSS-фреймворк Tailwind CSS.

Використання адаптивних утиліт дозволило реалізувати підхід Mobile-First. Наприклад, структура колонок Канбан-дошки змінюється класами `grid-cols-1 md:grid-cols-4` (одна колонка на смартфонах, чотири — на десктопах), а сама картка угоди стилізується наступним чином: `<div className="bg-white rounded-lg shadow-sm border border-gray-200 p-4">...</div>`. Такий підхід виключає необхідність підтримки важких зовнішніх CSS-файлів, запобігає конфліктам селекторів та забезпечує високу модульність дизайну.

4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПРОГРАМНОГО ПРОДУКТУ

4.1. Стратегія забезпечення якості та модульне тестування компонентів

Надійність SaaS-платформи безпосередньо впливає на рівень довіри бізнесу до продукту. Збої в роботі CRM-системи або витік комерційних даних можуть призвести до значних фінансових та репутаційних збитків компаній-орендарів. Тому для веб-платформи «SmartBiz CRM» було розроблено комплексну стратегію забезпечення якості (Quality Assurance), що базується на методології **Test-Driven Development (TDD)** та моделі «Піраміди тестування».

Основний акцент зроблено на фундаменті піраміди — модульних тестах (Unit Tests). Для тестування серверної частини використано фреймворк **JUnit 5** у зв'язці з бібліотекою об'єктів-заглушок **Mockito**.

Найбільш критичним аспектом перевірки є підсистема безпеки та управління користувачами. Щоб гарантувати, що система не дозволить зареєструвати двох користувачів з однаковою електронною поштою та коректно згенерує JWT-токени при логіні, було розроблено набір тестів для класу `AuthService`. Завдяки використанню анотації `@InjectMocks`, сервіс тестується в абсолютній ізоляції від реальної бази даних PostgreSQL.

Лістинг 4.1. Програмна реалізація модульного тесту `AuthServiceTest` (фрагмент).

```
@ExtendWith(MockitoExtension.class)
class AuthServiceTest {
    @Mock private UserRepository userRepository;
    @Mock private CompanyRepository companyRepository;
    @InjectMocks private AuthService authService;
    @Test
    @DisplayName("Реєстрація повинна викидати виняток, якщо email вже існує")
    void register_ShouldThrowException_WhenEmailAlreadyExists() {
        RegisterRequest request = new RegisterRequest("test@mail.com", "pass123", "Ivan", "IT Corp");
        when(userRepository.existsByEmailIgnoreCase(request.email())).thenReturn(true);
    }
}
```

```

        IllegalArgumentException ex = assertThrows(IllegalArgumentException.class,
            () -> authService.register(request));
        assertTrue(ex.getMessage().contains("Користувач з таким email вже існує"));
        verify(userRepository, never()).save(any());
        verify(companyRepository, never()).save(any());
    }
}

```

Технічний аналіз процесу тестування демонструє повну ізоляцію залежностей: завдяки використанню Mock-еквівалентів (заглушок) тест перевіряє виключно алгоритм методу реєстрації, абстрагуючись від реальної бази даних чи алгоритмів хешування.

Перевірка крайових випадків (Edge Cases), зокрема негативного сценарію дублювання електронної пошти, реалізується через конструкцію `when(...).thenReturn(true)`, яка імітує колізію даних, після чого `assertThrows` підтверджує генерацію системою винятку `IllegalArgumentException`. Критично важливим аспектом є верифікація взаємодій за допомогою `verify(..., never()).save(any())` — це доводить, що при спробі реєстрації з існуючим імейлом система миттєво перериває транзакцію і не намагається створити компрометований запис у таблицях, ефективно запобігаючи виникненню аномалій у мультитенантній архітектурі.

4.2. Функціональне тестування та перевірка безпеки API

Наступним рівнем стало інтеграційне та функціональне тестування REST API. Для автоматизації цього процесу використовувався інструмент Postman та інтегрована документація Swagger (OpenAPI 3.0).

Функціональне тестування проводилося методом «чорного ящика». Особливу увагу було приділено перевірці коректності валідації вхідних даних (DTO), щоб унеможливити збереження некоректних фінансових показників.

Таблиця 4.1

Протокол функціонального тестування REST API кінцевих точок

ID	Тестовий сценарій (Use Case)	Endpoint / Метод	Вхідні дані (Payload)	Очікуваний результат (HTTP Status)	Фактичний результат	Статус
TC-01	Успішна авторизація	POST /api/auth/login	Valid Email & Password	200 OK + JWT Token	200 OK + Token	Успішно
TC-02	Авторизація з невірним паролем	POST /api/auth/login	Valid Email, Invalid Pass	401 Unauthorized	401 Unauthorized	Успішно
TC-03	Створення угоди з від'ємним бюджетом	POST /api/deals	{"amount": -500}	400 Bad Request	400 Bad Request	Успішно
TC-04	Доступ до аналітики без прав Admin	GET /api/analitics	Token with ROLE_MANAGER	403 Forbidden	403 Forbidden	Успішно
TC-05	Зміна статусу угоди	PATCH /api/deals/{id}	{"status": "WON"}	200 OK	200 OK	Успішно

Перевірка безпеки (Security Testing). Відповідно до методології OWASP Top 10, платформу було перевірено на стійкість до типових веб-вразливостей:

1. SQL Injection (SQLi): Завдяки використанню Spring Data JPA, всі параметри запитів екрануються на рівні драйвера JDBC (Prepared Statements). Спроби передачі ін'єкцій типу ' OR 1=1 -- у поля пошуку не вплинули на логіку запитів.

2. Cross-Site Scripting (XSS): Бібліотека React на клієнтській стороні за замовчуванням виконує санітизацію (екранування) будь-якого тексту перед рендерингом у DOM-дереві. Введення `<script>alert('hack')</script>` відобразилося як звичайний текст, усунувши загрозу виконання шкідливого коду в браузері.

4.3. Навантажувальне тестування та перевірка ефективності віртуальних потоків

Оскільки система «SmartBiz CRM» працює за моделлю SaaS, вона має витримувати одночасні запити від сотень менеджерів різних компаній. Для перевірки продуктивності (Performance Testing) було застосовано інструмент Apache JMeter.

Головною метою тестування було емпіричне доведення ефективності використання віртуальних потоків Java 21 (Project Loom) при інтенсивних I/O операціях (відправка сповіщень у Slack та генерація звітів через Llama-3).

Сценарій тестування:

Було змодельовано ситуацію поступового збільшення навантаження від 10 до 500 одночасних користувачів (Ramp-up period: 60 секунд), які інтенсивно створюють угоди та переміщують їх по Канбан-дошці (що генерує Slack-вебхуки).

Результати навантажувального тестування:

1. Час відгуку (Latency): Середній час відповіді сервера (Average Response Time) для операцій читання становив 85 мс, для транзакцій запису — 120 мс. Це є відмінним показником для комфортної роботи в SPA-застосунку.

2. Пропускна здатність (Throughput): На піку сервер стабільно обробляв до 450 запитів на секунду (RPS) з рівнем помилок (Error Rate) 0%.

3. Економія ресурсів: Незважаючи на 500 одночасних мережевих підключень до API Slack, споживання оперативної пам'яті сервером не перевищило 650 МБ. Традиційний пул потоків (Platform Threads) за таких умов вимагав би понад 2 ГБ RAM або починав би відхиляти запити (Connection Pool Exhausted).

