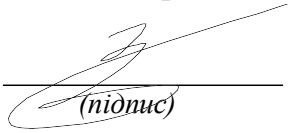
**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ
ТЕХНОЛОГІЙ НАВЧАЛЬНО-НАУКОВИЙ
ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інформаційна система підприємства на основі об'єктно-орієнтованих баз даних»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерна інженерія
освітньо-професійної програми «Комп'ютерна інженерія»

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



(підпис) Катерина ЗАКРУЖЕЦЬКА

Виконала: здобувачка вищої освіти групи КІД-41
Катерина ЗАКРУЖЕЦЬКА

Керівник: Наталія ЛАЩЕВСЬКА

Рецензент: Володимир САДОВЕНКО
к.ф.-м.н., доцент *Ім'я, ПРІЗВИЩЕ*

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерна інженерія

Ступінь вищої освіти Бакалавра

Спеціальність 123 Комп'ютерна інженерія

Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____Наталія ЛАЩЕВСЬКА
«_____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Закружецька Катерина Олександрівна

1. Тема кваліфікаційної роботи: «Інформаційна система підприємства на основі об'єктно-орієнтованих баз даних»

керівник кваліфікаційної роботи Наталія ЛАЩЕВСЬКА,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.
2. Строк подання кваліфікаційної роботи «05» червня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з інформаційних систем і баз даних, стандарти моделювання інформаційних систем, методики проєктування та тестування програмного забезпечення, приклади реалізації бізнес-логіки у системах управління даними.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз сучасного стану розвитку інформаційних систем з об'єктно-орієнтованими базами даних.
2. Порівняння архітектур об'єктно-орієнтованих і реляційних БД.
3. Огляд засобів розробки на базі об'єктно-орієнтованої моделі даних.

4.Узагальнення результатів і рекомендації щодо впровадження.

1. Перелік ілюстративного матеріалу: *презентація*

Актуальність переходу на об’єктно-орієнтовані бази даних у бізнес-середовищі.
Схема архітектури інформаційної системи підприємства з використанням об’єктно-орієнтованої моделі.
Порівняння реляційної та об’єктно-орієнтованої баз даних (таблиця або діаграма).
Візуалізація структури класів і зв’язків у базі даних.
Інтерфейс користувача CRM-системи (скріншоти прототипу).
Результати тестування продуктивності та ефективності системи.
Висновки щодо доцільності впровадження об’єктно-орієнтованих БД в інформаційні системи підприємств.

Дата видачі завдання «16» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	**Назва етапів кваліфікаційної роботи**	**Строк виконання етапів роботи**	**Примітка**
1	Аналіз науково-технічної літератури з питань інформаційних систем та об’єктно-орієнтованих баз даних	16.02–23.02.2025	
2	Вивчення архітектур та інструментів реалізації об’єктно-орієнтованих баз даних	23.02–02.03.2025	
3	Аналіз особливостей використання об’єктно-орієнтованого підходу в CRM-системах	02.03–09.03.2025	
4	Проєктування структури інформаційної системи підприємства	09.03–19.03.2025	
5	Реалізація прототипу інформаційної системи та його тестування	19.03–29.03.2025	
6	Оформлення пояснювальної записки, висновки, рекомендації	29.03–30.03.2025	
7	Підготовка презентації та демонстраційних матеріалів	30.04–31.04.2025	
8	Попередній захист роботи	28.05.2025	

Здобувачка вищої освіти _____ Катерина ЗАКРУЖЕЦЬКА
(підпис)

Керівник
кваліфікаційної роботи _____ Наталія ЛАЩЕВСЬКА
(підпис)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 80 сторінок, 29 рисунків, 30 джерел.

Об'єкт дослідження – інформаційні системи підприємств, що використовують об'єктно-орієнтовані бази даних.

Предмет дослідження – архітектура, інструменти та засоби реалізації об'єктно-орієнтованих баз даних у складі CRM-систем.

Мета роботи – розробити інформаційну систему підприємства на базі об'єктно-орієнтованої бази даних, яка забезпечить зручне управління клієнтськими даними, масштабованість і безпечну обробку запитів, а також порівняти її ефективність із традиційними рішеннями.

Методи дослідження:

1. Теоретичний аналіз ролі інформаційних систем у сучасному бізнесі, вивчення переваг об'єктно-орієнтованого підходу.
2. Порівняльний аналіз реляційних і об'єктно-орієнтованих баз даних.
3. Аналіз вимог до CRM-систем та формування структури системи управління клієнтами.
4. Проєктування та реалізація прототипу системи з використанням Java та об'єктної моделі даних.
5. Тестування функціоналу, оцінка зручності та ефективності розробленої системи.

Галузь використання – цифрова трансформація бізнесу, автоматизація CRM, управління взаємовідносинами з клієнтами, системи електронного документообігу.

ІНФОРМАЦІЙНА СИСТЕМА, CRM, ОБ'ЄКТНО-ОРІЄНТОВАНА БАЗА ДАНИХ, JAVA, МОДЕЛЮВАННЯ, УПРАВЛІННЯ ДАНИМИ, БІЗНЕС-ПРОЦЕСИ, АРХІТЕКТУРА, МАСШТАБОВАНІСТЬ, БЕЗПЕКА.

ABSTRACT

Text part of the bachelor's thesis: 80 pages, 29 figures, 30 references.

Object of research – information systems of enterprises that utilize object-oriented databases.

Subject of research – the architecture, tools, and implementation methods of object-oriented databases within CRM systems.

The aim of the work – to develop an enterprise information system based on an object-oriented database that ensures convenient client data management, scalability, and secure request processing, as well as to compare its efficiency with traditional solutions.

Research methods:

1. Theoretical analysis of the role of information systems in modern business, study of the advantages of the object-oriented approach.
2. Comparative analysis of relational and object-oriented databases.
3. Analysis of CRM system requirements and formation of a client management system structure.
4. Design and implementation of a system prototype using Java and an object data model.
5. Functionality testing, usability evaluation, and efficiency assessment of the developed system.

Field of application – digital business transformation, CRM automation, customer relationship management, electronic document management systems.

INFORMATION SYSTEM, CRM, OBJECT-ORIENTED DATABASE, JAVA, MODELING, DATA MANAGEMENT, BUSINESS PROCESSES, ARCHITECTURE, SCALABILITY, SECURITY.

ЗМІСТ

ВСТУП 8
1 ТЕОРЕТИЧНІ ОСНОВИ ІНФОРМАЦІЙНИХ СИСТЕМ І БАЗ ДАНИХ 10
1.1 Інформаційні системи в бізнес-середовищі 10
1.2 Порівняння реляційних та не реляційних баз даних 12
1.3 Особливості зберігання даних у різних типах СУБД 17
2 АНАЛІЗ ОБ'ЄКТНО-ОРІЄНТОВАНИХ ПІДХОДІВ ДО СТВОРЕННЯ БД 20
2.1 Об'єктно-орієнтовані середовища розробки 20
2.2 Об'єктно-орієнтовані бази даних: MongoDB, Oracle NoSQL 24
2.3 Створення, управління та моделювання документно-орієнтованих БД 28
3 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПІДПРИЄМСТВА 35
3.1 Вибір технологій для реалізації CRM-системи 35
3.2 Архітектура, інтерфейс і підтримка різних пристроїв 46
3.3 Реалізація та тестування функціоналу системи 56
ВИСНОВКИ 60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 62
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ) 66

ВСТУП

У сучасних умовах цифрової трансформації бізнесу ефективне управління інформаційними потоками стало критично важливим для підприємств. Традиційні реляційні бази даних, що домінували впродовж десятиліть, дедалі частіше виявляють обмеження у гнучкості, масштабованості та підтримці складних структур даних, що актуалізує потребу у використанні нових підходів до зберігання й обробки інформації.

Об’єктно-орієнтовані бази даних (ООБД) пропонують сучасне рішення, яке дозволяє моделювати бізнес-логіку ближче до реального світу, забезпечуючи природну інтеграцію з об’єктно-орієнтованими мовами програмування, такими як Java. Це суттєво спрощує розробку, супровід і масштабування інформаційних систем, особливо таких, як CRM, ERP та інші корпоративні платформи.

З огляду на зростання обсягів даних, необхідність персоналізованого підходу до клієнтів і стрімке впровадження хмарних технологій, дослідження інформаційних систем на базі об’єктно-орієнтованих БД є надзвичайно актуальним. Такі системи дозволяють не лише оптимізувати внутрішні бізнес-процеси, але й забезпечити гнучке реагування на зміну ринкових умов.

Тема дослідження є важливою як з наукової, так і з прикладної точки зору, оскільки відповідає потребам сучасного бізнес-середовища у надійних, масштабованих та адаптивних інформаційних рішеннях.

Мета роботи – розробити інформаційну систему підприємства на базі об’єктно-орієнтованої бази даних, яка забезпечить зручне управління клієнтськими даними, масштабованість і безпечну обробку запитів, а також порівняти її ефективність із традиційними рішеннями.

Об’єкт дослідження – інформаційні системи підприємств, що використовують об’єктно-орієнтовані бази даних.

Предмет дослідження – архітектура, інструменти та засоби реалізації об’єктно-орієнтованих баз даних у складі CRM-систем.

Методологія дослідження включає аналіз та узагальнення науково-технічної

літератури з тематики інформаційних систем, баз даних, зокрема – об’єктно-орієнтованих СУБД. Особлива увага приділяється вивченню архітектур об’єктно-орієнтованих баз даних, їхнього застосування в CRM-системах, а також технологічних рішень, що дозволяють ефективно інтегрувати ці бази з об’єктно-орієнтованими мовами програмування (Java, C# та інші).

У процесі дослідження застосовуються методи системного аналізу, моделювання, а також порівняльного аналізу реляційного та об’єктно-орієнтованого підходів до зберігання та обробки даних. Проводиться дослідження ключових параметрів – зручності масштабування, відповідності моделі реальним бізнес-процесам, ефективності доступу до даних і безпеки.

Практична частина включає розробку прототипу інформаційної системи підприємства з використанням об’єктно-орієнтованої бази даних, зокрема реалізацію структури даних, інтерфейсу користувача, бізнес-логіки та API-взаємодії. Система проходить тестування на предмет стабільності, продуктивності та масштабованості. За результатами експериментів проводиться порівняльна оцінка з традиційними рішеннями на основі реляційних СУБД.

1 ТЕОРЕТИЧНІ ОСНОВИ ІНФОРМАЦІЙНИХ СИСТЕМ І БАЗ ДАНИХ

1.1 Інформаційні системи в бізнес-середовищі

У сучасних умовах цифрової трансформації підприємства стикаються з необхідністю швидко адаптуватися до змін ринку, підвищувати ефективність внутрішніх процесів та забезпечувати високий рівень обслуговування клієнтів. У цьому контексті інформаційні системи (ІС) відіграють ключову роль, забезпечуючи основу для автоматизації, аналізу й прийняття обґрунтованих управлінських рішень [1].

ІС дозволяють підприємствам автоматизувати рутинні та повторювані завдання, скорочуючи час на їх виконання та знижуючи ймовірність людських помилок. Це охоплює як внутрішні процеси (облік, управління персоналом, фінансове планування), так і зовнішні взаємодії з клієнтами, постачальниками й партнерами. Автоматизація сприяє зниженню витрат, підвищенню продуктивності праці, зменшенню витрат на обслуговування техніки, людських ресурсів та оперативності бізнес-операцій.

Сучасні інформаційні системи є основою функціонування спеціалізованих корпоративних платформ, зокрема:

CRM (Customer Relationship Management) – система управління взаємовідносинами з клієнтами. Забезпечує зберігання й аналіз клієнтських даних, ведення історії взаємодій, планування комунікацій і маркетингових кампаній;

SCM (Supply Chain Management) – підтримує управління ланцюгами постачання, дозволяючи оптимізувати логістику, запаси та взаємодію з постачальниками;

ERP (Enterprise Resource Planning) – інтегрує дані з усіх підрозділів компанії для централізованого планування ресурсів, фінансів, виробництва тощо [2].

Завдяки цим системам підприємства можуть ефективніше координувати свою діяльність, скорочувати витрати та покращувати якість обслуговування клієнтів.

Інформаційні системи формують основу для data-driven управління, дозволяючи керівникам приймати рішення, базуючись на актуальних та достовірних даних. Інтеграція з аналітичними платформами (BI-системами), інструментами візуалізації та прогнозування (наприклад, засобами машинного навчання) відкриває можливості для побудови сценаріїв розвитку, виявлення трендів і ризиків, а також персоналізації продуктів і послуг.

Завдяки швидкому доступу до узагальненої інформації управлінці можуть оперативно реагувати на зміни середовища, адаптувати стратегії та підвищувати конкурентоспроможність підприємства.

У 2020-х роках особливу популярність набули хмарні інформаційні системи, мобільні додатки для бізнесу, а також інтеграція ІС з Big Data і штучним інтелектом (AI). Згідно з аналітичними звітами Gartner і McKinsey, компанії, які інвестують в інтелектуальні ІС, демонструють на 20–30% вищу продуктивність у довгостроковій перспективі. Ці тенденції підкреслюють необхідність не лише впровадження ІС, а й адаптації до найсучасніших технологій, що забезпечують конкурентні переваги на ринку [3].

Інформаційні системи є фундаментальним елементом цифрової економіки, який забезпечує ефективне управління ресурсами, оптимізацію процесів та прийняття стратегічних рішень. На рис. 1.1 представлено ключові напрями використання інформаційних систем у сучасному бізнес-середовищі.

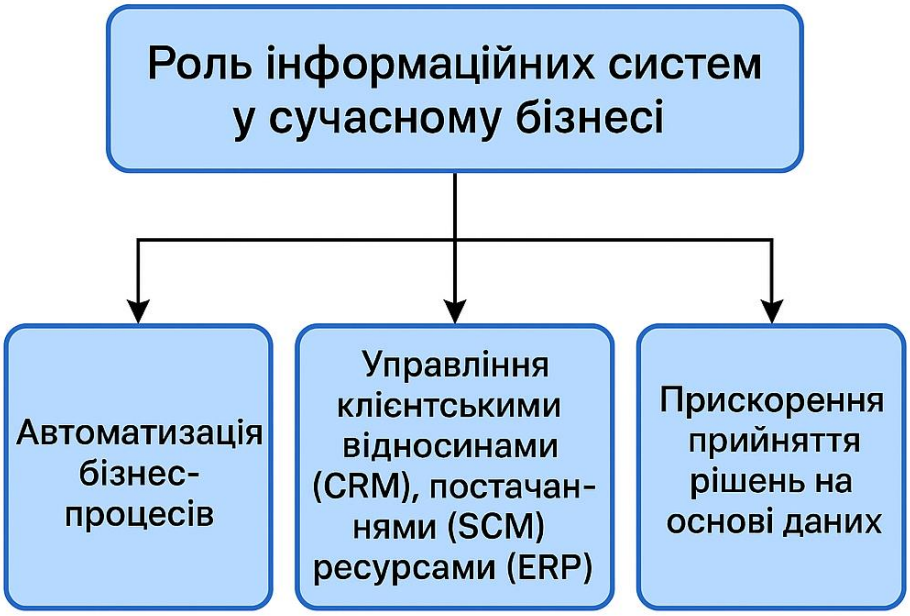


Рисунок 1.1 – Роль інформаційних систем у сучасному бізнесі

Успішна реалізація інформаційної системи неможлива без правильно обраної моделі бази даних, яка лежить в основі зберігання та обробки інформації. Саме тому доцільно розглянути різні типи баз даних, зокрема реляційні та об’єктно-орієнтовані підходи, що буде детально проаналізовано у наступному підрозділі.

