Міністерство освіти і науки України
Одеський національний технологічний університет
Кафедра комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему Розробка багатоплатформової системи керування замовленнями послуг
(назва кваліфікаційної роботи згідно наказу ОНТУ)

Здобувача *Гараса С. Я.*
(прізвище, ініціали)

2 (скор.) курсу 543а групи

Керівники: *д.т.н., проф. Артеменко С.В.*
(посада, прізвище та ініціали)
ст. викл. Сіренко О.І.
(посада, прізвище та ініціали)

Консультанти:
(посада, прізвище та ініціали)
Phd, ст.викл. Богданов О.О.
(посада, прізвище та ініціали)

Кваліфікаційна робота допускається до захисту

Рішення кафедри від *05.06* 2024 р., протокол № *8*

Завідувач кафедри комп. інженерії _____ Сергій АРТЕМЕНКО
(назва кафедри) (підпис) (Ім'я ПРІЗВИЩЕ)

Одеса – 2024 рік

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерної інженерії, програмування та кіберзахисту*
Кафедра *комп'ютерної інженерії*
Ступінь вищої освіти *бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*
Освітня програма *Мережеві технології та інтернет речей*

ЗАТВЕРДЖУЮ
Зав. кафедри *комп'ютерної інженерії*
Сергій АРТЕМЕНКО
« *30* » *серпня* 2023 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Гараса Сергія Ярославовича

1. Тема роботи *Розробка багатоплатформової системи керування замовленнями послуг*

Затверджена наказом університету від « *30* » *серпня* 2023 р., наказ № *442-03*

2 Термін здачі здобувачем закінченої роботи *28 травня 2024 р.*

3. Вихідні дані роботи

1. Середовища програмування Android Studio, VS Code. 2. Мова програмування Typescript.
3. Текстовий редактор Microsoft Word. 4. Редактор презентацій Microsoft PowerPoint.

4. Перелік питань, які потрібно розробити

1. Вступ. 2. Збір та аналіз інформації. 3. Проектування системи.
4. Розробка системи. 5. Економічні розрахунки.
6. Охорона праці. 7. Загальні висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайд 1. Тема роботи. Слайд 2. Об'єкт та предмет дослідження. Слайд 3. Актуальність.
Слайд 4. Аналоги. Слайд 5. Користувацький інтерфейс.
Слайд 6. Розгортання функціональних одиниць. Слайд 7. Економічні розрахунки.
Слайд 8. Схема БД. Слайд 9. Загальні висновки.

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Економіка	*Phd, ст. викл. Богданов О.О.*		
Охорона праці	*д.т.н., проф. Артеменко С.В.*		
Нормоконтроль	*ст. викл. Сіренко О.І.*		

7. Дата видачі завдання *30.08.2023*

Керівники *Сергій АРТЕМЕНКО*

Олександр СІРЕНКО

Завдання прийняв до виконання *Сергій ГАРАС*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Дослідженні предметної області*	*26.10.2023*	
2.	*Дослідження існуючих аналогів*	*30.11.2023*	
3.	*Дослідження методів реалізації*	*28.01.2023*	
	багатоплатформових систем		
4.	*Проектування*	*15.02.2024*	
5.	*Розробка демонстраційної версії ПЗ*	*27.03.2024*	
6.	*Підготовка техніко-економічної частини*	*15.04.2024*	
7.	*Підготовка розділу охорони праці*	*15.04.2024*	
8.	*Оформлення пояснювальної записки*	*27.05.2024*	
9.	*Оформлення графічної частини та лістингу*	*27.05.2024*	

Здобувач-дипломник _____ *Сергій ГАРАС*

Керівники роботи _____ *Сергій АРТЕМЕНКО*

_____ *Олександр СІРЕНКО*

Несу відповідальність за ідентичність електронного та друкованого варіантів кваліфікаційної роботи, даю згоду на обробку персональних даних та не заперечую проти розміщення кваліфікаційної роботи на офіційних web-ресурсах ОНТУ.

Підтверджую, що в кваліфікаційній роботі відсутні порушення норм академічної доброчесності.

Здобувач-дипломник *Сергій ГАРАС* _____

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці багатоплатформової системи управління замовленнями послуг. Ця система призначена для оптимізації процесу замовлення та надання послуг між користувачами та провайдерами, включаючи автоматизацію багатьох бізнес-процесів, що в свою чергу забезпечує підвищення ефективності роботи та задоволення клієнтів.

В першому розділі проведено аналіз предметної області та дослідження існуючих аналогів. Розглянуто сучасні тенденції та підходи у створенні систем управління замовленнями, а також основні вимоги до подібних систем

Другий розділ зосереджено на формуванні технічного завдання і розробці архітектури системи. Сформульовано основні вимоги до системи, включаючи її функціональність, безпеку та масштабованість.