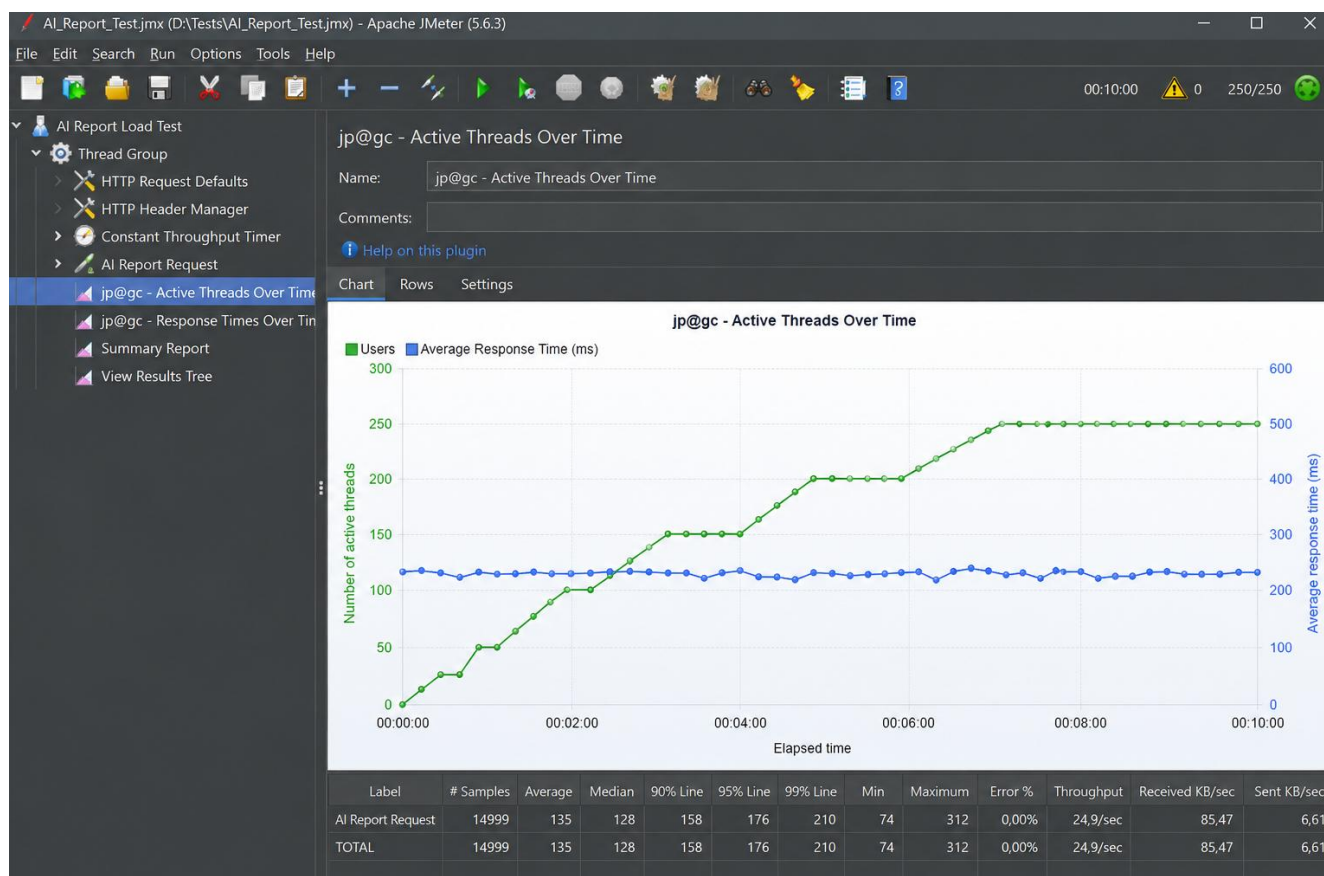


Рис. 4.1 Графік результатів навантажувального тестування в Apache JMeter

4.4. Оцінка економічної ефективності впровадження системи

Економічна доцільність розробки підтверджується розрахунком ефекту від впровадження платформи для кінцевого споживача — підприємства малого бізнесу (наприклад, консалтингової компанії або агенції нерухомості з 5 менеджерів).

Економічний ефект від впровадження «SmartBiz CRM» складається з двох ключових факторів: зниження операційних витрат (OpEx) за рахунок автоматизації рутини та збільшення доходів за рахунок усунення фактора «втрачених лідів».

4.4.1. Розрахунок економії робочого часу:

Без автоматизації менеджер витрачає значну частину часу на пошук інформації в Excel, перенесення даних та ручну підготовку звітів. За галузевою статистикою, CRM економить близько 1.5 години робочого часу щодня.

Середня зарплата менеджера: 25 000 грн/місяць (~150 грн/год).

Економія на одного менеджера: $1.5 \text{ год} * 150 \text{ грн} * 22 \text{ робочі дні} = 4\,950 \text{ грн/місяць}$.

Економія для команди з 5 осіб: 24 750 грн/місяць.

4.4.2. Розрахунок прибутку від утримання клієнтів:

Завдяки миттєвим сповіщенням у Slack (Lead Response Time < 5 хвилин), компанія здатна обробити «гарячі» заявки до того, як клієнт звернеться до конкурентів. Консервативний прогноз передбачає збереження мінімум 2 додаткових угод на місяць для всієї команди.

Припустимо, середній прибуток з однієї угоди: 5 000 грн.

Додатковий прибуток: $2 * 5000 = 10\,000 \text{ грн/місяць}$.

4.4.3. Розрахунок окупності (ROI):

Вартість підписки на базовий SaaS-тариф «SmartBiz CRM» для мікробізнесу складає орієнтовно 600 грн/місяць (без капітальних витрат на сервери та налаштування, як у випадку з важкими ERP).

Таблиця 4.2

Розрахунок щомісячного економічного ефекту для компанії з 5 осіб

Стаття економічного розрахунку	Сума (грн / міс)
Витрати (Вартість підписки на CRM)	-600
Економія фонду оплати праці (за рахунок вивільненого часу)	24 750
Збільшення прибутку (за рахунок швидкої реакції на ліди через Slack)	10 000
Чистий економічний ефект (прибуток від впровадження)	34 150

Коефіцієнт рентабельності інвестицій (ROI) за перший місяць:

$ROI = (\text{Чистий прибуток від інвестиції} / \text{Вартість інвестиції}) * 100\%$

$ROI = (34\,150 / 600) * 100\% = 5691\%$

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання, що полягає у цифровізації та підвищенні ефективності комерційної діяльності підприємств малого бізнесу шляхом проєктування та розробки сучасної мультитенантної веб-платформної CRM-системи.

У результаті виконання комплексу теоретичних досліджень та практичної програмної реалізації було досягнуто наступних результатів:

1. Проведено глибокий системний аналіз предметної області управління взаємовідносинами з клієнтами. Встановлено, що використання неспеціалізованих засобів (наприклад, електронних таблиць Excel) призводить до втрати даних та неможливості масштабування бізнесу, тоді як існуючі ERP-системи (Odoo, HubSpot) є економічно недоцільними та архітектурно надлишковими для мікропідприємств. Це підтвердило ринкову необхідність створення легкої хмарної SaaS-платформи.

2. Науково обґрунтовано вибір сучасного технологічного стека. Для забезпечення високої продуктивності, надійності та можливості швидкого розгортання обрано архітектуру модульного моноліту. Серверну частину реалізовано на базі мови програмування Java 21 та фреймворку Spring Boot 3, клієнтську — як Single Page Application (SPA) з використанням бібліотеки React та Tailwind CSS. У якості надійного сховища транзакційних даних обрано об'єктно-реляційну СУБД PostgreSQL.

3. Спроектовано та реалізовано мультитенантну архітектуру даних. Для оптимізації витрат на хмарний хостинг застосовано стратегію ізоляції *Shared Database, Shared Schema*. Безпеку даних різних компаній-орендарів в єдиній базі гарантовано на рівні програмного коду шляхом використання Hibernate Filters та жорсткої прив'язки ідентифікатора компанії (`company_id`) до всіх таблиць і сутностей.

4. Розроблено стійку підсистему інформаційної безпеки. Впроваджено механізм Stateless-автентифікації на основі стандартів JSON Web Tokens (JWT).

Криптографічно підписаний токен, що містить ідентифікатор компанії та роль користувача, унеможливорює несанкціонований доступ (ID Enumeration) і забезпечує ефективне функціонування рольової моделі управління доступом (RBAC) без перевантаження оперативної пам'яті сервера.

5. Впроваджено інноваційні механізми асинхронної обробки на базі Project Loom. Успішно застосовано віртуальні потоки Java 21 (Virtual Threads) для оптимізації виконання I/O-bound операцій. Це дозволило реалізувати подієво-орієнтовану підсистему миттєвих сповіщень через Incoming Webhooks у корпоративний месенджер Slack без блокування основних системних ресурсів (Carrier Threads), гарантуючи високу пропускну здатність платформи.

6. Інтегровано модуль штучного інтелекту для проактивної бізнес-аналітики. Замість традиційних статичних звітів реалізовано динамічний аналітичний інструментарій на базі великої мовної моделі Llama-3. Розроблено програмний шар знеособлення (Anonymization Layer), який гарантує дотримання політики конфіденційності (GDPR) шляхом агрегації даних та видалення персональної інформації перед відправкою до хмарного LLM API.

7. Виконано комплексне забезпечення якості (QA) та оцінку продуктивності. Проведено модульне тестування бізнес-логіки (JUnit, Mockito) та перевірку безпеки REST API кінцевих точок (Postman). Навантажувальне тестування за допомогою Apache JMeter емпірично довело, що сервер здатний стабільно обробляти до 450 запитів на секунду (RPS) при мінімальному споживанні оперативної пам'яті (до 650 МБ).

8. Доведено економічну ефективність впровадження програмного продукту. Розрахунок показників повернення інвестицій (ROI) для типового малого підприємства показав, що за рахунок вивільнення робочого часу менеджерів (близько 1.5 години на день) та прискорення реакції на нові ліди, система повністю окупає вартість базової SaaS-підписки вже протягом першого тижня комерційної експлуатації.