1.2 Порівняння реляційних та не реляційних баз даних

Реляційні бази даних (Relational Databases, RDB) є традиційною моделлю зберігання інформації, яка базується на чітко структурованій схемі даних у вигляді таблиць. У таких базах кожен запис представлено рядком, а кожен атрибут – стовпцем таблиці. Всі таблиці пов’язані між собою за допомогою первинних та зовнішніх ключів, що дозволяє підтримувати цілісність даних. Реляційні СУБД широко застосовуються у сфері бухгалтерського обліку, управління персоналом, фінансових операцій, де важлива транзакційність та жорстке дотримання схеми даних (рис. 1.2). До найбільш відомих представників реляційних СУБД належать MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server [4].

Реляційна БД

student

id	name	surname	age
1	John	Brown	19
2	Emma	Carly	23

Рисунок 1.2 – Приклад реляційної бази даних

Нереляційні бази даних (NoSQL – Not Only SQL) є альтернативним підходом до зберігання та обробки інформації, який виник у відповідь на зростання обсягів неструктурованих і напівструктурованих даних. Вони не потребують жорстко визначеної схеми та забезпечують вищу гнучкість у моделюванні даних. Залежно від типу, NoSQL-БД можуть бути документно-орієнтованими (наприклад, MongoDB), колонковими (Cassandra), графовими (Neo4j) або побудованими за принципом ключ-значення (Redis) [5]. Вони орієнтовані на горизонтальне масштабування, що дозволяє ефективно працювати з великими обсягами даних у розподіленому середовищі, що особливо актуально для веб-сервісів, CRM-систем, систем аналітики та IoT-платформ (рис. 1.3).

Нереляційна БД

Файл student.json:

```
[
  {
    "id": 1,
    "name": "John",
    "surname": "Brown",
    "age": 19,
  },
  {
    "id": 2,
    "name": "Emma",
    "surname": "Carly",
    "age": 23,
  }
]
```

Рисунок 1.3 – Приклад нереляційної бази даних

Однією з ключових причин, що визначають вибір між реляційними та нереляційними базами даних, є суттєві відмінності у принципах їхньої побудови, організації даних та забезпеченні масштабованості й узгодженості. На рис. 1.4 зображено порівняльну гістограму найпопулярніших СУБД, що демонструє розподіл популярності серед обох типів. Реляційні бази даних, такі як MySQL, PostgreSQL і Oracle, переважають у традиційних корпоративних системах, тоді як нереляційні, як-от MongoDB чи Redis, активно застосовуються в гнучких, високонавантажених системах із великими обсягами неструктурованих даних.

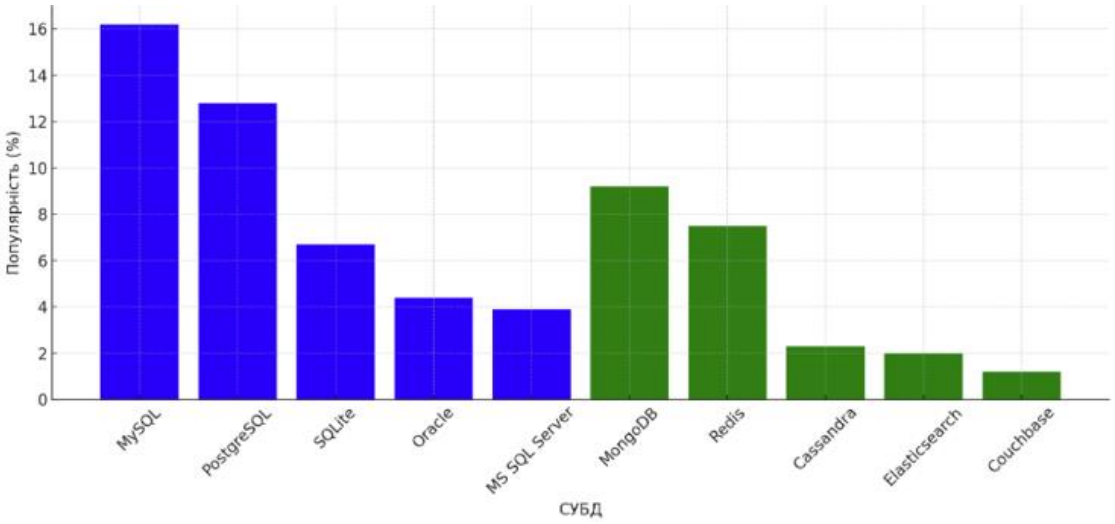


Рисунок 1.4 – Найпопулярніші реляційні та нереляційні СУБД

У реляційних базах даних інформація зберігається у вигляді таблиць із чітко визначеними стовпцями, де кожен рядок представляє окремий запис. Ця модель передбачає суворе дотримання структури та забезпечує високий рівень формалізації. У свою чергу, нереляційні бази даних (зокрема, документно-орієнтовані) оперують об’єктами або документами, які можуть мати довільну структуру. Кожен документ у такій системі є самодостатнім і зберігається у форматі JSON, BSON або XML, що дозволяє гнучко описувати дані без потреби у попередньо визначеній схемі [6].

У реляційних СУБД використовується жорстко фіксована схема, яка визначає типи даних, кількість полів, обмеження та зв’язки між таблицями. Зміна структури (наприклад, додавання нового поля) часто потребує змін у всій базі або

навіть порушує сумісність. У NoSQL базах навпаки – дані можуть мати різну структуру навіть у межах однієї колекції. Це забезпечує гнучкість при розробці, дозволяє швидко адаптуватися до змін у вимогах бізнесу без необхідності міграції даних.

Традиційні реляційні СУБД переважно орієнтовані на вертикальне масштабування – тобто підвищення продуктивності за рахунок потужнішого серверного обладнання. Проте це має обмеження з точки зору вартості та технічної реалізації. Натомість NoSQL-системи розроблені для горизонтального масштабування: дані розподіляються між кількома вузлами (нодами), що дозволяє легко збільшувати обсяг оброблюваної інформації та витримувати високі навантаження у розподілених середовищах.

Реляційні бази даних дотримуються принципів ACID (атомарність, узгодженість, ізольованість, довговічність), що гарантує надійність виконання операцій над даними, особливо важливо у фінансових та правових системах. Натомість більшість NoSQL-рішень реалізують менш суворий підхід – модель BASE (Basically Available, Soft state, Eventually consistent), яка допускає тимчасову неузгодженість даних заради підвищення масштабованості та продуктивності. Такий підхід є прийнятним у багатьох веб-сервісах, де затримка оновлення даних не є критичною.

У реляційних базах підтримка зв’язків реалізується через зовнішні ключі, що дозволяє зберігати логічну цілісність та будувати складні запити за допомогою операцій JOIN. Проте такі запити можуть значно знижувати продуктивність при роботі з великими обсягами даних. У нереляційних базах замість зовнішніх ключів часто використовуються вбудовані документи, коли пов’язані дані зберігаються разом в одному об’єкті. Це дозволяє уникати надлишкових запитів і підвищити швидкість доступу, проте вимагає ретельного проєктування структури для уникнення дублювання інформації.

З огляду на численні особливості реляційних та нереляційних підходів до зберігання даних, доцільно узагальнити основні характеристики кожного з них у порівняльній формі. Це дозволяє краще зрозуміти сферу застосування кожної

моделі, її переваги та обмеження залежно від вимог до інформаційної системи. Порівняльна табл. 1.1 містить ключові відмінності між реляційними та нереляційними базами даних за критеріями структури, масштабованості, схеми, транзакційності та підтримки зв’язків.

Таблиця 1.1 – Порівняння реляційних та нереляційних баз даних

Критерій	**Реляційні бази даних (SQL)**	**Нереляційні бази даних (NoSQL)**
Структура зберігання	Таблиці з фіксованими стовпцями та рядками	Документи, об’єкти, ключ-значення, графи
Схема даних	Жорстко визначена (статична) схема	Динамічна, може змінюватися від об’єкта до об’єкта
Масштабованість	Вертикальна (масштабування апаратних ресурсів)	Горизонтальна (додавання вузлів у кластері)
Транзакційність	Підтримка повної моделі ACID	Часткова або eventual consistency, модель BASE
Типові СУБД	MySQL, PostgreSQL, Oracle DB, Microsoft SQL Server	MongoDB, Cassandra, Couchbase, Redis, Neo4j
Підтримка зв’язків	Через зовнішні ключі та JOIN-запити	Через вбудовані документи або посилання, JOIN-запити не характерні
Гнучкість структури даних	Обмежена, потребує міграції при зміні структури	Висока, структура може бути унікальною для кожного запису
Придатність для	Транзакційні системи, фінанси, бухгалтерія	Big Data, IoT, веб-додатки, системи з високим навантаженням

Вибір типу системи управління базами даних має базуватися на характері задач, що вирішуються в межах конкретної інформаційної системи. У той час як реляційні СУБД забезпечують високий рівень надійності, транзакційності та структурної суворості, нереляційні бази даних пропонують гнучкість, масштабованість і зручність роботи з неструктурованими або напівструктурованими даними. Об’єктно-орієнтовані підходи, які лежать в основі сучасних мов програмування, краще інтегруються саме з NoSQL-рішеннями,

зокрема з документно-орієнтованими БД, що дає змогу безпосередньо відображати бізнес-логіку у структурі даних [7]. У контексті CRM-систем, де важлива адаптивність, швидкість обробки та персоналізація, використання NoSQL-технологій є доцільним та виправданим з позиції гнучкості і ефективності управління клієнтськими даними.

1.3 Особливості зберігання даних у різних типах СУБД

Системи управління базами даних (СУБД) реалізують різні підходи до зберігання, структурування та доступу до інформації, що безпосередньо впливає на ефективність роботи інформаційної системи, особливо в умовах масштабованих бізнес-процесів. У цьому контексті доцільно розглянути особливості зберігання даних у двох основних типах СУБД – реляційних та об’єктно-орієнтованих.

Реляційні бази даних (SQL) базуються на математичній теорії множин і алгебрі реляцій. Вони зберігають дані у вигляді двовимірних таблиць, де кожна таблиця представляє певну сутність або об’єкт предметної області. Наприклад, таблиця Customers може містити такі поля як CustomerID, Name, Email, а кожен запис – окремого клієнта.

Зв’язки між таблицями реалізуються через зовнішні ключі, які дозволяють створювати логічні зв’язки (один-до-багатьох, багато-до-багатьох). Наприклад, таблиця Orders, що зберігає інформацію про замовлення, може містити поле CustomerID, яке вказує на відповідного клієнта в таблиці Customers. Для вибірки пов’язаної інформації застосовуються операції JOIN [8].

Така модель вимагає попереднього проєктування фіксованої схеми бази даних, що обмежує гнучкість у зміні структури, але забезпечує сувору цілісність, контроль над типами даних, валідацією та підтримкою транзакційності.

На відміну від реляційної, об’єктно-орієнтована модель базується на принципах об’єктно-орієнтованого програмування. Тут основною одиницею

зберігання виступає об’єкт, що поєднує у собі дані (атрибути) та методи (функції), які оперують над цими даними.

Інформація в ООБД організовується у вигляді класів. Клас визначає шаблон (структуру) для об’єктів. Наприклад, клас Користувач може містити поля Ім’я, Прізвище, Email, а також методи login() чи updateProfile(). Об’єкти, створені на основі цього класу, зберігаються безпосередньо у базі у вигляді екземплярів із відповідними значеннями [9].

Особливістю об’єктно-орієнтованих баз є підтримка успадкування – можливість створення підкласів, які наслідують атрибути та методи базового класу. Наприклад, клас Клієнт може наслідувати базовий клас Користувач, додаючи поле Ідентифікаційний код клієнта або Історія покупок.

Це дозволяє моделювати складні бізнес-об'єкти природним чином, без потреби в нормалізації, як у реляційних БД (рис. 1.5). Водночас, збереження таких структур вимагає відповідної СУБД, яка підтримує об’єктну модель (наприклад, db4o, ObjectDB) або документно-орієнтовану модель (MongoDB, яка дозволяє зберігати вкладені структури, схожі на об’єкти).

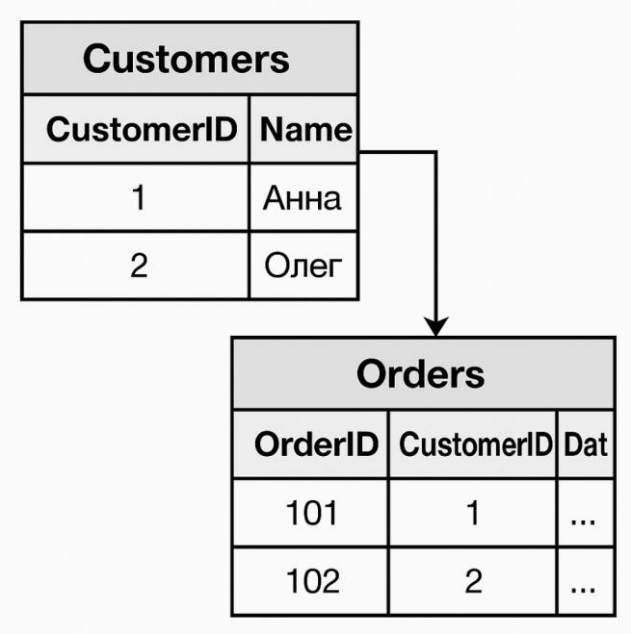
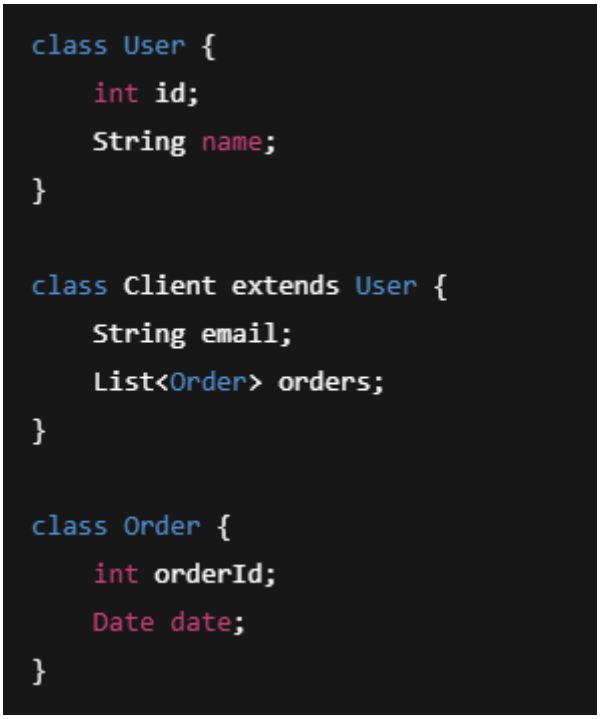


Рисунок 1.5 – Приклад логічної схеми реляційної бази даних

На рис. 1.6 зв’язок між клієнтом і замовленнями реалізується через вкладений список об’єктів, що дозволяє ефективно працювати з вкладеними структурами без складних SQL-запитів.



```
class User {
    int id;
    String name;
}

class Client extends User {
    String email;
    List<Order> orders;
}

class Order {
    int orderId;
    Date date;
}
```

Рисунок 1.6 – Структура класів у об’єктно-орієнтованій БД

У реляційній СУБД дані є суворо нормалізованими і розділеними по таблицях, що забезпечує високу надійність і уніфікованість, однак ускладнює інтеграцію з об’єктно-орієнтованими мовами. Об’єктно-орієнтовані СУБД, навпаки, дозволяють більш природно відображати структуру предметної області і логіку взаємодії об'єктів у бізнес-середовищі.