Третій розділ описує процес розробки системи, включаючи детальний опис реалізації бази даних, серверної логіки та клієнтського інтерфейсу. Обговорюються питання інтеграції різних модулів системи та заходи, прийняті для забезпечення високого рівня безпеки.

В четвертому розділі було проведено аналіз економічних показників розробленої системи, який підтвердив її високу економічну вигоду та ефективність.

У п'ятому розділі розглядаються питання охорони праці та забезпечення безпеки при використанні системи.

Результатом дипломної роботи є повнофункціональна система, що дозволяє користувачам ефективно управляти процесами замовлення та надання послуг, що підтверджує можливість її використання у реальних умовах.

Ключові слова: управління замовленнями послуг, багатоплатформова система, NestJS, React, база даних, безпека інформації, автоматизація бізнес-процесів.

ABSTRACT

The qualification work is devoted to the development of a multi-platform service order management system. This system is designed to optimize the process of ordering and providing services between users and providers, including the automation of many business processes, which in turn ensures increased work efficiency and customer satisfaction.

In the first chapter, an analysis of the subject area and a study of existing analogues were carried out. Modern trends and approaches in the creation of order management systems, as well as the main requirements for such systems, are considered

The second section focuses on the formation of the technical task and the development of the system architecture. The main requirements for the system are formulated, including its functionality, security and scalability.

The third section describes the system development process, including a detailed description of the database implementation, server logic, and client interface. Issues of integration of various system modules and measures taken to ensure a high level of security are discussed.

In the fourth chapter, an analysis of economic indicators of the developed system was conducted, which confirmed its high economic benefit and efficiency.

The fifth chapter deals with issues of labor protection and ensuring safety when using the system.

The result of the thesis is a fully functional system that allows users to effectively manage the processes of ordering and providing services, which confirms the possibility of its use in real conditions.

***Keywords**: service order management, multi-platform system, NestJS, React, database, information security, business process automation.*

ЗМІСТ

стор.

ВСТУП

Сфера послуг переживає стрімкий розвиток, що веде до зростання обсягів замовлень та потреб клієнтів. В таких умовах виникає гостра потреба в більш досконалих та універсальних системах управління замовленнями.

Існуючі на ринку рішення, незважаючи на свою різноманітність, часто мають ряд суттєвих недоліків. До них відносяться:

- обмежена функціональність. Багато систем не володіють всіма необхідними функціями для комплексного управління замовленнями;
- складний інтерфейс. Заплутаний інтерфейс може значно ускладнити роботу з системою, як для менеджерів, так і для клієнтів;
- нестача гнучкості. Багато систем не адаптовані до роботи на різних платформах, що обмежує їх використання в мобільних умовах;
- низька масштабованість. Деякі системи не здатні ефективно обробляти великі обсяги даних, що може призвести до проблем в роботі при зростанні навантаження.

Враховуючи вищезазначені проблеми, стає зрозумілою гостра потреба в розробці нових, більш досконалих систем управління замовленнями. Ці системи повинні відповідати сучасним вимогам ринку та мати такі характеристики:

- широкий спектр функцій. Система повинна володіти всіма необхідними функціями для комплексного управління замовленнями, включаючи прийом замовлень, їх обробку, відстеження статусу, контроль виконання, аналітику та звітність;
- зручний інтерфейс. Інтерфейс системи має бути простим, інтуїтивно зрозумілим та зручним для користування, як для менеджерів, так і для клієнтів;
- мультиплатформність. Система повинна бути доступною на різних платформах, включаючи веб-браузери, мобільні пристрої та планшети;
- висока масштабованість. Система повинна бути здатною обробляти великі обсяги даних без втрати продуктивності.

Розробка такої системи матиме значний вплив на сферу послуг, дозволяючи:

- підвищити ефективність роботи. Автоматизація рутинних завдань та оптимізація процесів обробки замовлень;
- поліпшити якість обслуговування клієнтів. Зручний інтерфейс та доступність системи на різних платформах дозволять клієнтам легко та швидко оформляти замовлення, а також відстежувати їх статус;
- підвищити конкурентоспроможність. Використання сучасних систем управління замовленнями може стати значною конкурентною перевагою для підприємств у сфері послуг.

Об'єктом дослідження є процес управління замовленнями в сфері послуг.

Предметом дослідження є дослідження методів та алгоритмів, що ґрунтуються на сучасних інформаційних технологіях, для вдосконалення процесів управління замовленнями.

Метою дослідження є розробка багатоплатформної системи управління замовленнями, що відповідає сучасним вимогам ринку та вирішує проблеми, пов'язані з існуючими системами.