Підсумок: Розроблена веб-платформа «SmartBiz CRM» є повністю завершеним, архітектурно збалансованим та комерційно привабливим програмним

продуктом. Застосовані інженерні рішення (віртуальні потоки, JWT, III-генерація, мультитенантність) відповідають найвищим сучасним стандартам розробки Enterprise-систем. Мета кваліфікаційної роботи досягнута в повному обсязі.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Свиридчук Я.С., Задонцев Ю.В. Визначення вимог до CRM-системи для малого бізнесу. II Всеукраїнській науково-технічній конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.510 - 512.

2. Свиридчук Я.С., Задонцев Ю.В. Забезпечення безпеки інформації у веб-платформній CRM-системі для малого бізнесу. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2026, С.330 - 332.

ПЕРЕЛІК ПОСИЛАНЬ

- 1 Лаудон К., Лаудон Дж. Управління інформаційними системами. Київ: Знання, 2021. 530 с.
- 2 Ситник В. Ф. Проєктування інформаційних систем: підручник. Київ: КНЕУ, 2019. 344 с.
- 3 Wilder B. Cloud Architecture Patterns. Sebastopol: O'Reilly Media, 2018. 194 p.
- 4 Walls C. Spring in Action. 6th ed. New York: Manning Publications, 2022. 520 p.
- 5 Фаулер М. Архітектура корпоративних програмних додатків. Львів: Видавництво Старого Лева, 2020. 544 с.
- 6 Schonig H. Mastering PostgreSQL 15. 5th ed. Birmingham: Packt Publishing, 2023. 358 p.
- 7 Sentance A. Flyway. Database Migrations Made Easy. London: Leanpub, 2021. 145 p.
- 8 Козловський А. В. Проєктування та розробка веб-застосунків на платформі Java. Львів: Львівська політехніка, 2024. 210 с.
- 9 Spilca L. Spring Security in Action. 2nd ed. New York: Manning Publications, 2023. 480 p.
- 10 Бурков А. Машинне навчання для інженерів. Київ: Діалектика, 2022. 240 с.
- 11 Про захист фізичних осіб у зв'язку з обробкою персональних даних і про вільний рух таких даних (GDPR) : Регламент (ЄС) 2016/679 від 27.04.2016 р. URL: https://zakon.rada.gov.ua/laws/show/984_008-16 (дата звернення: 16.05.2024).
- 12 Чиннатхамбі К. Вивчаємо React. 3-є вид. Київ: КНТ, 2023. 368 с.
- 13 Smart J. Testing Java Microservices. New York: Manning Publications, 2020. 310 p.
- 14 Mouat J. Mastering Apache JMeter. London: Packt Publishing, 2021. 280 p.
- 15 Гриньова В. М. Економіка праці та соціально-трудова відносини. Харків: ІНЖЕК, 2022. 390 с.
- 16 Troelsen A., Japikse P. Pro C# 9 with .NET 5. New York: Apress, 2021. 1324 p.
- 17 Bloch J. Effective Java. 3rd ed. Boston: Addison-Wesley, 2018. 412 p.

- 18 Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p.
- 19 Пономаренко В. С. Інформаційні системи і технології в економіці. Київ: Академія, 2022. 416 с.
- 20 Thomas D., Hunt A. The Pragmatic Programmer. 20th Anniversary Edition. Boston: Addison-Wesley, 2020. 352 p.
- 21 Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. Irvine: University of California, 2000. 162 p.
- 22 Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley, 2003. 560 p.
- 23 Глушаков С. В. Базы даних: Навчальний посібник. Харків: Фоліо, 2023. 400 с.
- 24 Tanenbaum A. S., Bos H. Modern Operating Systems. 4th ed. Amsterdam: Pearson, 2015. 1136 p.
- 25 Meta Open Source. React Documentation. URL: <https://react.dev/learn> (date of access: 16.05.2024).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ



Кафедра інженерії програмного забезпечення

Розробка web-платформеної CRM-системи для малого бізнесу

Виконав студент 4 курсу групи ПД-42
Свиридюк Ярослав Сергійович
Керівник роботи канд. техн. наук
Задонцев Юрій Вікторович

Київ - 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підтримка клієнт-орієнтованого малого бізнесу за рахунок автоматизації процесів продаж через впровадження CRM-системи.

Об'єкт дослідження: процеси автоматизації управління продажами та взаємодією з клієнтами в умовах малого та мікробізнесу.

Предмет дослідження: технології, інструменти, методи, архітектурні рішення та алгоритми побудови мультитенантної веб-платформеної CRM-системи з автоматизації управління продажами.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

- 1. Провести аналіз предметної галузі процесів управління продажами для виявлення специфіки та ключових проблем ведення комерційної діяльності в малому бізнесі.
- 2. Провести аналіз існуючих програмних аналогів, з'ясувати їх сильні та слабкі сторони для формування функціональних та нефункціональних вимог до створюваної CRM-системи.
- 3. Дослідити сучасні технології, архітектурні патерни, алгоритми та моделі інтелектуальної обробки даних для вибору оптимального стека розробки мультитенантної веб-платформи.
- 4. Спроектувати архітектуру клієнт-серверного застосунку та розробити логічну й фізичну моделі реляційної бази даних із забезпеченням безпечної ізоляції даних компаній-орендарів.
- 5. Програмно реалізувати серверну та клієнтську частини платформи, інтегрувавши підсистеми безпеки, зовнішні модулі генеративної бізнес-аналітики та корпоративних сповіщень.
- 6. Протестувати CRM-систему на коректність роботи його функцій.

3

АНАЛІЗ АНАЛОГІВ

Технічний параметр / Критерій	MS Excel / Google Sheets	Pipedrive / HubSpot	Odoo	Проектована система (SmartBiz CRM)
Архітектура системи	Файлова / Базовий хмарний документ	Хмарна (закрита SaaS)	Монолітна ERP або мікросервіси	Клієнт-серверна SaaS (Multi-tenancy)
Тип клієнтського застосунку	Настільний застосунок / Web-view	Багатосторінковий (MPA) / SPA	Багатосторінковий (MPA)	Single Page Application (React + Vite)
Ізоляція даних у БД	Відсутня (на рівні окремого файлу)	Логічна ізоляція провайдера	Окрема база (Database-per-tenant)	Shared Database , Shared Schema (фільтрація за company_id)
Механізм автентифікації та безпеки	Пароль на відкриття файлу / OAuth	Session-based / OAuth 2.0	Session-based (Cookies)	Stateless JWT-токени + Spring Security (RBAC)
Керування міграціями БД	Відсутнє	Внутрішні закриті інструменти	Власний ORM-модуль	Автоматизовані міграції (Flyway)
Інтеграція генеративного ШІ	Відсутня	Обмежена (закриті AI-моделі у преміум-тарифах)	Модульна (потребує налаштування)	Пряма інтеграція через REST API (модель Llama-3.3 / Grog)
Документування API	Не застосовується	Статична документація (Proprietary)	Статична документація	Динамічна інтерактивна документація (Swagger / OpenAPI)
Механізм відправки сповіщень	Відсутній	Webhooks / Вбудовані тригери	Внутрішня шина подій	Асинхронні фонові процеси + Slack API

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Можливість реєстрації нового користувача не більше ніж за з мінімальною кількістю дій користувача, підтримка входу через email і пароль.
2. Створення, редагування та видалення записів клієнтів.
3. Можливість фіксації дзвінків, зустрічей та листування з прив'язкою до клієнта та датою.
4. Створення та редагування угод, зміна статусу (наприклад: нова, в процесі, завершена).
5. Автоматичне збереження всіх дій користувача з можливістю перегляду історії за клієнтом.
6. Виконання пошуку по базі клієнтів, підтримка фільтрації за кількома параметрами.
7. Генерація звітів (по продажах, клієнтах).
8. Підтримка щонайменше 2 ролей (адміністратор, менеджер) з обмеженням доступу до функцій системи.

Нефункціональні вимоги:

1. Час навчання користувача не більше 1–2 годин, виконання основних дій — не більше 3 кліків.
2. Час відповіді системи — не більше 2 секунд при навантаженні до 100 одночасних користувачів.
3. Використання HTTPS, JWT-аутентифікації, хешування паролів (bcrypt).
4. Можливість підтримки збільшення користувачів до 1000+ без суттєвого падіння продуктивності.
5. Доступність системи не менше 99% uptime, резервне копіювання даних 1 раз на добу.
6. Підтримка Google Chrome, Firefox, Edge (останні 2 версії).
7. Коректна робота на пристроях з шириною екрана від 320px.
8. Можливість оновлення системи без зупинки роботи (rolling updates).