2 АНАЛІЗ ОБ'ЄКТНО-ОРІЄНТОВАНИХ ПІДХОДІВ ДО СТВОРЕННЯ БД

2.1 Об'єктно-орієнтовані середовища розробки

У процесі створення інформаційних систем на основі об'єктно-орієнтованих баз даних важливу роль відіграють інструменти розробки, які забезпечують підтримку принципів об'єктно-орієнтованого програмування (ООП), інтеграцію з базами даних, тестування, налагодження та розгортання програмного забезпечення. Такі середовища дозволяють розробникам реалізовувати складні проєкти відповідно до принципів інкапсуляції, наслідування та поліморфізму [9].

Серед найбільш популярних середовищ, які широко використовуються для розробки програмного забезпечення мовами Java, C++ та іншими ОО-мовами, варто виділити Eclipse та IntelliJ IDEA. Обидва середовища забезпечують гнучке управління проєктами, автоматизацію компіляції, підтримку сучасних фреймворків і засобів доступу до баз даних.

Eclipse – це багатофункціональне відкрите середовище розробки, яке підтримує численні мови програмування, зокрема Java, Python, C/C++, PHP тощо. Завдяки модульній архітектурі Eclipse дозволяє легко розширювати функціональність за допомогою плагінів. Інтеграція з системами контролю версій (Git), а також плагіни для роботи з реляційними та NoSQL базами даних роблять це середовище зручним для розробки великих інформаційних систем. Крім того, Eclipse підтримує ORM-фреймворки, такі як Hibernate, що спрощує обробку об'єктно-реляційних перетворень [10].

IntelliJ IDEA – це потужне комерційне середовище розробки від компанії JetBrains, яке вирізняється високим рівнем автоматизації, глибокою інтеграцією з сучасними технологіями та зручним інтерфейсом. IntelliJ IDEA підтримує створення корпоративних застосунків з використанням Java EE, Spring, Hibernate, а також дозволяє безпосередньо працювати з базами даних завдяки вбудованому

редактору SQL-запитів та засобам візуалізації структури БД. Середовище також надає засоби для моделювання структури класів, діаграм UML і автоматичного рефакторингу коду, що є надзвичайно корисним у розробці ОО-орієнтованих систем.

Використання таких середовищ дозволяє не лише ефективно реалізовувати бізнес-логіку системи у вигляді об’єктів і класів, а й спростити процес взаємодії з об’єктно-орієнтованими або документно-орієнтованими базами даних. У рамках реалізації прототипу CRM-системи в даній роботі також буде використано IntelliJ IDEA як основне середовище розробки, що забезпечує всі необхідні інструменти для побудови масштабованої та зручної інформаційної системи [11].

Порівняльна характеристика подана в табл. 2.1 демонструє переваги IntelliJ IDEA у контексті розробки об’єктно-орієнтованих інформаційних систем.

Таблиця 2.1 – Порівняння можливостей середовищ Eclipse та IntelliJ IDEA

Критерій	**Eclipse**	**IntelliJ IDEA**
Тип ліцензії	Відкрита (Open Source)	Безкоштовна (Community) та комерційна (Ultimate)
Підтримка мов програмування	Java, C/C++, Python, PHP, інші	Java, Kotlin, Groovy, Scala, SQL, JavaScript тощо
Підтримка ООП	Повна підтримка принципів ООП	Повна підтримка, глибока інтеграція з ОО-архітектурою
Інтеграція з БД	Плагіни для роботи з SQL та NoSQL	Вбудована підтримка БД, редактор SQL, ER-діаграми
Підтримка ORM	Hibernate, JPA (через плагіни)	Spring Data, Hibernate, JPA (вбудовано)
Інструменти рефакторингу	Стандартні	Розширені, автоматичне перейменування, безпечне переміщення класів
Моделювання UML	Через плагіни	Вбудовано (в Ultimate версії)
Робота з Git	Плагіни (EGit)	Повна інтеграція з Git, GitHub
Продуктивність та UX	Менш зручний інтерфейс, більша ресурсомісткість	Інтуїтивний інтерфейс, оптимізоване середовище
Підтримка розширень	Дуже висока (через Eclipse Marketplace)	Висока (через JetBrains Plugins Repository)

Застосування вказаних середовищ розробки широко поширене у практиці реального програмного забезпечення. За даними аналітичних звітів компанії JetBrains, понад 75% Java-розробників у світі надають перевагу IntelliJ IDEA у комерційних проєктах завдяки її продуктивності, розширеній інтеграції з фреймворками та високому рівню автоматизації [12]. Це середовище активно використовується у розробці складних корпоративних рішень, зокрема у фінансовому секторі, телекомунікаціях, логістиці та CRM-системах.

У той самий час Eclipse залишається популярним вибором серед дослідницьких проєктів, академічного середовища та відкритих спільнот. Його гнучка архітектура з великою кількістю плагінів дозволяє налаштовувати середовище під специфічні потреби, що особливо цінно в освітніх та наукових установах.

У процесі побудови об’єктно-орієнтованої інформаційної системи наступним етапом є моделювання структури взаємозв’язків між класами. Таке моделювання дозволяє візуалізувати логіку взаємодії між сутностями системи та визначити основні атрибути і методи для подальшої реалізації. Одним із найпоширеніших способів візуалізації є діаграма класів UML, яка ілюструє ієрархію класів, принципи наслідування, а також асоціативні зв’язки між об’єктами. На рис. 2.3 наведено приклад структури базових класів для CRM-системи з використанням об’єктно-орієнтованого підходу.

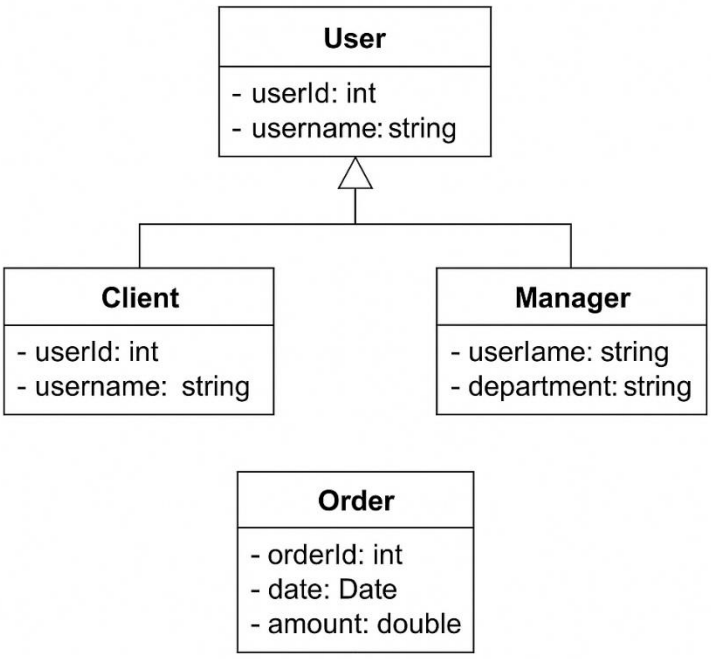


Рисунок 2.1 – UML-діаграма класів для CRM-системи

Об’єктно-орієнтовані середовища розробки, зокрема IntelliJ IDEA та Eclipse, відіграють ключову роль у проєктуванні та реалізації складних інформаційних систем. Вони забезпечують інтеграцію з мовами програмування, що підтримують ООП, дозволяють ефективно працювати з базами даних, виконувати автоматичне тестування, рефакторинг та моделювання структури системи. Розширені можливості IntelliJ IDEA, особливо в роботі з Java, ORM-фреймворками та SQL/NoSQL, роблять її найбільш придатною для реалізації CRM-систем об’єктно-орієнтованого типу [13].

Однак ефективність інформаційної системи значною мірою залежить не лише від середовища розробки, а й від обраної системи управління базами даних, яка повинна підтримувати гнучку, масштабовану й об’єктно-сумісну модель. Саме тому у наступному підпункті доцільно розглянути об’єктно-орієнтовані бази даних, зокрема MongoDB та Oracle NoSQL, які широко застосовуються у сучасних розподілених інформаційних системах.

2.2 Об’єктно-орієнтовані бази даних: MongoDB, Oracle NoSQL

У рамках розробки інформаційних систем, що базуються на об’єктно-орієнтованому підході, особливе значення мають системи управління базами даних, які підтримують гнучке зберігання складних структур даних, властивих об’єктам програмної моделі. На відміну від традиційних реляційних СУБД, об’єктно-орієнтовані та документно-орієнтовані бази даних дозволяють напряму зберігати об’єкти, вкладені структури та динамічно розширювати модель даних без зміни схеми.

Серед найбільш популярних представників таких рішень виділяються MongoDB та Oracle NoSQL Database [14]. Обидві системи розроблені з урахуванням вимог до масштабованості, гнучкості структури, швидкості доступу та ефективності зберігання неструктурованої або напівструктурованої інформації, що робить їх оптимальними для сучасних CRM-, ERP- та e-commerce-рішень.

MongoDB є документно-орієнтованою базою даних, яка зберігає дані у форматі BSON (бінарне представлення JSON). Це дозволяє працювати з об’єктами без потреби в попередньому визначенні суворої схеми. Кожен запис у MongoDB є окремим документом, який може містити вкладені поля, масиви, об'єкти та списки – таким чином структура класу з програмного коду напряму відповідає структурі документа в базі (рис. 2.2).

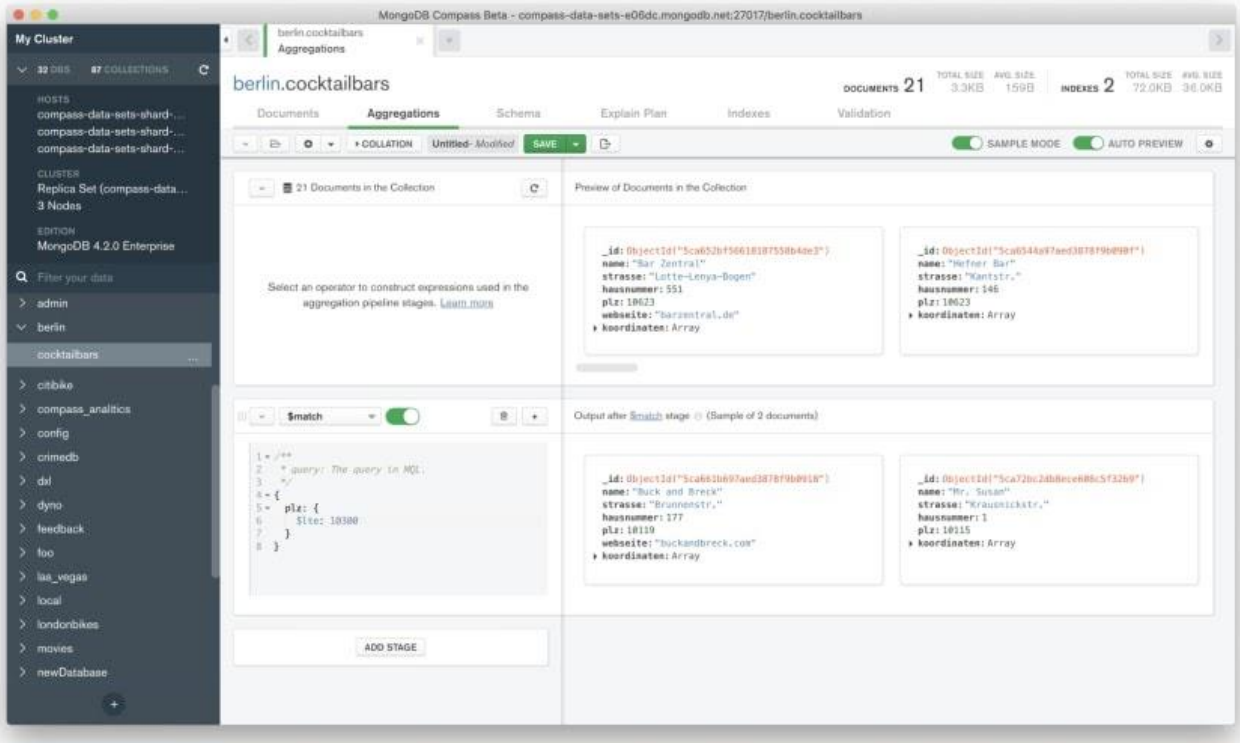


Рисунок 2.2 – Графічний інтерфейс MongoDB

MongoDB підтримує динамічну зміну структури документів, що забезпечує високу адаптивність у процесі розробки. Завдяки горизонтальному масштабуванню (через шардинг), високій швидкості читання/запису та інтеграції з мовами програмування (Java, Python, Node.js тощо), MongoDB активно використовується для розробки веб-додатків, мобільних платформ, аналітичних систем і IoT-проєктів [15].

Для забезпечення цілісності та аналітичної обробки MongoDB реалізує механізми індексування, агрегації, транзакцій і реплікації, що дозволяє використовувати її в критичних бізнес-застосуваннях. Зокрема, у CRM-системах документи можуть відповідати сутностям «Клієнт», «Менеджер», «Замовлення», а вкладені структури дозволяють уникати надлишкових запитів JOIN.

Oracle NoSQL Database є високопродуктивною розподіленою СУБД, яка поєднує принципи ключ-значення та документно-орієнтованого підходу. Вона призначена для обробки великих обсягів даних у реальному часі та має потужні механізми масштабування, реплікації та балансування навантаження (рис. 2.3).

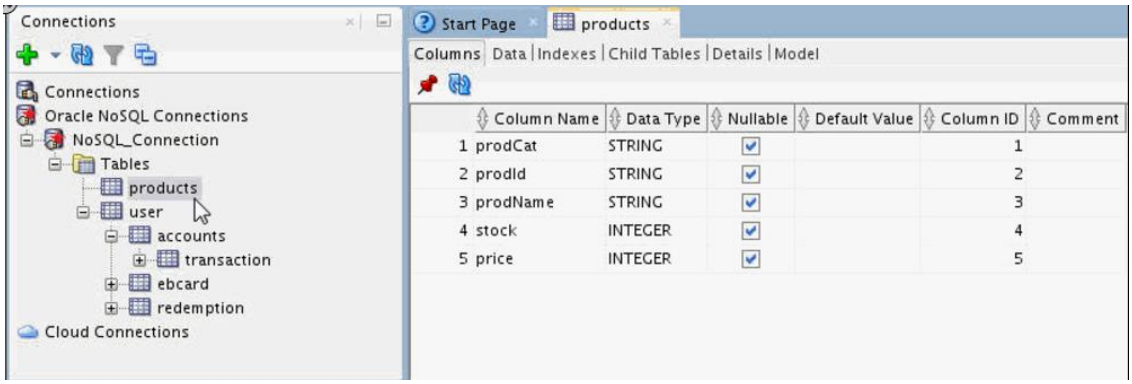


Рисунок 2.3 – Графічний інтерфейс Oracle NoSQL

На відміну від MongoDB, Oracle NoSQL надає більше контрольованих механізмів для керування транзакціями, збереження консистентності та безпеки. Система підтримує API для Java, C та Python, а також можливість використання JSON-документів з гнучкою структурою [16].

