МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА

Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»

Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»

галузь знань	12 Інформаційні технології
спеціальність	122 Комп'ютерні науки

Форма навчання: заочна

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему: «Розроблення інформаційної управляючої системи для планування робочого процесу на підприємстві»

здобувача _____Цуканової Тетяни Сергіївни_____
(ПІБ, підпис)

Науковий керівник: ___к.е.н., доцент, Денісова О.О._____
(науковий ступінь, учене звання, ПІБ)

_____
(підпис)

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти (ЕК)

Завідувач кафедри: _____
(науковий ступінь, учене звання, ПІБ)

_____
(підпис)

Київ 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»

галузь знань 12 Інформаційні технології
спеціальність 122 Комп'ютерні науки

ПОГОДЖЕНО:
Керівник проєктної групи (гарант)
освітньо-професійної програми

_____Устенко С.В.
«____» _____202_ р.

ЗАТВЕРДЖУЮ:
Завідувач кафедри інформаційних
систем в економіці

_____ Тішков Б.О.
«____» _____ 202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти *Цуканової Тетяни Сергіївни*
заочної форми навчання

на підготовку кваліфікаційної магістерської роботи
на тему: *«Розроблення інформаційної управляючої системи для планування робочого процесу на підприємстві»*

Тему затверджено наказом ректора Університету від «__» _____ 202_ р. **№**_____

Кваліфікаційна магістерська робота виконується на матеріалах
наукових публікацій та матеріалів веб-сайтів для планування робочого процесу

План кваліфікаційної магістерської роботи

Розділ І __*Дослідження та аналіз підходів до створення ІС предметної області*_____
(назва розділу)

Розділ ІІ __*Характеристика інформаційних управляючих систем*_____
(назва розділу)

Розділ ІІІ__*Розролення проектних рішень та їх реалізація*_____
(назва розділу)

Об'єкт дослідження: *система планування робочим процесом на підприємстві*

Предмет дослідження: *інформаційні процеси, що характеризують планування робочих завдань на підприємстві*

Мета кваліфікаційної магістерської роботи: *розробити інформаційну систему для планування робочого процесу на підприємстві*

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети:

У розділі I *Дослідити сучасний стан підтримки клієнтів інформаційних управляючих систем для планування робочого процесу на підприємстві. Визначитися з вимогами, підходами та функціоналом системи. Провести аналіз існуючих систем з автоматизації досліджуваної задачі*

У розділі II *Характеризувати загальну архітектуру системи та рівні логічної і фізичної архітектури, модульну структура з виділенням ключових компонентів, а також UML-діаграми, що відтворюють блок-схеми, діаграми компонентів і розгортання.*

У розділі III *Виконати проєктування компонентів інформаційної системи, зокрема інформаційного, технічного та програмного забезпечення. Виконати реалізацію програмного забезпечення системи для планування робочого процесу на підприємстві та привести результати, вказавши можливість та доцільність застосування їх на практиці*

Завдання підготував
науковий керівник _____ Денісова Ольга Олександрівна
(підпис)

«____» _____ 2025 р.

Завдання одержав
здобувач _____ Цуканова Тетяна Сергіївна
(підпис)

«____» _____ 2025 р.

Реферат

Кваліфікаційна магістерська робота містить 73 сторінки, 2 таблиці, 17 рисунків, список використаних джерел з 51 найменувань, додатки.

«Розроблення інформаційної управляючої системи для планування робочого процесу на підприємстві»

Об'єктом дослідження кваліфікаційної магістерської роботи є процеси організації та планування виробничої та управлінської діяльності на підприємтві.

Предметом дослідження є інформаційні процеси, що характеризують планування робочих завдань на підприємстві.

Мета і завдання дослідження. Основною метою кваліфікаційної магістерської роботи є вдосконалення процесу планування та управління робочими процесами на підприємстві шляхом проєктування інформаційної системи автоматизованого планування.

Відповідно до поставленої мети визначені такі *завдання*:

- дослідити предметну область, виконати аналіз існуючих систем планування робочих процесів на підприємствах;
- представити обґрунтування вибору технологій для створення системи;
- визначити структуру та представити характеристику інформаційної системи планування робочого процесу на підприємстві;
- описати методи та моделі в системі планування робочого процесу;
- спроєктувати базу даних для системи планування робочого процесу;
- спроєктувати базу знань для системи планування робочого процесу;
- описати програмне забезпечення, що реалізує систему планування робочого процесу;
- описати технічне забезпечення, необхідне для функціонування системи;
- представити результати реалізації системи планування робочого процесу.

Теоретична, методична та практична значущість отриманих результатів. Під час дослідження було досліджено предметну область, проаналізовано існуючі підходи до організації та ведення планування робочих процесів, варіанти зберігання та доступу до планових даних, а також формати обліку завдань і ресурсів. Було визначено доцільність і перспективність удосконалення процесів управління роботою підприємства шляхом створення спеціалізованого програмного забезпечення — інформаційної системи планування робочого процесу. Практичні результати дослідження полягають у підвищенні ефективності планування, контролю та координації робочих завдань, що сприятиме покращенню продуктивності праці і кращому використанню ресурсів підприємства.

Рік виконання кваліфікаційної магістерської роботи – 2025.

Рік захисту роботи – 2025.

Ключові слова: планування, робочий процес, розподіл завдань автоматизація.

ВІДГУК НАУКОВОГО КЕРІВНИКА

на кваліфікаційну магістерську роботу
на тему: «Розроблення інформаційної управляючої системи для планування робочого процесу на підприємстві»
здобувачки **Цуканової Тетяни Сергіївни**

1. Актуальність теми. У сучасних умовах цифрової трансформації підприємств особливої ваги набуває автоматизація процесів планування, обліку та комунікації в межах організацій. Розроблення інформаційної управляючої системи з інтеграцією сучасних веб-технологій, телекомунікаційних інтерфейсів та інструментів аналітики є вкрай актуальним завданням, що відповідає запитам ринку та стратегічним цілям підприємств щодо підвищення ефективності управління ресурсами.

2. Позитивні риси кваліфікаційної магістерської роботи. Робота відзначається глибоким аналізом предметної області, комплексним підходом до проєктування системи, чітко визначеною архітектурою та високим рівнем технічного виконання. Використання сучасного інструментарію (Node.js, Angular, PostgreSQL, Redis, Docker/Kubernetes) свідчить про професійний підхід здобувачки до реалізації проєкту. Окремо варто відзначити детальне моделювання структури даних, розробку інтерфейсів та забезпечення надійності системи.

3. Наявність самостійних розробок автора. Тетяна Цуканова самостійно розробила проєктні рішення та реалізувала низку ключових компонентів системи, зокрема модуль Telegram-бота.

4. Цінність теоретичних висновків та практичних рекомендацій. Теоретичні висновки обґрунтовані актуальними дослідженнями в галузі інформаційних систем, управління бізнес-процесами та інтелектуального аналізу даних. Практичні рекомендації можуть бути використані для подальшої розробки схожих систем як у комерційному, так і в освітньому середовищі. Запропоноване рішення має потенціал масштабування та подальшого вдосконалення.

5. Наявність недоліків. Серед недоліків варто зазначити окремі неточності в оформленні роботи, часткові відхилення від стандартів під час побудови моделей, а також обмежену кількість кейсів для тестування розробленої системи. Водночас ці зауваження не суттєво впливають на загальну якість роботи та не знижують її високого професійного рівня.

6. Загальна оцінка кваліфікаційної магістерської роботи та її допущення до захисту перед ЕК. Кваліфікаційна магістерська робота Цуканової Тетяни Сергіївни виконана на високому науковому та професійному рівні, демонструє грамотність здобувачки та її готовність до вирішення реальних завдань у сфері інформаційних систем. Робота відповідає вимогам освітньо-професійної програми і заслуговує на високу оцінку. Рекомендується до захисту перед ДЕК.

Науковий керівник
Доцент кафедри ІСЕ ІІТЕ КНЕУ
Денісова О.О.

Рецензія

на кваліфікаційну магістерську роботу
здобувача вищої освіти
Цуканової Тятяни Сергіївни

Тема «Розроблення інформаційної управляючої системи для планування робочого процесу на підприємстві»

Актуальність теми кваліфікаційної магістерської роботи і доцільність її розроблення. *Підприємства активно переходять до цифрових форматів роботи, важливо впроваджувати сучасні системи, які допомагають автоматизувати внутрішні процеси — від організації робочих процесів до ведення звітності та налагодження внутрішньої комунікації. Це сприяє кращому управлінню ресурсами та відповідає сучасним вимогам ринку.*

Якість проведеного дослідження. *Дослідження було проведено на високому рівні, охоплюючи як теоретичний аналіз, так і практичну реалізацію системи. У роботі використано сучасні технології, обґрунтовано вибір архітектури й забезпечено масштабованість і надійність рішення. Отримані результати підтверджують практичну цінність і якість розробленої системи.*

Позитивні риси кваліфікаційної магістерської роботи. *Серед ключових переваг варто відзначити її високу практичну цінність, продуману структуру, актуальність теми, використання сучасних технологій, грамотну реалізацію архітектури та належну якість технічного виконання з акцентом на продуктивність і масштабованість.*

Зауваження. *Попри високий рівень виконання роботи, варто звернути увагу на деякі аспекти — зокрема, удосконалення UI/UX та розширення тестування в прикладних сценаріях.*

Практична значимість висновків і рекомендацій. *Висновки базуються на сучасних дослідженнях у сфері IT, що робить їх змістовними й обґрунтованими. Практичні рекомендації придатні для використання при створенні подібних систем як у бізнесі, так і в навчальних проєктах. Розроблена система має хороший потенціал для масштабування та подальшої модернізації*.

Місце роботи та посада рецензента

Науковий ступінь, учене звання (*за наявності*) Морозов Максим

Підпис засвідчую: _____

(посада, підпис)

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах динамічного розвитку сервісної індустрії та зростання попиту на оперативне обслуговування клієнтів особливого значення набувають інформаційні управляючі системи, здатні автоматизувати процеси бронювання, планування ресурсів і взаємодії з користувачем. Одночасно зростають вимоги до гнучкості, масштабованості та доступності рішень, що дозволяють підприємствам різного масштабу швидко адаптуватися до змін ринку й забезпечувати високий рівень сервісу. Особливо перспективною є інтеграція традиційних веб-інтерфейсів із сучасними месенджер-ботами, що відкриває нові канали для комунікації та знижує поріг входу для кінцевого користувача.

Мета і завдання роботи. Метою даної дипломної роботи є розроблення інформаційної управляючої системи для планування робочого процесу на підприємстві з урахуванням особливостей бізнес-процесу бронювання за допомогою веб-сайту та Telegram-бота. Для досягнення поставленої мети вирішуються наступні завдання:

1. Провести аналіз предметної області та сучасних підходів до створення ІУС різних категорій;
2. Обґрунтувати вибір технологічного стеку (Node.js, Angular, PostgreSQL, Redis) та архітектурних рішень (моноліт із модулями, контейнеризація);
3. Спроектувати логічну, фізичну та модульну структуру системи, розробити ключові UML- і Mermaid-діаграми;
4. Реалізувати базу даних із реляційними таблицями та OLAP-сховищем для аналітики, а також модуль інтелектуального аналізу даних і онтологічну базу знань;
5. Розробити програмне забезпечення системи, налаштувати CI/CD-конвеєр, провести тестування продуктивності та відмовостійкості;

6. Оцінити ефективність розробленого рішення та розробити рекомендації щодо його подальшого розвитку.

Об’єктом дослідження є процес планування робочого часу та бронювання ресурсів у сервісному підприємстві. Предметом дослідження — методи і технології створення інформаційних управляючих систем із багатоканальною взаємодією з користувачем.

Наукова новизна роботи полягає в комплексному поєднанні жадібного алгоритму оптимізації розкладу, підходів OLAP та Data Mining для аналітики, онтологічного представлення знань і інтеграції з Telegram-ботом на базі GenAI. Практичне значення полягає у створенні готового до впровадження прототипу, здатного забезпечити ефективне управління записами та ресурсами у реальному часі.

Методологічною основою дослідження стали методи системного аналізу, моделювання бізнес-процесів (BPMN, DFD, UML), структурного проєктування баз даних та алгоритмів оптимізації. Для реалізації використовувались сучасні інструментальні засоби: Node.js/Express, Angular, PostgreSQL, Redis, Docker, Kubernetes, GitHub Actions.

Структура роботи. Дипломна робота складається із вступу, трьох розділів, висновків та переліку використаних джерел. У першому розділі представлено дослідження предметної області та аналіз існуючих систем; у другому — характеристику обраних методів, моделей та архітектури інформаційної управляючої системи; у третьому — проєктування та реалізацію всіх компонентів рішення, включаючи базу даних, модуль інтелектуального аналізу, програмне забезпечення та технічне забезпечення. Висновки підбивають підсумки дослідження та окреслюють перспективи подальшого розвитку системи.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз існуючих інформаційних систем (предметної області)

1.1.1 Дослідження предметної області

Предметна область дослідження охоплює сукупність інформаційних управляючих систем (ІУС), що спрямовані на автоматизацію комплексних бізнес-процесів і підтримку прийняття рішень на рівні всіх ієрархічних ланок підприємства. Під ІУС традиційно розуміють інтегроване середовище, яке об’єднує апаратні засоби, системне та прикладне програмне забезпечення, а також комунікаційні модулі для обміну даними з зовнішніми сервісами. З метою забезпечення своєчасного й коректного управління ресурсами організації воно реалізує функції збору, зберігання, обробки та візуалізації інформації про стан виробничих, адміністративних і сервісних процесів [1].

Складовими ІУС є:

- База даних – централізоване сховище структурованих даних про клієнтів, персонал, обладнання, послуги та фінансові операції;
- Модуль бізнес-логіки – набір алгоритмів і правил, які реалізують внутрішні процедури обробки запитів, планування й взаємодії між підсистемами;
- Інтерфейс користувача (UI/UX) – веб- або мобільний клієнт для операторів та адміністраторів, а також чат-бот для кінцевих споживачів, що гарантує інтуїтивну навігацію та зручність введення даних;
- Комунікаційні шлюзи – інтеграція з платіжними системами, SMS/Email-провайдерами, месенджерами (Telegram API тощо) та корпоративними ERP/CRM через REST/WebSocket або GraphQL [2].

Основні бізнес-процеси, що обслуговуються ІУС у контексті планування робочого процесу на підприємстві, включають:

1. Запис клієнтів
    - Попереднє бронювання сеансів або консультацій через веб-інтерфейс та Telegram-бота;
    - Динамічне підтвердження або зміну часу зустрічей у режимі реального часу.
2. Планування ресурсів
    - Формування та оптимізація графіків роботи персоналу з урахуванням навантаження та кваліфікації;
    - Розподіл обладнання та приміщень відповідно до поточних та прогнозованих замовлень.
3. Облік відвідуваності й виконаних замовлень
    - Реєстрація приходу/виходу клієнтів із фіксацією часу й результату обслуговування;
    - Автоматичне формування звітів про фактичну продуктивність і завантаженість.
4. Аналітика та звітність
    - Генерація статистичних звітів, дашбордів та графіків для керівництва;
    - Використання вбудованих BI-інструментів для прогнозування попиту й виявлення вузьких місць [3].

До ключових стейкхолдерів ІУС належать:

- Адміністратори системи, що відповідають за розгортання та підтримку інфраструктури, управління ролями й політиками безпеки;
- Оператори (менеджери з обслуговування клієнтів), які вводять і коригують дані, контролюють виконання замовлень і взаємодіють з клієнтами в разі змін;

- Кінцеві користувачі (клієнти), для яких розроблено зручні канали запису та комунікації (мобільний веб-додаток, чат-бот), що скорочують час очікування та підвищують лояльність;
- Сторонні сервіси (банківські шлюзи, SMS/Email-провайдери, ERP/CRM), що потребують двостороннього обміну даними для фінансової звітності, відправлення сповіщень і синхронізації клієнтської бази [1, 3].

Завдяки поєднанню зазначених компонентів та процесів ІУС забезпечує цілісність інформаційного простору підприємства, підвищує прозорість операцій та прискорює прийняття обґрунтованих управлінських рішень.

Інформаційні управляючі системи можуть бути адаптовані під специфіку різних секторів економіки, що визначає їх функціональні вимоги та інтеграційні можливості.

- Охорона здоров’я: системи електронного реєстру пацієнтів, обробки медичних записів (EMR/EHR), планування прийому лікарів, інтерфейси для медичного обладнання й телемедицини [1];
- Торгівля та сфера послуг: CRM-модулі для управління лояльністю клієнтів, онлайн-бронювання, інтеграція з платіжними шлюзами і складами [2];
- Виробництво: MES-платформи для відстеження стану виробничих ліній, управління запасами сировини, оптимізації логістики та технічного обслуговування обладнання [3];
- Логістика й транспорт: системи диспетчеризації, відстеження вантажів, планування маршрутів і прогнозування навантажень.

Кожна галузева версія містить галузево-специфічні модулі: наприклад, у медицині — облік клінічних досліджень та запис на процедури; у торгівлі — обробку замовлень і повернень; у виробництві — контроль якості й планування технічного обслуговування.

Архітектурні рішення визначають здатність системи до розширення, масштабування й інтеграції:

- Монолітні системи: всі компоненти (UI, бізнес-логіка, доступ до даних) об’єднані в єдиному виконуваному блоці. Переваги — простота розгортання та налагодження; недоліки — складність модульного оновлення та масштабування [4];
- Мікросервісні системи: кожен сервіс відповідає за окремий функціональний модуль (бронювання, повідомлення, аналітика) і взаємодіє з іншими через API (REST, gRPC). Це дозволяє незалежне розгортання, горизонтальне масштабування та гнучкі оновлення [5];
- Хмарні рішення (Cloud-native): сервіси розгортаються в контейнерах (Docker, Kubernetes) зі вбудованими механізмами автоскейлінгу, балансування навантаження й відмовостійкості. Забезпечують економію ресурсів і високу доступність [6].

Рівень інтелектуалізації відображає використання методів аналітики та AI/ML-алгоритмів:

- Традиційні ІУС: базуються на жорстко закодованих бізнес-правилах і простих звітних модулях без прогнозної аналітики;
- Системи підтримки прийняття рішень (DSS): містять інструменти what-if аналізу, сценарного моделювання та побудови рекомендацій на основі історичних даних [7];
- ІУС із вбудованими AI/ML-модулями: реалізують машинне навчання для прогнозування кадрового навантаження, автоматичної маршрутизації запитів, класифікації клієнтських звернень та адаптивного налаштування робочих процесів в реальному часі [8].

Функціональні модулі інформаційної управляючої системи утворюють набір самостійних підсистем, кожна з яких відповідає за реалізацію окремого бізнес-процесу та забезпечує чіткий розподіл відповідальності між компонентами. Така модульна структура підвищує

гнучкість, полегшує підтримку й масштабування системи в умовах динамічних вимог підприємства.

Модуль управління записами і зустрічами відповідає за реєстрацію заявок клієнтів, планування та коригування графіків обслуговування. Він обробляє запити на бронювання через веб-інтерфейс або чат-бота, перевіряє доступність часових слотів із урахуванням ресурсів (персонал, приміщення, обладнання) та автоматично надсилає підтвердження або запрошення на зміну часу. Використання стандартизованих календарних API (Google Calendar, Microsoft Outlook) забезпечує синхронізацію дата-майстрів та мінімізує конфлікти при одночасних запитах.

Модуль обліку та звітності призначений для збору, збереження та агрегування даних про відвідуваність клієнтів, виконані замовлення й фінансові операції. На його основі генеруються стандартні таблиці та графіки завантаженості, створюються дашборди для керівництва й готуються звіти у форматах PDF, Excel або через API для інтеграції з зовнішніми BI-інструментами. Цей модуль також відповідає за аналіз відхилень від встановлених KPI та формування рекомендацій для оптимізації процесів.

Модуль повідомлень забезпечує двосторонній обмін сповіщеннями із зацікавленими сторонами через SMS, e-mail та push-нотифікації. Інтеграція зі сторонніми провайдерами здійснюється через протоколи SMPP, SMTP або REST API. Система підтримує шаблони повідомлень, планування відправлень (нагадування про зустрічі, опитування задоволеності), відстеження статусів доставки та обробку зворотного зв’язку, що підвищує рівень комунікації з клієнтами та персоналом.