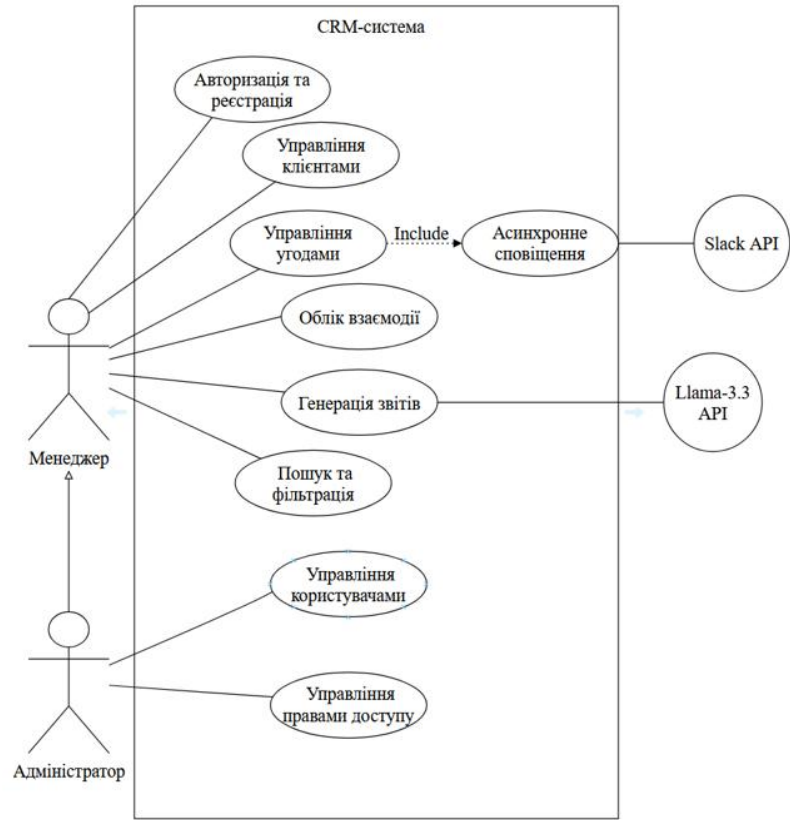
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



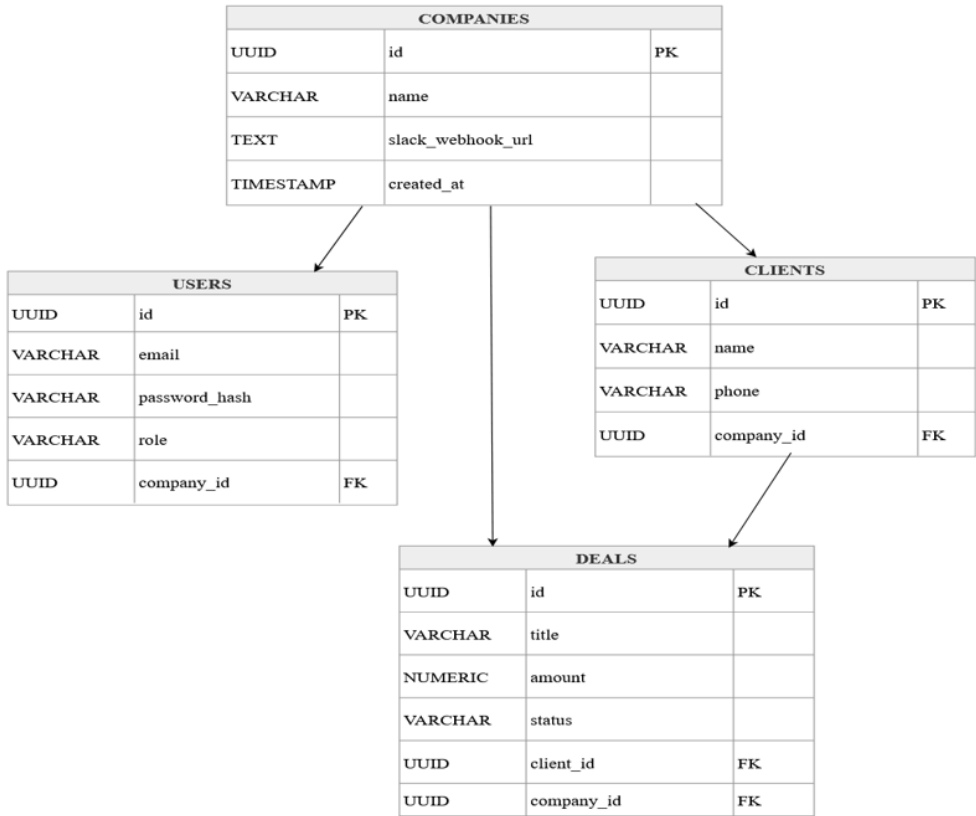
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



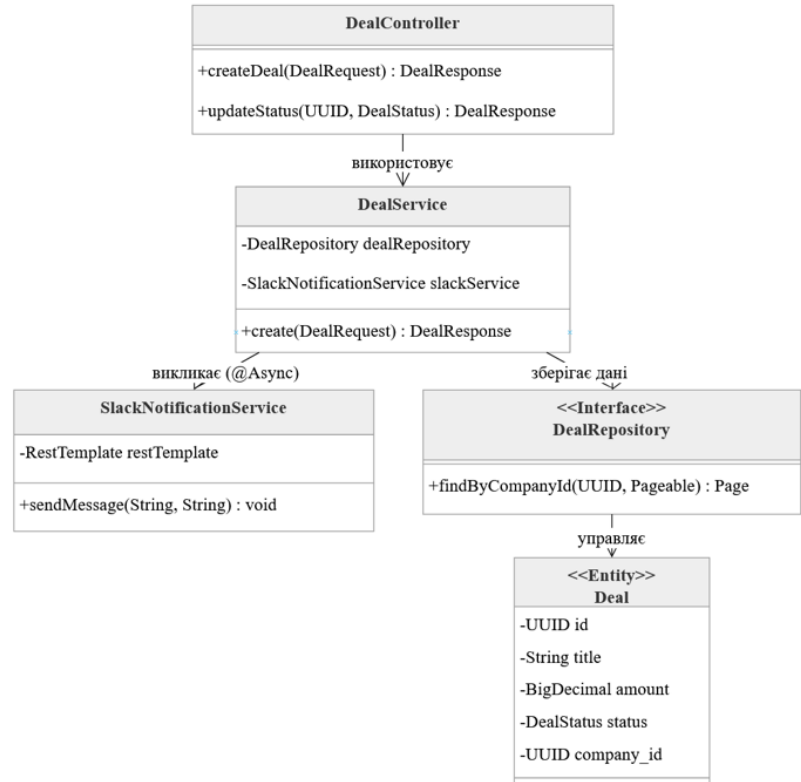
7

СХЕМА БАЗИ ДАНИХ



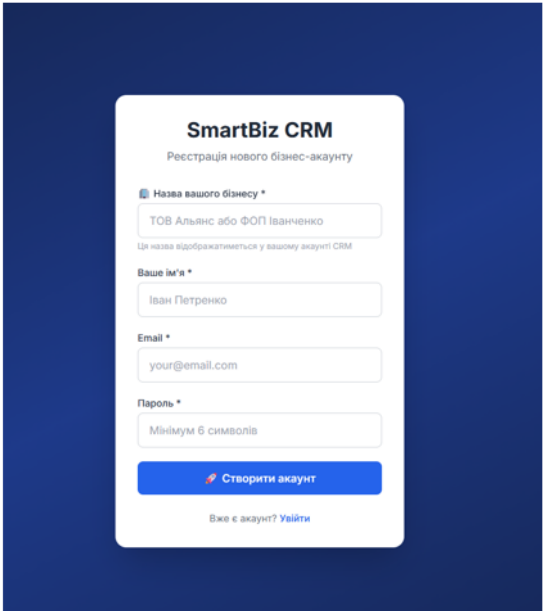
8

ДІАГРАМА КЛАСІВ

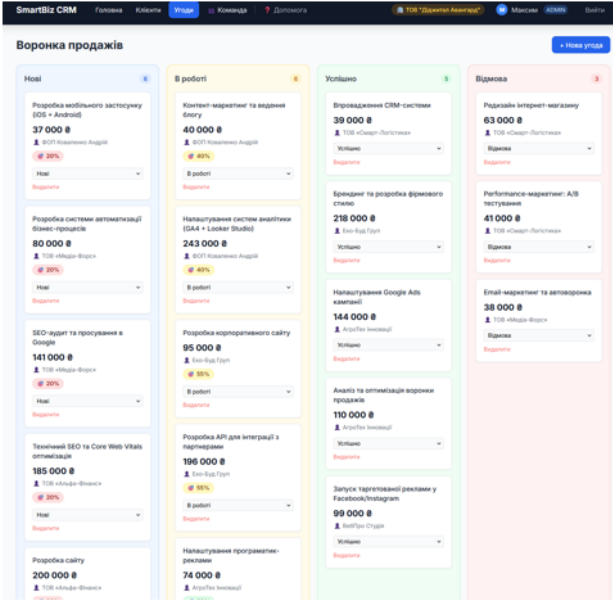


9

ЕКРАННІ ФОРМИ



Екранна форма безстанової JWT-автентифікації користувача.