Завдяки тісній інтеграції з іншими продуктами Oracle (Oracle Cloud, Oracle Database, Oracle Analytics), ця СУБД часто використовується у великих підприємствах, які потребують високого рівня надійності, доступності та інтеграції з аналітичними системами.

Для кращого розуміння особливостей та порівняльних характеристик двох популярних об’єктно-орієнтованих систем управління базами даних – MongoDB і Oracle NoSQL – доцільно узагальнити їх функціональні властивості у табличній формі (табл. 2.2). Це дозволяє виокремити ключові параметри вибору відповідного інструменту залежно від потреб конкретної інформаційної системи.

Таблиця 2.2 – Порівняння MongoDB і Oracle NoSQL

Критерій	**MongoDB**	**Oracle NoSQL**
Тип	Документно-орієнтована	Документно-орієнтована, ключ-значення
Формат даних	BSON (розширений JSON)	JSON, ключ-значення
Масштабування	Горизонтальне (sharding)	Горизонтальне
Мова запитів	MongoDB Query Language	JSON API, Java API
Підтримка транзакцій	Обмежена, ACID-підтримка з 4.0	Модель BASE з окремими ACID-функціями
Індексація	Підтримується	Підтримується
Платформа	Відкрите ПЗ, доступна спільнота	Комерційне ПЗ від Oracle
Застосування	Веб-додатки, CRM, IoT, аналітика	Корпоративні системи, хмарні рішення, Big Data

Таким чином, об’єктно-орієнтовані бази даних, зокрема MongoDB та Oracle NoSQL, демонструють високу ефективність у зберіганні складних структур даних, притаманних сучасним інформаційним системам. Їх використання дозволяє відмовитися від суворої схеми даних, забезпечити масштабованість, швидкий доступ до інформації та природне представлення об’єктів реального світу. MongoDB є оптимальним вибором для розробки високонавантажених веб-додатків завдяки простоті інтеграції з мовами програмування, тоді як Oracle NoSQL частіше застосовується у великих підприємствах з підвищеними вимогами до консистентності, безпеки та хмарної сумісності [17].

Для повноцінної реалізації об’єктно-орієнтованого підходу недостатньо лише вибору відповідної СУБД – важливо також правильно спроєктувати

структуру бази даних, включаючи моделювання документів, класів і зв'язків між ними. Саме ці аспекти будуть розглянуті у наступному підпункті, присвяченому створенню, управлінню та моделюванню документно-орієнтованих баз даних.

2.3 Створення, управління та моделювання документно-орієнтованих БД

Для наочного розуміння організації документно-орієнтованої бази даних у MongoDB, що використовується в CRM-системі компанії, розглянемо структуру даних та характер їх взаємодії в межах бази. Нижче подано фрагмент з колекції Clients, який демонструє збереження ключової інформації про клієнтів, зокрема їхні контактні дані, історію звернень та статус обслуговування:

У межах колекції Clients зберігається структурована інформація про кожного клієнта CRM-системи. Нижче наведено фрагмент документа, який демонструє основні поля, необхідні для ефективного управління взаємодією з клієнтами [22]:

```
{
  "_id": ObjectId("..."),
  "clientId": "C1001",
  "name": "John Doe",
  "email": "johndoe@example.com",
  "phone": "123-456-7890",
  "organization": "Doe Drones",
  "interactionHistory": [
    {
      "date": "2025-04-21",
      "type": "Email",
      "details": "Запит на інформацію про нові моделі дронів"
    }
  ]
}
```

Рисунок 2.4 – Структура документа клієнта в колекції Clients бази даних MongoDB для CRM-системи

Цей фрагмент демонструє, як зберігаються персональні та контактні дані клієнта (name, email, phone), а також його зв'язок із певною організацією (organization). Поле interactionHistory реалізоване у вигляді масиву, що дозволяє фіксувати хронологію взаємодії з клієнтом, включаючи дату, тип звернення та його зміст. Такий підхід дає змогу забезпечити повноцінну історію комунікації, що є критично важливим для підвищення якості обслуговування та персоналізації пропозицій [18].

Кожен елемент у колекції Projects відображає окремий проєкт, що реалізується компанією. Нижче подано фрагмент документа, який демонструє збереження основної інформації про проєкт, включаючи його ідентифікатор, назву, опис, відповідального менеджера та стан виконання:

```
{
  "_id": ObjectId("..."),
  "projectId": "P2002",
  "clientReference": "C1001",
  "projectName": "Розробка навігаційного ПЗ для дронів",
  "startDate": "2025-04-20",
  "endDate": "2025-08-20",
  "status": "В розробці"
}
```

Рисунок 2.5 – Структура документа проєкту в колекції Projects бази даних MongoDB

Колекція SupportRequests призначена для фіксації звернень клієнтів до служби підтримки. Кожен документ у цій колекції містить інформацію про конкретний запит, включаючи ідентифікатор клієнта, дату звернення, опис проблеми та її поточний статус. Така структура дозволяє ефективно контролювати обробку звернень та забезпечувати належний рівень сервісу.

Нижче подано фрагмент, який демонструє організацію даних у межах одного звернення (рис. 2.6).

```
{
  "_id": ObjectId("..."),
  "requestId": "R3003",
  "clientReference": "C1001",
  "date": "2025-04-15",
  "issue": "Помилка під час оновлення програмного забезпечення",
  "status": "У роботі"
}
```

Рисунок 2.6 – Структура документа звернення у колекції SupportRequests бази даних MongoDB

Колекція SoftwareUpdates зберігає дані про випущені оновлення програмного забезпечення. Кожен документ у цій колекції включає відомості про ідентифікатор оновлення, його назву, дату релізу, опис змін та номер версії. Такий формат дозволяє систематизувати історію змін і забезпечити прозорість розвитку програмного продукту.

```
{
  "_id": ObjectId("..."),
  "updateId": "U4004",
  "title": "Оновлення безпеки – квітень 2025",
  "releaseDate": "2025-04-20",
  "description": "Виправлено критичні вразливості у модулі авторизації",
  "version": "v3.2.1"
}
```

Рисунок 2.7 – Структура документа оновлення у колекції SoftwareUpdates бази даних MongoDB

Колекція DroneModels призначена для зберігання технічних характеристик та описів моделей дронів, які використовуються або пропонуються компанією. Кожен документ у цій колекції містить детальну інформацію про конкретну модель, включаючи її ідентифікатор, назву, основні параметри та опис призначення. Така структура дозволяє швидко отримувати технічну інформацію для консультацій або

порівняння моделей. На рис. 2.8 наведено фрагмент, що демонструє структуру одного з документів колекції.

```
{
  "_id": ObjectId("..."),
  "modelId": "D5005",
  "modelName": "SkyTracker X7",
  "manufacturer": "AeroTech",
  "specs": {
    "flightTime": "45 хв",
    "range": "10 км",
    "camera": "4K UHD"
  },
  "purpose": "Аерозйомка та моніторинг об'єктів"
}
```

Рисунок 2.8 – Структура документа моделі дрона в колекції DroneModels бази даних MongoDB

Розглянуті колекції формують ядро інформаційної структури CRM-системи, що використовується для управління проєктами, клієнтами, технічною підтримкою та обслуговуванням дронів. Вони виступають основою функціонування всієї системи, забезпечуючи логічну організацію даних та підтримуючи ефективну взаємодію між її компонентами.

MongoDB як документно-орієнтована база даних дозволяє гнучко управляти вмістом колекцій, легко модифікувати структуру документів, оперативно отримувати доступ до потрібної інформації та ефективно відслідковувати взаємодії з клієнтами. Це робить її надзвичайно зручною для застосування в динамічному середовищі CRM, орієнтованому на обслуговування високотехнологічної продукції – зокрема, дронів [19].

Окрему увагу необхідно приділити питанням індексації та оптимізації запитів, адже саме ці аспекти безпосередньо впливають на продуктивність системи, особливо при роботі з великими обсягами даних і складними аналітичними вибірками.

Види індексації, які можуть бути реалізовані в MongoDB для підвищення продуктивності CRM-системи:

1. *Базова індексація.* За замовчуванням MongoDB створює унікальний індекс на полі _id, що дає змогу здійснювати швидкий доступ до окремих документів. Крім того, для покращення вибірки даних доцільно створювати індекси на найбільш часто використовуваних полях, таких як clientId у *Clients* або projectId у *Projects*.

2. *Складені (композитні) індекси.* Їх застосування дозволяє підвищити ефективність запитів, що охоплюють кілька полів одночасно. Наприклад, індекс на комбінацію clientId та startDate у колекції *Projects* значно прискорює отримання даних про проєкти конкретного клієнта у визначений період.

3. *Текстова індексація.* Для полів, що містять описові текстові значення, наприклад issueDescription у зверненнях до техпідтримки, доцільно використовувати текстові індекси. Це забезпечує ефективний повнотекстовий пошук за ключовими словами.

4. *Індекси для сортування.* Коли запити містять операції сортування (sort()), створення відповідних індексів на такі поля, як startDate (у *Projects*) або submissionDate (у *SupportRequests*), дозволяє суттєво знизити навантаження на систему при генерації звітів або формуванні списків [20].

Методи оптимізації запитів у MongoDB, які сприяють підвищенню ефективності CRM-системи:

1. *Використання оператора $explain.* Цей механізм дає змогу проаналізувати, як саме виконується запит, які індекси застосовуються, та оцінити загальну ефективність вибірки.

2. *Аналіз та корекція плану виконання запитів.* За допомогою отриманої інформації з $explain можна коригувати індексацію або змінювати структуру запитів, що особливо важливо при виконанні складних умов вибірки.

3. *Пагінація результатів.* Для обмеження обсягу даних, що повертаються у відповідь на запит, застосовуються методи розбиття на сторінки через limit() та skip(), що значно знижує навантаження на сервер при великій кількості результатів.

4. *Оптимізація агрегаційних операцій.* У випадках, коли потрібно здійснювати об'єднання, фільтрацію або трансформацію даних (наприклад, під час формування аналітичних звітів), варто застосовувати агрегаційні пайплайни з належною оптимізацією.

Завдяки впровадженню ефективних підходів до індексації та оптимізації запитів, CRM-система демонструє високу продуктивність і стабільність під час обробки даних, що особливо критично для компаній, які займаються розробкою та технічною підтримкою програмного забезпечення для дронів.

У практиці використання MongoDB в рамках даної системи, оптимізація іноді передбачає відхід від типової моделі запитів. Наприклад, при формуванні повної історії взаємодії клієнта із системою виникає потреба у вибірці даних одночасно з декількох колекцій – Clients, Projects, SupportRequests. У таких випадках оптимально використовувати агрегаційні пайплайни, які дозволяють ефективно поєднувати, фільтрувати та трансформувати дані, мінімізуючи навантаження на базу.

Ключовим фактором у забезпеченні високої продуктивності системи є регулярний моніторинг продуктивності бази даних. Це охоплює аналіз часу виконання запитів, використання оперативної пам’яті та процесора, а також частоту звернень до окремих колекцій. Подібна аналітика дозволяє своєчасно виявляти "вузькі місця", переглядати стратегії індексації або змінювати схему збереження даних без шкоди для цілісності системи.

Окремої уваги заслуговує оптимізація аналітичних та звітних запитів, які зазвичай вимагають значних обчислювальних ресурсів. Для підвищення ефективності рекомендується застосовувати кешування результатів часто виконуваних запитів, агрегування даних у фоновому режимі та побудову матеріалізованих подань (pre-aggregated views) для звітів із високим пріоритетом.

Крім технічних аспектів, важливо враховувати здатність системи до масштабування та адаптації до бізнес-змін. Зміна вимог або розширення функціоналу (наприклад, інтеграція нових модулів, облік нових типів клієнтів або послуг) потребують гнучкої моделі даних, яка може бути динамічно адаптована без потреби в повному рефакторингу бази.

Крім того, доцільним є застосування автоматизованих систем моніторингу (наприклад, MongoDB Atlas Performance Advisor або сторонніх APM-рішень), які надають рекомендації щодо створення нових індексів, виявлення неефективних запитів і аналізу змін у шаблонах навантаження в режимі реального часу [21-22].

3 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПІДПРИЄМСТВА

3.1 Вибір технологій для реалізації CRM-системи

Ефективне управління структурою документів, індексацією та оптимізацією запитів у MongoDB має вирішальне значення для забезпечення стабільної роботи CRM-системи, орієнтованої на обслуговування й розвиток програмного забезпечення для дронів. Завдяки використанню інструментів MongoDB, система демонструє високу адаптивність до змін бізнес-процесів, масштабованість і продуктивність при роботі з великими обсягами даних.

У наступному розділі буде представлено детальну модель даних MongoDB, яка є основою функціонування CRM-системи. Розгляд охоплює ключові сутності – Clients, Projects, SupportRequests, SoftwareUpdates та DroneModels, кожна з яких виконує важливу роль у забезпеченні ефективного управління клієнтськими відносинами, технічною підтримкою, розробкою продуктів і контролем за технічними параметрами дронів [23].

Таблиця 3.1 Основні сутності моделі даних CRM-системи на базі MongoDB

Сутність	Ключові поля	Опис	Призначення
Клієнти	clientId, name, email	Дані про клієнтів	Взаємодія з клієнтами
Проєкти	projectId, clientReference, status	Інформація про проєкти	Управління проєктною діяльністю
Тікети	ticketId, issueDescription, resolutionStatus	Запити на підтримку	Обробка та реагування на звернення

Продовження таблиці 3.1

Основні сутності моделі даних CRM-системи на базі MongoDB

Оновлення ПЗ	updateId, version, changeLog	Відомості про оновлення ПЗ	Відстеження релізів та змін
Моделі дронів	modelId, modelName, specifications	Характеристики моделей дронів	Технічні параметри й опис дронів

Сутність Clients є однією з ключових у моделі даних об'єктно-орієнтованої бази MongoDB, оскільки саме вона відповідає за зберігання та керування інформацією про клієнтів у межах CRM-системи. Її значення полягає у забезпеченні централізованого доступу до даних про взаємодії з клієнтами, що критично важливо для ефективної роботи компанії.

MongoDB, як документно-орієнтована база даних, забезпечує високу гнучкість у зберіганні таких даних, дозволяючи зберігати структури, які легко адаптуються до змін. Це особливо актуально в умовах, коли дані про клієнтів можуть розширюватися – наприклад, за рахунок додаткових атрибутів, історії взаємодії чи нових параметрів, які з’являються в процесі розвитку бізнесу.