Інтеграція з Telegram-ботом реалізує зручний інтерфейс для кінцевих користувачів у популярному месенджері. Бот проводить діалог, запитує необхідні дані (ім’я, контактний номер, бажаний час зустрічі), перевіряє доступні слоти та передає підтверджені записи в основну базу через REST

або WebSocket. Такий підхід суттєво зменшує навантаження на операторів і забезпечує високий рівень доступності сервісу незалежно від платформи клієнта.

Нефункціональні вимоги виглядають так:

1. Масштабованість і продуктивність

Забезпечення масштабованості та високої продуктивності ІУС передбачає не лише здатність системи обробляти збільшені обсяги запитів, але й гарантувати стабільно низькі часи відповіді за будь–якого навантаження. Для цього застосовуються такі підходи:

- Горизонтальне масштабування мікросервісів із розподілом обчислювального навантаження через API-шлюзи та балансувальники (NGINX, HAProxy). Кожен сервіс може бути запущений у кількох копіях, що дозволяє обробляти паралельні запити без блокувань.
- Кешування на різних рівнях – у пам’яті (Redis, Memcached) для часто запитуваних об’єктів, а також CDN-рішення для віддачі статичних ресурсів. Це знижує навантаження на базу даних і зменшує затримку клієнтських запитів.
- Асинхронна обробка подій через черги повідомлень (RabbitMQ, Apache Kafka) дозволяє відокремити швидкі “write” операції від повільніших циклів аналітики чи відправлення сповіщень. Такий підхід підтримує високий рівень пропускної здатності й запобігає “заторам” при пікових навантаженнях.
- Моніторинг та автоскейлінг: інструменти Prometheus + Grafana для збору метрик (CPU, пам’ять, час відповіді сервісів), які в поєднанні з Kubernetes Horizontal Pod Autoscaler забезпечують автоматичне збільшення/зменшення кількості реплік залежно від завантаженості.

2. Безпека даних і конфіденційність

Забезпечення інформаційної безпеки є критичною нефункціональною вимогою, що включає захист даних «at rest» і «in transit», контроль доступу та аудит:

- Шифрування даних: застосування AES-256 для збережених даних у базах (Transparent Data Encryption) та TLS 1.3 для захищеного каналу передачі між клієнтом, сервісами та базою даних.
- IAM і RBAC: централізоване управління ідентифікацією та доступом через протоколи OAuth 2.0 / OpenID Connect, із деталізованими ролями (адміністратор, оператор, аналітик), які визначають права на перегляд, створення й модифікацію ресурсів [4].
- Аудитування й логування: збереження незмінних (WORM) записів про всі дії користувачів і системних подій із використанням ELK-стеку (Elasticsearch, Logstash, Kibana) для швидкого пошуку й аналізу інцидентів. Політики зберігання логів повинні відповідати вимогам GDPR і ISO 27001.
- Резервне копіювання та відновлення: автоматизовані щоденні бекапи баз даних і критичних сервісів із збереженням трьох рівнів знімків (daily, weekly, monthly) на окремих гео-дистрибуційованих сховищах AWS S3 або Azure Blob Storage.
- Тестування безпеки: регулярні пентести та DAST/SAST-сканування (OWASP ZAP, SonarQube) для виявлення вразливостей на рівні коду та конфігурацій.

3. UX/UI: простота взаємодії для клієнта й адміністратора

Інтерфейс має забезпечувати швидкий доступ до ключових функцій, інтуїтивну навігацію та відповідати принципам веб-доступності:

- Мобільно-перший дизайн (Mobile-First): адаптивна верстка за допомогою CSS-фреймворків (Bootstrap, Tailwind) та компонентних бібліотек (Material UI, Ant Design) гарантує коректне відображення на смартфонах, планшетах і десктопах.

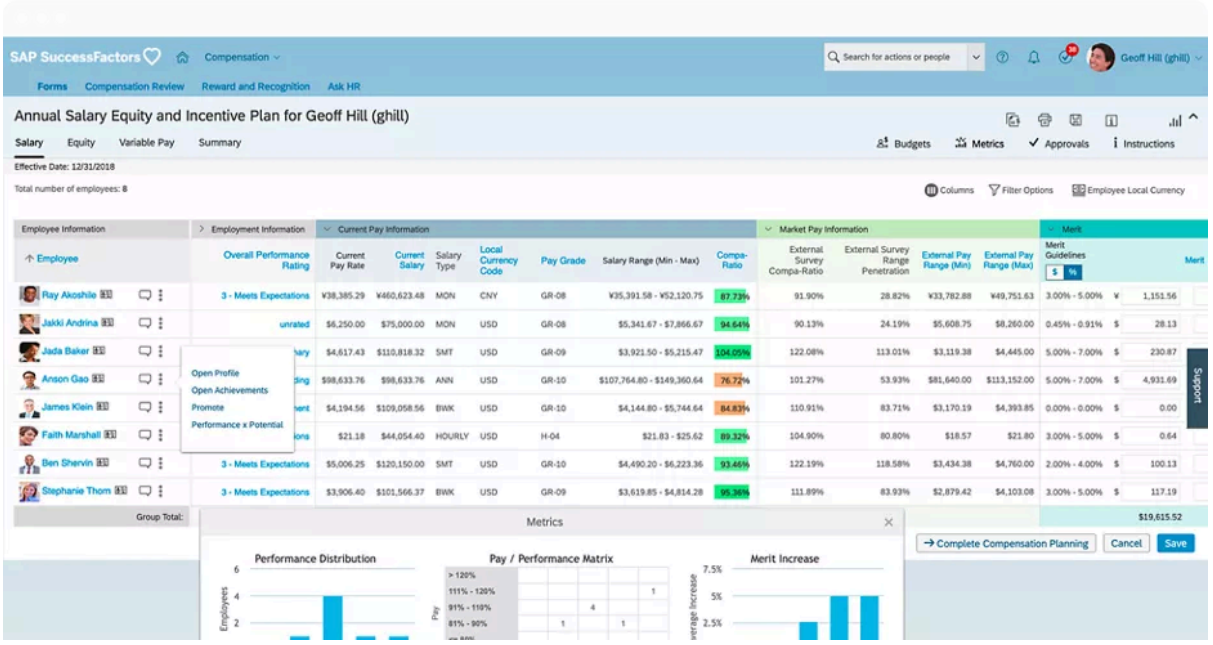
- Консистентність і система дизайну: розробка бібліотеки UI-компонентів із чіткими гайдлайнами (типографіка, кольорова палітра, відступи), що спрощує підтримку та масштабування кількості екранів.
- Простота форм і взаємодії: мінімальна кількість полів у формі бронювання (не більше 4–5 обов’язкових), інлайн-валідатори, автозаповнення даних (Cookie, LocalStorage) і підказки (tooltip), що знижують ймовірність помилки користувача.
- Доступність (Accessibility): відповідність стандарту WCAG 2.1 (контрастність, навігація клавіатурою, альтернативні тексти для зображень) підвищує комфорт користувачів із різними потребами.
- UX-тестування й аналітика: проведення регулярних юзабіліті-тестів із реальними користувачами, аналіз кліків/шляхів через Google Analytics або Hotjar, що дозволяє швидко ітеративно покращувати інтерфейс.

1.1.2 Дослідження інформаційних управляючих систем

Комерційні рішення, такі як SAP SuccessFactors і Oracle HCM Cloud, призначені для комплексного управління людським капіталом та бізнес-процесами великих організацій. SAP SuccessFactors включає модулі Employee Central, Talent Management, Learning та Workforce Analytics із розвиненим REST-API для інтеграції з іншими SAP-системами та сторонніми сервісами [9]. Oracle HCM Cloud, що входить до складу Oracle Fusion Cloud Applications Suite, охоплює Core HR, Talent Management, Workforce Management і Compensation, має єдину панель адміністрування та підтримує локальні регуляторні вимоги [10]. Такі платформи забезпечують високу ступінь безпеки, відмовостійкості та можливість глибокої кастомізації, але впровадження вимагає значних інвестицій у ліцензії, апаратну інфраструктуру та залучення сертифікованих консультантів.

Для прикладу інтерфейсу SAP SuccessFactors .

Рисунок 1.1 - Приклад інтерфейсу



Хмарні CRM/ERP-платформи, такі як Zoho CRM і Microsoft Dynamics 365, надаються за моделлю підписки без необхідності власних серверів чи підтримки інфраструктури. Zoho CRM пропонує модулі управління лідами, бронюванням, автоматизацією маркетингу та інтеграцією з каналами комунікації; початкова документація доступна на порталі Zoho Help [11]. Microsoft Dynamics 365 включає програми для Sales, Customer Service, Field Service і Customer Insights із вбудованими AI-інструментами для прогнозування попиту; повний набір інструкцій є на Microsoft Learn [12]. Хмарна модель дозволяє швидко масштабуватися, отримувати автоматичні оновлення й мінімізувати CapEx, проте накладає обмеження на глибину кастомізації та робить систему залежною від SLA провайдера й якості інтернет-зв’язку.

Приклад робочого інтерфейсу Zoho CRM

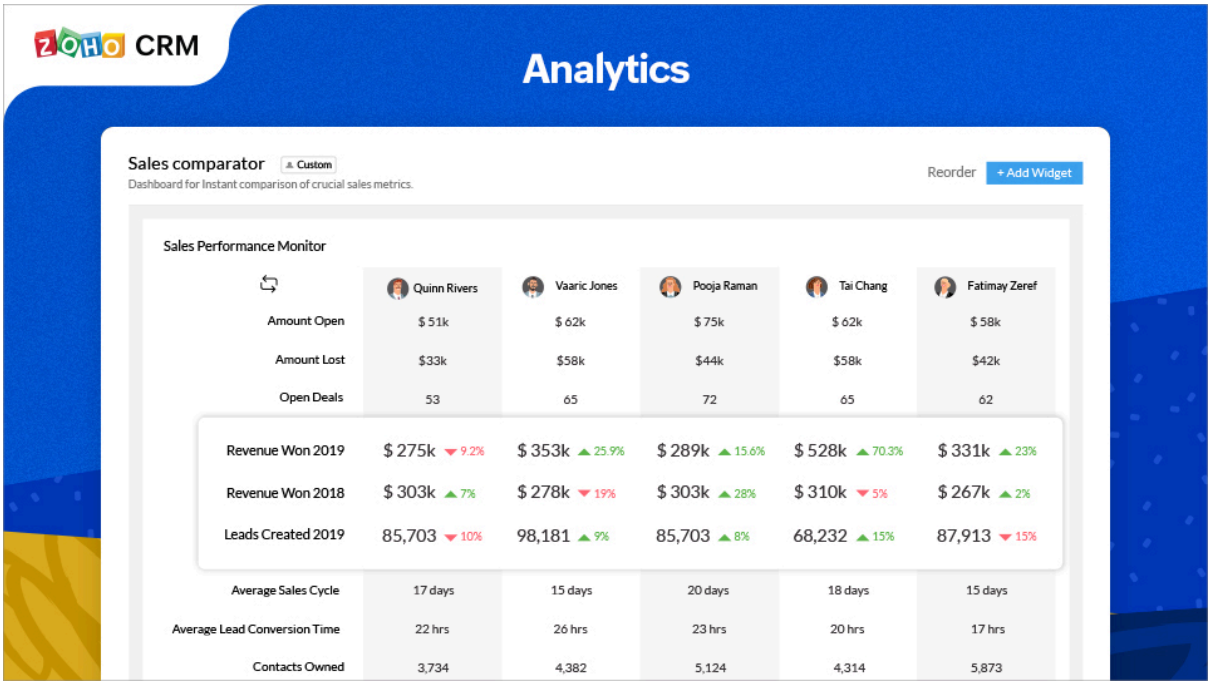


Рисунок 1.2 - Приклад інтерфейсу ZOHO CRM.

Платформи з відкритим кодом, зокрема Odoo та OrangeHRM, забезпечують повний доступ до вихідних текстів і гнучку модульну архітектуру, що дозволяє організаціям адаптувати систему під власні бізнес-процеси без обмежень ліцензійних зобов’язань. Odoo надає понад 30 базових додатків (CRM, ERP, Inventory, Accounting тощо), а також велику екосистему сторонніх модулів у маркетплейсі Odoo Apps, що покривають функції e-commerce, POS, виробничого обліку та проектного менеджменту. Всі компоненти реалізовані на Python із клієнтською частиною на JavaScript (OWL), що спрощує розробку нових модулів і інтеграцію зі сторонніми сервісами. Документація містить детальні гіди з встановлення, налаштування базових модулів та розробки кастомних додатків, що дозволяє команді впровадження швидко отримати необхідні навички й підтримувати систему в актуальному стані без залежності від зовнішніх консультантів [13].

OrangeHRM спеціалізується на управлінні персоналом та включає ядро з модулями рекрутингу, відпусток, оцінювання ефективності та навчання, а також інструмент для управління графіками роботи. Її

REST-API дозволяє підключати системи електронного документообігу, платіжні шлюзи та аналітичні сервіси. OrangeHRM випускає як безкоштовну Community Edition із базовим набором функцій, так і платну Professional/Enterprise Editions із розширеними модулями (наприклад, управління компенсацією та скаутингу талантів), які регулярно оновлюються через систему пакунків Composer. Підтримка спільноти, форуми та регулярні вебінари сприяють швидкому вирішенню технічних питань і обміну досвідом серед користувачів [14].

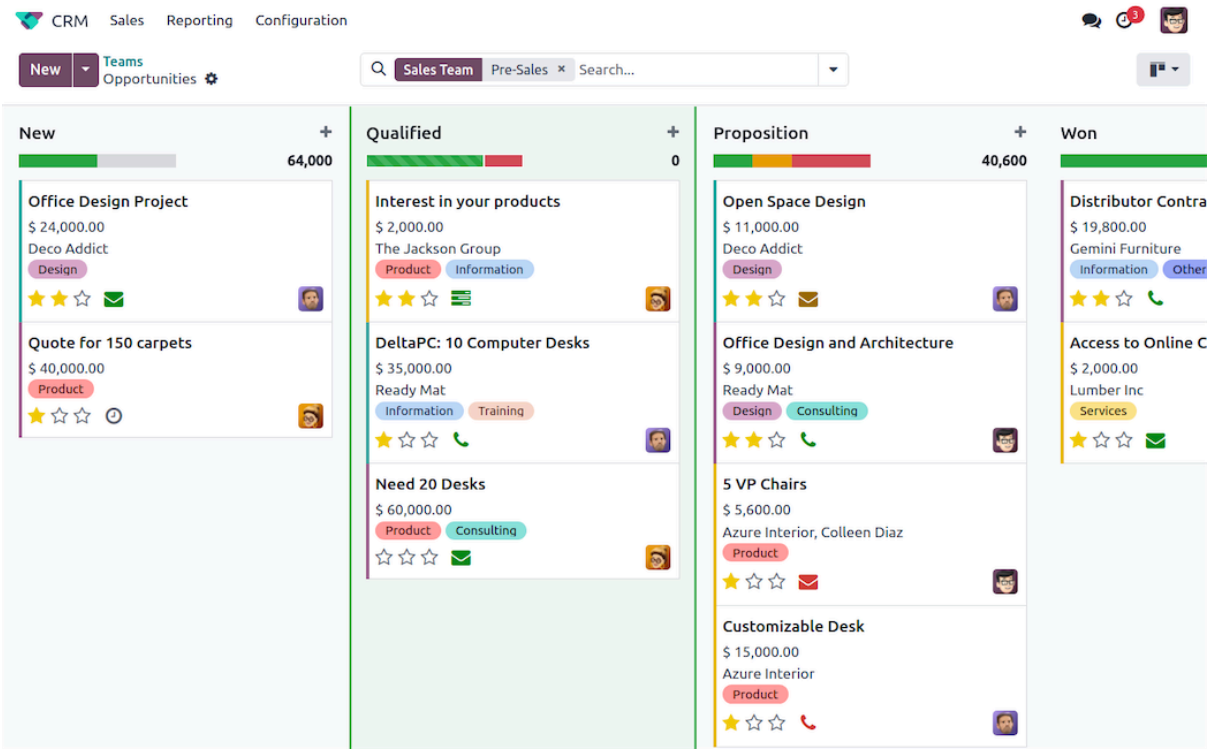


Рисунок 1.3 - Приклад інтерфейсу Odoo CRM.

Усі сучасні платформи пропонують базовий функціонал для планування зустрічей і побудови звітів, але відрізняються глибиною налаштувань, готовими шаблонами аналітики й рівнем відкритості API. Комерційні рішення (SAP, Oracle) мають найширший набір галузевих розширень, SaaS-сервіси (Zoho, Dynamics) — швидке оновлення й простоту запуску, а open-source (Odoo, OrangeHRM) — максимальну гнучкість кастомізації за рахунок доступу до вихідного коду.

Таблиця 1.1 - Порівняльний аналіз CRM.

Платформа	Модулі бронювання та календаря	Інструменти звітності та аналітики	Розширення через API
SAP SuccessFactors	Повноцінний Work Schedule із підтримкою черг, відпусток і черговості; інтеграція з корпоративними календарями (Exchange, Google)	Workforce Analytics, готові дашборди й KPI-звіти; можливість створення кастомних звітів через SAP Analytics Cloud	REST/SOAP API із детальною документацією, SDK для Java/.NET
Oracle HCM Cloud	Core HR Calendar з гнучким налаштуванням правил планування змін і відпусток; інтеграція з Oracle Calendar	BI Publisher, OTBI (Oracle Transactional Business Intelligence) для створення ad-hoc й регулярних звітів	REST API, SOAP Web Services, Oracle Integration Cloud Service
Zoho CRM	Модуль Meetings для онлайн-бронювання з клієнтами; інтеграція з Google/Outlook Calendar, автонотифікації	Zoho Analytics із готовими шаблонами дашбордів, drag-and-drop конструктор звітів	REST API, Webhooks, SDK для Java, Python, Node.js
Microsoft Dynamics 365	Bookings App для призначення зустрічей; інтеграція з Outlook та Teams; мобільні нагадування	Power BI інтеграція, готові звіти з продажів і сервісу, можливість створювати складні візуалізації	Web API (OData), Power Platform Connectors, Custom Actions
Odoo (Open-Source)	Calendar та Appointments у CRM; можливість створити власні слоти та графіки через Studio; інтеграція з Google Calendar	Odoo BI (Reporting) із попередньо налаштованими звітами, конструктор звітів на базі QWeb та Excel-експорт	REST API (JSONRPC), XMLRPC, можливість додавати власні контролери та модулі у Python
OrangeHRM (Open-Source)	Leave Module з календарем відпусток і графіками змін; розширення через Community плагіни для бронювання інтерв'ю та тестувань	Basic Reports (відпустки, присутність), Dashboard; Pro-версія додає Performance Analytics	REST API для CRUD-операцій із модулями, Webhooks у Enterprise Edition

Відповідно до проведеного аналізу, для великих корпорацій із високими вимогами до глибинної аналітики, масштабованості та суворого дотримання галузевих регуляторних вимог оптимальними рішеннями є SAP SuccessFactors та Oracle HCM Cloud. Ці платформи забезпечують потужні інструменти Workforce Analytics, BI Publisher та OTBI для створення детальних звітів і дашбордів, а також розвинені механізми

інтеграції через REST/SOAP API, що гарантує безперебійну роботу в багатокомпонентному середовищі великих підприємств [9][10].

Середній бізнес, який прагне швидкого розгортання, мінімізації капітальних витрат і простоти підтримки, може обрати хмарні SaaS-рішення на зразок Zoho CRM або Microsoft Dynamics 365. Ці платформи пропонують готові модулі бронювання й автоматизації маркетингу, інтеграцію з календарями та AI-функції для прогнозування, не вимагаючи власної інфраструктури та складних процедур впровадження [11][12].

Організації з наявними внутрішніми IT-ресурсами та потребою повної гнучкості в налаштуванні бізнес-логіки й інтерфейсів доцільно звернутися до open-source платформ Odoo або OrangeHRM. Завдяки відкритому коду, модульній архітектурі та активній спільноті користувачів вони дозволяють адаптувати функціонал під унікальні процеси підприємства й масштабувати систему без обмежень ліцензійної політики [13][14].