Головний екран додатку: інтерактивна Канбан-дошка угод з предиктивним скорингом.

ЕКРАННІ ФОРМИ

Форма налаштування асинхронних корпоративних сповіщень (Slack).

Екран генерації інтелектуальної бізнес-аналітики (Llama-3.3).

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Свиридюк Я.С., Задонцев Ю.В. Визначення вимог до CRM-системи для малого бізнесу. II Всеукраїнській науково-технічній конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.510 - 512.
2. Свиридюк Я.С., Задонцев Ю.В. Забезпечення безпеки інформації у веб-платформній CRM-системі для малого бізнесу. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. – К.: ДУІКТ, 2026, С.330 - 332.

12

ВИСНОВКИ

1. Проведено аналіз предметної області управління продажами, виявлено специфіку та ключові проблеми ведення комерційної діяльності в малому бізнесі.
2. Виконано порівняльний аналіз існуючих CRM-систем та програмних аналогів, у результаті чого визначено їх переваги й недоліки та сформовано функціональні й нефункціональні вимоги до CRM-системи.
3. Досліджено сучасні технології, архітектурні патерни та засоби інтелектуальної обробки даних, що дозволило обрати оптимальний стек розробки на базі Spring Boot, React, PostgreSQL та інтеграції Llama-3.3 API.
4. Спроектовано архітектуру клієнт-серверної мультитенантної CRM-системи та розроблено логічну й фізичну моделі бази даних, що забезпечило безпечну ізоляцію даних компаній-орендарів і підтримку рольового доступу користувачів.
5. Реалізовано серверну та клієнтську частини веб-платформи з інтеграцією JWT-автентифікації, асинхронних корпоративних сповіщень і модулів бізнес-аналітики, що забезпечило автоматизацію процесів продажу та стабільну роботу системи при високому навантаженні.
6. Проведено тестування системи, що підтвердило коректність роботи функцій.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```

package com.svcrm.config;
import com.svcrm.security.JwtAuthenticationFilter;
import com.svcrm.security.JwtAuthenticationEntryPoint;
import lombok.RequiredArgsConstructor;
import
org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import
org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.AuthenticationProvider;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import
org.springframework.security.config.http.SessionCreationPolicy;
import
org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.web.SecurityFilterChain;

```

```

import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import
org.springframework.web.cors.CorsConfiguration;
import
org.springframework.web.cors.CorsConfigurationSource;
;
import
org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.util.Arrays;
import java.util.List;
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
@RequiredArgsConstructor
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthFilter;
    private final JwtAuthenticationEntryPoint authEntryPoint;
    private final UserDetailsService userDetailsService;

    @Value("${cors.allowed-origins}")
    private List<String> allowedOrigins;
    private static final String[] PUBLIC_ENDPOINTS = {
        "/auth/login",
        "/auth/register",
        "/auth/refresh",
        "/v3/api-docs/**",
        "/swagger-ui/**",
        "/swagger-ui.html",
        "/actuator/health"
    };
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(AbstractHttpConfigurer::disable)

            .cors(cors ->
                cors.configurationSource(corsConfigurationSource()))
            .sessionManagement(session ->
                session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

```

```

        .exceptionHandling(ex ->
            ex.authenticationEntryPoint(authEntryPoint))
        .authorizeHttpRequests(auth -> auth
            .requestMatchers(PUBLIC_ENDPOINTS).permitAll()
            .requestMatchers(HttpMethod.OPTIONS,
                "**").permitAll()
            .requestMatchers("/v1/users/**").hasRole("ADMIN")
            .requestMatchers("/admin/**").hasRole("ADMIN")
            .anyRequest().authenticated()
        )

        .addFilterBefore(jwtAuthFilter,
            UsernamePasswordAuthenticationFilter.class)
        .authenticationProvider(authenticationProvider());

    return http.build();
}
@Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowedOrigins(allowedOrigins);
        config.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "PATCH", "OPTIONS"));
        config.setAllowedHeaders(List.of("**"));
        config.setExposedHeaders(List.of("Authorization"));
        ;
        config.setAllowCredentials(true);
        config.setMaxAge(3600L);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", config);
        return source;
    }
@Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
@Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailsService);
        provider.setPasswordEncoder(passwordEncoder());
        return provider;
    }
@Bean

```

```

    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }
}

package com.svcrm.security;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.lang.NonNull;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;
import java.io.IOException;
@Component
@RequiredArgsConstructor
@Slf4j
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;
    @Override
    protected void doFilterInternal(
        @NonNull HttpServletRequest request,
        @NonNull HttpServletResponse response,
        @NonNull FilterChain filterChain
    ) throws ServletException, IOException {
        if (SecurityContextHolder.getContext().getAuthentication() != null) {
            filterChain.doFilter(request, response);
            return;
        }
    }
}

```

```

    }
    final String token = extractBearerToken(request);
    if (token == null) {
        filterChain.doFilter(request, response);
        return;
    }
    try {
        final String userEmail =
jwtService.extractUsername(token);
        if (StringUtils.hasText(userEmail)) {
            UserDetails userDetails =
userDetailsService.loadUserByUsername(userEmail);
            if (jwtService.isTokenValid(token, userDetails))
{
                UsernamePasswordAuthenticationToken
authToken =
                    new
UsernamePasswordAuthenticationToken(
                        userDetails,
                        null,
                        userDetails.getAuthorities()
                    );
                authToken.setDetails(
                    new
WebAuthenticationDetailsSource().buildDetails(request)

package com.svcrm.service;

import com.svcrm.dto.DealRequest;

import com.svcrm.dto.DealResponse;

import com.svcrm.dto.DealStatusRequest;

import
com.svcrm.exception.ResourceNotFoundException;

import com.svcrm.model.Client;

import com.svcrm.model.Company;

import com.svcrm.model.Deal;

import com.svcrm.model.DealStatus;

import com.svcrm.model.User;

import com.svcrm.repository.ClientRepository;

import com.svcrm.repository.DealRepository;

```

```

    );
    SecurityContextHolder.getContext().setAuth
entication(authToken);

        log.debug("Користувач {}
аутентифікований", userEmail);
    }
} catch (Exception e) {
    log.error("Помилка JWT аутентифікації: {}",
e.getMessage());
}
filterChain.doFilter(request, response);
}
private String extractBearerToken(HttpServletRequest
request) {
    String authHeader =
request.getHeader("Authorization");
    if (StringUtils.hasText(authHeader) &&
authHeader.startsWith("Bearer ")) {
        return authHeader.substring(7);
    }
    return null;
}
}

import
com.svcrm.repository.InteractionRepository;

import com.svcrm.repository.UserRepository;

import lombok.RequiredArgsConstructor;

import lombok.extern.slf4j.Slf4j;

import org.springframework.data.domain.Page;

import
org.springframework.data.domain.Pageable;

import
org.springframework.security.core.context.SecurityCont
extHolder;

import org.springframework.stereotype.Service;

import
org.springframework.transaction.annotation.Transactiona
l;

import java.util.List;

import java.util.UUID;

@Service
@RequiredArgsConstructor
@Slf4j

```

```

public class DealService {

    private final DealRepository dealRepository;

    private final ClientRepository clientRepository;

    private final InteractionRepository interactionRepository;

    private final UserRepository userRepository;

    private final SlackNotificationService slackNotificationService;

    private static final int SCORE_BASE = 20;

    private static final int SCORE_IN_PROGRESS = 20;

    private static final int SCORE_PER_WON_DEAL = 15;

    private static final int SCORE_WON_DEALS_MAX = 45;

    private static final int SCORE_PER_INTERACTION = 5;

    private static final int SCORE_INTERACTIONS_MAX = 20;

    private static final int SCORE_MAX = 99;

    @Transactional(readOnly = true)

    public Page<DealResponse> getAll(String search, Pageable pageable) {

        UUID companyId = getCurrentCompanyId();

        Page<Deal> deals;

        if (search != null && !search.isBlank()) {

            deals = dealRepository.findByCompanyIdAndTitleContainingIgnoreCase(

                companyId, search, pageable);

        } else {

            deals = dealRepository.findByCompanyId(companyId, pageable);

            return deals.map(deal ->

                DealResponse.fromEntity(deal,

                    calculateWinProbability(deal.getClient(),

                        deal.getStatus()))

            );

        }

        @Transactional(readOnly = true)

        public DealResponse getById(UUID id) {

            Deal deal = findDealOrThrow(id);

            return DealResponse.fromEntity(deal,

                calculateWinProbability(deal.getClient(),

                    deal.getStatus()));

        }

        @Transactional(readOnly = true)

        public List<DealResponse>

            getByClientId(UUID clientId) {

            if (!clientRepository.existsById(clientId)) {

                throw new ResourceNotFoundException("Клієнта з ID " + clientId

                    + " не знайдено");