Сутність *Clients* у MongoDB представлена у вигляді документа, який зберігає ключову інформацію про клієнта. Її структура включає наступні поля:

– *clientId (String).* Унікальний ідентифікатор клієнта, що використовується для однозначної ідентифікації особи або організації в системі. Це поле є критично важливим для зв’язку з іншими сутностями, зокрема *Projects* і *Tickets*;

– *name (String).* Ім’я клієнта або назва компанії. Це поле використовується для персоналізації сервісу та може містити ім’я контактної особи або юридичну назву організації, що підвищує зручність ідентифікації;

– *email (String).* Електронна адреса, яка виступає основним каналом зв’язку з клієнтом. Вона застосовується для надсилання сповіщень, оновлень, технічної інформації чи маркетингових повідомлень;

– *phone (String).* Контактний номер телефону. Поле може підтримувати декілька варіантів номерів, що забезпечує швидкий доступ до представників різних підрозділів або відповідальних осіб [23];

– *organization (String).* Назва компанії або організації, до якої належить клієнт. Це поле дозволяє чітко розмежовувати персональні та корпоративні обліки, особливо коли значення поля name є іменем фізичної особи;

– *interactionHistory (Array).* Масив об’єктів, кожен з яких представляє окрему взаємодію клієнта з компанією. Кожен елемент містить дату взаємодії, її тип (наприклад, електронний лист, телефонна розмова, зустріч) і опис події. Це поле є основою для побудови повної історії комунікацій з клієнтом та аналізу взаємодій у динаміці.

Сутність Clients у MongoDB надає широкі можливості для гнучкої роботи з клієнтськими даними. Завдяки відсутності жорстко визначеної схеми, характерної для реляційних баз даних, система дозволяє вільно змінювати структуру документів – додавати нові поля, змінювати наявні або видаляти непотрібні без потреби у глобальному рефакторингу всієї бази. Така адаптивність є надзвичайно важливою для сучасної CRM-системи, яка повинна швидко підлаштовуватись під динамічні вимоги бізнесу.

Особливу цінність становить масив interactionHistory, який зберігає послідовність взаємодій з клієнтом. Кожен запис у цьому масиві може містити дату, тип комунікації та її зміст, що дозволяє відстежувати повну історію контактів. Цей масив є надзвичайно корисним для аналізу ефективності комунікаційних стратегій, виявлення пріоритетів клієнтів та планування подальших дій у межах клієнтського циклу.

Поля email та phone виконують не лише роль каналів зворотного зв’язку, а й можуть бути інтегровані з зовнішніми сервісами – зокрема, з платформами масових розсилок, CRM-автоматизації або аналітики поведінки користувачів. Це відкриває

можливості для запуску персоналізованих маркетингових кампаній, сповіщень та побудови консистентної моделі взаємодії з клієнтами.

Поле organization дозволяє відобразити організаційну приналежність клієнта, що актуально при роботі з корпоративними структурами. Зібрані таким чином дані сприяють сегментації клієнтської бази, виявленню ринкових тенденцій, а також формуванню цільових комерційних пропозицій відповідно до галузевих або організаційних характеристик.

Окрім основних полів, сутність Clients може бути легко розширена новими атрибутами, такими як preferences (налаштування), feedbacks (відгуки), contractDetails (деталі контрактів) тощо. Це перетворює її з простого сховища інформації на повноцінний аналітичний інструмент, здатний підтримувати стратегічне планування та управління взаєминами на високому рівні [22-24].

Структура сутності Projects

Сутність *Projects* у MongoDB використовується для зберігання інформації про всі проєкти, що реалізуються в межах CRM-системи. Вона забезпечує структурований підхід до управління проєктною діяльністю та дозволяє пов’язувати кожен проєкт із відповідним клієнтом і супровідними елементами (наприклад, зверненнями або оновленнями програмного забезпечення). Основні поля цієї сутності включають:

– *projectId (String).* Унікальний ідентифікатор проєкту, що виступає основним ключем. Це поле забезпечує однозначну ідентифікацію кожного проєкту та використовується для зв’язку з іншими сутностями, зокрема *Tickets* та *SwUpdates*;

– *clientReference (String).* Посилання на ідентифікатор клієнта (*clientId*) із сутності *Clients*. Це поле дозволяє встановити зв’язок між проєктом та замовником, забезпечуючи прозоре відстеження того, для кого виконується відповідна робота;

– *projectName (String).* Назва проєкту, яка стисло характеризує його мету або напрям. Це поле сприяє зручній ідентифікації, каталогізації та пошуку проєктів у системі;

– *startDate та endDate (Date).* Дати початку та завершення проєкту. Завдяки цим полям можна контролювати часові рамки виконання завдань, оцінювати тривалість розробки та планувати розподіл ресурсів;

– *budget (Number).* Запланований бюджет проєкту. Це поле є основою для контролю фінансових витрат, оцінки рентабельності та прийняття рішень щодо подальших інвестицій;

– *status (String).* Поточний стан проєкту. Наприклад: *«У розробці»*, *«На тестуванні»*, *«Завершено»* тощо. Це поле використовується для моніторингу прогресу та своєчасного інформування всіх зацікавлених учасників про хід реалізації.

Сутність Projects виконує ключову функцію в управлінні проєктною діяльністю компанії, координуючи основні процеси розробки, тестування та впровадження програмних продуктів. Вона забезпечує централізований доступ до інформації про кожен проєкт, що дозволяє учасникам команди – від менеджерів до розробників – контролювати прогрес виконання, вчасно виявляти відхилення від початкового плану та оперативно вносити корективи.

Завдяки використанню поля clientReference, яке пов’язує проєкт із конкретним клієнтом, система дає змогу глибше розуміти потреби замовника й адаптувати реалізацію під його індивідуальні вимоги. Це не лише підвищує ефективність комунікації, а й сприяє формуванню довготривалих партнерських відносин через персоналізований підхід [24].

Наявність полів для фіксації дат початку та завершення, бюджету та статусу проєкту створює основу для стратегічного планування ресурсів, контролю витрат і моніторингу загальної ефективності. Усі ці елементи є критично важливими для підтримання стабільності розробки та своєчасного досягнення поставлених цілей.

Завдяки своїй гнучкості MongoDB дозволяє легко оновлювати й масштабувати структуру сутності Projects, що особливо актуально в умовах швидкозмінного середовища, характерного для галузі розробки програмного забезпечення. Можливість динамічного доповнення документа новими атрибутами

(наприклад, етапи реалізації, відповідальні особи чи пріоритетність) дозволяє системі зберігати актуальність та відповідати сучасним вимогам бізнесу.

Сутність Tickets у MongoDB

У контексті CRM-системи, що обслуговує розробку програмного забезпечення для дронів, сутність *Tickets* виконує важливу функцію в організації та супроводі служби підтримки. Вона забезпечує централізоване збереження інформації про звернення клієнтів, дозволяє ефективно координувати процес обробки запитів, фіксувати історію обслуговування та здійснювати аналітику щодо частоти, типів і складності проблем.

MongoDB як гнучка документно-орієнтована система ідеально підходить для представлення таких звернень у вигляді окремих документів, структура яких може адаптуватися під різні типи запитів або внутрішні процеси компанії.

Основні поля структури сутності *Tickets*:

– *ticketId (String).* Унікальний ідентифікатор звернення, який дозволяє здійснювати точне відстеження та керування окремим тікетом у системі;

– *projectId (String).* Посилання на проєкт, до якого відноситься дане звернення. Завдяки цьому полю забезпечується зв’язок між тікетами та конкретними ініціативами, що дає змогу враховувати контекст проблеми;

– *clientReference (String).* Ідентифікатор клієнта (*clientId*) з сутності *Clients*, який подав запит. Це поле дозволяє пов’язати кожен тікет із конкретною особою або компанією, що звернулася до служби підтримки;

– *issueDescription (String).* Деталізований опис проблеми, який надається клієнтом. Це ключовий атрибут для глибокого розуміння запиту та ефективного його вирішення;

– *submissionDate (Date).* Дата реєстрації звернення. Дане поле важливе для обчислення часу обробки запиту та формування звітності щодо якості сервісного обслуговування;

– *resolutionStatus (String).* Поточний статус тікету – наприклад: *«В обробці», «Вирішено», «Закрито»*. Це поле дає змогу оперативно контролювати хід вирішення запиту та сповіщати клієнта про його прогрес.

Сутність Tickets є важливою складовою CRM-системи, оскільки забезпечує структуроване управління зверненнями клієнтів, пов’язаними з технічною підтримкою, обслуговуванням та вирішенням проблем. Завдяки цій сутності команда підтримки може оперативно реагувати на запити, відслідковувати їхній статус у реальному часі, а також виявляти повторювані проблеми для подальшого вдосконалення продукту або сервісу.

Інтеграція Tickets із сутностями Projects та Clients дозволяє створити цілісне уявлення про взаємодію з кожним клієнтом, формуючи так звану «360-градусну модель» обслуговування. Це забезпечує не лише технічне вирішення окремих запитів, а й аналітичний підхід до оцінки рівня клієнтської задоволеності, якості сервісу та досвіду користування продуктом загалом.

Крім того, Tickets може бути інтегрована з системами сповіщень, службами автоматичних відповідей або чат-ботами, що дозволяє автоматизувати частину процесів підтримки. Наприклад, система може автоматично розпізнавати типові звернення, класифікувати їх за рівнем пріоритетності та надавати клієнтам шаблонні відповіді без участі операторів. Такий підхід суттєво скорочує час реакції та водночас підвищує загальний рівень обслуговування [23-25].

Сутність SwUpdates у MongoDB

Сутність *SwUpdates* відіграє стратегічну роль у системі управління оновленнями програмного забезпечення для дронів, забезпечуючи централізоване збереження даних про всі релізи, їх версії, зміни та прив’язку до відповідних проєктів. У межах CRM-системи ця сутність дозволяє розробникам, інженерам підтримки та тестувальникам контролювати життєвий цикл програмного забезпечення, забезпечуючи безперервну підтримку та відповідність сучасним стандартам.

Структура сутності SwUpdates включає наступні поля:

– *updateId (String).* Унікальний ідентифікатор оновлення, що дозволяє точно ідентифікувати кожен реліз і забезпечує його відстеження в системі.

– *projectId (String).* Посилання на проєкт, у межах якого було створено або застосовано оновлення. Це дозволяє підтримувати зв’язок між розробкою нових функціональностей і відповідними змінами в системі.

– *releaseDate (Date).* Дата публікації оновлення. Цей атрибут є основою для формування графіків релізів і аналітики щодо частоти оновлень.

– *version (String).* Версія програмного забезпечення. Наявність чіткої версіонності є критично важливою для підтримання сумісності з моделями дронів, а також для забезпечення цілісності системи в разі оновлень.

– *changeLog (String або Array).* Детальний перелік змін, що були внесені під час даного оновлення. Це можуть бути виправлення помилок, впровадження нових функцій, оновлення системи безпеки або зміни інтерфейсу.

Сутність SwUpdates сприяє підвищенню прозорості та контролю за всіма етапами розробки і впровадження оновлень. Її наявність забезпечує простежуваність еволюції програмного забезпечення та дозволяє системно підходити до оцінки його стабільності та безпеки. Особливо цінним є використання поля changeLog, яке дозволяє здійснювати ретроспективний аналіз змін, ідентифікувати потенційні ризики або недоліки, що могли з’явитися у нових версіях, а також документувати відповідність вимогам сертифікації та нормативних стандартів [18, 22, 24].

MongoDB у цьому контексті виконує роль ефективної платформи для масштабованого зберігання та аналізу даних про оновлення. Завдяки підтримці гнучких структур документів, системі агрегацій та високій продуктивності при обробці великих обсягів даних, MongoDB забезпечує швидкий пошук релізів, аналіз історії оновлень та моніторинг їх впливу на функціонування програмних рішень у дронах.

Сутність DroneModels у MongoDB

Сутність *DroneModels* відіграє важливу роль у зберіганні й систематизації інформації про різні моделі дронів, що використовуються або підтримуються компанією в межах CRM-системи. Ця структура дозволяє централізовано керувати технічними характеристиками, забезпечити сумісність програмного забезпечення з

конкретними моделями, а також вести облік супровідної інформації, яка є критично важливою для розробки, тестування та підтримки ПЗ.

Структура сутності DroneModels включає:

– modelId (String). Унікальний ідентифікатор моделі дрона, що забезпечує точну ідентифікацію кожного пристрою в системі. Це поле використовується для внутрішніх посилань та відстеження моделі протягом її життєвого циклу;

– modelName (String). Назва конкретної моделі, що дозволяє легко її розпізнавати серед асортименту пристроїв. Поле використовується як у внутрішніх документах, так і в інтерфейсах користувача;

– specifications (Embedded Document або String). Технічні параметри дрона, такі як час польоту, вага, радіус дії, тип камери або сенсорів тощо. Для зручності й логічного групування ці характеристики можуть бути структуровані у вигляді вкладеного документа (*embedded document*);

– compatibleSoftwareVersions (Array). Масив версій ПЗ, з якими дана модель дрона є сумісною. Це поле критично важливе для уникнення проблем із сумісністю, особливо під час оновлення або впровадження нових функцій;

– notes (String). Додаткові зауваження або примітки щодо моделі. Тут можуть зберігатися особливі умови експлуатації, обмеження, рекомендації щодо обслуговування або важливі оновлення [26].

Сутність *DroneModels* є основою для технічної інтеграції програмного забезпечення з фізичними пристроями. Вона дозволяє компанії ефективно адаптувати свої рішення до конкретних характеристик дронів, забезпечуючи стабільну роботу та відповідність функціональних можливостей вимогам апаратної частини.

Використання MongoDB для реалізації цієї сутності забезпечує високу гнучкість у зберіганні технічних даних, підтримку вкладених структур і масштабованість при роботі з великою кількістю моделей. Агрегаційні функції бази дозволяють швидко аналізувати технічні параметри, знаходити відповідні

моделі для певних вимог та здійснювати фільтрацію за будь-якими характеристиками.

Використання MongoDB для реалізації сутності DroneModels забезпечує гнучке та масштабоване зберігання технічної інформації, що є критично важливим для динамічного середовища розробки програмного забезпечення для дронів. Однією з ключових переваг є можливість безболісно оновлювати або доповнювати структуру документа – без необхідності перебудови всієї бази, що дозволяє оперативно реагувати на нові вимоги ринку та технологічні зміни.

Сутність DroneModels тісно пов’язана з іншими компонентами системи, зокрема з Projects та SwUpdates. Це забезпечує цілісне представлення процесів розробки, тестування й обслуговування. Розробники можуть з легкістю отримати повну картину щодо того, які саме моделі дронів були використані в рамках певного проєкту, які версії програмного забезпечення були з ними сумісні та які оновлення виявилися найефективнішими.

Гнучка структура MongoDB дає змогу розширювати модель DroneModels відповідно до змін бізнес-логіки або технологічних вимог, зокрема шляхом додавання нових характеристик, впровадження сучасних технологій або розширення асортименту пристроїв. Це робить сутність не просто технічним сховищем, а стратегічним інструментом, який підтримує інновації, дозволяє зберігати конкурентоспроможність компанії та адаптуватися до стрімких змін у галузі.

Крім того, поля на зразок notes та compatibleSoftwareVersions можуть використовуватися для збору зворотного зв’язку, виявлення тенденцій, аналізу сумісності та побудови прогнозів щодо розвитку продуктів. Це дозволяє не лише забезпечити відповідність програмного забезпечення характеристикам апаратного забезпечення, а й формувати дорожні карти продуктового розвитку.