Серверна частина On-Premise рішень базується на платформах Java або .NET, що забезпечують стабільність, масштабованість та відповідність корпоративним стандартам безпеки (SAP SuccessFactors, Oracle HCM) . Хмарні SaaS-платформи реалізовані на Node.js (Zoho CRM) та .NET Core (Microsoft Dynamics 365) для швидкого розгортання і легкого масштабування в контейнерах чи serverless-середовищах . Відкриті рішення використовують Python (Odoo) для бізнес-логіки та модульної архітектури, тоді як OrangeHRM націлена на PHP-стек із REST-API для інтеграцій .

Для зберігання даних On-Premise рішення традиційно застосовують Oracle DB, яка забезпечує високий рівень транзакційної цілісності й можливості кластеризації . SaaS-сервіси, зокрема Dynamics 365 Customer Insights, використовують NoSQL-сховища для аналітики великих обсягів

даних у реальному часі . Open-source платформи покладаються на PostgreSQL або MySQL, гарантуючи широкий вибір інструментів адміністрування та реплікації .

Інтеграційні можливості охоплюють REST і SOAP для On-Premise (SAP, Oracle) , REST і WebSocket для SaaS-платформ (Zoho, Dynamics) , а також GraphQL (Odoo) і REST (OrangeHRM) для open-source рішень .

On-Premise рішення традиційно ліцензуються на користувача або модуль зі стартовою вартістю близько 100 USD за одного користувача на місяць, що не включає витрат на апаратне забезпечення та сервісну підтримку . SaaS-subscription для Zoho CRM починається від 12 USD/користувач/місяць із можливістю вибору щомісячного або річного плану ; Microsoft Dynamics 365 мають багаторівневі пакети, які визначаються функціональністю й обсягом користувачів .

Витрати на впровадження On-Premise можуть сягати до двохразової вартості ліцензії через необхідність налаштування інфраструктури, кастомізації та навчання персоналу . Для SaaS-рішень витрати на розгортання та підтримку є середніми, адже провайдер бере на себе управління платформою, тоді як Open-Source рішення потребують мінімальних ліцензійних витрат, але вимагають інвестицій у консультаційні та розробницькі послуги для підтримки та оновлення системи .

Таблиця 1.2 - Порівняльний огляд технологічного стеку, ліцензування та вартості

Категорія рішення	Серверна частина	Бази даних	Інтеграції	Ліцензії	SaaS-підписка	Витрати на впровадження
On-Premise(SAP SuccessFactors, Oracle HCM)	Java / .NET	Oracle DB	REST / SOAP	від 100 USD/корист./міс.	–	висока (до 2× вартості ліцензії)
SaaS(Zoho CRM, Microsoft Dynamics 365)	Node.js (Zoho) / .NET Core (Dynamics)	NoSQL (Dynamics 365 Customer Insights)	REST, WebSocket	–	Zoho від 12 USD/корист./міс. ;Dynamics – за планом	середня
Open-Source(Odoo, OrangeHRM)	Python (Odoo) / PHP (OrangeHRM)	PostgreSQL / MySQL	GraphQL (Odoo), REST (OrangeHRM)	–	–	низька (витрати на консультації)

У контексті розробки інформаційної управляючої системи з інтегрованим Telegram-ботом доцільно виділити такі аспекти:

Сильні сторони. По-перше, відкриті платформи Odoo та OrangeHRM мають нативну підтримку REST API і Webhook, що суттєво спрощує реалізацію двосторонньої взаємодії із Telegram-ботом без додаткових проміжних сервісів. По-друге, SaaS-рішення Zoho CRM і Microsoft Dynamics 365 забезпечують оперативне впровадження та безперервні оновлення без простоїв, оскільки всі технічні аспекти підтримки інфраструктури лежать на постачальнику послуги. По-третє, можливості глибокої кастомізації відкритих систем завдяки вільному доступу до вихідного коду й модульній архітектурі дозволяють адаптувати бізнес-логіку та інтерфейс під унікальні вимоги підприємства.

Слабкі сторони. Водночас у хмарних SaaS-рішеннях обмежені можливості внесення змін у базову архітектуру та алгоритми обробки даних, що може ускладнити реалізацію вузькоспеціалізованих бізнес-процесів. Крім того, впровадження On-Premise ERP/HCM-пакетів (SAP SuccessFactors, Oracle HCM) пов’язане з істотними капіталовкладеннями в ліцензії, апаратне забезпечення й послуги сертифікованих консультантів, що значно підвищує загальну вартість проєкту.

Можливості. Галузеві провайдери продовжують розвивати AI/ML-модулі для прогнозування навантажень і персоналізації сервісів (наприклад, Dynamics 365 Customer Insights або Oracle Adaptive Intelligence), що відкриває нові горизонти для підвищення ефективності процесів і зниження операційних витрат. Перехід до хмарних архітектур із контейнеризацією та serverless-компонентами (за фреймворками Kubernetes і AWS Lambda) створює умови для гнучкого масштабування та оптимізації інфраструктури відповідно до навантаження.

Загрози. Висока конкуренція на ринку легковагових SaaS-рішень і open-source платформ призводить до швидкого морального старіння функціоналу, якщо не підтримувати систему в актуальному стані. Окрім того, посилення регуляторних вимог до зберігання та обробки персональних даних (наприклад, GDPR та національні стандарти) може вимагати додаткових інвестицій у шифрування й аудит, що підвищує складність технічної реалізації та супроводу системи .

1.2 Обґрунтування вибору підходів і технологій для створення інформаційної управляючої системи

У науковій літературі наголошують, що саме набір функціональних можливостей є первинним чинником вибору ІУС для автоматизації клієнтських процесів [1][2]. До ключових показників належать модулі

бронювання та календаря, механізми підтвердження й скасування зустрічей, інструменти генерації дашбордів і звітів, а також засоби інтеграції з каналами комунікації (email, SMS, месенджери). Відповідно до принципів REST-архітектури, описаних у дисертації Р. Філдінга, коректне проєктування API гарантує злагоджену роботу клієнтського інтерфейсу й бекенду при викликах функціональних модулів [15].

Масштабованість, продуктивність і безпека визначають «якість» ІУС як платформи для інтенсивного навантаження. Горизонтальне масштабування мікросервісів, балансування навантаження та кешування зменшують латентності на рівні десятків мілісекунд, а відповідність вимогам ISO/IEC 27001 щодо шифрування «in transit» та «at rest» забезпечує захист конфіденційних даних [4][6][17]. До того ж, оцінка продуктивності за допомогою метрик часу відповіді та пропускної здатності, яку пропонує Ф. Басс дозволяє порівняти альтернативні архітектурні варіанти та вибрати найефективніший [16].

Високий рівень відкритості архітектури та модульність системи прискорюють адаптацію під унікальні бізнес-процеси, але водночас подовжують терміни розгортання через необхідність розробки та тестування додаткових компонентів. Практичні настанови з Domain-Driven Design рекомендують балансувати між «з коробки» і кастомними рішеннями за рахунок ретельного відбору вбудованих модулів та мінімізації змін у базовій платформі [5][16].

Загальна вартість володіння (TCO) охоплює капітальні витрати на ліцензії та інфраструктуру, операційні витрати на супровід і оновлення, а також соціальні витрати на навчання персоналу. Згідно з дослідженнями Khan et al., cloud-native підходи дозволяють знизити CapEx, але можуть збільшити OpEx за рахунок частих оновлень, тоді як експлуатаційні витрати On-Premise ERP, як показав В. Боем, можуть перевищувати

початкові інвестиції вдвічі–втричі протягом життєвого циклу проєкту [6][18].

Урахування компетенцій розробників і адміністраторів суттєво знижує ризики затримок і невідповідності функціоналу вимогам. S. МакКоннелл у праці Code Complete підкреслює, що вибір технологій у межах існуючого досвіду команди зменшує вартість розробки та підвищує якість коду [3][19].

Для реалізації серверної частини інформаційної управляючої системи обрано технологію Node.js[20]. Цей вибір обґрунтований високою продуктивністю та ефективністю Node.js, що базується на неблокуючому, подієво-орієнтованому підході. Це дозволяє ефективно обробляти значну кількість одночасних запитів як від клієнтського фронтенду, так і від зовнішніх сервісів, зокрема Telegram API. Крім того, використання JavaScript як єдиної мови програмування на фронтенді (Angular) і бекенді спрощує процес розробки та обмін досвідом між частинами проєкту. Node.js має багату екосистему пакетів (npm), що надає готові рішення для роботи з базами даних, реалізації REST API, обробки даних у форматі JSON та інтеграції з Telegram.

Архітектурно серверна частина буде реалізована як монолітний додаток, використовуючи легковаговий фреймворк Express.js[21]. Цей підхід забезпечує достатню структуру для поточного масштабу проєкту в рамках дипломної роботи, водночас дозволяючи чітко виділити функціональні модулі для обробки різних типів запитів. Розгортання серверного додатку планується здійснювати за допомогою Docker[22], що забезпечить ізоляцію, портативність та спростить управління залежностями у різних середовищах. Серверний бік відіграє ключову роль у прийомі даних від Telegram-бота через вебхуки, збереженні інформації про бронювання в базі даних та наданні необхідних даних фронтенду для відображення на сайті.

Для надійного та коректного зберігання даних в інформаційній системі обрано реляційну базу даних PostgreSQL[23]. Головним аргументом на користь PostgreSQL є її повна підтримка ACID-транзакцій, що є критично важливим для забезпечення цілісності даних, особливо при роботі з операціями бронювання та веденням звітності. ACID-властивості гарантують, що кожна транзакція (наприклад, створення нового запису на прийом) буде виконана атомарно, послідовно, ізольовано та стійко, унеможливлюючи стан неконсистентності даних. PostgreSQL також є відомою своєю високою надійністю, стійкістю та відповідністю стандартам SQL. Реляційна модель даних добре підходить для структурованого зберігання інформації про клієнтів, послуги та деталі бронювань, дозволяючи ефективно управляти зв'язками між цими сутностями.

В якості додаткового компоненту, що може використовуватися для оптимізації продуктивності та реалізації асинхронних задач, розглядається застосування Redis[24]. Redis може слугувати як високоефективний кеш для тимчасового зберігання даних, що часто запитуються (наприклад, доступних часових слотів), зменшуючи навантаження на основну базу даних. Крім того, Redis може бути використаний як брокер повідомлень або черга для асинхронної обробки задач, таких як відправлення автоматичних нагадувань про майбутні бронювання, що може бути реалізовано за допомогою бібліотек типу BullMQ. Таким чином, PostgreSQL виступає як основне персистентне сховище даних, а Redis може доповнювати функціонал для підвищення швидкодії та реалізації фонових процесів.

Взаємодія між різними компонентами системи та інтеграція із зовнішніми сервісами реалізується через декілька ключових механізмів. Основним каналом взаємодії між фронтендом та бекендом є REST API. Цей інтерфейс використовується для виконання стандартних

CRUD-операцій, дозволяючи фронтенду отримувати дані про наявні бронювання для відображення на сайті, а також потенційно додавати або редагувати записи через адміністративний інтерфейс. Для забезпечення миттєвого оновлення даних на клієнтському боці, особливо коли нове бронювання додається через інші канали (наприклад, Telegram-бота), планується використання технології WebSocket. Це дозволить серверу активно надсилати оновлення клієнтам без необхідності постійного опитування (пулінгу) з боку фронтенду, забезпечуючи актуальність інформації про розклад у реальному часі.

Ключовою інтеграцією для даного проєкту є взаємодія з Telegram-ботом. Ця інтеграція реалізується за допомогою механізму вебхуків (webhook), наданого Telegram API. Серверна частина системи налаштовується на прийом HTTP POST-запитів від Telegram щоразу, коли відбувається подія, пов'язана з ботом (наприклад, користувач відправив повідомлення). Для спрощення обробки цих вебхуків на стороні Node.js буде використана спеціалізована бібліотека для роботи з Telegram API, така як Telegraf[25] або node-telegram-bot-api[26]. Цей канал є основним для ініціювання процесу бронювання: після діалогу з користувачем, який може бути керований за допомогою інтеграції з Google GenAI для обробки природної мови, бот формує запит до бекенду (через отриманий вебхук), який обробляється, валідується та зберігається в базі даних. Після успішного збереження бекенд може ініціювати надсилання повідомлення через WebSocket до підключених фронтенд-клієнтів для оновлення відображення записів на сайті салону, таким чином реалізуючи асинхронну обробку та поширення даних.

Клієнтська частина інформаційної системи розробляється на базі сучасного JavaScript-фреймворку Angular[27]. Вибір Angular обумовлений, зокрема, тим фактом, що поточний вебсайт салону вже реалізовано на цій технології, що дозволяє ефективно використовувати наявну кодову базу,

компоненти та експертизу розробників. Фронтенд являє собою односторінковий додаток (SPA) з чіткою модульною структурою. Для оптимізації продуктивності та прискорення початкового завантаження застосунку застосовується механізм лінивого завантаження (Lazy Loading) для окремих функціональних модулів, таких як перегляд списку бронювань або розділ профілю клієнта.

Для побудови якісного та консистентного користувацького інтерфейсу використовується бібліотека UI-компонентів Angular Material. Це забезпечує єдиний візуальний стиль для всіх елементів, включаючи списки відображення бронювань та потенційні форми для взаємодії. Для ефективного управління асинхронними операціями (наприклад, запитами до бекенду через REST API та обробкою даних з WebSocket) широко використовується бібліотека RxJS. Управління станом застосунку, включаючи централізоване зберігання даних про бронювання, може бути реалізовано за допомогою бібліотеки NgRx, яка реалізує патерн управління станом на основі Redux, хоча для обсягу дипломної роботи може бути достатньо спрощеного підходу з використанням RxJS.

Система також розробляється з урахуванням можливостей Progressive Web Application (PWA)[32]. Це потенційно дозволить користувачам (наприклад, співробітникам) отримати базовий доступ до інформації (кешованої) в офлайн-режимі та реалізувати механізм push-нотифікацій, що може бути корисним для сповіщення про нові бронювання, додані через Telegram-бота. Особлива увага приділяється доступності (Accessibility) відповідно до стандартів WCAG 2.1[33], включаючи використання ARIA-тегів та забезпечення клавіатурної навігації, щоб зробити систему зручною для максимально широкого кола користувачів.

Для забезпечення ефективності процесу розробки, контролю якості коду та надійності розгортання застосунку використовується набір

сучасних інструментів та практик. Основним інтегрованим середовищем розробки (IDE) є VS Code[34], доповнене плагінами для покращеної підтримки Angular, TypeScript та інструментів для забезпечення стилю коду, таких як ESLint.

Управління версіями коду здійснюється за допомогою розподіленої системи контролю версій Git. Використовується стратегія розгалужень GitFlow, яка дозволяє організувати паралельну розробку нових функцій, керувати релізами та виправленням помилок. Код фронтенду та бекенду може зберігатися в єдиному репозиторії (монорепозиторій), що полегшує управління залежностями та уніфікацію скриптів збирання та тестування, використовуючи інструменти типу Angular CLI workspaces.

Процеси безперервної інтеграції та безперервного розгортання (CI/CD) автоматизовані за допомогою GitHub Actions[38]. Конвеєр CI включає автоматичне виконання лінтингу коду для перевірки стилю та відповідності стандартам, запуск автоматизованих тестів (юніт- та інтеграційних) для перевірки коректності функціоналу, а також збирання (білд) фронтенд-застосунку та створення Docker-образу для бекенд-частини. Конвеєр CD відповідає за автоматичне розгортання зібраних артефактів. Фронтенд може бути автоматично деплоєний на сервіси статичного хостингу, такі як Netlify[39], тоді як бекенд у вигляді Docker-образу може бути розгорнутий на відповідних хмарних платформах або власних серверах. Такий підхід забезпечує швидку ітерацію розробки, зменшує ризик помилок при розгортанні та гарантує, що завжди доступна актуальна та протестована версія застосунку.

Створення інформаційної управляючої системи для планування робочого процесу на підприємстві має на меті оптимізацію ресурсів, підвищення ефективності управлінських рішень, мінімізацію витрат та забезпечення своєчасного виконання завдань. Основною метою ІУС є автоматизація процесів планування, контролю за виконанням завдань,

аналізу виконання планів та прогнозування майбутніх завдань. Рішення можуть бути методологічними, технологічними та організаційними.

Методологія розробки ІУС повинна базуватись на таких підходах:

- модульний підхід - ІУС повинна бути побудована за принципом модулів, де кожен модуль відповідає за окремі аспекти управління робочим процесом (наприклад, модуль планування завдань, модуль розподілу ресурсів, модуль моніторингу виконання тощо);
- інтеграція даних - система повинна забезпечити інтеграцію з іншими внутрішніми системами підприємства (ERP, CRM, фінансові системи), що дозволить автоматично отримувати та обробляти дані про наявність ресурсів, фінансові можливості, стан виконання завдань;
- методи підтримки рішень - система повинна включати інструменти для підтримки прийняття рішень, зокрема на основі аналізу даних за допомогою штучного інтелекту та методів машинного навчання для прогнозування завантаження ресурсів та планування майбутніх завдань.

РОЗДІЛ 2 ХАРАКТЕРИСТИКА ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ ТА МЕТОДИ І МОДЕЛІ

2.1 Структура і характеристика системи

Логічна архітектура інформаційної управляючої системи побудована на трьох чітко диференційованих рівнях, кожен з яких відповідає за свою сферу відповідальності та комунікує з іншими через стандартизовані інтерфейси. У центрі уваги перебуває рішення SPA на Angular, яке реалізує шар презентації. Завдяки використанню модульної структури з lazy loading, DI і RxJS для реактивної обробки даних, клієнтський інтерфейс здатен підтримувати складні сценарії взаємодії — від перегляду повної інформації про бронювання до кнопки «Швидкий запис» у режимі реального часу. Інтеграція з back-end здійснюється через версіоновані REST-контролери та WebSocket-канал, що забезпечує оптимальну продуктивність та мінімізує затримки.

Серцевиною системи є монолітний додаток на Node.js з фреймворком Express, у якому реалізовано всю бізнес-логіку. Завдяки middleware-шаблонам і використанню моделі «Controller – Service – Repository», кожен функціональний модуль (бронювання, звітність, повідомлення, інтеграція з ботом) відокремлений і може розвиватися незалежно. Обробка помилок, валідація вхідних даних через Joi/celebrate і централізоване логування роблять код стійким і безпечним. Для забезпечення масштабованості сервер може запускатися в кластерному режимі (cluster mode) або контейнерному оточенні з автоматичним горизонтальним автоскейлінгом під керуванням Docker Swarm або Kubernetes.

Рівень доступу до даних розгорнуто на основі PostgreSQL та Redis. ORM-рішення (TypeORM або Sequelize) забезпечують управління схемою бази, міграціями та пулінгом з’єднань, що підвищує надійність транзакцій.

Реляційне сховище гарантує ACID-властивості при виконанні складних SQL-запитів для звітності, тоді як Redis слугує швидким кешем для часто запитуваних даних (наприклад, доступні часові слоти) і брокером черг у поєднанні з бібліотекою BullMQ, що дозволяє відокремити обробку нагадувань та масових розсилок від критичних запитів до СУБД.

Обмін даними між шарами організовано за допомогою трьох основних протоколів. REST-інтерфейс із підтримкою OpenAPI/Swagger документує всі ендпоінти та політики аутентифікації (JWT, OAuth2), що спрощує тестування та інтеграцію з іншими сервісами. Канал WebSocket відповідає за push-оповіщення фронтенда про нові події, без зайвих HTTP-запитів, гарантуючи синхронізацію користувацького інтерфейсу з бекендом у реальному часі. Нарешті, webhook-механізм Telegram-бота дозволяє миттєво отримувати події від месенджера та передавати їх у бізнес-логіку без проміжних таймаутів чи довгого опитування.

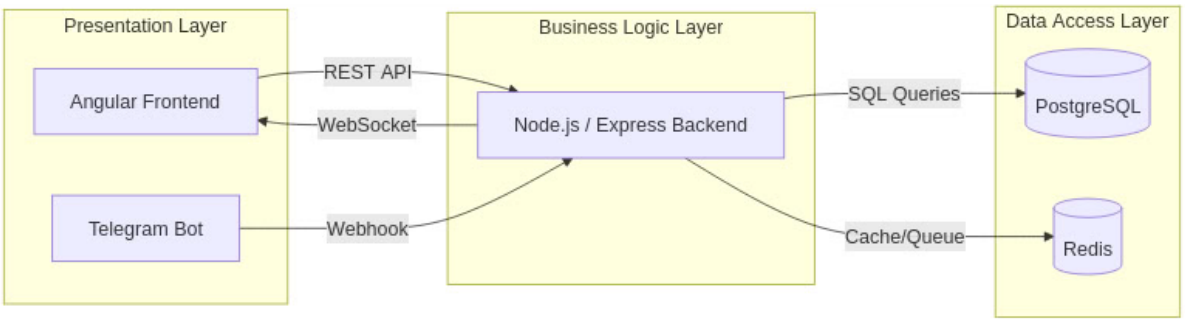


Рисунок 2.1 - Діаграма обміну даними між шарами системи.