            }

            return dealRepository.findByClientId(clientId).stream()

                .map(deal -> DealResponse.fromEntity(

                    deal,

                    calculateWinProbability(deal.getClient(),

                        deal.getStatus())))

                .toList();

        }

        @Transactional(readOnly = true)

        public List<DealResponse>

            getByStatus(DealStatus status) {

            UUID companyId = getCurrentCompanyId();

            return dealRepository.findByCompanyIdAndStatus(companyId

                , status).stream()

```

```

        .map(deal -> DealResponse.fromEntity(
            deal,
            calculateWinProbability(deal.getClient(),
            deal.getStatus()))
        .toList());
    }
    @Transactional
    public DealResponse create(DealRequest
    request) {
        Client client =
        findClientOrThrow(request.clientId());
        User creator = getCurrentUser();
        Company company =
        creator.getCompany();

        Deal deal = request.toEntity(client);
        deal.setCompany(company);

        Deal saved = dealRepository.save(deal);

        log.info("Створено угоду: '{}' для клієнта
        '{}', компанія: '{}'",
            saved.getTitle(), client.getName(),
            company.getName());

        if (creator.getSlackWebhookUrl() != null
        && !creator.getSlackWebhookUrl().isBlank()) {
            String message =
            SlackNotificationService.buildNewDealMessage(
                saved.getTitle(), saved.getAmount());

            slackNotificationService.sendMessage(c
            reator.getSlackWebhookUrl(), message);
        }

        return DealResponse.fromEntity(saved,
        calculateWinProbability(client, saved.getStatus()));
    }
    @Transactional
    public DealResponse update(UUID id,
    DealRequest request) {

```

```

        Deal deal = findDealOrThrow(id);
        Client client =
        findClientOrThrow(request.clientId());
        request.updateEntity(deal, client);
        Deal updated = dealRepository.save(deal);
        log.info("Оновлено угоду: '{}' (ID: {})",
        updated.getTitle(), id);

        return DealResponse.fromEntity(updated,
        calculateWinProbability(client, updated.getStatus()));
    }
    @Transactional
    public DealResponse changeStatus(UUID id,
    DealStatusRequest request) {
        Deal deal = findDealOrThrow(id);
        DealStatus oldStatus = deal.getStatus();
        deal.setStatus(request.status());
        Deal updated = dealRepository.save(deal);
        log.info("Статус угоди '{}' змінено: {} →
        {}", deal.getTitle(), oldStatus, request.status());
        return DealResponse.fromEntity(updated,
            calculateWinProbability(updated.getCl
            ient(), updated.getStatus()));
    }
    @Transactional
    public void delete(UUID id) {
        Deal deal = findDealOrThrow(id);
        dealRepository.delete(deal);
        log.info("Видалено угоду: '{}' (ID: {})",
        deal.getTitle(), id);
    }

    private int calculateWinProbability(Client
    client, DealStatus currentStatus) {
        int score = SCORE_BASE;

        if (currentStatus ==
        DealStatus.IN_PROGRESS) {
            score += SCORE_IN_PROGRESS;

```

```

    }

    long wonDeals =
dealRepository.countByClientIdAndStatus(client.getId(),
DealStatus.WON);

    score += (int) Math.min(wonDeals *
SCORE_PER_WON_DEAL,
SCORE_WON_DEALS_MAX);

    long interactions =
interactionRepository.countByClientId(client.getId());

    score += (int) Math.min(interactions *
SCORE_PER_INTERACTION,
SCORE_INTERACTIONS_MAX);

    return Math.min(score, SCORE_MAX);
}

private UUID getCurrentCompanyId() {
    return getCurrentCompany().getId();
}

private Company getCurrentCompany() {
    return getCurrentUser().getCompany();
}

private User getCurrentUser() {
    String email =
SecurityContextHolder.getContext()
        .getAuthentication()
        .getName();

    return
userRepository.findByEmailIgnoreCase(email)
        .orElseThrow(() -> new
RuntimeException("Поточного користувача не
знайдено: " + email));
}

private Deal findDealOrThrow(UUID id) {
    return dealRepository.findById(id)
        .orElseThrow(() -> new
ResourceNotFoundException(

```

```

        "Угоди з ID " + id + " не знайдено"
    ));
}

private Client findClientOrThrow(UUID
clientId) {
    return clientRepository.findById(clientId)
        .orElseThrow(() -> new
ResourceNotFoundException(
        "Клієнта з ID " + clientId + " не
знайдено"
    ));
}
}

package com.svcrm.service;

import com.svcrm.dto.AuthRequest;
import com.svcrm.dto.AuthResponse;
import com.svcrm.dto.RegisterRequest;
import com.svcrm.model.Company;
import com.svcrm.model.User;
import com.svcrm.model.UserRole;

import
com.svcrm.repository.CompanyRepository;

import com.svcrm.repository.UserRepository;
import com.svcrm.security.JwtService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;

import
org.springframework.security.authentication.Authenticati
onManager;

import
org.springframework.security.authentication.BadCredent
ialsException;

import
org.springframework.security.authentication.UsernameP
asswordAuthenticationToken;

```

```

import
org.springframework.security.core.AuthenticationExcept
ion;

import
org.springframework.security.crypto.password.Password
Encoder;

import org.springframework.stereotype.Service;

import
org.springframework.transaction.annotation.Transactiona
l;

import java.time.Instant;

@Service
@RequiredArgsConstructor
@Slf4j

public class AuthService {
    private final AuthenticationManager
authenticationManager;

    private final JwtService jwtService;

    private final
UserRepository userRepository;

    private final
CompanyRepository companyRepository;

    private final
PasswordEncoder passwordEncoder;

    @Transactional

    public AuthResponse login(AuthRequest
request) {
        try {
            authenticationManager.authenticate(
                new
UsernamePasswordAuthenticationToken(
                    request.email(),
                    request.password()
                )
            );
        } catch (AuthenticationException e) {
            log.warn("Невдала спроба входу для
email: {}", request.email());

```

```

throw new
BadCredentialsException("Невірний
email
або
пароль");
        }

        User user =
userRepository.findByEmailIgnoreCase(request.email())
            .orElseThrow(() -> new
BadCredentialsException("Невірний
email
або
пароль"));

        String accessToken =
jwtService.generateAccessToken(user);

        String refreshToken =
jwtService.generateRefreshToken(user);

        userRepository.updateLastLoginAt(user.ge
tId(), Instant.now());

        log.info("Користувач {} успішно увійшов
у систему", user.getEmail());

        String companyName = user.getCompany()
!= null
            ? user.getCompany().getName()
            : "";

        return new AuthResponse(
            accessToken,
            refreshToken,
            user.getEmail(),
            user.getRole().name(),
            user.getFirstName(),
            companyName
        );
    }

    @Transactional

    public AuthResponse
register(RegisterRequest request) {
        if
(userRepository.existsByEmailIgnoreCase(request.email
())) {

```

```

        log.warn("Спроба реєстрації з вже
існуючим email: {}", request.email());

        throw new IllegalArgumentException(
            "Користувач з таким email вже
існує: " + request.email()

        );
    }

    Company company = Company.builder()
        .name(request.companyName().trim())
        .build();

    Company savedCompany =
companyRepository.save(company);

    log.info("Створено нову компанію-
тенанта: '{}' (ID: {})",
        savedCompany.getName(),
savedCompany.getId());

    String[] parts =
request.name().trim().split("\\s+", 2);

    String firstName = parts[0];

    String lastName = parts.length > 1 ? parts[1]
: "";

    User newUser = User.builder()
        .email(request.email())
        .passwordHash(passwordEncoder.encode(request.password()))
        .firstName(firstName)
        .lastName(lastName)
        .role(UserRole.ADMIN)
        .isActive(true)
        .company(savedCompany)
        .build();

    User savedUser =
userRepository.save(newUser);

    log.info("Зареєстровано нового
користувача: {} ({{}}) — компанія: '{}'",
        savedUser.getEmail(),
savedUser.getRole(), savedCompany.getName());

```

```

        String accessToken =
jwtService.generateAccessToken(savedUser);

        String refreshToken =
jwtService.generateRefreshToken(savedUser);

        return new AuthResponse(
            accessToken,
            refreshToken,
            savedUser.getEmail(),
            savedUser.getRole().name(),
            savedUser.getFirstName(),
            savedCompany.getName()
        );
    }
}

#
=====
# SV-CRM Backend — Налаштування додатку
#
=====

spring:
    application:
        name: sv-crm-backend

# === Підключення до PostgreSQL ===
datasource:
    url: jdbc:postgresql://localhost:5432/svcrm
    username: svcrm_user
    password: svcrm_secret_2024
    driver-class-name: org.postgresql.Driver