У перспективі DroneModels може бути інтегрована з зовнішніми джерелами даних – наприклад, базами постачальників, каталогами виробників або ринковими аналітичними системами. Це відкриває нові можливості для адаптації до

технологічних трендів, виявлення інноваційних ніш і забезпечення сталого розвитку бізнесу.

3.2 Архітектура, інтерфейс і підтримка різних пристроїв

Архітектура розробленої CRM-системи базується на класичній клієнт-серверній моделі, що забезпечує розподіл функціональних обов'язків між інтерфейсом користувача та серверною частиною. Такий підхід гарантує високу гнучкість, масштабованість і модульність рішення, що є особливо важливим для систем, орієнтованих на динамічну взаємодію з користувачами та обробку великих обсягів даних [27].

Клієнтська частина (Front-End)

– Користувацький інтерфейс реалізовано із застосуванням сучасних вебтехнологій – HTML5, CSS3, а також JavaScript-фреймворків на зразок *React* або *Angular*. Завдяки цьому забезпечено інтуїтивно зрозумілий, адаптивний та відгуковий інтерфейс, який дозволяє користувачам швидко орієнтуватися в системі, переглядати звіти, керувати клієнтськими даними та формувати запити.

– Для обміну інформацією між клієнтом і сервером використовується API, побудований на базі REST або GraphQL. Така реалізація забезпечує стандартизовану структуру запитів і відповідей, спрощує інтеграцію з іншими сервісами та сприяє повторному використанню компонентів.

Серверна частина (Back-End)

– Серверна логіка реалізується за допомогою мов програмування, таких як Python, Java або Node.js, з використанням відповідних фреймворків (наприклад, *Express.js, Spring Boot, Django*). Основна відповідальність бекенду полягає у прийомі запитів, виконанні бізнес-логіки, взаємодії з базою даних та сторонніми сервісами.

– У якості системи управління базами даних використовується MongoDB, яка надає можливість зберігати інформацію у вигляді документів у колекціях. Такий формат зберігання ідеально підходить для CRM-систем, де структура даних може змінюватися з часом. MongoDB забезпечує високу продуктивність, горизонтальне масштабування та легкість адміністрування.

– Серверна частина також забезпечує взаємодію з зовнішніми сервісами, зокрема системами електронної пошти, інструментами бізнес-аналітики, хмарними платформами (AWS, Google Cloud, Azure) та службами автентифікації.

Мікросервісний підхід

Система може бути реалізована за принципами мікросервісної архітектури, де кожна ключова функція (керування користувачами, обробка звернень, генерація звітів, оновлення ПЗ) представлена окремим мікросервісом. Така організація дозволяє:

- масштабувати компоненти незалежно;
- оновлювати функціональність без зупинки всієї системи;
- підвищити fault-tolerance (стійкість до збоїв);
- забезпечити технологічну різноманітність (наприклад, кожен сервіс може бути реалізований іншою мовою програмування чи з використанням іншої бази даних).

Інтеграція MongoDB у CRM-систему, реалізовану мовою Java, вимагає послідовного підходу до налаштування з'єднання, організації структури даних і обробки запитів до бази. Це особливо важливо в системах, орієнтованих на галузь обслуговування дронів, де обсяги, структура і контекст даних можуть змінюватися динамічно.

У якості основної бази даних було обрано MongoDB, що забезпечує гнучкість у роботі з неструктурованими або напівструктурованими даними, високу продуктивність при виконанні запитів і підтримку горизонтального масштабування. Це дозволяє системі адаптуватися до змін без необхідності складної модифікації схеми даних [28].

Підключення MongoDB до Java-проєкту

Для інтеграції MongoDB у Java-застосунок необхідно додати відповідні залежності у файл керування бібліотеками — наприклад, pom.xml у разі використання ***Maven***. Бібліотека ***MongoDB Java Driver*** забезпечує низькорівневий доступ до бази та реалізацію повного спектра CRUD-операцій.

Конфігурація підключення охоплює:

- адресу хоста MongoDB-сервера;
- порт підключення (за замовчуванням — 27017);
- назву бази даних;
- облікові дані для автентифікації (за потреби).

Після створення об'єкта підключення до MongoDB ініціюється сесія, через яку виконуються запити до колекцій: зчитування, створення нових документів, оновлення записів або їх видалення.

CRUD-операції

Основна логіка взаємодії з даними реалізується через створення спеціалізованих методів для виконання CRUD-операцій:

- Create (створення): додавання нових документів у відповідні колекції (*Clients*, *Projects*, тощо);
- Read (читання): виконання пошукових запитів із використанням фільтрів, сортування і пагінації;
- Update (оновлення): зміна наявних даних із можливістю часткового або повного оновлення полів;
- Delete (видалення): безпечне видалення документів із урахуванням залежностей.

Особливу увагу приділено обробці виключень та помилок, що можуть виникати при з’єднанні з базою або під час виконання запитів. Це дозволяє гарантувати стабільність і надійність роботи всієї системи.

Оптимізація взаємодії з базою

Для забезпечення ефективності MongoDB-запитів реалізовано:

- використання індексів на найбільш використовуваних полях (наприклад, clientId, projectId);
- впровадження агрегаційних операцій для статистики та аналітики;

Інтеграція MongoDB у Java-проєкт CRM-системи здійснюється з використанням офіційного драйвера MongoDB. Нижче наведено покрокову інструкцію з реалізації базових операцій (CRUD) для роботи з сутністю Clients, яка є однією з ключових у системі.

Крок 1. Додавання залежності до Maven-проєкту Першим кроком є підключення MongoDB Java Driver до проєкту за допомогою системи керування залежностями Maven. Для цього необхідно додати відповідний фрагмент у файл pom.xml.

```xml
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>4.11.1</version>
</dependency>
```

Рисунок 3.1 – Підключення бібліотеки MongoDB Java Driver у файлі pom.xml

Крок 2. Встановлення з'єднання з базою даних Наступним етапом є створення з'єднання з MongoDB. Для цього необхідно вказати URI сервера, ім'я бази даних та вибрану колекцію.

```java
import com.mongodb.client.*;
import org.bson.Document;

public class MongoConnection {
    public static void main(String[] args) {
        String uri = "mongodb://localhost:27017";
        MongoClient mongoClient = MongoClients.create(uri);

        MongoDatabase database = mongoClient.getDatabase("crm_system");
        MongoCollection<Document> clientsCollection = database.getCollection("Clients");

        System.out.println("Підключення до бази даних встановлено.");
    }
}
```

Рисунок 3.2 – Створення підключення до бази даних MongoDB у Java

Крок 3. Додавання нового клієнта (Create) Для створення нового запису в колекції Clients використовується метод insertOne(), до якого передається об'єкт типу Document.

```
Document newClient = new Document("clientId", "C1001")
        .append("name", "Іван Петренко")
        .append("email", "ivan.petrenko@example.com")
        .append("phone", "+380991234567")
        .append("organization", "DroneTech");

clientsCollection.insertOne(newClient);
System.out.println("Новий клієнт доданий.");
```

Рисунок 3.3 – Додавання нового документа до колекції Clients

Крок 4. Отримання списку клієнтів (Read) Зчитування даних з бази виконується за допомогою методу find(), який повертає колекцію всіх документів у вигляді ітератора.

```
FindIterable<Document> clients = clientsCollection.find();
for (Document client : clients) {
    System.out.println(client.toJson());
}
```

Рисунок 3.4 – Зчитування документів із колекції Clients

Крок 5. Оновлення даних клієнта (Update) Для оновлення значення певного поля в документі використовується оператор $set у поєднанні з методом updateOne().

```
clientsCollection.updateOne(
        new Document("clientId", "C1001"),
        new Document("$set", new Document("email", "new.email@example.com"))
);
System.out.println("Електронну адресу оновлено.");
```

Рисунок 3.5 – Оновлення поля електронної адреси у документі клієнта

Крок 6. Видалення клієнта (Delete) Видалення документа з колекції виконується за допомогою методу deleteOne() із передачею умовного фільтра.

```
clientsCollection.deleteOne(new Document("clientId", "C1001"));
System.out.println("Клієнта видалено.");
```

Рисунок 3.6 – Видалення документа з колекції Clients за умовою

Для зручності повторного використання та інкапсуляції логіки взаємодії з

базою даних створено окремий клас MongoDBConnector. Він дозволяє централізовано управляти підключенням до бази MongoDB, виконувати базові CRUD-операції та працювати з різними колекціями [28].

```java
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoCollection;
import org.bson.Document;

public class MongoDBConnector {

    private MongoDatabase database;

    // Конструктор для підключення до бази даних
    public MongoDBConnector() {
        MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017");
        database = mongoClient.getDatabase("crmDB");
    }

    // Метод отримання колекції за її назвою
    public MongoCollection<Document> getCollection(String collectionName) {
        return database.getCollection(collectionName);
    }

    // Метод для створення нового документа
    public void createDocument(String collectionName, Document doc) {
        MongoCollection<Document> collection = getCollection(collectionName);
        collection.insertOne(doc);
    }

    // Метод для зчитування документа за умовою
    public Document readDocument(String collectionName, Document query) {
        MongoCollection<Document> collection = getCollection(collectionName);
        return collection.find(query).first();
    }

    // Метод для оновлення документа
    public void updateDocument(String collectionName, Document query, Document update) {
        MongoCollection<Document> collection = getCollection(collectionName);
        collection.updateOne(query, new Document("$set", update));
    }

    // Метод для видалення документа
    public void deleteDocument(String collectionName, Document query) {
        MongoCollection<Document> collection = getCollection(collectionName);
        collection.deleteOne(query);
    }
}
```

Рисунок 3.7 – Клас MongoDBConnector для управління з’єднанням та CRUD-операціями в MongoDB

У поданому прикладі клас MongoDBConnector реалізує основну логіку для підключення до бази даних MongoDB та виконання базових CRUD-операцій.

Методи createDocument, readDocument, updateDocument та deleteDocument демонструють універсальні підходи до роботи з документами у базі даних, використовуючи клас Document з офіційного MongoDB Java Driver.

Цей клас служить інфраструктурною основою, яка може бути використана у всіх інших компонентах системи для взаємодії з різними колекціями, такими як Clients, Projects, Tickets, SwUpdates та DroneModels. Завдяки цьому досягається централізоване керування базою даних, повторне використання коду та покращена підтримуваність системи.

Після реалізації базового конектора доцільно створити окремі класи або сервіси для кожної з логічних сутностей CRM-системи. Ці класи міститимуть опис відповідної моделі даних (наприклад, у вигляді POJO), а також набір методів для взаємодії з колекцією відповідної сутності.

У наступному підпункті буде наведено приклад структурування таких моделей, а також реалізації CRUD-операцій для однієї з ключових сутностей — Clients.

Реалізація сутностей у Java-проєкті передбачає створення відповідних моделей, які відображають структуру даних у базі MongoDB, та розробку методів для виконання CRUD-операцій. Нижче подано основні етапи цього процесу:

Крок 1. Створення класів моделей Для кожної логічної сутності CRM-системи (наприклад, Client, Project, Ticket) створюється окремий Java-клас. У межах класу описуються всі необхідні поля, які відповідають структурі документів у відповідних колекціях MongoDB [29].

```
public class Client {
    private String clientId;
    private String name;
    private String email;
    private String phone;
    private String organization;

    // Геттери та сеттери
    // Конструктори
    // toString(), equals(), hashCode() – за потреби
}
```

Рисунок 3.8 – Java-модель для сутності Client

Крок 2. Використання анотацій для MongoDB (за допомогою Spring Data) У разі використання фреймворку Spring Boot із бібліотекою Spring Data MongoDB, можна використовувати анотації для автоматичного зв'язку класів з документами бази даних.

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "Clients")
public class Client {

    @Id
    private String clientId;
    private String name;
    private String email;
    private String phone;
    private String organization;

    // Конструктори, геттери, сеттери
}
```

Рисунок 3.9 – Модель Client з анотаціями Spring Data MongoDB

Крок 3. Реалізація CRUD-операцій Після створення моделей необхідно реалізувати CRUD-функціональність (створення, читання, оновлення та видалення документів) для кожної сутності.

Це можна зробити:

вручну — за допомогою MongoDBConnector та Document;

або автоматизовано — через MongoRepository у Spring.

```
public class ClientService {
    private MongoDBConnector connector = new MongoDBConnector();

    public void addClient(Client client) {
        Document doc = new Document("clientId", client.getClientId())
                .append("name", client.getName())
                .append("email", client.getEmail())
                .append("phone", client.getPhone())
                .append("organization", client.getOrganization());

        connector.createDocument("Clients", doc);
    }
}
```

Рисунок 3.10 – Метод додавання клієнта у ClientService за допомогою MongoDBConnector

Після створення моделей сутностей, таких як Client, Project, Ticket тощо, наступним логічним кроком є визначення зв’язків між цими сутностями, що забезпечує цілісність і взаємозв’язок даних у системі. Наприклад, у класі Project може бути поле clientId, яке вказує на відповідного клієнта. Це дозволяє легко встановлювати логічний зв’язок між проєктами та замовниками, забезпечуючи можливість фільтрації, групування та аналітики за клієнтським критерієм.

Подібні зв’язки можуть бути реалізовані й між іншими сутностями, наприклад:

- Ticket → projectId (прив’язка звернення до конкретного проєкту),
- SwUpdates → projectId або modelId (залежність оновлень від моделей або проєктів),
- DroneModels → compatibleSoftwareVersions (зв’язок із оновленнями ПЗ).

Ще одним важливим аспектом побудови моделі є валідація даних. Вона необхідна для забезпечення достовірності, повноти та узгодженості інформації, яка зберігається в базі. Для цього можуть бути використані як стандартні механізми валідації, доступні у фреймворках типу Spring (наприклад, анотації @NotNull, @Email, @Pattern), так і власноруч реалізовані методи перевірки у класах моделей.

Валідація сприяє:

- запобіганню збереженню некоректних або неповних даних;
- виявленню помилок ще до етапу збереження;
- підвищенню надійності CRM-системи загалом.

Усі ці кроки — від побудови моделей і реалізації CRUD-логіки до встановлення міжсистемних зв’язків і валідації — формують фундамент для стійкої, масштабованої та зручно підтримуваної CRM-системи. Такий підхід дозволяє ефективно керувати інформацією про клієнтів, проєкти, звернення і технічні характеристики продукту, забезпечуючи відповідність вимогам сучасного бізнесу.

У межах CRM-системи, реалізованої з використанням Spring Framework, бізнес-логіка обробки даних клієнтів, проєктів та інших сутностей зосереджується у спеціалізованих сервісних класах. Ці класи є основою сервісного шару застосунку

та відповідають за взаємодію між контролерами (рівень представлення) та репозиторіями (рівень доступу до даних).

Spring підтримує принципи інверсії керування (IoC) та впровадження залежностей (DI), що дозволяє ефективно організовувати взаємозв'язки між компонентами. Сервісні класи позначаються анотацією @Service, що вказує Spring-контейнеру на необхідність їх створення як керованих компонентів (Spring beans).