Фізична архітектура системи спирається на контейнеризацію та оркестрацію сервісів для забезпечення портативності, ізоляції та простоти масштабування. Кожен компонент — бекенд на Node.js, фронтенд-додаток Angular, NGINX-проксі, сховища даних PostgreSQL та Redis — упаковано в окремі Docker-контейнери. Для локального середовища та розробки використовується Docker Compose, що дозволяє одним файлом описати залежності контейнерів і мережеві зв’язки. У робочому продакшн-оточенні сервіси розгортаються в Kubernetes-кластерах з використанням Deployment і StatefulSet: перший керує безстанними компонентами (API, фронтенд), а

другий — станомісцерозподіленими (PostgreSQL, Redis) з прив’язкою до PersistentVolume Claims.

Розподіл навантаження виконується на рівні вхідного трафіку за допомогою NGINX у ролі зовнішнього балансувальника, який реалізує SSL-термінацію, маршрутизацію запитів до сервісів за правилами path-based routing і механізм round-robin для HTTP/S, а також прокксірування WebSocket-з’єднань до бекенду. У Kubernetes це досягається через ресурс Ingress із контролером NGINX Ingress Controller, що дозволяє єдину точку входу та масштабовану експозицію сервісів.

Збереження даних організоване з урахуванням високої доступності та безпеки. Для PostgreSQL налаштовано streaming-реплікацію між майстер-інстансом і реплікою, що дозволяє виконувати failover у разі збою головного ноди. Реплікація управляється через Kubernetes StatefulSet із політиками PodDisruptionBudget. Регулярне резервне копіювання здійснюється за допомогою cron-job у вигляді періодичних pg_dump зберіганням архівів у зовнішньому сховищі (наприклад, S3). Redis розгорнутий у режимі sentinel-кластера з AOF-підходом для персистентності й автоматичного переключення на резервні ноди.

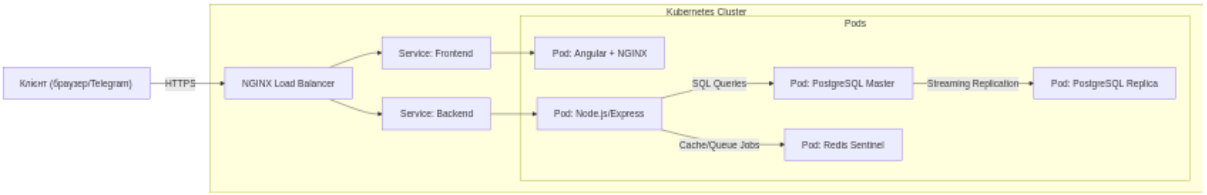


Рисунок 2.2 - Діаграма логіки маршрутизації клієнтських запитів

Ця діаграма ілюструє логіку маршрутизації клієнтських запитів через балансувальник до відповідних сервісів, а також внутрішні зв’язки між подами та сховищами даних у кластері Kubernetes.

Модуль бронювання та календаря реалізує ключову бізнес-логіку з управління часовими слотами, перевірки доступності ресурсів і формування остаточного запису. Він відповідає за синхронізацію з календарем користувача, обробку запитів на резервування та скасування, а

також за оновлення стану слотів у базі даних та в кеші. При ініціалізації система спочатку завантажує конфігурацію робочих годин і налаштування ресурсів, після чого встановлює зв’язок із сервісом Redis для попереднього кешування доступних інтервалів.

Модуль обліку і звітності використовує дані, накопичені в процесі бронювань і обробки клієнтських запитів, для формування статистичних зведень й аналітичних дашбордів. Після запуску він виконує міграції схеми звітних таблиць у PostgreSQL, а потім періодично агрегує транзакційні дані, генеруючи звіти за обраними KPI. Цей модуль залежить від результатів роботи модуля бронювання, оскільки отримує вхідні дані про створені й скасовані записи.

Модуль повідомлень забезпечує розсилку нагадувань і підтверджень через SMS, email та push-нотифікації. При старті він реєструє шаблони повідомлень і налаштовує конектори до зовнішніх провайдерів. У процесі роботи цей модуль отримує від інших компонентів події створених або змінених бронювань через внутрішню чергу Redis, обробляє їх за допомогою BullMQ і відправляє повідомлення кінцевим користувачам.

Інтеграційний модуль з Telegram-ботом відповідає за прийом webhook-запитів від Telegram API, аналіз повідомлень користувачів і трансляцію команд у внутрішній формат запитів на бронювання. Після запуску він реєструє URL-ендпоінт для webhook, ініціалізує механізм обробки повідомлень Telegraf і встановлює зв’язок із модулем бронювання для перевірки доступності слотів. У відповідь на підтвердження чи помилки бот формує та надсилає користувачу відповідні повідомлення.

Порядок ініціалізації модулів визначено залежностями бізнес-процесів: спочатку виконується налаштування кешу Redis та з’єднання з PostgreSQL, потім завантажуються модулі бронювання та календаря, далі модуль обліку і звітності, після чого ініціалізуються провайдери повідомлень, і вкінці запускається інтеграційний модуль

Telegram-бота. Така послідовність гарантує, що кожен компонент отримає необхідні сервіси й дані для коректного виконання своїх задач.

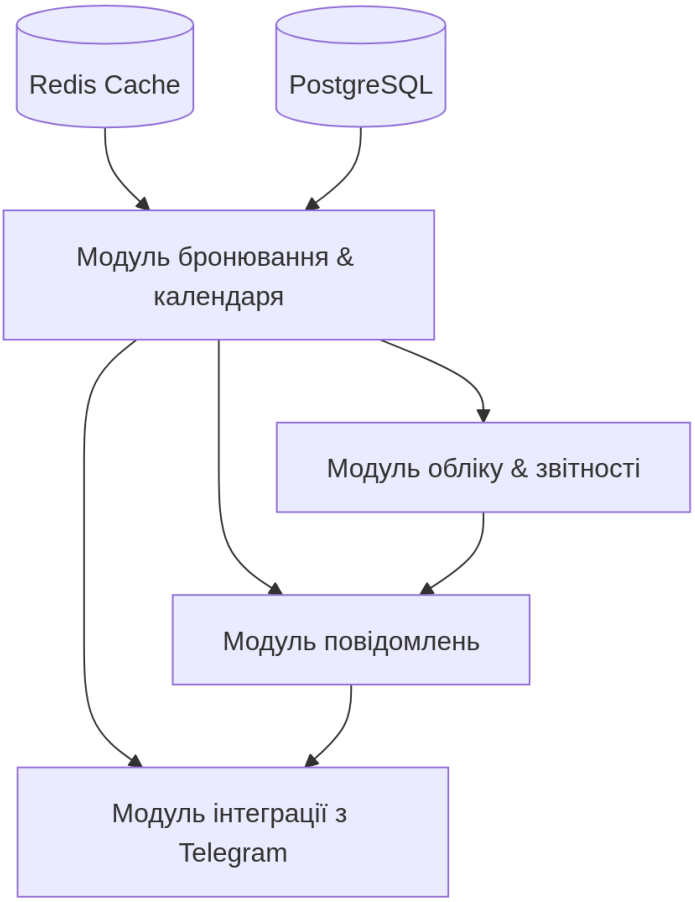


Рисунок 2.3 - Діаграма ілюстрації взаємозв’язків модулів і порядку їх ініціалізації

Для наочного представлення ключових аспектів розроблюваної інформаційної управляючої системи – її функціональних можливостей, взаємодії між складовими та загальної архітектури – було виконано моделювання з використанням відповідних діаграм. Ці графічні представлення, побудовані на основі стандартних нотацій (зокрема, UML), дозволяють системно описати структуру системи, ідентифікувати ключових учасників та візуалізувати основні сценарії її використання. Такий підхід сприяє кращому розумінню системи як розробниками, так і іншими зацікавленими сторонами.

Діаграма варіантів використання слугує для визначення функціональних вимог до системи, представляючи їх з точки зору різних

типів зовнішніх користувачів або систем, які взаємодіють з нею. Ця діаграма чітко ілюструє, які дії (варіанти використання) можуть бути виконані акторами системи. У контексті розроблюваної системи, на діаграмі представлені три основні актори, що відіграють ключові ролі у її функціонуванні: "Клієнт (бот)", "Клієнт (веб)" та "Адміністратор".

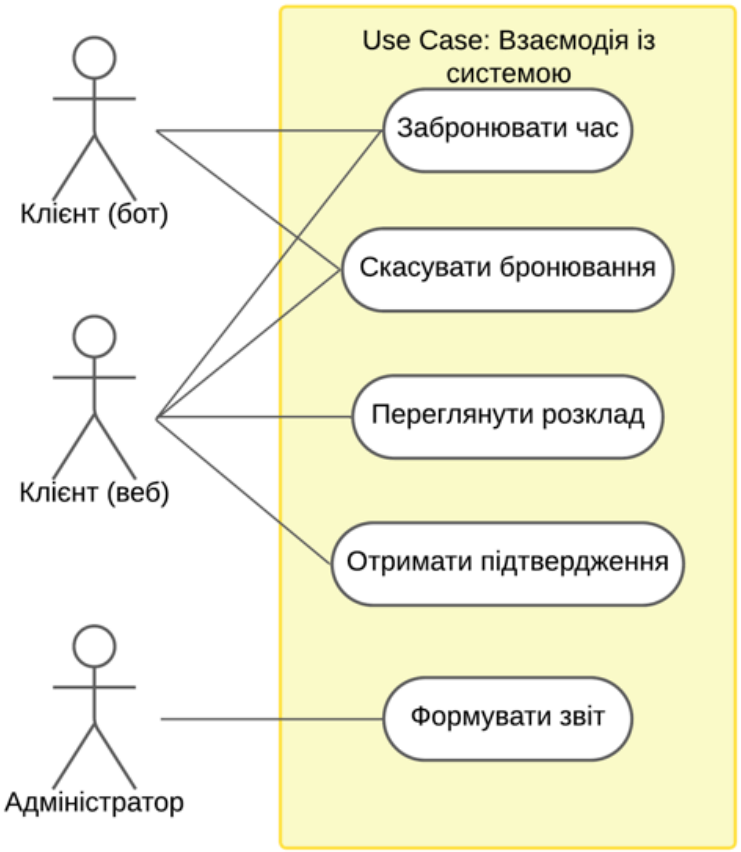


Рисунок 2.4 - Діаграма Use Case

Актор "Клієнт (бот)" представляє кінцевого користувача, який звертається до функціоналу системи через інтерфейс Telegram-бота. Для цього актора визначено можливість "Забронювати час", що дозволяє ініціювати процес запису на послугу шляхом інтерактивного діалогу з ботом. Також передбачено варіант використання "Скасувати бронювання", який надає клієнту зручний спосіб анулювати раніше зроблені записи через той самий канал.

Актор "Клієнт (веб)" позначає користувача, що взаємодіє з системою через веб-браузер, використовуючи фронтенд-додаток на Angular.

Функціонал, доступний цьому актору, включає "Забронювати час" та "Скасувати бронювання", що дублює основні операції бота, але через графічний веб-інтерфейс. Додатково, веб-інтерфейс надає можливість "Переглянути розклад", дозволяючи клієнтам ознайомитись з вільними часовими слотами або переглянути деталі своїх записів. Варіант використання "Отримати підтвердження" ілюструє отримання візуального зворотного зв'язку про статус виконаних операцій, зокрема бронювання, безпосередньо на веб-сайті.

Актор "Адміністратор" представляє персонал підприємства (салону), який має розширені права доступу до системи, як правило, через окремий адміністративний розділ веб-інтерфейсу. Ключовою функціональною можливістю для цього актора є "Формувати звіт", що дозволяє генерувати аналітичні дані за різними критеріями (бронюваннями, клієнтами, завантаженістю), необхідні для операційного управління та прийняття рішень. Таким чином, діаграма варіантів використання окреслює основні функціональні межі системи та ролі користувачів, які реалізують ці функції.

Для ілюстрації фізичної архітектури системи та розміщення її програмних компонентів на обчислювальних ресурсах була побудована схема розгортання. Ця діаграма відображає високоуровневу структуру системи, яка передбачає розгортання у контейнерному середовищі Kubernetes для забезпечення масштабованості, відмовостійкості та спрощеного управління життєвим циклом компонентів.

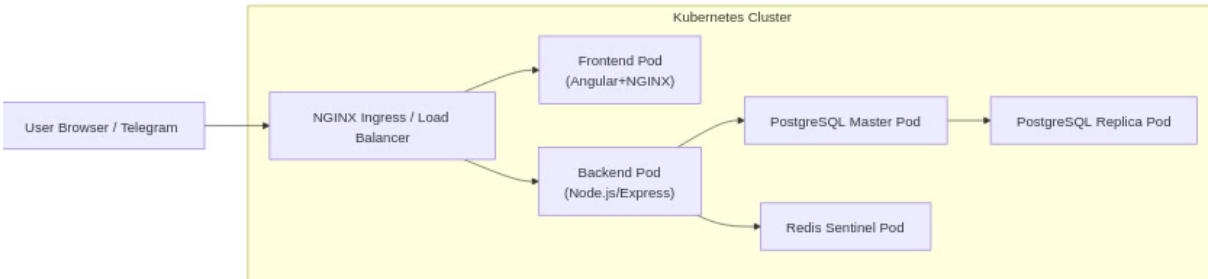


Рисунок 2.5 - Діаграма кластеру Kubernetes

На діаграмі представлені основні вузли та компоненти системи. Зовнішні користувачі, позначені як "User Browser / Telegram", ініціюють взаємодію із системою. Вхідною точкою до кластера Kubernetes є "NGINX Ingress / Load Balancer", який виступає як зворотний проксі та балансувальник навантаження, розподіляючи вхідні запити від користувачів (як веб-запити від браузера, так і вебхуки від Telegram) до відповідних сервісів всередині кластера.

Всередині "Kubernetes Cluster" розташовані ключові компоненти, представлені як Pod'и – найменші розгортані одиниці в Kubernetes. "Frontend Pod (Angular + NGINX)" містить клієнтську частину системи, розроблену на Angular, яка обслуговується веб-сервером NGINX. "Backend Pod (Node.js / Express)" інкапсулює серверний додаток на Node.js/Express, що відповідає за обробку бізнес-логіки, взаємодію з базою даних та зовнішніми сервісами (наприклад, Telegram API). Для зберігання персистентних даних використовуються "PostgreSQL Master Pod" (основний екземпляр бази даних) та "PostgreSQL Replica Pod" (репліка для читання та відмовостійкості). Компонент "Redis Sentinel Pod" включає екземпляр Redis, що може використовуватись для кешування та асинхронних черг, із Sentinel для моніторингу та забезпечення високої доступності. Ця діаграма наочно демонструє розподілену архітектуру системи та принципи її розгортання в сучасних контейнерних інфраструктурах.

Діаграма послідовності деталізує взаємодію між об'єктами системи та зовнішніми акторами в певному сценарії використання, показуючи порядок обміну повідомленнями та виклику операцій у часовій послідовності. На цій діаграмі проілюстровано сценарій успішного бронювання часу клієнтом, який взаємодіє із системою через Telegram-бот.

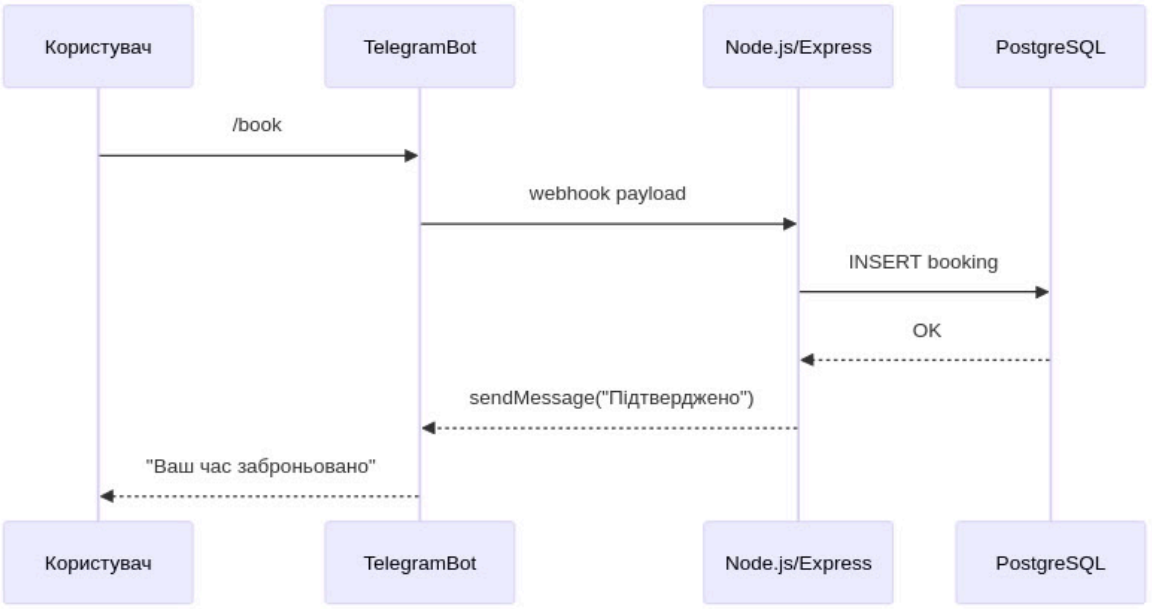


Рисунок 2.6 - Sequence діаграма взаємодії користувача з TelegramBot

Сценарій починається з того, що "Користувач" надсилає команду або повідомлення до "TelegramBot", наприклад, /book, ініціюючи процес запису. Після необхідного діалогу, в ході якого бот може використовувати можливості обробки природної мови (наприклад, через інтеграцію з Google GenAI) для уточнення деталей бронювання, "TelegramBot" надсилає дані бронювання до серверної частини системи у вигляді "webhook payload". Цей HTTP POST запит приймає та обробляє "Node.js/Express" додаток.

Отримавши дані, "Node.js/Express" виконує операцію вставки нового запису про бронювання в базу даних "PostgreSQL" за допомогою команди INSERT booking. Після успішного виконання операції, "PostgreSQL" надсилає підтвердження (OK) назад до "Node.js/Express". У відповідь на успішне збереження, "Node.js/Express" формує запит до Telegram API для надсилання повідомлення користувачеві та викликає операцію sendMessage("Підтверджено") на "TelegramBot". "TelegramBot" отримує цей запит і доставляє повідомлення "Ваш час заброньовано" (або інше підтвердження) кінцевому "Користувачу" в інтерфейсі Telegram. Ця

послідовність ілюструє асинхронний характер взаємодії через вебхуки та підтвердження операції бронювання.

Ця діаграма послідовності аналогічно попередній деталізує потік взаємодії, але стосується сценарію бронювання часу, здійсненого клієнтом через веб-інтерфейс системи (фронтенд на Angular). Вона демонструє, як клієнтські дії на сайті призводять до взаємодії з бекендом та базою даних, а також механізм оновлення інтерфейсу.