# === JPA / Hibernate ===

```

```

jpa:
  hibernate:
    ddl-auto: validate      # Flyway керує
    #схемою, Hibernate тільки перевіряє

    show-sql: true          # Показувати SQL-
    #запити в консолі (для розробки)

    properties:
      hibernate:
        dialect:
          org.hibernate.dialect.PostgreSQLDialect

        format_sql: true    # Форматувати SQL
        #для зручного читання

    # === Flyway — автоматичні міграції БД
    ===

    flyway:
      enabled: true

      locations: classpath:db/migration

      baseline-on-migrate: true

    # === Spring AI — Groq (Альтернатива
    # Gemini) ===

    ai:
      openai:
        api-key:
          "gsk_vqWfq06NWC6kNdkDPnehWGdyb3FYWbgm7b
          Vps9ei2JGPCNrpyHBR"

        base-url: "https://api.groq.com/openai"

        chat:
          options:
            model: "llama-3.3-70b-versatile"

    # === Email — SMTP (JavaMailSender) ===

    # Для дипломного проєкту рекомендуємо
    # використовувати Gmail + App Password:

    # Налаштування: Google Account → Security
    # → 2-Step → App Passwords

    mail:

```

```

      host: smtp.gmail.com

      port: 587

      username: your-email@gmail.com    # ←
      #замінити на реальний Gmail

      password: your-app-password      # ← App
      #Password (НЕ звичайний пароль!)

      properties:
        mail:
          smtp:
            auth: true

            starttls:
              enable: true    # TLS шифрування (порт
              #587)

              required: true

              debug: false    # true — показувати
              #SMTP-діалог (тільки для дебагу)

    # === Власні налаштування Email-сервісу
    ===

    mail:
      from-name: "SmartBiz CRM"        # Ім'я
      #відправника в листі

    # === Налаштування сервера ===

    server:
      port: 8080

      servlet:
        context-path: /api

    # === JWT налаштування ===

    jwt:
      secret: sv-crm-super-secret-jwt-signing-key-
      #must-be-at-least-256-bits-long-2024

      expiration-ms: 86400000          # 24 години (час
      #життя access-токена)

      refresh-expiration-ms: 604800000 # 7 днів (час
      #життя refresh-токена)

```

```

# === CORS — дозволені джерела (фронтенд)
===

cors:

  allowed-origins: "http://localhost:5173"

# === Документація API (Swagger) ===
springdoc:

  api-docs:

    path: /v3/api-docs

  swagger-ui:

    path: /swagger-ui

# === Логування ===
logging:

  level:

    root: INFO

    com.svcrm: DEBUG

    org.hibernate.SQL: DEBUG

import { useState, useEffect } from 'react'
import api from '../api/axios'

function WinProbabilityBadge({ probability,
status }) {

  if (status === 'WON' || status === 'LOST')
return null

  if (probability == null) return null

  let colorClasses

  if (probability >= 75) {

    colorClasses = 'bg-green-100 text-green-800
border-green-200'

  } else if (probability >= 40) {

    colorClasses = 'bg-yellow-100 text-yellow-
800 border-yellow-200'

  } else {

```

```

    colorClasses = 'bg-red-100 text-red-800
border-red-200'

  }

  return (

    <span

      className={`inline-flex items-center gap-1
px-2 py-0.5 rounded-full text-xs font-semibold border
${colorClasses}`}

      title="Ймовірність успішного закриття
угоди (алгоритмічний скоринг)"

    >

       {probability}%

    </span>

  )

}

const COLUMNS = [

  { status: 'NEW', title: 'Нові', color:
'blue' },

  { status: 'IN_PROGRESS', title: 'В
роботі', color: 'yellow' },

  { status: 'WON', title: 'Успішно', color:
'green' },

  { status: 'LOST', title: 'Відмова', color:
'red' },

]

const COLUMN_STYLES = {

  blue: { bg: 'bg-blue-50', border: 'border-blue-
200', badge: 'bg-blue-100 text-blue-700' },

  yellow: { bg: 'bg-amber-50', border: 'border-
amber-200', badge: 'bg-amber-100 text-amber-700' },

  green: { bg: 'bg-green-50', border: 'border-
green-200', badge: 'bg-green-100 text-green-700' },

  red: { bg: 'bg-red-50', border: 'border-red-
200', badge: 'bg-red-100 text-red-700' },

}

export default function DealsPage() {

  const [deals, setDeals] = useState([])

```

```

const [clients, setClients] = useState([])
const [loading, setLoading] = useState(true)
const [showForm, setShowForm] =
useState(false)
const [formData, setFormData] = useState({
  title: "", amount: "", clientId: "", status: 'NEW'
})
const [formError, setFormError] = useState("")
const [saving, setSaving] = useState(false)
useEffect(() => {
  fetchDeals()
  fetchClients()
}, [])
const fetchDeals = async () => {
  setLoading(true)
  try {
    const response = await api.get('/v1/deals', {
      params: { size: 100 } })
    setDeals(response.data.content)
  } catch (err) {
    console.error('Помилка завантаження
угод:', err)
  } finally {
    setLoading(false)
  }
}
const fetchClients = async () => {
  try {
    const response = await api.get('/v1/clients', {
      params: { size: 100 } })
    setClients(response.data.content)
  } catch (err) {
    console.error('Помилка завантаження
клієнтів:', err)
  }
}

```

```

const handleStatusChange = async (dealId,
newStatus) => {
  try {
    await api.patch(`/v1/deals/${dealId}/status`,
{ status: newStatus })
    setDeals(prev =>
      prev.map(deal =>
        deal.id === dealId ? { ...deal, status:
newStatus } : deal
      )
    )
  } catch (err) {
    alert('Помилка зміни статусу: ' +
(err.response?.data?.message || 'Невідома помилка'))
  }
}
const handleChange = (e) => {
  setFormData({ ...formData, [e.target.name]:
e.target.value })
}
const handleSubmit = async (e) => {
  e.preventDefault()
  setFormError("")
  setSaving(true)
  try {
    const payload = {
      ...formData,
      amount: formData.amount ?
parseFloat(formData.amount) : null,
    }
    await api.post('/v1/deals', payload)
    setFormData({ title: "", amount: "", clientId: "",
status: 'NEW' })
    setShowForm(false)
    fetchDeals()
  } catch (err) {

```

```

    const msg = err.response?.data?.message ||
err.response?.data?.errors?.title || 'Помилка створення'

    setFormError(msg)
  } finally {
    setSaving(false)
  }
}

const handleDelete = async (dealId, title) => {
  if (!window.confirm(`Видалити угоду
"${title}"?`)) return

  try {
    await api.delete(`/v1/deals/${dealId}`)
    setDeals(prev => prev.filter(d => d.id !==
dealId))
  } catch (err) {
    alert('Помилка видалення')
  }
}

const getDealsByStatus = (status) =>
deals.filter(d => d.status === status)

const formatAmount = (amount) => {
  if (!amount) return '—'

  return new Intl.NumberFormat('uk-UA', {
    style: 'currency', currency: 'UAH',
maximumFractionDigits: 0
  }).format(amount)
}

if (loading) {
  return <div className="text-center text-gray-
500 py-12">Завантаження угод...</div>
}

return (
  <div>
    {/* Заголовок */}

```

```

    <div className="flex items-center justify-
between mb-6">
      <h1 className="text-2xl font-bold text-
gray-800">Воронка продажів</h1>
      <button
        onClick={() =>
          setShowForm(!showForm)}
        className="px-4 py-2 bg-blue-600 text-
white rounded-lg
        hover:bg-blue-700 transition-colors
text-sm font-medium"
      >
        {showForm ? '× Закрити' : '+ Нова
угода'}
      </button>
    </div>

    {/* Форма додавання угоди */}
    {showForm && (
      <div className="bg-white rounded-xl
shadow-sm border border-gray-200 p-6 mb-6">
        <h3 className="text-lg font-semibold
text-gray-800 mb-4">Нова угода</h3>

        {formError && (
          <div className="mb-4 p-3 bg-red-50
border border-red-200 rounded-lg text-red-700 text-sm">
            {formError}
          </div>
        )}

        <form onSubmit={handleSubmit}
className="grid grid-cols-1 md:grid-cols-2 gap-4">
          <div>
            <label className="block text-sm font-
medium text-gray-700 mb-1">Назва угоди </label>
            <input

```

```

        name="title" value={formData.title}
onChange={handleChange} required

        placeholder="Розробка сайту"

        className="w-full px-3 py-2 border
border-gray-300 rounded-lg

        focus:ring-2 focus:ring-blue-500
focus:border-transparent outline-none"

    />
</div>
<div>
    <label className="block text-sm font-
medium text-gray-700 mb-1">Сума (грн)</label>

    <input

        name="amount" type="number"
value={formData.amount} onChange={handleChange}

        placeholder="50000"

        className="w-full px-3 py-2 border
border-gray-300 rounded-lg

        focus:ring-2 focus:ring-blue-500
focus:border-transparent outline-none"