```
import org.springframework.stereotype.Service;
import org.springframework.beans.factory.annotation.Autowired;
import com.example.crm.repository.ClientRepository;
import com.example.crm.model.Client;

@Service
public class CrmManagementService {

    private final ClientRepository clientRepository;

    @Autowired
    public CrmManagementService(ClientRepository clientRepository) {
        this.clientRepository = clientRepository;
    }

    // Метод для отримання клієнта за ідентифікатором
    public Client getClientById(String id) {
        return clientRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Client not found"));
    }

    // Метод для створення нового клієнта
    public Client createClient(Client client) {
        return clientRepository.save(client);
    }

    // За потреби: методи для оновлення, видалення, пошуку тощо
}
```

Рисунок 3.11 – Сервісний клас CrmManagementService для управління клієнтами в Spring

Тут клас CrmManagementService реалізує основні операції над клієнтськими даними, зокрема пошук клієнта за ID та створення нового запису. Для взаємодії з базою даних використовується ClientRepository — інтерфейс, що реалізує CRUD-функціональність за допомогою Spring Data MongoDB.

Залежність від репозиторію впроваджується через конструктор за допомогою анотації @Autowired, що відповідає принципам розділення відповідальностей і

дозволяє легко тестувати або модифікувати компонент.

Клас може бути розширений додатковими методами для управління іншими сутностями — Projects, Tickets, SwUpdates, DroneModels — у межах централізованої бізнес-логіки CRM-системи [27-29].

3.3 Реалізація та тестування функціоналу системи

У межах розробки CRM-системи було реалізовано повний набір функціональних можливостей для управління сутностями Clients, Projects, Tickets, SwUpdates та DroneModels на основі MongoDB. Система побудована за архітектурою клієнт-сервер з використанням Spring Framework, що забезпечує модульність, масштабованість і гнучкість.

Функціональність була реалізована через такі ключові компоненти:

- Моделі (POJO-класи) — для відображення даних з колекцій MongoDB.
- Репозиторії (Spring Data MongoDB) — для взаємодії з базою даних.
- Сервіси — для втілення бізнес-логіки та організації CRUD-операцій.
- REST-контролери — для взаємодії з клієнтською частиною.
- Тести — для перевірки коректності реалізації функцій системи.

```
@PostMapping("/clients")
public ResponseEntity<Client> addClient(@RequestBody Client client) {
    Client created = crmManagementService.createClient(client);
    return new ResponseEntity<>(created, HttpStatus.CREATED);
}
```

Рисунок 3.12 – Створення нового клієнта через REST API

Після реалізації функціональних компонентів було проведено модульне тестування з використанням JUnit і Spring Boot Test, що дозволило перевірити коректність реалізації CRUD-операцій і взаємодії з MongoDB [30].

Таблиця 3.1 Основні кейси для тестування

№	Назва тесту	Очікувана поведінка
1	Створення клієнта	Клієнт додається до бази, повертається HTTP 201
2	Отримання клієнта за ID	Повертається валідний об’єкт Client
3	Спроба отримати неіснуючого	Повертається HTTP 404 Client not found
4	Оновлення клієнта	Дані клієнта оновлюються у базі
5	Видалення клієнта	Клієнт видаляється, запит findById повертає null
6	Створення проєкту з неіснуючим ID	Система блокує зв’язування з неіснуючим клієнтом

Тест-кейс з JUnit для createClient().

```
@SpringBootTest
@AutoConfigureMockMvc
class ClientControllerTests {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void testCreateClient() throws Exception {
        String clientJson = """
            {
              "clientId": "C2001",
              "name": "Тестовий Клієнт",
              "email": "test@example.com",
              "phone": "+380501112233",
              "organization": "TestOrg"
            }
        """;

        mockMvc.perform(post("/clients")
                .contentType(MediaType.APPLICATION_JSON)
                .content(clientJson))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.clientId").value("C2001"))
                .andExpect(jsonPath("$.name").value("Тестовий Клієнт"));
    }
}
```

Рисунок 3.13 – Тестування створення клієнта за допомогою MockMvc

```
@SpringBootTest
@AutoConfigureMockMvc
class ProjectControllerTests {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void testCreateProject() throws Exception {
        String projectJson = """
            {
              "projectId": "P3001",
              "clientReference": "C2001",
              "projectName": "Інтеграція AI-модуля",
              "startDate": "2025-05-01",
              "endDate": "2025-08-01",
              "budget": 150000,
              "status": "У розробці"
            }
        """;

        mockMvc.perform(post("/projects")
                .contentType(MediaType.APPLICATION_JSON)
                .content(projectJson))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.projectId").value("P3001"))
                .andExpect(jsonPath("$.status").value("У розробці"));
    }
}
```

Рисунок 3.14 – Тест створення нового проєкту через REST API

Проєкт успішно створюється у базі даних, повертається HTTP 201 Created, значення projectId і projectName коректно записані.

```java
@SpringBootTest
@AutoConfigureMockMvc
class TicketControllerTests {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void testCreateTicket() throws Exception {
        String ticketJson = """
            {
              "ticketId": "T5005",
              "projectId": "P3001",
              "clientReference": "C2001",
              "issueDescription": "Дрон не підключається до мережі",
              "submissionDate": "2025-05-10",
              "resolutionStatus": "В обробці"
            }
        """;

        mockMvc.perform(post("/tickets")
                .contentType(MediaType.APPLICATION_JSON)
                .content(ticketJson))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.ticketId").value("T5005"))
                .andExpect(jsonPath("$.resolutionStatus").value("В обробці"));
    }
}
```

Рисунок 3.15 – Тест створення звернення до техпідтримки (тікета)

Тікет додається до колекції Tickets, система повертає код 201 і в JSON-відповіді відображається issueDescription.

Результати реалізації підтверджують працездатність CRUD-операцій для ключових сутностей системи. Застосування Spring Boot Test дозволило перевірити окремі компоненти в умовах, максимально наближених до продакшен-середовища.

Реалізовані тести дозволяють [30]:

- виявляти помилки на ранніх етапах;
- підтверджувати функціональність API;
- забезпечити цілісність даних при оновленнях і видаленнях;
- знизити ризики при подальшому масштабуванні.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було комплексно розглянуто ключові теоретичні засади побудови інформаційних систем та баз даних у контексті сучасних бізнес-потреб. Особливу увагу приділено аналізу ролі інформаційних систем у підприємницькому середовищі та порівнянню особливостей реляційних і нереляційних систем управління базами даних. Зроблено акцент на підвищенні ефективності обробки даних за допомогою гнучких та масштабованих технологій.

У ході дослідження проаналізовано переваги використання об’єктно-орієнтованих підходів до моделювання даних, а також розглянуто функціональні особливості сучасних документно-орієнтованих СУБД, зокрема MongoDB. Визначено, що застосування MongoDB забезпечує ефективне зберігання, швидкий доступ та динамічне оновлення даних, що особливо актуально для CRM-систем, які працюють із великою кількістю користувацьких та транзакційних даних.

В аналітичному розділі роботи виокремлено структуру ключових сутностей CRM-системи, таких як Clients, Projects, Tickets, SwUpdates та DroneModels, а також побудовано модель їх взаємодії. Визначено механізми оптимізації запитів, індексації, агрегацій та масштабування, що дозволяє забезпечити стабільну продуктивність системи навіть у випадках інтенсивного навантаження.

У практичній частині проєкту було розроблено повнофункціональний прототип CRM-системи на основі Java та MongoDB, побудований за клієнт-серверною архітектурою. Реалізовано модулі для управління клієнтами, проєктами, технічною підтримкою та оновленням програмного забезпечення. Для взаємодії між компонентами використано REST API, а також впроваджено принципи інверсії контролю та розділення відповідальностей через Spring Framework.

Також було проведено тестування функціоналу системи з використанням JUnit та Spring Boot Test. Реалізовані тест-кейси підтвердили коректність виконання CRUD-операцій, стабільність підключення до MongoDB та

правильність обробки запитів REST-контролерами. Це підтверджує надійність архітектури та готовність системи до масштабування і впровадження в реальні бізнес-процеси.

Таким чином, розроблена CRM-система демонструє можливість ефективного управління даними підприємства на основі об’єктно-орієнтованої бази даних MongoDB. Запропоновані підходи можуть бути адаптовані для розширення системи, інтеграції з іншими сервісами, а також подальшого вдосконалення функціоналу відповідно до специфіки галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Codd, E. F. A Relational Model of Data for Large Shared Data Banks [Електронний ресурс] / E. F. Codd // Communications of the ACM. – 1970. – Vol. 13, No. 6. – URL: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf (дата звернення: 11.05.2025).
2. PostgreSQL Documentation. Chapter 8: Data Types [Електронний ресурс]. – URL: https://www.postgresql.org/docs/current/datatype.html (дата звернення: 11.05.2025).
3. MongoDB Manual. Data Modeling [Електронний ресурс]. – URL: https://www.mongodb.com/docs/manual/data-modeling/ (дата звернення: 11.05.2025).
4. ObjectDB. ObjectDB for Java/JDO Developer's Guide [Електронний ресурс]. – URL: https://www.objectdb.com/database/jdo/manual (дата звернення: 11.05.2025).
5. ACID vs. BASE Database Model: Differences Explained [Електронний ресурс] // phoenixNAP. – URL: https://phoenixnap.com/kb/acid-vs-base (дата звернення: 11.05.2025).
6. NoSQL Databases: A Comparison of the Top Contenders [Електронний ресурс] // DataStax. – URL: https://www.datastax.com/blog/nosql-databases-comparison-top-contenders (дата звернення: 11.05.2025).
7. Digital Transformation: Rewiring for Digital and AI [Електронний ресурс] // McKinsey & Company. – 2024. – URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/leadership-and-digital-transformation (дата звернення: 11.05.2025).
8. Gartner Magic Quadrant for CRM Customer Engagement Center (CEC) 2024 [Електронний ресурс] // CX Today. – URL: https://www.cxtoday.com/crm/gartner-magic-quadrant-for-crm-customer-engagement-center-cec-2024-the-rundown/ (дата звернення: 11.05.2025).

9. Data Consistency Models: ACID vs. BASE Explained [Електронний ресурс] // Neo4j Blog. – URL: https://neo4j.com/blog/graph-database/acid-vs-base-consistency-models-explained/ (дата звернення: 11.05.2025).

10. Types of NoSQL Databases [Електронний ресурс] // GeeksforGeeks. – URL: https://www.geeksforgeeks.org/types-of-nosql-databases/ (дата звернення: 11.05.2025).

11. NoSQL Databases Explained: Advantages, Types, and Use Cases [Електронний ресурс] // InterSystems Resources. – URL: https://www.intersystems.com/resources/nosql-databases-explained-advantages-types-and-use-cases/ (дата звернення: 11.05.2025).

12. MongoDB Documentation. Data Modeling [Електронний ресурс]. – URL: https://www.mongodb.com/docs/manual/data-modeling/ (дата звернення: 11.05.2025).

13. Oracle. Table Modeling and Design in Oracle NoSQL [Електронний ресурс]. – URL: https://docs.oracle.com/en/database/other-databases/nosql-database/22.3/nsdev/table-modelling-and-design1.html (дата звернення: 11.05.2025).

14. MyScale. MongoDB vs Oracle NoSQL Database: A Model Sample [Електронний ресурс]. – URL: https://myscale.com/blog/mongodb-vs-oracle-nosql-database-model-comparison/ (дата звернення: 11.05.2025).

15. Oracle. Oracle NoSQL Database: Technical Overview [Електронний ресурс]. – URL: https://www.oracle.com/database/nosql/technologies/nosql/ (дата звернення: 11.05.2025).

16. GeeksforGeeks. Data Modeling in MongoDB [Електронний ресурс]. – URL: https://www.geeksforgeeks.org/mongodb-data-modelling/ (дата звернення: 11.05.2025).

17. MongoDB University. Overview of MongoDB and the Document Model [Електронний ресурс]. – URL: https://learn.mongodb.com/courses/overview-of-mongodb-and-the-document-model (дата звернення: 11.05.2025).

18. Oracle. Oracle NoSQL Database Documentation [Електронний ресурс]. – URL: https://docs.oracle.com/en/database/other-databases/nosql-database/index.html (дата звернення: 11.05.2025).

19. DB-Engines. MongoDB vs. Oracle vs. Oracle NoSQL Comparison [Електронний ресурс]. – URL: https://db-engines.com/en/system/MongoDB%3BOracle%3BOracle%2BNoSQL (дата звернення: 11.05.2025).

20. Codecademy. MongoDB Data Modeling Basics [Електронний ресурс]. – URL: https://www.codecademy.com/article/mongo-db-data-modeling-basics (дата звернення: 11.05.2025).

21. TutorialsPoint. MongoDB Data Modeling [Електронний ресурс]. – URL: https://www.tutorialspoint.com/mongodb/mongodb_data_modeling.htm (дата звернення: 11.05.2025).

22. C. J. Date. SQL and Relational Theory: How to Write Accurate SQL Code 2nd edition – O'reilly Media, Inc. – 2012. – 428 p.

23. Embley, David W. Object Database Development – Addison Wesley. – 1998. – 887 p.

24. Connolly T., Begg C. Database Systems A Practical Approach to Design, Implementation, and Management: Global Edition – Pearson Education, 2014. – 1440 p.

25. Baeldung. A Practical Guide to Spring Data MongoDB [Електронний ресурс]. – URL: https://www.baeldung.com/spring-data-mongodb-tutorial (дата звернення: 11.05.2025).

26. GeeksforGeeks. CRUD Operations in MongoDB using Java [Електронний ресурс]. – URL: https://www.geeksforgeeks.org/crud-operations-in-mongodb-using-java/ (дата звернення: 11.05.2025).

27. Spring Framework Documentation. Core Technologies: Dependency Injection [Електронний ресурс]. – URL: https://docs.spring.io/spring-framework/reference/core/beans/dependencies.html (дата звернення: 11.05.2025).

28. DigitalOcean. How To Build a Java Application with MongoDB [Електронний ресурс]. – URL: https://www.digitalocean.com/community/tutorials/java-mongodb-application (дата звернення: 11.05.2025).

29. Fowler M. Microservices: A Definition of This New Architectural Term [Електронний ресурс]. – URL: https://martinfowler.com/articles/microservices.html (дата звернення: 11.05.2025).

30. JUnit 5. User Guide [Електронний ресурс]. – URL: https://junit.org/junit5/docs/current/user-guide/ (дата звернення: 11.05.2025).

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ
ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ

Кафедра Комп'ютерної інженерії

ІНФОРМАЦІЙНА СИСТЕМА ПІДПРИЄМСТВА НА ОСНОВІ ОБ'ЄКТНО-ОРІЄНТОВАНИХ БАЗ ДАНИХ

Виконала:
Закружецька Катерина
Олександрівна
Науковий керівник:
Лащевська Наталія Олександрівна

Київ 2025

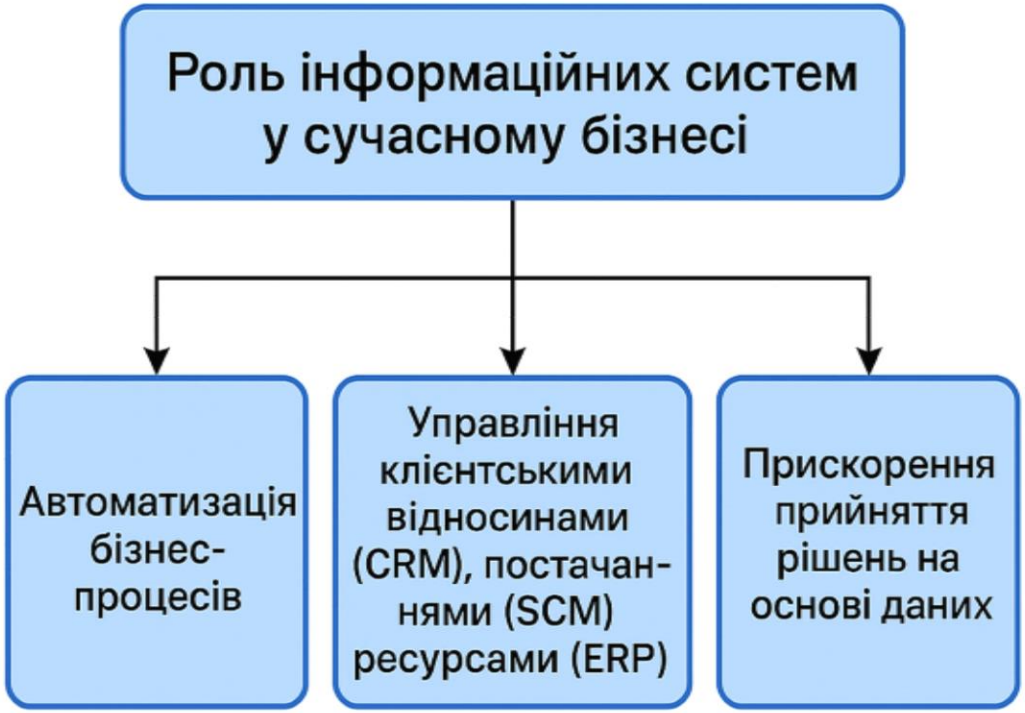
Мета, об’єкт, предмет

- **Метою роботи** є дослідити інформаційну систему підприємства на базі об’єктно-орієнтованої бази даних, яка забезпечить зручне управління клієнтськими даними, масштабованість і безпечну обробку запитів, а також порівняти її ефективність із традиційними рішеннями.
- **Об’єктом дослідження** є інформаційна система підприємства та об`єктно орієнтовані бази даних.
- **Предметом дослідження є** архітектура, інструменти та засоби реалізації об’єктно-орієнтованих баз даних у складі CRM-систем.

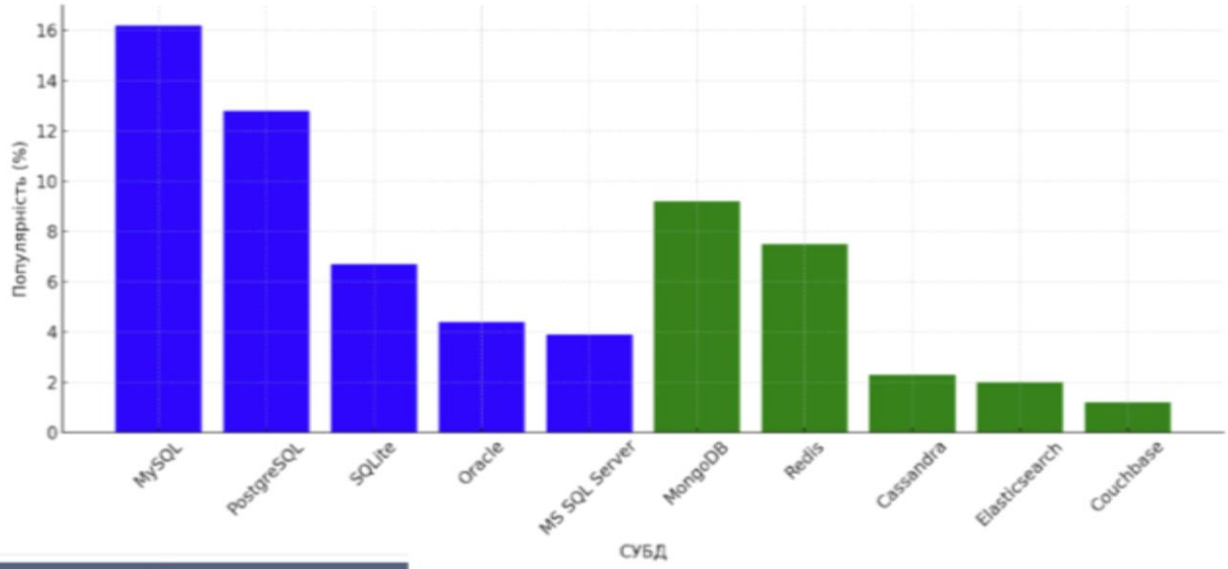
Актуальність

- Актуальність теми полягає в тому, що сучасні підприємства потребують гнучких, масштабованих та ефективних інформаційних систем. Традиційні реляційні бази даних мають обмеження, тому все більшого значення набувають об'єктно-орієнтовані СУБД, які краще інтегруються з сучасними мовами програмування та дозволяють моделювати складні бізнес-процеси, особливо в CRM-системах.

Роль інформаційних систем у сучасному бізнесі



Реляційні та нереляційні СУБД



Реляційна БД

student			
id	name	surname	age
1	John	Brown	19
2	Emma	Carly	23

Нереляційна БД

Файл student.json:

```
[
  {
    "id": 1,
    "name": "John",
    "surname": "Brown",
    "age": 19,
  },
  {
    "id": 2,
    "name": "Emma",
    "surname": "Carly",
    "age": 23,
  }
]
```

Порівняння MongoDB і Oracle NoSQL

Критерій	MongoDB	Oracle NoSQL
Тип	Документно-орієнтована	Документно-орієнтована, ключ-значення
Формат даних	BSON (розширений JSON)	JSON, ключ-значення
Масштабування	Горизонтальне (sharding)	Горизонтальне
Мова запитів	MongoDB Query Language	JSON API, Java API
Підтримка транзакцій	Обмежена, ACID-підтримка з 4.0	Модель BASE з окремими ACID-функціями
Індексація	Підтримується	Підтримується
Платформа	Відкрите ПЗ, доступна спільнота	Комерційне ПЗ від Oracle
Застосування	Веб-додатки, CRM, IoT, аналітика	Корпоративні системи, хмарні рішення, Big Data

Створення, управління та моделювання документно-орієнтованих БД

```
{
  "_id": ObjectId("..."),
  "clientId": "C1001",
  "name": "John Doe",
  "email": "johndoe@example.com",
  "phone": "123-456-7890",
  "organization": "Doe Drones",
  "interactionHistory": [
    {
      "date": "2025-04-21",
      "type": "Email",
      "details": "Запит на інформацію про нові моделі дронів"
    }
  ]
}
```

Рисунок 2.4 – Структура документа клієнта в колекції Clients бази даних MongoDB для CRM-системи

```
{
  "_id": ObjectId("..."),
  "projectId": "P2002",
  "clientReference": "C1001",
  "projectName": "Розробка навігаційного ПЗ для дронів",
  "startDate": "2025-04-20",
  "endDate": "2025-08-20",
  "status": "В розробці"
}
```

Рисунок 2.5 – Структура документа проєкту в колекції Projects бази даних MongoDB

```
{
  "_id": ObjectId("..."),
  "requestId": "R3003",
  "clientReference": "C1001",
  "date": "2025-04-15",
  "issue": "Помилка під час оновлення програмного забезпечення",
  "status": "У роботі"
}
```

Рисунок 2.6 – Структура документа звернення у колекції SupportRequests бази даних MongoDB

Створення, управління та моделювання документно-орієнтованих БД

У MongoDB для підвищення продуктивності CRM-системи застосовують різні типи індексації (базова, складена, текстова, для сортування) та методи оптимізації запитів (аналіз $explain, пагінація, агрегаційні пайплайни). Це дозволяє ефективно обробляти великі обсяги даних, особливо при формуванні звітів і запитах до кількох колекцій. Важливим є також постійний моніторинг продуктивності БД і адаптація моделі даних до бізнес-змін. Рекомендується використовувати інструменти автоматизованого аналізу, зокрема MongoDB Atlas Performance Advisor.

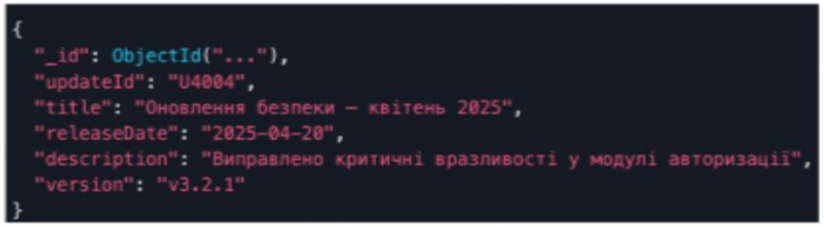


Рисунок 2.7 – Структура документа оновлення у колекції SoftwareUpdates бази даних MongoDB



Рисунок 2.8 – Структура документа моделі дрона в колекції DroneModels бази даних MongoDB

Вибір технологій для реалізації CRM-системи

Сутність	Ключові поля	Опис	Призначення
Клієнти	clientId, name, email	Дані про клієнтів	Взаємодія з клієнтами
Проєкти	projectId, clientReference, status	Інформація про проєкти	Управління проєктною діяльністю
Тікети	ticketId, issueDescription, resolutionStatus	Запити на підтримку	Обробка та реагування на звернення
Оновлення ПЗ	updateId, version, changeLog	Відомості про оновлення ПЗ	Відстеження релізів та змін
Моделі дронів	modelId, modelName, specifications	Характеристики моделей дронів	Технічні параметри й опис дронів

Таблиця 3.1 Основні сутності моделі даних CRM-системи на базі MongoDB

Архітектура, інтерфейс і підтримка різних пристроїв

```xml
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>4.11.1</version>
</dependency>
```

Рисунок 3.1 – Підключення бібліотеки MongoDB Java Driver у файлі pom.xml

```java
import com.mongodb.client.*;
import org.bson.Document;

public class MongoConnection {
    public static void main(String[] args) {
        String uri = "mongodb://localhost:27017";
        MongoClient mongoClient = MongoClients.create(uri);

        MongoDatabase database = mongoClient.getDatabase("crm_system");
        MongoCollection<Document> clientsCollection = database.getCollection("Clients");

        System.out.println("Підключення до бази даних встановлено.");
    }
}
```

Рисунок 3.2 – Створення підключення до бази даних MongoDB у Java

```java
Document newClient = new Document("clientId", "C1001")
        .append("name", "Іван Петренко")
        .append("email", "ivan.petrenko@example.com")
        .append("phone", "+380991234567")
        .append("organization", "DroneTech");

clientsCollection.insertOne(newClient);
System.out.println("Новий клієнт доданий.");
```

Рисунок 3.3 – Додавання нового документа до колекції Clients

```java
FindIterable<Document> clients = clientsCollection.find();
for (Document client : clients) {
    System.out.println(client.toJson());
}
```

Рисунок 3.4 – Зчитування документів із колекції Clients

```java
clientsCollection.updateOne(
        new Document("clientId", "C1001"),
        new Document("$set", new Document("email", "new.email@example.com"))
);
System.out.println("Електронну адресу оновлено.");
```

Рисунок 3.5 – Оновлення поля електронної адреси у документі клієнта

Архітектура, інтерфейс і підтримка різних пристроїв

```
clientsCollection.deleteOne(new Document("clientId", "C1001"));
System.out.println("Клієнта видалено.");
```

Рисунок 3.6 – Видалення документа з колекції Clients за умовою



Рисунок 3.7 – Клас MongoDBConnector для управління з'єднанням та CRUD-операціями в MongoDB

```
public class Client {
    private String clientId;
    private String name;
    private String email;
    private String phone;
    private String organization;

    // Геттери та сеттери
    // Конструктори
    // toString(), equals(), hashCode() – за потреби
}
```

Рисунок 3.8 – Java-модель для сутності Client

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "Clients")
public class Client {

    @Id
    private String clientId;
    private String name;
    private String email;
    private String phone;
    private String organization;

    // Конструктори, геттери, сеттери
}
```

Рисунок 3.9 – Модель Client з анотаціями Spring Data MongoDB

Реалізація та тестування функціоналу системи

```java
@PostMapping("/clients")
public ResponseEntity<Client> addClient(@RequestBody Client client) {
    Client created = crmManagementService.createClient(client);
    return new ResponseEntity<>(created, HttpStatus.CREATED);
}
```

Рисунок 3.10 – Створення нового клієнта через REST API

```java
@SpringBootTest
@AutoConfigureMockMvc
class ClientControllerTests {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void testCreateClient() throws Exception {
        String clientJson = """
            {
              "clientId": "C2001",
              "name": "Тестовий Клієнт",
              "email": "test@example.com",
              "phone": "+380501112233",
              "organization": "TestOrg"
            }
        """;

        mockMvc.perform(post("/clients")
                .contentType(MediaType.APPLICATION_JSON)
                .content(clientJson))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.clientId").value("C2001"))
                .andExpect(jsonPath("$.name").value("Тестовий Клієнт"));
    }
}
```

Рисунок 3.11 – Тестування створення клієнта за допомогою MockMvc

```java
@SpringBootTest
@AutoConfigureMockMvc
class ProjectControllerTests {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void testCreateProject() throws Exception {
        String projectJson = """
            {
              "projectId": "P3001",
              "clientReference": "C2001",
              "projectName": "Інтеграція AI-модуля",
              "startDate": "2025-05-01",
              "endDate": "2025-08-01",
              "budget": 150000,
              "status": "У розробці"
            }
        """;

        mockMvc.perform(post("/projects")
                .contentType(MediaType.APPLICATION_JSON)
                .content(projectJson))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.projectId").value("P3001"))
                .andExpect(jsonPath("$.status").value("У розробці"));
    }
}
```

Рисунок 3.12 – Тест створення нового проєкту через REST API

```java
@SpringBootTest
@AutoConfigureMockMvc
class TicketControllerTests {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void testCreateTicket() throws Exception {
        String ticketJson = """
            {
              "ticketId": "T5005",
              "projectId": "P3001",
              "clientReference": "C2001",
              "issueDescription": "Дрон не підключається до мережі",
              "submissionDate": "2025-05-10",
              "resolutionStatus": "В обробці"
            }
        """;

        mockMvc.perform(post("/tickets")
                .contentType(MediaType.APPLICATION_JSON)
                .content(ticketJson))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.ticketId").value("T5005"))
                .andExpect(jsonPath("$.resolutionStatus").value("В обробці"));
    }
}
```

Рисунок 3.13 – Тест створення звернення до техпідтримки (тікета)

Висновки

- У роботі розглянуто теоретичні засади побудови інформаційних систем і баз даних відповідно до сучасних бізнес-вимог.
- Проаналізовано переваги об'єктно-орієнтованого моделювання даних і функціональність MongoDB для CRM-систем.
- Уточнено структуру основних сутностей системи та визначено ефективні підходи до індексації, агрегації та масштабування.
- Реалізовано прототип CRM-системи на основі Java, Spring Framework і MongoDB з REST-архітектурою.
- Проведено тестування функціональності, що підтвердило стабільність і коректність роботи CRUD-операцій і API-запитів.
- Система готова до впровадження, розширення та адаптації до галузевих потреб бізнесу.