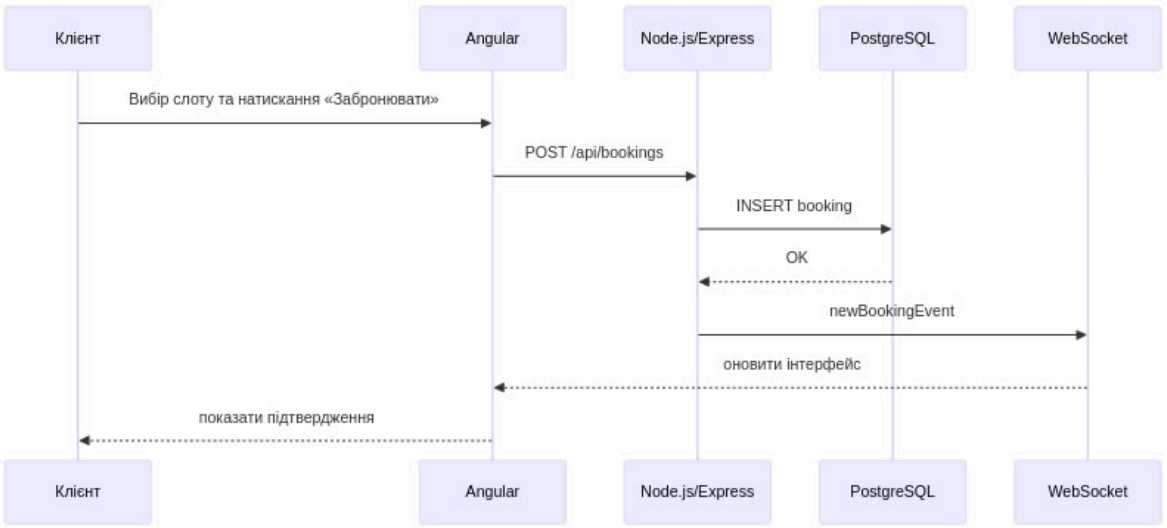


Рисунок 2.7 - Діаграма взаємодії з браузеру

Сценарій починається з того, що "Клієнт" в інтерфейсі "Angular" (на веб-сайті) обирає бажаний часовий слот та натискає кнопку "Забронювати". "Angular" додаток, обробивши цю дію, формує HTTP POST запит до бекенду, який зазвичай спрямовується на API-ендпоінт на зразок /api/bookings. Цей запит надсилається до "Node.js/Express" додатку, що виконує роль серверного API.

"Node.js/Express" приймає POST-запит, обробляє дані бронювання та виконує операцію вставки нового запису в базу даних "PostgreSQL" за допомогою команди INSERT booking. Після успішного збереження, "PostgreSQL" повертає підтвердження (OK) до "Node.js/Express". Важливою відмінністю від сценарію з ботом є використання WebSocket: після успішного бронювання, "Node.js/Express" ініціює подію

(newBookingEvent) та надсилає її через з'єднання "WebSocket" до всіх підключених "Angular" клієнтів.

"WebSocket" сервер передає цю подію "Angular" додатку. "Angular" додаток, отримавши через WebSocket повідомлення про нове бронювання, виконує операцію "оновити інтерфейс", щоб негайно відобразити зміни в розкладі або списку бронювань для користувача. Паралельно з надсиланням WebSocket-події, "Node.js/Express" надсилає відповідь на початковий HTTP POST запит назад до "Angular", підтверджуючи успішне виконання операції. "Angular" додаток, отримавши цю відповідь, відображає "показати підтвердження" в інтерфейсі для "Клієнта", візуально підтверджуючи, що бронювання здійснено. Ця діаграма ілюструє комбіноване використання REST API для ініціації операцій та WebSocket для реактивного оновлення інтерфейсу користувача.

Класова діаграма є фундаментальною структурною діаграмою UML, яка надає статичне уявлення про модель системи. Вона визначає типи об'єктів у системі (класи), їхні властивості (атрибути), поведінку (операції або методи) та різноманітні статичні зв'язки між цими класами, такі як асоціації, агрегації, композиції та успадкування. Ця діаграма слугує основою для проектування бази даних та реалізації об'єктно-орієнтованої логіки застосунку.

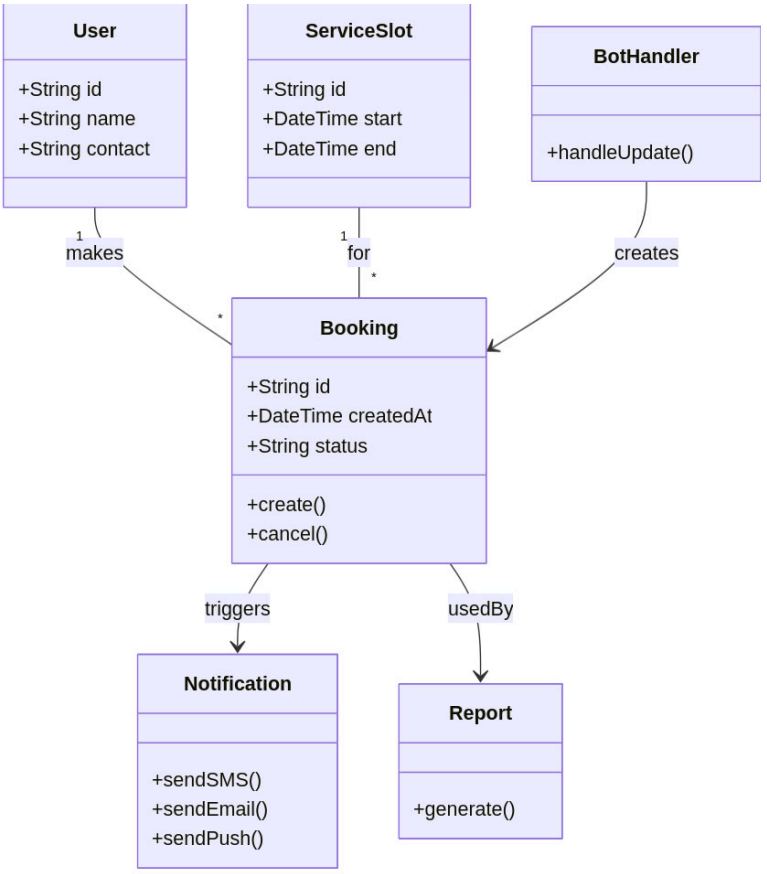


Рисунок 2.8 - Діаграма класів.

На діаграмі представлені ключові класи, що моделюють основні сутності предметної області та компоненти системи. Клас "User" представляє користувача системи (клієнта) з атрибутами id (унікальний ідентифікатор), name (ім'я) та contact (контактна інформація). Клас "ServiceSlot" моделює доступний для бронювання часовий інтервал, маючи атрибути id, start (час початку слоту) та end (час закінчення слоту). Центральним класом є "Booking", який інкапсулює інформацію про здійснене бронювання з атрибутами id, createdAt (час створення запису) та status (поточний стан бронювання). Клас "Booking" також визначає операції, такі як create() для створення нового запису та cancel() для його анулювання.

Зв'язки між класами відображають асоціації: один "User" може зробити "багато" (*) "Booking", і кожне "Booking" відноситься до одного "ServiceSlot". Клас "BotHandler" представляє компонент, відповідальний за

обробку вхідних даних від Telegram-бота, маючи операцію handleUpdate(). Цей клас пов'язаний зі створенням ("creates") об'єктів "Booking". Клас "Notification" моделює функціональність надсилання сповіщень, визначаючи операції для відправки повідомлень через різні канали: sendSMS(), sendEmail(), sendPush(). Діаграма показує, що "Booking" "запускає" ("triggers") "Notification", що означає, що успішне бронювання або його скасування може ініціювати відправку сповіщення. Нарешті, клас "Report" представляє функціональність генерації звітів з операцією generate(). Цей клас "використовує" ("usedBy") дані з "Booking" для формування аналітичної звітності. Ця класова діаграма слугує структурною основою для проектування бази даних та об'єктно-орієнтованої реалізації бізнес-логіки системи.

У процесі обробки запиту на бронювання даних проглядається кілька поєднаних потоків інформації. Спершу кінцевий користувач через веб-інтерфейс обирає дату і час зустрічі, після чого Angular-додаток формує REST-запит і надсилає його на сервер. Серверна логіка виконує валідацію даних, зберігає бронювання в PostgreSQL і публікує відповідну подію в чергу Redis для фонової обробки нагадувань. Після успішного збереження сервер негайно надсилає подію WebSocket-каналом усім підключеним клієнтам, щоб синхронізувати відображення розкладу в реальному часі.

Діаграма знаходиться в додатку Б

Обмін даними через WebSocket та webhook реалізує двосторонню взаємодію в реальному часі. WebSocket-канал використовується для трансляції змін у розкладі (нові або скасовані бронювання) без потреби постійного опитування сервера, що знижує затримки й навантаження на мережу . Водночас webhook-механізм забезпечує прийом подій від Telegram-бота: коли користувач надсилає команду боту, Telegram надсилає HTTP POST-запит на попередньо визначений URL у вашому бекенді, де

Express-сервер розбирає повідомлення й ініціює відповідні бізнес-процеси (створення або скасування бронювання).

Асинхронна обробка завдань організована через чергу повідомлень у Redis і фонові воркери (наприклад, BullMQ). Це дозволяє відокремити критичні шляхи обробки REST-запитів від триваліших операцій — надсилання SMS, email чи push-повідомлень — та гарантувати, що відповіді на клієнтські запити повертаються максимально швидко. Крім того, такий підхід підвищує відмовостійкість: у разі тимчасових проблем із зовнішніми провайдерами повідомлень завдання залишаються в черзі й обробляться автоматично після відновлення сервісів.

2.2 Методи та моделі в інформаційних управляючих системах і технологіях

У сучасному підході до проєктування інформаційних систем моделювання бізнес-процесів виконує дві головні ролі: формалізацію очікувань стейкхолдерів та відпрацювання алгоритмів взаємодії компонентів перед початком кодування. Для цього широко застосовують BPMN (Business Process Model and Notation) та DFD (Data Flow Diagram).

BPMN — це стандартизована нотація, затверджена Object Management Group, яка дозволяє описати потік робіт як послідовність завдань, подій і шлюзів у єдиному графічному форматі . Елементи BPMN-діаграми включають початкові й кінцеві події, активності (tasks), розгалуження (gateways) та підпроцеси (sub-processes)[40]. Застосування BPMN дає змогу:

- чітко визначити ролі та відповідальність учасників;
- аналізувати варіанти розвитку сценаріїв через різні шлюзи;
- виявити вузькі місця й точки оптимізації перед імплементацією.

Нижче — спрощена BPMN-діаграма сценарію «Бронювання»:

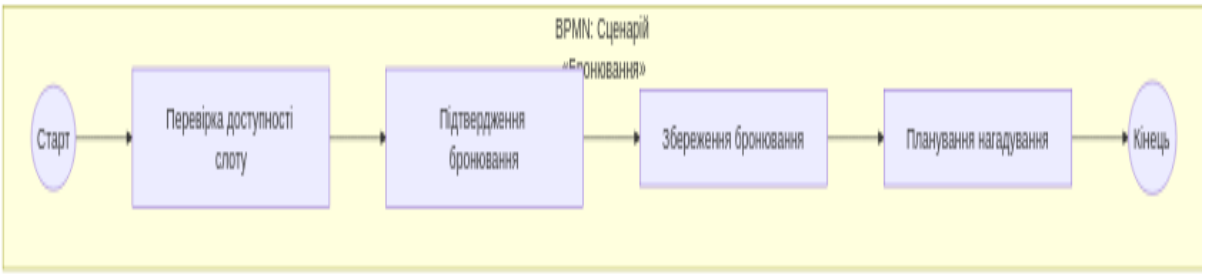
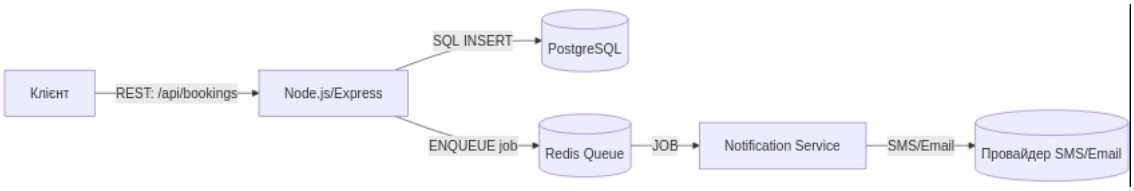


Рисунок 2.9 - BPMN-діаграма сценарію «Бронювання»

DFD (Data Flow Diagram) — структурний метод аналізу, що відображає, як дані рухаються між зовнішніми суб'єктами, процесами та сховищами системи . Графічні нотації DFD — це потоки даних (стрілки), процеси (кола або прямокутники), сховища (паралельні лінії) і зовнішні сутності (квадрати)[41]. Використання DFD допомагає:

- виявити, які процеси змінюють або передають дані;
- спланувати структуру бази даних і API-ендпоінти;
- забезпечити узгодженість між технічним дизайном і бізнес-вимогами.

Приклад DFD для обробки запиту на бронювання:



Об'єднуючи результати BPMN та DFD, ми переходимо до детальних UML-моделей і безпосередньої імплементації функціональних компонентів із упевненістю, що всі процеси й потоки даних були ретельно опрацьовані та задокументовані.

У проєктуванні інформаційної управляючої системи ключовим етапом є створення концептуальної та логічної моделі даних, яка відображає взаємозв'язки між сутностями предметної області та забезпечує коректне зберігання інформації.

Першою складовою цієї моделі є ER-діаграма (Entity–Relationship), що ілюструє сутності, їх атрибути та зв'язки. Відповідно до методології

Elmasri & Navathe, ER-модель дозволяє за допомогою нотацій «сутність–зв’язок» формалізувати структуру даних, а подальша нормалізація (1NF, 2NF, 3NF) усуває надлишковість та забезпечує цілісність даних, знижуючи ризик аномалій при оновленні й видаленні записів .

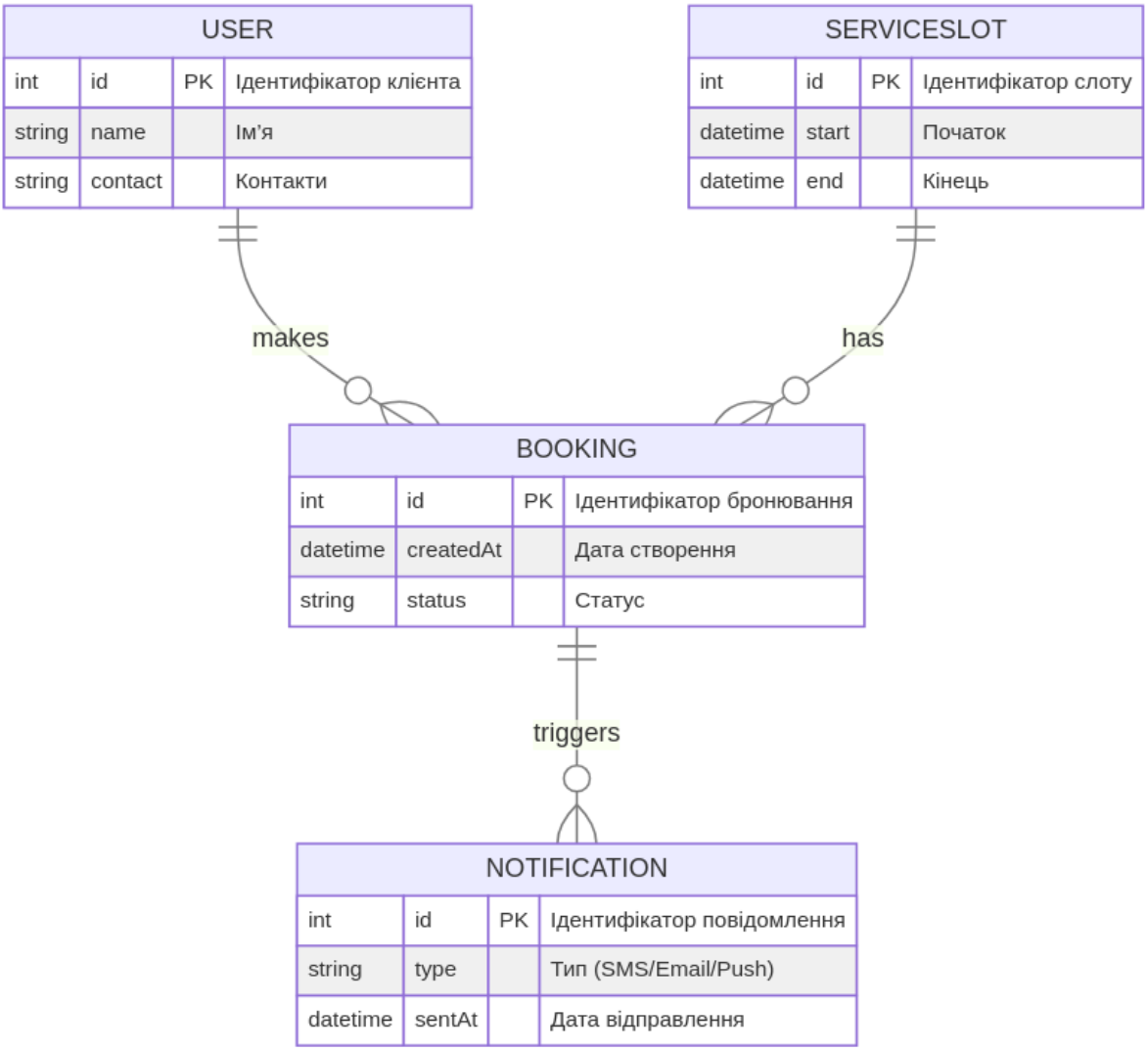


Рисунок 2.10 - ER-діаграма сутностей.

У наведеній ER-діаграмі сутність USER зв’язана із BOOKING через асоціацію «1–до–багатьох», що відображає можливість множинних бронювань одного клієнта. Сутність SERVICESLOT відображає часові інтервали, які також асоціюються з бронюваннями. Сутність NOTIFICATION пов’язана з бронюваннями як подія надсилання

повідомлень. Після побудови ER-моделі виконується нормалізація: переведення в першу нормальну форму (видалення груп повторюваних даних), другу (усунення часткових залежностей) і третю (усунення транзитивних залежностей), що дає змогу уникнути дублювання та аномалій оновлення .

Для підтримки аналітики й формування складних звітів використовується OLAP-куб (Online Analytical Processing), збудований за принципами зіркового (star) схеми. Згідно з рекомендаціями Kimball, куб складається з факт-таблиці BookingFact, що містить числові показники (кількість бронювань, сума оплат тощо), та набору довідкових таблиць-вимірів (dimensions): DateDimension, ClientDimension, ServiceDimension та ResourceDimension.

У рамках оптимізації розкладу на підприємстві застосування жадібного алгоритму дозволяє швидко отримати робочий варіант графіка з урахуванням ключових вимог: рівномірності навантаження, пріоритетів клієнтів і часових обмежень. Спочатку кожна подія (запис клієнта) описується тривалістю, набором допустимих часових вікон і ваговим коефіцієнтом пріоритету. Далі всі події сортуються у порядку спадання їхньої пріоритетності або за датою завершення («earliest-deadline-first»), що знижує ймовірність фрагментації розкладу та гарантує, що найважливіші записи отримають найкращі часові слоти.

Після сортування жадібний алгоритм ітерує кожну подію, намагаючись призначити її в той часовий інтервал, який на поточному кроці має мінімальне накопичене навантаження. Для цього використовується мін-купа (min-heap), де ключем є загальна кількість подій або сумарна тривалість у кожному слоті. Якщо виявляється, що жоден з вільних слотів не відповідає часовим обмеженням події, вона відкладається у відкладений список для подальшої обробки або ручного коригування.

Перевагою жадібного підходу є його простота та ефективність: сортування n подій виконується за O(n log n), а пошук найменш навантаженого слота — за O(log m) для кожної події, де m — кількість часових інтервалів. Таким чином, загальна складність алгоритму оцінюється як O(n log n + n log m), що забезпечує його застосовність у режимі реального часу навіть для кількох сотень запитів.

Жадібний алгоритм не гарантує знаходження глобального оптимуму, оскільки під час кожного кроку він приймає локально найкраще рішення, не оцінюючи вплив на подальші події. Проте в контексті планування записів у салоні краси, де важлива оперативність і гнучкість, такий компроміс між якістю та швидкістю є прийнятним. Алгоритм дозволяє швидко генерувати чернетки розкладу, які потім можуть бути кориговані адміністратором або доповнені еволюційними методами для досягнення вищої якості.

Таким чином, жадібний підхід служить основною технікою початкового розподілу ресурсів і часу в інформаційній управляючій системі планування робочого процесу на підприємстві, забезпечуючи мінімальні затримки при обробці запитів клієнтів і зручний інструмент для подальшої оптимізації та інтеграції з Telegram-ботом.

Аналітичні можливості інформаційної управляючої системи покликані забезпечити керівництво підприємства інструментами швидкого та глибокого аналізу даних для прийняття обґрунтованих рішень. Поєднання OLAP-сегментації, Data Mining, ML-рекомендацій та DSS створює єдину аналітичну екосистему, здатну обробляти як оперативні, так і стратегічні завдання.