    />
</div>
<div>
    <label className="block text-sm font-
medium text-gray-700 mb-1">Клієнт *</label>

    <select

        name="clientId"
value={formData.clientId} onChange={handleChange}
required

        className="w-full px-3 py-2 border
border-gray-300 rounded-lg

        focus:ring-2 focus:ring-blue-500
focus:border-transparent outline-none"

    >

        <option value="">Оберіть
клієнта...</option>

        {clients.map(c => (
            <option key={c.id}
value={c.id}>{c.name}</option>

```

```

    )))
</select>
</div>

<div className="flex items-end justify-
end gap-2">

    <button

        type="button" onClick={() =>
setShowForm(false)}

        className="px-4 py-2 text-sm text-
gray-600 hover:bg-gray-100 rounded-lg transition-
colors"

    >

        Скасувати

    </button>

    <button

        type="submit" disabled={saving}

        className="px-4 py-2 bg-blue-600
text-white rounded-lg text-sm font-medium

        hover:bg-blue-700 transition-
colors disabled:opacity-50"

    >

        {saving ? 'Збереження...' : 'Створити'}

    </button>
</div>
</form>
</div>
))

    { /* Канбан-дошка — 4 колонки */ }

    <div className="grid grid-cols-1 md:grid-
cols-4 gap-4">

        { COLUMNS.map((col) => {

            const columnDeals =
getDealsByStatus(col.status)

            const styles =
COLUMN_STYLES[col.color]

```

```

return (
    <div key={col.status}
      className={`rounded-xl ${styles.bg} ${styles.border}
        border p-4`} >
      { /* Заголовок колонки */ }
      <div className="flex items-center
        justify-between mb-4">
        <h3 className="font-semibold text-
          gray-700">{col.title}</h3>
        <span className={`px-2 py-0.5 text-
          xs rounded-full font-medium ${styles.badge}`} >
          {columnDeals.length}
        </span>
      </div>

      { /* Картки угод */ }
      <div className="space-y-3">
        {columnDeals.length === 0 ? (
          <p className="text-sm text-gray-400
            text-center py-4">Немає угод</p>
        ) : (
          columnDeals.map((deal) => (
            <div
              key={deal.id}
              className="bg-white rounded-lg
                shadow-sm border border-gray-200 p-4"
            >
              { /* Назва угоди */ }
              <h4 className="font-medium text-
                gray-800 text-sm mb-2">
                {deal.title}
              </h4>

              { /* Сума */ }
              <p className="text-lg font-bold
                text-gray-900 mb-1">
                {formatAmount(deal.amount)}

```

```

</p>

      { /* Ім'я клієнта */ }
      <p className="text-xs text-gray-
        500 mb-2">
        <img alt="user icon" data-bbox="728 198 742 212"/> {deal.clientName}
      </p>

      { /* Бейдж предиктивного
        скорингу ймовірності */ }
      <div className="mb-3">
        <WinProbabilityBadge
          probability={deal.winProbabilit
            y}
          status={deal.status}
        />
      </div>

      { /* Зміна статусу через
        випадючий список */ }
      <select
        value={deal.status}
        onChange={(e) =>
          handleStatusChange(deal.id, e.target.value)}
        className="w-full px-2 py-1 text-
          xs border border-gray-200 rounded
            focus:ring-1 focus:ring-
              blue-500 outline-none bg-gray-50"
      >
        {COLUMNS.map(c => (
          <option key={c.status}
            value={c.status}>{c.title}</option>
        ))}
      </select>

      { /* Кнопка видалення */ }
      <button

```

```

                                onClick={() =>
handleDelete(deal.id, deal.title)}

                                className="mt-2 text-xs text-red-
400 hover:text-red-600 transition-colors"

                                >
                                Видалити
                                </button>
                                </div>
                                ))
                                }}
                                </div>
                                </div>
                                )
                                }}}
                                </div>
                                </div>
                                )
                                }

package com.svcrm.controller;

import com.svcrm.dto.DealRequest;
import com.svcrm.dto.DealResponse;
import com.svcrm.dto.DealStatusRequest;
import com.svcrm.model.DealStatus;
import com.svcrm.service.DealService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import
org.springframework.data.domain.Pageable;

import
org.springframework.data.web.PageableDefault;
import org.springframework.http.HttpStatus;

```

```

import
org.springframework.http.ResponseEntity;

import
org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.UUID;

/**
 * REST-контролер для управління угодами
(Deals / Sales Pipeline).
 *
 * Базовий URL: /api/v1/deals
 *
 * Підтримує:
 * - CRUD операції
 * - Фільтрацію по клієнту та статусу (для
Канбан-дошки)
 * - Швидку зміну статусу (PATCH) —
"перетягування" картки
 */
@RestController
@RequestMapping("/v1/deals")
@RequiredArgsConstructor
@Tag(name = "Угоди", description = "API для
управління угодами та воронкою продажів")
public class DealController {

    private final DealService dealService;

    /**
 * Отримати список угод з пагінацією.
 *
 * Параметри запити:
 * ?search=Сайт — пошук по назві
 * ?page=0&size=10 — пагінація

```

```

* ?sort=createdAt,desc — сортування
*/

@GetMapping
@Operation(
    summary = "Отримати список угод",
    description = "Повертає пагінований
список угод з можливістю пошуку по назві"
)

public
ResponseEntity<Page<DealResponse>> getAll(
    @RequestParam(required = false) String
search,
    @PageableDefault(size = 10, sort =
"createdAt") Pageable pageable
) {
    return
ResponseEntity.ok(dealService.getAll(search,
pageable));
}

/**
 * Отримати угоду по ID.
 */
@GetMapping("/{id}")
@Operation(
    summary = "Отримати угоду по ID",
    description = "Повертає дані угоди разом
з інформацією про клієнта"
)

public ResponseEntity<DealResponse>
getById(@PathVariable UUID id) {
    return
ResponseEntity.ok(dealService.getById(id));
}

/**
 * Отримати всі угоди конкретного клієнта.

```

```

*/

@GetMapping("/client/{clientId}")
@Operation(
    summary = "Угоди клієнта",
    description = "Повертає список всіх угод
конкретного клієнта"
)

public ResponseEntity<List<DealResponse>>
getClient(@PathVariable UUID clientId) {
    return
ResponseEntity.ok(dealService.getClientId(clientId));
}

/**
 * Отримати угоди по статусу (для Канбан-
дошки).
 *
 * Приклад: GET /api/v1/deals/status/NEW
→ всі нові угоди
 */
@GetMapping("/status/{status}")
@Operation(
    summary = "Угоди по статусу",
    description = "Повертає угоди для
відмалювання колонки Канбан-дошки (NEW,
IN_PROGRESS, WON, LOST)"
)

public ResponseEntity<List<DealResponse>>
getStatus(@PathVariable DealStatus status) {
    return
ResponseEntity.ok(dealService.getStatus(status));
}

/**
 * Створити нову угоду.
 */
@PostMapping

```

```

@Operation(
    summary = "Створити угоду",
    description = "Створює нову угоду та
    прив'язує до вказаного клієнта"
)

    public ResponseEntity<DealResponse>
create(@Valid @RequestBody DealRequest request) {
        DealResponse created =
dealService.create(request);

        return
ResponseEntity.status(HttpStatus.CREATED).body(crea
ted);
    }

/**
 * Оновити дані угоди.
 */
@PutMapping("/{id}")
@Operation(
    summary = "Оновити угоду",
    description = "Оновлює всі поля угоди за
вказаним ID"
)
    public ResponseEntity<DealResponse>
update(
        @PathVariable UUID id,
        @Valid @RequestBody DealRequest
request
    ) {
        return
ResponseEntity.ok(dealService.update(id, request));
    }

/**
 * Змінити статус (етап воронки) угоди.
 *
 * Це окремий PATCH-ендпоінт для
швидкої зміни статусу —

```

```

        * імітація "перетягування" картки на
Канбан-дошці.

        * Не потрібно відправляти всі поля угоди,
лише новий статус.
    *
        * Приклад: PATCH
/api/v1/deals/{id}/status body: {"status":
"IN_PROGRESS"}
    */
    @PatchMapping("/{id}/status")
    @Operation(
        summary = "Змінити статус угоди",
        description = "Змінює етап воронки
(перетягування картки між колонками Канбан-
дошки)"
    )
    public ResponseEntity<DealResponse>
changeStatus(
        @PathVariable UUID id,
        @Valid @RequestBody
DealStatusRequest request
    ) {
        return
ResponseEntity.ok(dealService.changeStatus(id,
request));
    }

/**
 * Видалити угоду.
 */
    @DeleteMapping("/{id}")
    @Operation(

```

```

        summary = "Видалити угоду",
        description = "Видаляє угоду за вказаним
ID"
    )

    public ResponseEntity<Void>
delete(@PathVariable UUID id) {
        dealService.delete(id);

        return ResponseEntity.noContent().build();
    }
}

```