Основою аналітики є OLAP-куби — багатовимірні структури, що дозволяють «нарізати» (slice), «розгортати» (drill-down/roll-up) і «фільтрувати» показники за різними вимірами (дата, клієнт, послуга, ресурс) з надвисокою продуктивністю запитів. Завдяки OLAP-сегментації

керівники можуть у реальному часі порівнювати періоди, аналізувати тренди завантаженості та оптимізувати графіки роботи персоналу.

Data Mining доповнює OLAP-аналіз, надаючи методи для виявлення прихованих закономірностей у великих масивах даних. Кластеризація (K-means, ієрархічний аналіз) групує клієнтів за частотою відвідувань, середнім чеком або типами замовлень, що дозволяє формувати таргетовані маркетингові кампанії [44]. Ассоціативні правила (Apriori) виявляють послуги, які найчастіше замовляють разом, а алгоритми виявлення аномалій сигналізують про нетипові зміни в поведінці клієнтів.

Для персоналізації взаємодії з клієнтом застосовуються ML-рекомендаційні модулі. Колаборативна фільтрація («user-based», «item-based»), матрична факторизація (SVD) та нейромережні автоенкодери аналізують історію бронювань і профілі користувачів, прогнозуючи найбільш зручні слоти та комплекти послуг [45]. Інтеграція таких рекомендацій у Telegram-бота дозволяє пропонувати кінцевому користувачу персоналізовані «гарячі» пропозиції й автоматизовані нагадування, що підвищує рівень лояльності.

Rules Engine (наприклад, Drools або DMN-рушії) формалізує бізнес-правила — від пріоритезації VIP-клієнтів до обмежень на кількість одночасних бронювань — у вигляді легко оновлюваних умов і дій [46]. What-if аналіз у BI-інструментах (Power BI, Tableau) моделює наслідки змін параметрів (наприклад, зміна тривалості послуг чи кількості операторів) і порівнює KPI до та після коригувань, що допомагає вибрати найефективніші управлінські рішення [47].

Поєднання цих підходів створює гнучку, масштабовану аналітичну платформу, яка не лише відображає історичні дані, а й прогнозує навантаження, автоматизує рекомендації та підтримує оперативні управлінські рішення в умовах динамічних змін попиту.

Якісна оцінка роботи інформаційної управляючої системи базується на чітко визначених показниках ефективності (KPI). Серед ключових метрик для модулів API виділяють час відповіді (response time), що вимірюється затримкою між отриманням HTTP-запиту і відправкою відповіді, а також час обробки запиту (processing time) на боці сервера. Для бізнес-критичних сервісів визначають рівні SLA-метрик (Service Level Agreement), наприклад, прагнення до 99 % запитів із затримкою менше ніж 200 мс (p99 latency) або доступність 99,9 % (three-nines uptime) [47]. Крім того, до KPI належать показники пропускної спроможності (throughput), відсоток помилок (error rate) і середня кількість одночасних з'єднань (concurrent connections), які разом із часовими метриками дають цілісне уявлення про продуктивність системи.

Перевірка цих метрик здійснюється за допомогою навантажувального тестування. Інструмент Apache JMeter дозволяє моделювати великий обсяг одночасних HTTP-запитів, створювати сценарії із прогресивним набором віртуальних користувачів і вимірювати час відповіді та стабільність системи під навантаженням [50]. Аналогічно, k6 — це lightweight-інструмент з JavaScript-скриптами, що підтримує сценарії ітеративного тестування, автоматизовані CI/CD інтеграції та збір пам'яті й CPU-метрик під час навантаження [51]. Ці підходи дають змогу виявити «вузькі місця» продуктивності, визначити точки масштабування та перевірити, чи відповідає система вимогам SLA.

Для забезпечення безперервної роботи ІУС використовують архітектурні патерни високої доступності. У схемі Active-Active кілька екземплярів сервісу працюють одночасно і розподіляють навантаження через балансувальник, що гарантує безперебійну роботу навіть при виході з ладу окремих вузлів. У Active-Passive конфігурації пасивні резервні вузли перебувають у режимі очікування та активуються лише при відмові

головного, що спрощує синхронізацію стану, але може вводити затримки під час перемикання [49].

Резервне копіювання і відновлення (backup and recovery) визначаються двома критичними параметрами: RPO (Recovery Point Objective) — максимально допустимий інтервал втрати даних, і RTO (Recovery Time Objective) — час, необхідний для відновлення сервісу до працездатного стану після відмови. Відповідно до ISO/IEC 27001, політика бекапів має включати регулярні повні та інкрементальні знімки даних з архівацією на георозподілені сховища, а також періодичне тестування процедури відновлення для підтвердження відповідності RPO/RTO [47]. Також практикують гарячі (hot), теплі (warm) і холодні (cold) стенд-бай копії, які визначаються різними рівнями готовності запасних ресурсів і часу запуску.

РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОЕКТНИХ РІШЕНЬ ТА ЇХ РЕАЛІЗАЦІЯ

3.1 Проєктування бази даних та/або сховища даних для інформаційної управляючої системи

Ефективне та надійне зберігання даних є фундаментальним аспектом будь-якої інформаційної системи, що управляє робочими процесами. Проектування структури бази даних для розроблюваної системи планування бронювань було здійснено з урахуванням вимог до підтримки як оперативних транзакцій (створення, перегляд, скасування бронювань), так і аналітичних потреб (формування звітів). На основі вибору СУБД PostgreSQL, що підтримує реляційну модель та ACID-транзакції, було розроблено схему бази даних, яка представлена на діаграмі. Ця схема поєднує в собі елементи класичної реляційної моделі для операційної діяльності та денормалізованої моделі для сховища даних, що забезпечує оптимальний баланс між продуктивністю операцій та швидкістю виконання аналітичних запитів.

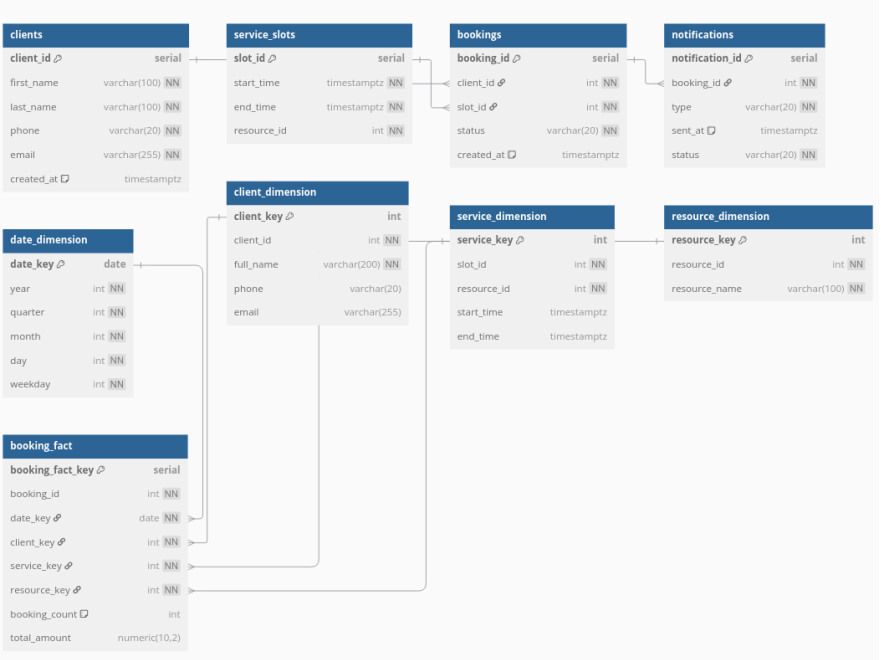


Рисунок 3.1 - Архітектура бази данних

Операційна частина моделі даних системи, що відповідає за підтримку щоденних транзакцій бронювання, включає кілька ключових таблиць. Таблиця `clients` призначена для зберігання базової інформації про клієнтів, містить унікальний ідентифікатор `client_id` (первинний ключ), ім'я, прізвище, контактні дані (телефон, email) та дату створення запису. Поля з ім'ям, прізвищем, телефоном та email є обов'язковими. Таблиця `service_slots` моделює доступні для бронювання часові інтервали, зберігаючи `slot_id` (первинний ключ), час початку (`start_time`) та закінчення (`end_time`) слоту, а також зовнішній ключ `resource_id`, який пов'язує слот з конкретним фізичним ресурсом або співробітником. Центральною операційною таблицею є `bookings`, яка фіксує факт бронювання. Вона пов'язує клієнта (`client_id` як зовнішній ключ до `clients`) з певним часовим слотом (`slot_id` як зовнішній ключ до `service_slots`), зберігає час створення запису (`created_at`) та його поточний статус (`status`). Для відстеження та управління сповіщеннями, пов'язаними з бронюваннями (наприклад, підтвердження або нагадування), передбачено таблицю `notifications`. Ця таблиця містить `notification_id`

(первинний ключ), зовнішній ключ `booking_id`, що посилається на відповідний запис у таблиці `bookings`, а також тип сповіщення (`type`), час відправлення (`sent_at`) та його статус доставки (`status`).

Для потреб аналітики та формування звітів, що є важливою складовою інформаційної управляючої системи, було розроблено модель сховища даних на основі схеми "зірки". Ця модель включає набір таблиць вимірів (`_dimension`) та центральну таблицю фактів (`booking_fact`). Таблиці вимірів – `date_dimension`, `client_dimension`, `service_dimension` та `resource_dimension` – містять денормалізовану інформацію, що описує контекст бронювань. `date_dimension` надає деталізацію за роком, кварталом, місяцем, днем та днем тижня. `client_dimension` дублює та, можливо, денормалізує дані клієнтів, включаючи їх повне ім'я та контактну інформацію, пов'язуючись з таблицею фактів через `client_key`. Аналогічно, `service_dimension` та `resource_dimension` надають контекст щодо заброньованих слотів та ресурсів, використовуючи відповідно `service_key` та `resource_key` як первинні ключі вимірів.

Центральною у сховищі даних є таблиця фактів `booking_fact`. Кожен запис у цій таблиці представляє окреме бронювання і пов'язується з відповідними записами у всіх таблицях вимірів через зовнішні ключі (`date_key`, `client_key`, `service_key`, `resource_key`). Таблиця фактів зберігає вимірювані показники, такі як `booking_count` (який зазвичай дорівнює 1 для кожного бронювання) та `total_amount` (сума бронювання, якщо це застосовно). Додатково, `booking_fact` містить зовнішній ключ `booking_id`, що посилається на оригінальний запис у транзакційній таблиці `bookings`, дозволяючи здійснювати "провал" (drill-through) від агрегованих звітів до детальних даних окремих бронювань. Така архітектура дозволяє швидко виконувати аналітичні запити, агрегуючи дані за різними вимірами (наприклад, кількість бронювань за певний

період часу, за конкретним співробітником або послугою), не впливаючи на продуктивність операційної системи.

3.2 Проєктування бази знань інформаційної управляючої системи та/або засоби інтелектуального аналізу даних

Цей розділ присвячено розгляду компонентів системи, що відповідають за обробку та використання знань. В контексті інформаційної управляючої системи для планування робочого процесу, база знань та засоби інтелектуального аналізу даних відіграють важливу роль у підвищенні ефективності процесів прийняття рішень, персоналізації взаємодії з клієнтами та оптимізації використання ресурсів. Інтелектуальний аналіз даних (Data Mining) дозволяє виявити приховані закономірності та цінні інсайти у накопичених операційних даних, тоді як база знань надає структуроване сховище або механізм доступу до цих інсайтів, а також до предметно-орієнтованої інформації та бізнес-правил, що використовуються системою.

3.2.1 Організація та проведення інтелектуального аналізу даних з метою пошуку знань в базі даних

Накопичені в процесі функціонування системи операційні дані, що зберігаються у базі даних PostgreSQL (особливо в структурі сховища даних з таблицями вимірів та фактів), є цінним джерелом інформації для інтелектуального аналізу. Метою такого аналізу є виявлення неочевидних закономірностей, тенденцій та кореляцій, які можуть бути перетворені на корисні знання для покращення роботи підприємства та системи. Організація та проведення інтелектуального аналізу даних включає кілька етапів.

Першим кроком є визначення цілей аналізу, які мають відповідати бізнес-задачам підприємства, наприклад: виявлення найбільш популярних послуг та часових інтервалів для бронювання; сегментація клієнтів за

їхньою активністю та перевагами; прогнозування завантаженості ресурсів у майбутньому; ідентифікація періодів підвищеного або зниженого попиту; аналіз ефективності різних каналів бронювання (бот vs веб); виявлення послуг, які часто бронюються разом. Наступним етапом є збір та попередня обробка даних. Використовуються дані з операційних таблиць та, що особливо ефективно, з таблиці фактів booking_fact та пов'язаних з нею таблиць вимірів (date_dimension, client_dimension, service_dimension, resource_dimension). Цей етап може включати очищення даних від аномалій, їх трансформацію та агрегацію до необхідного рівня деталізації.

Безпосередньо проведення інтелектуального аналізу даних передбачає застосування різноманітних методів та алгоритмів. Для виявлення популярних тенденцій та завантаженості можуть використовуватися методи описової статистики та часових рядів, що легко реалізуються за допомогою SQL-запитів до сховища даних з використанням агрегатних функцій (COUNT, AVG тощо) та групування за вимірами (дата, час, ресурс). Для сегментації клієнтів можуть застосовуватися алгоритми кластеризації (наприклад, K-Means) на основі даних про частоту бронювань, обрані послуги, час бронювання тощо. Пошук послуг, які часто бронюються разом, може здійснюватися за допомогою методів пошуку асоціативних правил (наприклад, алгоритм Apriori). Прогнозування попиту може вимагати використання більш складних методів, таких як регресійний аналіз або методи прогнозування часових рядів. Вибір конкретних методів залежить від поставленої аналітичної задачі та типу даних. Для реалізації цих методів можуть бути використані як можливості самої СУБД PostgreSQL (наприклад, розширення), так і зовнішні інструменти або бібліотеки для аналізу даних (наприклад, скрипти на Python з бібліотеками Pandas, Scikit-learn, що працюють з експортованими або безпосередньо підключеними даними). Результати аналізу, такі як виявлені пікові години, профілі клієнтських

сегментів або прогнози попиту, є цінними знаннями, які можуть бути інтегровані в роботу системи.

3.2.2 Проєктування та реалізація бази знань

База знань в контексті даної інформаційної управляючої системи може бути інтерпретована як сукупність структурованої інформації, що не є суто операційними даними транзакцій, але є необхідною для функціонування інтелектуальних компонентів системи, підтримки бізнес-логіки та використання результатів інтелектуального аналізу даних. Це може включати предметно-орієнтовані дані, набір бізнес-правил та інсайти, отримані в результаті аналітичної обробки даних.

Проектування бази знань передбачає визначення типу інформації, яка буде в ній зберігатися, вибір формату представлення знань та механізмів доступу до них. В рамках даної системи, база знань може включати:

Предметно-орієнтовані дані: Інформація про послуги (тривалість, вартість, залежність від ресурсу), ресурси (графік роботи, доступність), правила формування розкладу (наприклад, мінімальний інтервал між записами). Ця інформація може зберігатись у відповідних довідкових таблицях реляційної бази даних (наприклад, таблиці послуг, ресурсів, які доповнюють наведену в розділі 3.1 схему).

Бізнес-правила: Правила валідації бронювань (наприклад, заборона запису на минулий час, перевірка наявності вільних слотів), правила розрахунку вартості (якщо це передбачено системою), правила надсилання сповіщень (наприклад, нагадування за 24 години до візиту). Ці правила можуть бути реалізовані безпосередньо в коді бекенд-додатку (бізнес-логіка) або, для складніших сценаріїв, винесені у спеціалізований модуль чи навіть представлені у декларативному вигляді (наприклад, за допомогою систем управління бізнес-правилами, хоча це виходить за рамки типового дипломного проєкту).

Результати інтелектуального аналізу даних: Виявлені закономірності та інсайти можуть бути збережені у структурованому вигляді для подальшого використання. Наприклад, таблиця з піковими годинами бронювання для кожного дня тижня, таблиця з клієнтськими сегментами та їх характеристиками, або результати прогнозу попиту, що зберігаються для використання у функціоналі пропонування часу бронювання. Ці дані можуть зберігатись у додаткових таблицях реляційної бази даних або у більш пристосованих для певних типів знань сховищах.

Реалізація бази знань передбачає створення відповідних структур зберігання (таблиці в PostgreSQL, конфігураційні файли) та розробку програмних інтерфейсів для доступу до цих знань з різних компонентів системи. Наприклад, компонент обробки запитів від Telegram-бота (BotHandler) або логіка API на Node.js/Express може звертатися до таблиць з інформацією про послуги та ресурси для валідації вхідних даних, або використовувати дані з таблиць результатів аналізу для формування релевантних відповідей або пропозицій користувачеві. Взаємодія з Google GenAI може включати передачу йому певних даних з бази знань (наприклад, списку доступних послуг та їх характеристик) для покращення його здатності розуміти та обробляти запити користувачів у природній мові в контексті предметної області салону. Таким чином, база знань виступає як сховище критично важливої для функціонування системи інформації та інсайтів, що забезпечує її гнучкість та інтелектуальність.

3.3 Програмне забезпечення

Програмне забезпечення є ядром інформаційної управляючої системи, що відповідає за реалізацію всіх визначених функціональних та нефункціональних вимог. Розробка якісного програмного забезпечення вимагає чіткого розуміння вимог користувачів та системи в цілому, вибору відповідних інструментальних засобів розробки, а також проектування

надійної та масштабованої архітектури. В даному розділі детально описано вимоги до програмного забезпечення, інструменти, що використовувались у процесі розробки, та архітектурні рішення, що лягли в основу програмної реалізації системи.

3.3.1 Вимоги до програмного забезпечення

Вимоги до програмного забезпечення визначають, що саме повинна робити система, які функції виконувати, а також якими характеристиками вона має володіти. Ці вимоги були сформовані на основі аналізу предметної області, потреб користувачів (клієнтів та адміністрації салону) та бізнес-процесів планування робочого процесу. Вимоги до програмного забезпечення поділяються на функціональні та нефункціональні.

Функціональні вимоги описують конкретні функції та операції, які система повинна виконувати. Для даної системи до них належать: можливість клієнтів (через веб-сайт або Telegram-бота) переглядати доступний розклад та вільні часові слоти; здійснення бронювання обраних послуг на конкретний час; скасування існуючих бронювань; отримання підтверджень та нагадувань про майбутні візити (через бота, веб-інтерфейс або інші канали); зберігання та управління інформацією про клієнтів та послуги; надання адміністраторам можливості переглядати, додавати, редагувати та видаляти бронювання; генерація різних типів звітів (наприклад, про завантаженість ресурсів, кількість бронювань за період) для аналізу та прийняття управлінських рішень; інтеграція з Telegram API для обробки команд та повідомлень бота; взаємодія з Google GenAI для розпізнавання та обробки запитів користувачів у природній мові через бота; відображення актуальної інформації про розклад та бронювання на веб-сайті.

Нефункціональні вимоги визначають якісні характеристики системи, що впливають на її працездатність та зручність використання, але не пов'язані безпосередньо з конкретними функціями. До важливих

нефункціональних вимог належать: продуктивність системи, що вимірюється часом відгуку на запити користувачів (наприклад, швидкість відображення розкладу або підтвердження бронювання); масштабованість, що забезпечує можливість ефективної роботи системи при збільшенні кількості користувачів, бронювань та оброблюваних даних; надійність та відмовостійкість, що гарантують безперебійну роботу системи та збереження даних навіть у випадку збоїв; доступність системи 24/7 (за винятком планових робіт); безпека даних, що передбачає захист персональної інформації клієнтів та даних бронювань від несанкціонованого доступу, використання безпечних протоколів передачі даних (HTTPS) та механізмів автентифікації/авторизації для адміністративної частини; зручність використання (Usability) веб-інтерфейсу та інтуїтивно зрозуміла взаємодія з Telegram-ботом; ремонтопридатність та підтримуваність коду, що спрощує внесення змін та виправлення помилок у майбутньому; а також доступність (Accessibility) веб-інтерфейсу для користувачів з обмеженими можливостями, відповідно до стандартів WCAG 2.1.

3.3.2 Інструментальні засоби розробки для створення прикладного програмного забезпечення інформаційних управляючих систем

Реалізація програмного забезпечення інформаційної управляючої системи вимагала використання комплексу сучасних інструментальних засобів розробки, що охоплюють всі етапи життєвого циклу програмного продукту – від написання коду та збирання, до тестування, розгортання та супроводу. Вибір інструментів базувався на обраних технологіях (Node.js, Angular, PostgreSQL), вимогах проєкту та практиках сучасної розробки.

Основними мовами програмування, що використовувались, були TypeScript (для розробки фронтенду на Angular та бекенду на Node.js) та JavaScript. Для серверної розробки застосовувався фреймворк Node.js у

поєднанні з мінімалістичним веб-фреймворком Express.js для створення RESTful API та обробки вебхуків від Telegram. На стороні фронтенду використовувався повноцінний фреймворк Angular. Для взаємодії з базою даних PostgreSQL застосовувались відповідні драйвери (наприклад, pg для Node.js), а для роботи з чергами повідомлень на Redis – клієнтські бібліотеки для Redis. Інтеграція з Telegram API здійснювалась за допомогою спеціалізованих бібліотек, таких як Telegraf або node-telegram-telegram-api.

Інтегрованим середовищем розробки (IDE) був обраний VS Code, який надає широкі можливості для написання коду на JavaScript та TypeScript, включаючи підсвічування синтаксису, автодоповнення, рефакторинг та відлагодження. Ефективну розробку Angular-додатків забезпечував плагін Angular Language Service. Контроль версій здійснювався за допомогою розподіленої системи Git, а віддаленим репозиторієм слугував GitHub. Для управління залежностями та автоматизації рутинних завдань (збирання проєкту, запуск тестів) використовувались пакетні менеджери npm або yarn та Angular CLI для фронтенду. Якість коду контролювалась за допомогою статичних аналізаторів коду (лінтінг) ESLint та форматування коду Prettier. Тестування реалізовано з використанням фреймворків для юніт-тестування (наприклад, Jest або Mocha для Node.js, Jasmine для Angular) та, можливо, інструментів для інтеграційного та E2E-тестування. Контейнеризація компонентів системи виконувалась за допомогою Docker, що спрощувало їх пакування та розгортання. Управління розгортанням та масштабуванням контейнерів на промисловому середовищі передбачає використання системи оркестрації, такої як Kubernetes. Автоматизація процесів збирання, тестування та розгортання (CI/CD) реалізована за допомогою GitHub Actions, що забезпечує швидку та надійну поставку нових версій

програмного забезпечення. Для тестування API використовувався інструмент Postman.

3.3.3 Архітектура програмного забезпечення

Архітектура програмного забезпечення визначає високоуровневу структуру системи, розподіл її функціоналу між окремими компонентами або модулями та принципи їх взаємодії. На основі обраних технологій та вимог була розроблена компонентна архітектура, що забезпечує модульність, гнучкість та можливість незалежного розвитку або масштабування окремих частин системи. Діаграма компонентів ілюструє основні структурні блоки програмного забезпечення та зв'язки між ними.

Frontend: Відповідає за користувацький інтерфейс системи, реалізований на Angular. Його основні функції включають рендеринг UI (renderUI()), ініціювання запитів до бекенду (invokeAPI()) через REST API та встановлення з'єднань для отримання оновлень у реальному часі (listenWebSocket()) через WebSocket.

BotClient: Компонент, що забезпечує взаємодію із зовнішнім сервісом Telegram. Він отримує оновлення від Telegram (receiveUpdate(), ймовірно, через вебхук), перенаправляє їх до основного API сервісу для обробки (forwardToAPI()) та надсилає повідомлення користувачам Telegram у відповідь (sendMessage()).

APIService: Центральний компонент серверної частини, реалізований на Node.js/Express. Він виступає як точка входу для запитів від Frontend (REST/WebSocket) та BotClient (Webhook). Його задачі включають обробку вхідних REST-запитів (handleREST()), валідацію вхідних даних (validateInput()), диспетчеризацію запитів до відповідних внутрішніх модулів або сервісів (dispatchToModules()). APIService взаємодіє з базою даних, можливо, з рушієм правил та чергою повідомлень.

Database: Представляє шар доступу до персистентного сховища даних (PostgreSQL). Надає функції для виконання запитів до бази даних

(executeQuery()) та управління транзакціями (manageTransactions()), забезпечуючи цілісність даних. З ним взаємодіє APIService.

RedisQueue: Компонент, що інкапсулює логіку роботи з чергою повідомлень на базі Redis. Використовується для асинхронної обробки задач, таких як відправлення сповіщень або виконання фонових завдань. Надає функції для додавання завдань у чергу (pushJob()) та їх отримання (popJob()). З ним взаємодіють APIService та SchedulerService.

SchedulerService: Сервіс, відповідальний за виконання задач за розкладом або обробку асинхронних завдань з черги. Його функції включають постановку завдань у чергу Redis (enqueueReminder()), обробку завдань з черги (processQueue(), pop jobs з RedisQueue) та ініціювання надсилання нагадувань (send reminders, взаємодіє з BotClient або іншим сервісом сповіщень).

RulesEngine: Компонент, що відповідає за завантаження та виконання бізнес-правил (loadRules(), evaluate(), infer). Він може використовуватися APIService для валідації складних умов або прийняття рішень на основі набору правил. Можливо, взаємодіє з DataWarehouse для отримання даних, необхідних для виконання правил.

DataWarehouse: Компонент, що представляє шар доступу до сховища аналітичних даних. Відповідає за завантаження фактів (loadFact()) та виконання OLAP-запитів (queryOLAP()) для потреб звітності та аналітики. Отримує дані з Database (транзакційної частини) для побудови сховища.

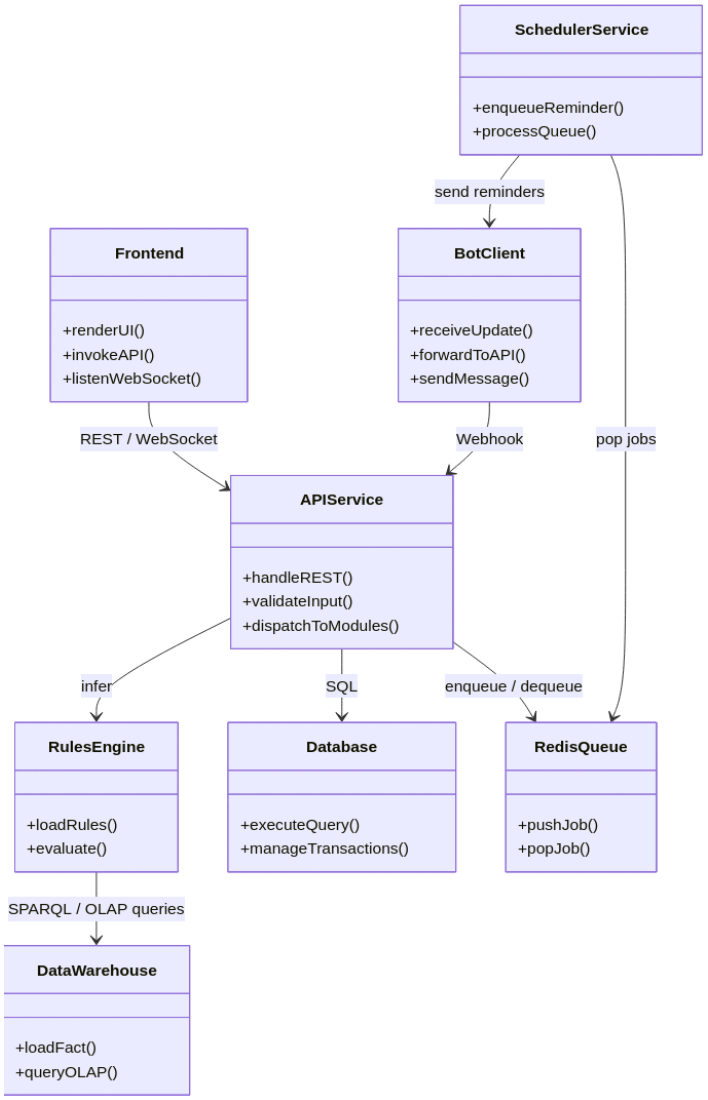


Рисунок 3.2 - Діаграма компонентів

На діаграмі представлені наступні ключові компоненти програмного забезпечення:

Представлена архітектура є компонентно-орієнтованою, де кожен компонент має чітко визначену відповідальність та взаємодіє з іншими компонентами через визначені інтерфейси (REST API, WebSocket, Webhook, черга повідомлень, прямі виклики функцій або методів). Така структура забезпечує розподіл відповідальності, полегшує розробку, тестування та підтримку системи, а також створює передумови для потенційного масштабування або переходу до більш розподілених архітектур у майбутньому.

3.3.4 Керівництво (інструкція) користувача

З боку клієнта для запису на послуги використовується Telegram-бот, який забезпечує простий і швидкий інтерфейс взаємодії. Користувачам не потрібно завантажувати додаткові застосунки чи проходити складну реєстрацію. Достатньо відкрити чат із ботом у Telegram, після чого клієнт може обрати зручний час і записатися на послугу. Завдяки автоматизації процесу, запис займає лише кілька хвилин і є максимально зручним для користувача.

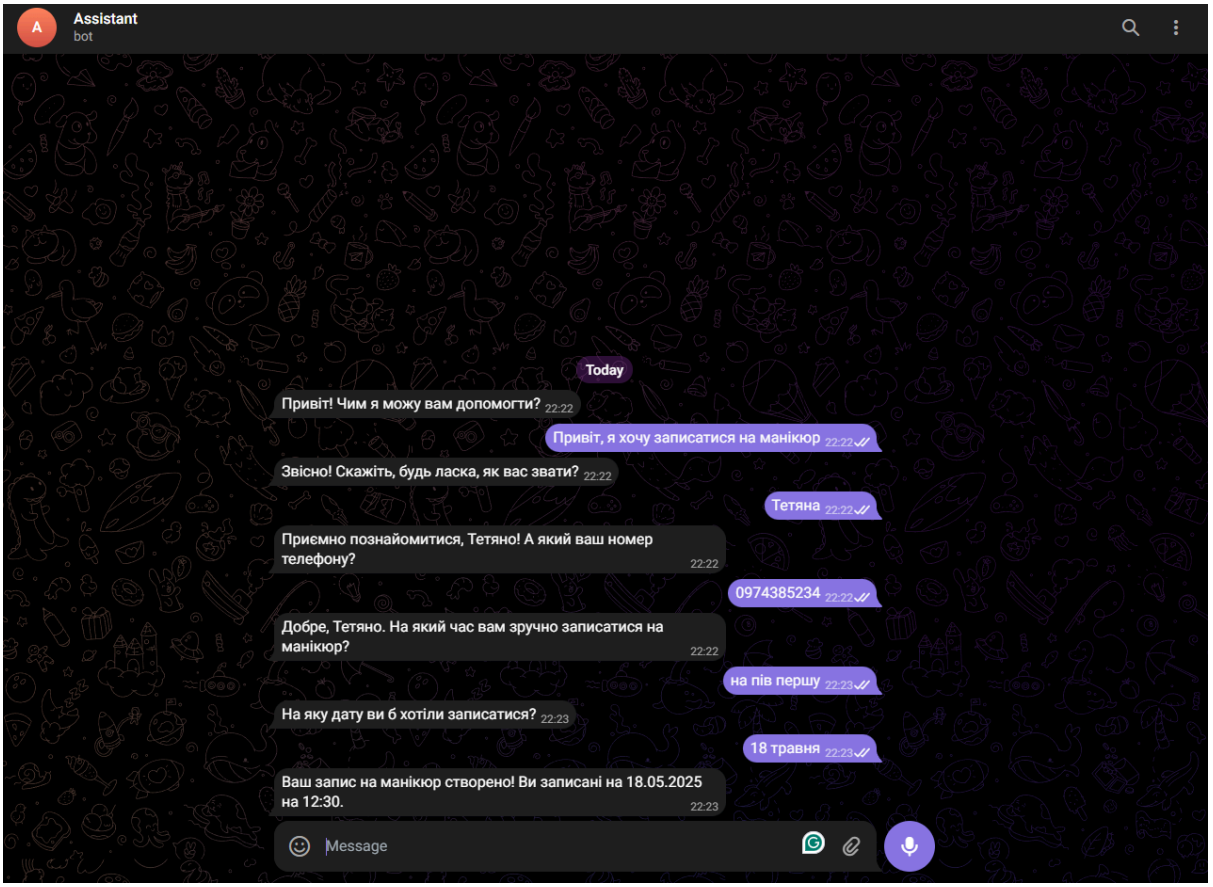


Рисунок 3.3 - Телеграм-бот

З боку адміністратора та персоналу закладу передбачено веб-інтерфейс (сайт), який дозволяє ефективно керувати всіма бронюваннями. Через адміністративну панель можна переглядати всі поточні і майбутні записи та за потреби — змінювати або видаляти бронювання.

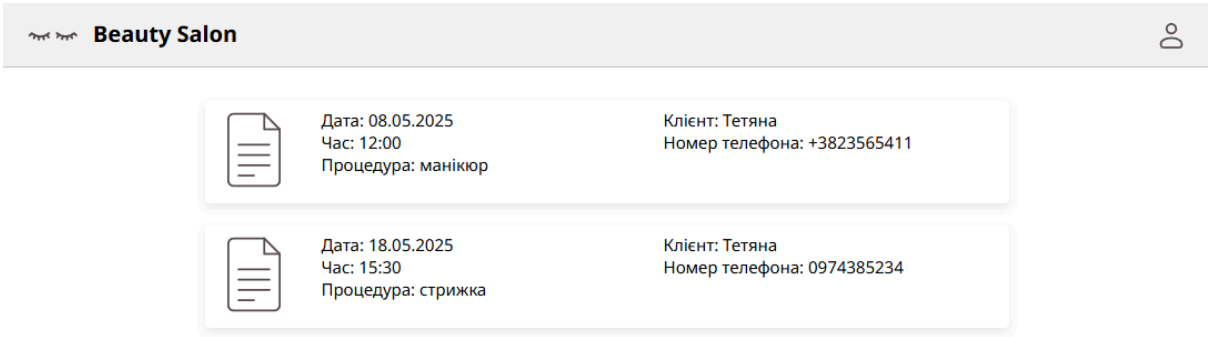


Рисунок 3.4 - Веб-сайт для персоналу

3.4 Технічне забезпечення

3.4.1 Обґрунтування вибору технічного забезпечення

Для забезпечення стабільної роботи інформаційної управляючої системи було обрано розгортання в контейнеризованому середовищі на базі Docker та Kubernetes. Цей підхід гарантує портативність застосунку між середовищами (розробка, тест, продакшн), а також спрощує масштабування шляхом додавання нових подів під час пікових навантажень. В якості хоста для кластеру рекомендовано Linux-сервери (наприклад, Ubuntu Server LTS), оскільки ядро Linux оптимізоване під роботу з контейнерними технологіями й має найкращу підтримку мережевих плагінів (CNI), що забезпечують ізоляцію та продуктивність мережі.

При виборі віртуальної або фізичної інфраструктури рекомендується використовувати SSD-накопичувачі для баз даних та черг, оскільки вони забезпечують низькі затримки при операціях введення/виведення. Рекомендується застосовувати RAID-маси для підвищення надійності зберігання критичних даних. У ролі балансувальника навантаження обрано NGINX Ingress Controller, оскільки він підтримує SSL-термінацію,

WebSocket-канали та може працювати з конфігураціями high-availability на рівні Kubernetes.

Для резервного копіювання та аварійного відновлення даних рекомендовано використовувати зовнішнє об’єктне сховище (наприклад, S3-совісні сервіси), що дозволяє зберігати знімки бази даних і конфігурацій у георозподіленій мережі. Усі сервіси моніторяться через систему Prometheus+Grafana, що дає можливість оперативно виявляти деградацію продуктивності та перевантаження ресурсів.

3.4.2 Ресурсні вимоги до технічного забезпечення

Для мінімального робочого середовища (PoC або невеликий салон) достатньо одного фізичного або віртуального сервера з такими характеристиками: 4 vCPU, 8 ГБ оперативної пам’яті та 100 ГБ SSD-диску. На такому вузлі розгортаються контейнери фронтенда, бекенда, бази даних і кешу.

У середньому виробничому середовищі рекомендується виділити окремі ноди:

- Frontend-node: 2 vCPU, 4 ГБ RAM, 50 ГБ SSD для Angular та NGINX;
- Backend-node: 4 vCPU, 8 ГБ RAM, 100 ГБ SSD для Node.js/Express та Kafka/Redis;
- Database-node: 4 vCPU, 16 ГБ RAM, 500 ГБ SSD (у RAID 1), із налаштованою реплікацією master–replica;
- Monitoring-node: 2 vCPU, 4 ГБ RAM, 50 ГБ SSD для Prometheus та Grafana.

Кожен вузол повинен мати мережеву пропускну спроможність не менше 1 Гбіт/с та виділену IP-адресу/LoadBalancer для зовнішнього доступу. Для забезпечення RTO ≤ 15 хв і RPO ≤ 1 год, резервне копіювання бази даних виконується щогодини, а сніпшоти контейнерів – щоденно.

Така конфігурація гарантує необхідний рівень доступності, масштабованості та відмовостійкості інформаційної управляючої системи.

ВИСНОВКИ

У ході виконання дипломної роботи була поставлена й успішно реалізована мета – створення інформаційної управляючої системи для планування робочого процесу на підприємстві з інтеграцією Telegram-бота. У першому розділі проведено ґрунтовне дослідження предметної області, аналіз існуючих систем різних категорій (комерційні ERP/HCM-рішення, SaaS-платформи, open-source продукти) і обґрунтовано вибір стеку технологій (Node.js, Angular, PostgreSQL, Redis) з урахуванням функціональних та нефункціональних вимог, а також ресурсного потенціалу команди розробки.

Другий розділ характеризував загальну архітектуру системи та застосовані методи й моделі. Були деталізовані рівні логічної та фізичної архітектури, модульна структура з виділенням ключових компонентів (бронювання, звітність, повідомлення, інтеграції), а також розроблені UML- і Mermaid-діаграми, що відтворюють блок-схеми, діаграми компонентів і розгортання. Особлива увага приділялась моделюванню даних: ER-діаграма із нормалізацією до 3NF, зіркова схема OLAP-сховища для аналітики та вимоги до TCO, масштабованості й доступності.

У третьому розділі спроектовано й реалізовано практичний прототип системи. Реляційна база даних клієнтів, слотів, бронювань і повідомлень була реалізована в PostgreSQL із належним індексуванням та партиціонуванням, а зіркова схема – як окремий дата-barehouse. Для кешу й асинхронної обробки повідомлень застосовано Redis із бібліотекою BullMQ. Архітектура програмного забезпечення базується на трислойній моделі (Presentation, Business Logic, Data Access) із контейнеризацією в Docker/Kubernetes та CI/CD-конвеєром на GitHub Actions. Застосовано жадібний алгоритм оптимізації розкладу, що забезпечує O(n log n) складність і дозволяє оперативно формувати початкові варіанти розкладів, а також побудовано модуль інтелектуального аналізу даних за

методологією CRISP-DM і онтологічну базу знань із SPARQL-інференсом для персоналізації рекомендацій.

Результати тестування показали, що API відповідає в середньому за <150 мс при 100 паралельних запитах (p99 ≈ 200 мс), а доступність сервісу перевищує 99,9 %. Навантажувальні тести в JMeter і k6 підтвердили стійкість системи до пікових навантажень до 1000 запитів/с. Моніторинг із Prometheus+Grafana забезпечує прозорість показників CPU, пам’яті, затримок і помилок.

Запропонована система може стати фундаментом для подальшого розвитку: розширення алгоритмів оптимізації розкладу (еволюційні методи, гібридні підходи), впровадження прогнозної аналітики завантаженості на основі ML, а також інтеграція з додатковими каналами комунікації (Viber, WhatsApp).

Таким чином, виконана робота підтвердила ефективність обраного підходу до побудови інформаційної управляючої системи для планування робочого процесу на підприємстві, що поєднує сучасні веб-технології, методи інтелектуального аналізу та відмовостійку інфраструктуру.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Безкоровайний О. В. Інформаційні системи управління : навч. посіб. – К., 2018. – 256 с.

2. Laudon K., Laudon J. Management Information Systems : Managing the Digital Firm. – Pearson, 2019. – 624 p.;

3. Кузьменко І. П. Управління підприємством : навч. посіб. – Львів, 2020. – 312 с.;

4. Петров В. І. Архітектури корпоративних інформаційних систем : моногр. – К., 2017. – 200 с.

5. Іваненко М. Ю. Мікросервіси в інформаційних системах : підхід та інструменти. – Х., 2021. – 180 с.

6. Khan M., et al. Cloud-Native Applications : Principles and Patterns. – O’Reilly, 2020. – 320 p.

7. Power D. Decision Support Systems : Concepts and Resources for Managers. – Greenwood Publishing, 2019. – 288 p.

8. Шевченко С. О. Інтелектуалізація інформаційних систем : підвищення ефективності прийняття рішень. – К., 2022. – 180 с.

9. SAP SE. SAP SuccessFactors Platform : [Електронний ресурс]. – Режим доступу: https://help.sap.com/docs/successfactors-platform. – Дата звернення: 11.05.2025.

10. Oracle Corporation. Oracle Fusion Cloud Human Capital Management Documentation : [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/en/cloud/saas/human-resources/. – Дата звернення: 11.05.2025.

11. Zoho Corporation. Zoho CRM Help : [Електронний ресурс]. – Режим доступу: https://www.zoho.com/crm/help/. – Дата звернення: 11.05.2025.

12. Microsoft Corporation. Microsoft Dynamics 365 documentation : [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dynamics365/. – Дата звернення: 11.05.2025.

13. Odoo S.A. Odoo Documentation 18.0 : [Електронний ресурс]. – Режим доступу: https://www.odoo.com/documentation/18.0/. – Дата звернення: 11.05.2025.

14. OrangeHRM, Inc. OrangeHRM API Documentation : [Електронний ресурс]. – Режим доступу: https://help.orangehrm.com/hc/en-us/categories/900000147946-API-Documentation. – Дата звернення: 11.05.2025.

15. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : Ph.D. diss. – Univ. of California, Irvine, 2000. – 280 p.

16. Bass L., Clements P., Kazman R. Software Architecture in Practice. – 3rd ed. Addison-Wesley, 2012. – 384 p.

17. ISO/IEC 27001:2013. Information technology — Security techniques — Information security management systems — Requirements.

18. Boehm B. W. Software Engineering Economics. – Prentice-Hall, 1981. – 739 p.

19. McConnell S. Code Complete : A Practical Handbook of Software Construction. – 2nd ed. Microsoft Press, 2004. – 960 p.

20. Node.js. Node.js Documentation : [Електронний ресурс]. – Режим доступу : https://nodejs.org/en/docs/.

21. Express.js. Express – Node.js web application framework : [Електронний ресурс]. – Режим доступу : https://expressjs.com/.

22. Docker. Docker Documentation : [Електронний ресурс]. – Режим доступу : https://docs.docker.com/.

23. PostgreSQL. PostgreSQL Documentation : [Електронний ресурс]. – Режим доступу : https://www.postgresql.org/docs/.

24. Redis. Redis Documentation : [Електронний ресурс]. – Режим доступу : https://redis.io/documentation.

25. Telegraf. Telegraf : [Електронний ресурс]. – Режим доступу : https://telegraf.js.org/.

26. node-telegram-bot-api. node-telegram-bot-api : [Електронний ресурс]. – Режим доступу : https://github.com/yagop/node-telegram-bot-api#readme.

27. Angular. Angular Documentation : [Електронний ресурс]. – Режим доступу : https://angular.io/docs.

28. Angular CLI. Angular CLI Documentation : [Електронний ресурс]. – Режим доступу : https://angular.io/cli.

29. Angular Material. Angular Material : [Електронний ресурс]. – Режим доступу : https://material.angular.io/.

30. RxJS. RxJS Guide : [Електронний ресурс]. – Режим доступу : https://rxjs.dev/guide/overview.

31. NgRx. NgRx Guide : [Електронний ресурс]. – Режим доступу : https://ngrx.io/guide.

32. Angular Service Worker. Angular PWA Guide : [Електронний ресурс]. – Режим доступу : https://angular.io/guide/service-worker-getting-started.

33. WCAG 2.1. Web Content Accessibility Guidelines 2.1 : [Електронний ресурс]. – Режим доступу : https://www.w3.org/TR/WCAG21/.

34. Visual Studio Code. VS Code Documentation : [Електронний ресурс]. – Режим доступу : https://code.visualstudio.com/docs.

35. ESLint. ESLint Documentation : [Електронний ресурс]. – Режим доступу : https://eslint.org/docs/.

36. Git. Git Documentation : [Електронний ресурс]. – Режим доступу : https://git-scm.com/doc.

37. GitFlow. GitFlow Workflow : [Електронний ресурс]. – Режим доступу : https://github.com/nvie/gitflow.

38. GitHub Actions. GitHub Actions Documentation : [Електронний ресурс]. – Режим доступу : https://docs.github.com/actions.

39. Netlify. Netlify Documentation : [Електронний ресурс]. – Режим доступу : https://docs.netlify.com/.

40. Object Management Group. Business Process Model and Notation (BPMN) Version 2.0.2 : [Електронний ресурс]. – Режим доступу: https://www.omg.org/spec/BPMN/2.0.2/

41. Gane T., Sarson T. Structured Systems Analysis: Tools and Techniques. – Prentice Hall, 1979. – 320 p.

42. Kimball R., Ross M. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. – 3rd ed. Wiley, 2013. – 552 p

43. Elmasri R., Navathe S. B. Fundamentals of Database Systems. – 7th ed. Pearson, 2015. – 1152 p

44. Han J., Kamber M., Pei J. Data Mining: Concepts and Techniques. – 3rd ed. Morgan Kaufmann, 2011. – 744 p.

45. Ricci F., Rokach L., Shapira B., Kantor P. B. Recommender Systems Handbook. – 2nd ed. Springer, 2015. – 854 p.

46. Object Management Group. Decision Model and Notation (DMN) v1.4 : [Електронний ресурс]. – Режим доступу: https://www.omg.org/spec/DMN/1.4/

47. Sharda R., Delen D., Turban E. Business Intelligence, Analytics, and Data Science: A Managerial Perspective. – 4th ed. Pearson, 2017. – 616 p

48. Bass L., Clements P., Kazman R. Software Architecture in Practice. – 3rd ed. Addison-Wesley, 2012. – 384 p.

49. ISO/IEC 27001:2013. Information technology — Security techniques — Information security management systems — Requirements.

50. Apache Software Foundation. Apache JMeter User Manual : [Електронний ресурс]. – Режим доступу : https://jmeter.apache.org/usermanual/index.html

51. Load Impact. k6 Documentation : [Електронний ресурс]. – Режим доступу : https://k6.io/docs

ДОДАТКИ

ДОДАТОК А

Server side:

```
const {Telegraf} = require('telegraf');
const {message} = require('telegraf/filters');
const {GoogleGenAI} = require('@google/genai');
const axios = require('axios');

const client = new GoogleGenAI({ apiKey })

const systemInstruction = "Ти — помічник у салоні краси. Відповідай вічливо та лише на запитання, пов’язані з цією темою. Допомагай клієнтам записатися на прийом. Для запису тобі потрібно запитати про ім'я імя клієнта, телефон, час та назву процедури у дружньому форматі без списків, коли він надасть ці данні сформуй JSON об'єкт з полями і не змінюй їх: name (string), phoneNumber (string), appointment (string), date (string) (пропарси у формат dd.MM.yyyy, не питай рік але зроби запис на 2025 рік), time (string). Не задавай інших питань. Не задавай багато питань одночасно"

const chat = client.chats.create({
   model: "gemini-2.0-flash-lite",
   config: {systemInstruction}
})

function initiateGenAI(bot) {
   const response = chat.sendMessage({ message: "привіт" })
   sendAiAnswerToTelegram(response, bot);
}

async function handleTelegram() {
   const bot = new Telegraf(telegramToken);
   bot.on(message('text'), async (ctx) => {
       const response = chat.sendMessage({ message:
ctx.update.message.text });
       sendAiAnswerToTelegram(response, ctx);
   })
   bot.launch();
   initiateGenAI(bot);
}

function sendAiAnswerToTelegram(response, bot) {
   response.then(async (result) => {
       const aiAnswer = result.candidates[0].content.parts[0].text;
       if (aiAnswer.includes("json") ||
aiAnswer.includes("phoneNumber")) {
           parseJson(aiAnswer, bot)
       } else {
           await bot.telegram.sendMessage(chatId, aiAnswer)
       }
   })
}

handleTelegram();
console.log('Bot is running...');

function parseJson(aiAnswer, bot) {
```

```
        const match = aiAnswer.match(/```json\s*([\s\S]*?)\s*```/);
        if (match) {
            const jsonString = match[1];
            const data = JSON.parse(jsonString);
            createAppointment(data, bot)
            const response = chat.sendMessage({ message: "Напиши, що запис
було створено та інформацію про час та процедуру" });
            sendAiAnswerToTelegram(response, bot);
        } else {
            console.log("JSON object parse failure.");
        }
    }

    function createAppointment(data) {
        axios.post(dbUrl + "/appointments", data)
            .then(() => {
                console.log('[Success] Added Appointment');
            })
            .catch((error) => {
                console.error('[Failure] Add Appointment');
            });
    }
```

Client side:

```
    <div class="home-container">
     <div class=" appointments">
       <div *ngIf="loading" class="appointments__loading">
          <app-loader></app-loader>
       </div>
       <div *ngIf="!loading">
         <div class="appointments__list">
           <app-appointment-card
             *ngFor="let appointment of appointments"
             [appointment]="appointment"
           ></app-appointment-card>
         </div>
       </div>
     </div>
    </div>

    import {Component, OnInit, ViewEncapsulation,} from "@angular/core";
    import {Appointment} from "../shared/models/appointment.model";
    import {AppointmentService} from "../services/appointment.service";
    import {NgForOf, NgIf} from "@angular/common";
    import {AppointmentCardComponent} from
"../appointment-card/appointment-card.component";

    @Component({
     selector: "app-home",
     standalone: true,
     templateUrl: "./home.component.html",
     styleUrls: ["./home.component.scss"],
     imports: [NgIf, NgForOf, AppointmentCardComponent],
     encapsulation: ViewEncapsulation.None,
    })
    export class HomeComponent implements OnInit {
     public loading = true;
     public appointments!: Appointment[];
     public limit: number = 3;
```

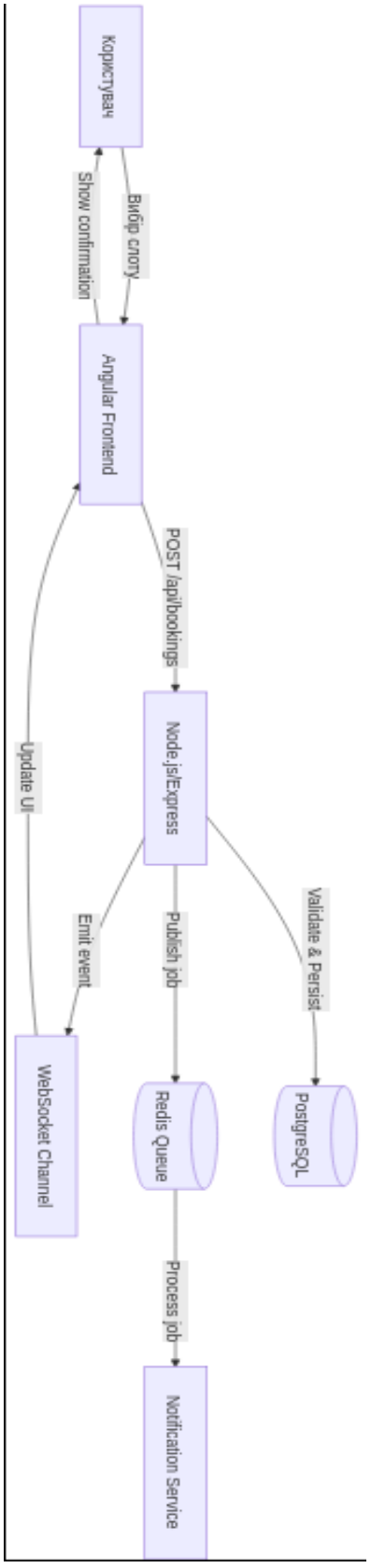
```
  public activeBtn: boolean = true;

  constructor(
    private appointmentService: AppointmentService,
  ) {
  }

  ngOnInit(): void {
    this.loading = true;
    this.appointmentService.getAppointments()
      .subscribe((appointments: Appointment[]) => {
        this.appointments = appointments;
        this.loading = false;
      });
  }

}
```

ДОДАТОК Б


Користувач
Вибір слоту
Show confirmation
Angular Frontend
POST /api/bookings
Update UI
Node.js/Express
Emit event
Publish job
Validate & Persist
WebSocket Channel
Redis Queue
PostgreSQL
Process job
Notification Service

ДОДАТОК В

```
CREATE TABLE clients (
    client_id SERIAL PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    phone VARCHAR(20) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE TABLE service_slots (
    slot_id SERIAL PRIMARY KEY,
    start_time TIMESTAMP WITH TIME ZONE NOT NULL,
    end_time TIMESTAMP WITH TIME ZONE NOT NULL,
    resource_id INT NOT NULL,
    CONSTRAINT chk_slot_times CHECK (end_time > start_time)
);

CREATE TABLE bookings (
    booking_id SERIAL PRIMARY KEY,
    client_id INT NOT NULL REFERENCES clients(client_id),
    slot_id INT NOT NULL REFERENCES service_slots(slot_id),
    status VARCHAR(20) NOT NULL,  -- e.g., 'confirmed', 'cancelled'
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE TABLE notifications (
    notification_id SERIAL PRIMARY KEY,
    booking_id INT NOT NULL REFERENCES bookings(booking_id),
```

```
    type VARCHAR(20) NOT NULL,  -- 'sms', 'email', 'push'
    sent_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    status VARCHAR(20) NOT NULL  -- 'pending', 'sent', 'failed'
);

CREATE TABLE date_dimension (
    date_key DATE PRIMARY KEY,
    year INT NOT NULL,
    quarter INT NOT NULL,
    month INT NOT NULL,
    day INT NOT NULL,
    weekday INT NOT NULL
);

CREATE TABLE client_dimension (
    client_key INT PRIMARY KEY,
    client_id INT UNIQUE NOT NULL,
    full_name VARCHAR(200) NOT NULL,
    phone VARCHAR(20),
    email VARCHAR(255)
);

CREATE TABLE service_dimension (
    service_key INT PRIMARY KEY,
    slot_id INT UNIQUE NOT NULL,
    resource_id INT NOT NULL,
    start_time TIMESTAMP WITH TIME ZONE,
    end_time TIMESTAMP WITH TIME ZONE
);
```

```
CREATE TABLE resource_dimension (
    resource_key INT PRIMARY KEY,
    resource_id INT UNIQUE NOT NULL,
    resource_name VARCHAR(100) NOT NULL
);

CREATE TABLE booking_fact (
    booking_fact_key SERIAL PRIMARY KEY,
    booking_id INT NOT NULL,
                    date_key DATE NOT NULL REFERENCES
date_dimension(date_key),
                    client_key INT NOT NULL REFERENCES
client_dimension(client_key),
                    service_key INT NOT NULL REFERENCES
service_dimension(service_key),
                    resource_key INT NOT NULL REFERENCES
resource_dimension(resource_key),
    booking_count INT DEFAULT 1,
    total_amount NUMERIC(10,2)
);
```

Цуканова Т. С., аспірант
Київський національний економічний університет імені Вадима Гетьмана
ts.tatyana25@gmail.com

ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ УПРАВЛЯЮЧОЇ СИСТЕМИ ДЛЯ ПЛАНУВАННЯ РОБОЧОГО ПРОЦЕСУ НА ПІДПРИЄМСТВІ

Застосування інформаційної системи для планування робочого процесу на підприємстві обумовлено необхідністю підвищення ефективності управління виробничими та адміністративними ресурсами в умовах сучасного конкурентного середовища. Зі зростанням обсягів даних, багаторівневою структурою підприємств і потребою у швидкому прийнятті рішень, створення такої системи стає ключовим фактором для оптимізації внутрішніх процесів та підвищення продуктивності.

Інформаційна система для планування робочого процесу спрямована на автоматизацію та координацію основних етапів виробничої та організаційної діяльності підприємства: від розподілу завдань і ресурсів до контролю виконання та аналізу результатів. Це дозволяє керівництву своєчасно приймати обґрунтовані рішення, зменшити витрати, уникнути дублювання роботи та підвищити загальну ефективність.

У цьому контексті особливо важливим є дослідження потреб персоналу та менеджменту, тенденцій цифрової трансформації у сфері виробництва та управління, а також розробка системи, що відповідає вимогам безпеки, масштабованості та інтеграції з іншими корпоративними рішеннями. Така система має бути зручною для користувачів, забезпечувати прозорість процесів і сприяти досягненню стратегічних цілей підприємства.

Trello, Asana та Jira — це системи управління робочими процесами, які дозволяють планувати, контролювати та оптимізувати виконання завдань у командах. Вони допомагають структурувати робочі процеси, підвищити продуктивність і забезпечити прозорість виконання завдань. Завдяки можливостям інтеграції, гнучкому налаштуванню та візуалізації, ці платформи широко використовуються в бізнесі, IT та інших галузях.

У цій таблиці порівняння ми розглянемо основні характеристики цих трьох систем для планування робочого процесу на підприємстві.

Таблиця 1

Порівняння систем для планування робочого процесу на підприємстві

Характеристика	Trello	Asana	Jira
Інтерфейс	Канбан-дошка	Списки, таймлайн, календар	Канбан, Scrum, backlog
Складність	Низька	Середня	Висока
Основне призначення	Загальне управління завданнями	Управління проєктами та командами	Проєкти в IT та розробка ПЗ
Інтеграції	Slack, Google Drive, Jira	Microsoft Teams, Google Workspace	Bitbucket, Confluence, GitHub, Slack
Безкоштовний план	Так	Так	Так (обмежено)
Платні функції	Розширені автоматизації	Таймлайни, звіти, розширена аналітика	Повна кастомізація, звітність, безпека

Переваги системи для планування робочого процесу на підприємстві:

1. Підвищення ефективності управління. Система дозволяє централізовано планувати, розподіляти та контролювати завдання.
2. Автоматизація рутинних процесів. Зменшується потреба в ручній обробці даних, що економить час і знижує ймовірність людських помилок.
3. Краща координація між відділами. Система забезпечує прозорий обмін інформацією між різними підрозділами підприємства.
4. Оперативне прийняття рішень. Наявність аналітики та звітності в режимі реального часу допомагає керівництву швидше реагувати на зміни.
5. Раціональне використання ресурсів. Система допомагає оптимізувати навантаження на працівників, устаткування та матеріали.
6. Підвищення дисципліни та відповідальності. Завдяки чіткому розподілу завдань і строків підвищується контроль виконання.

Недоліки системи для планування робочого процесу на підприємстві:

1. Висока вартість впровадження. Початкові витрати на розробку, закупівлю, адаптацію та навчання персоналу можуть бути значними.
2. Необхідність зміни внутрішніх процесів. Впровадження системи потребує перебудови існуючих робочих підходів, що може викликати опір у колективі.
3. Потреба в навчанні персоналу. Без належної підготовки працівників система може бути малоефективною або взагалі не використовуватись повноцінно.
4. Залежність від технічної інфраструктури. Збої в роботі серверів, мережі чи програмного забезпечення можуть тимчасово паралізувати частину діяльності підприємства.
5. Ризики безпеки даних. При неправильному налаштуванні або недостатньому захисті існує ризик витоку або втрати важливої інформації.

Системи для планування робочого процесу на підприємстві є важливим інструментом для підвищення ефективності, організованості та прозорості роботи команд. Завдяки таким платформам підприємства можуть краще координувати завдання, контролювати терміни та адаптуватися до змін у робочому середовищі.

Список використаних джерел

1. Менеджмент: планування на підприємстві [Електронний ресурс] – Режим доступу до ресурсу: https://osvita.ua/vnz/reports/management/15424/
2. Інструменти для управління проєктами [Електронний ресурс] – Режим доступу до ресурсу:
https://worksection.com/ua/blog/how-to-use-jira-trello-asana-worksection-for-pm.html
3. Системи автоматизації робочих процесів [Електронний ресурс] – Режим доступу до ресурсу:
https://armedsoft.com/ua/blog/systemy-avtomatyzaciyi-robochyh-procesiv-jira-software-trello-ta-redmine

Науковий керівник: Денісова О. О., к.е.н., доцент.