Для досягнення поставленої мети будуть вирішені наступні завдання:

- аналіз сучасних методів та алгоритмів управління замовленнями;
- виявлення основних проблем та недоліків існуючих систем;
- розробка нових методів та алгоритмів, що враховують сучасні тенденції та потреби ринку;
- проектування та розробка багатоплатформної системи управління замовленнями;
- тестування та оцінка ефективності розробленої системи.

Наукова новизна полягатиме в розробці нових методів управління замовленнями, які враховуватимуть сучасні тенденції та вимоги ринку. Практичне значення отриманих результатів полягатиме в можливості використання цих розробок для покращення ефективності бізнес-процесів, оптимізації взаємодії з клієнтами та підвищення конкурентоспроможності підприємств на ринку послуг.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Система керування замовленнями послуг – це програмне забезпечення, яке автоматизує процес замовлення та виконання послуг, що надаються компанією. Ця система спрощує взаємодію між клієнтами, виконавцями та керівництвом компанії, забезпечуючи ефективне управління замовленнями, відстеження статусів, комунікацію та аналітику.

Актуальність тематики дослідження полягає в тому, що сучасний світ нерозривно пов'язаний з інформаційними технологіями, які стають ключовим фактором у розвитку різних сфер діяльності. У контексті розробки багатоплатформової системи керування замовленнями послуг, інформаційні технології відіграють критичну роль у забезпеченні ефективного управління бізнес-процесами, забезпеченні взаємодії з клієнтами та партнерами, а також у підтримці конкурентоспроможності компанії на ринку.

Перш за все, інформаційні технології дозволяють автоматизувати та оптимізувати бізнес-процеси, що стає особливо важливим у сферах, де великий обсяг замовлень та їх швидка обробка є ключовими факторами успіху. Система керування замовленнями, побудована на сучасних інформаційних технологіях, дозволяє забезпечити швидке оброблення замовлень, ефективне планування ресурсів та моніторинг виконання завдань.

Далі, інформаційні технології розширюють можливості взаємодії з клієнтами та партнерами. Завдяки розвитку онлайн-комунікацій та електронної комерції, компанії можуть легко спілкуватися зі своїми клієнтами, приймати та обробляти замовлення в реальному часі, а також надавати індивідуалізовану підтримку.

Крім того, інформаційні технології дозволяють компаніям збирати та аналізувати великі обсяги даних про замовлення, що дозволяє виявляти тенденції

та прогнозувати попит, оптимізувати запаси та ресурси, а також розробляти персоналізовані пропозиції для клієнтів.

У цілому, інформаційні технології в сфері керування замовленнями послуг відіграють важливу роль у підвищенні ефективності бізнесу, покращенні якості обслуговування клієнтів та забезпеченні конкурентоспроможності компанії. Тому дослідження та розробка сучасних багатоплатформових систем керування замовленнями є актуальним завданням, яке відповідає вимогам сучасного ринку та сприяє подальшому розвитку бізнесу.

Предметна галузь управління замовленнями послуг стикається з рядом значних проблем, які впливають на ефективність та конкурентоспроможність компаній у цьому секторі.

Перша проблема полягає в управлінні замовленнями в умовах зростаючого обсягу та різноманітності послуг. Сучасні компанії пропонують широкий спектр послуг, що призводить до складності управління замовленнями та їх оптимізації. Це може призвести до затримок у виконанні замовлень, невідповідності очікуванням клієнтів та зниження рівня задоволеності користувачів.

Друга проблема пов'язана з нестабільністю попиту та непередбачуваністю ринкових умов. Управління замовленнями у таких умовах стає складним завданням, оскільки компаніям потрібно швидко адаптуватися до змін в попиті та реагувати на конкурентні та економічні чинники.

Третя проблема полягає в забезпеченні високої якості обслуговування клієнтів. У сучасному світі клієнти мають високі вимоги до швидкості, зручності та персоналізації обслуговування. Компанії повинні забезпечити ефективне управління замовленнями, щоб забезпечити високу задоволеність та лояльність клієнтів.

Четверта проблема стосується складнощів у координації та співпраці з партнерами та постачальниками послуг. Велика кількість сторін, що беруть участь у процесі виконання замовлень, може призвести до затримок, помилок та недоліків у роботі.

Крім того, важливо враховувати проблеми безпеки даних та конфіденційності інформації в управлінні замовленнями послуг. У зв'язку з ростом кількості та значущості цифрових даних, компанії повинні приділяти особливу увагу захисту інформації від несанкціонованого доступу та кібератак.

Узагальнюючи, проблеми в управлінні замовленнями послуг включають в себе складності управління обсягами та різноманітністю замовлень, нестабільність ринкових умов, потребу у високій якості обслуговування, проблеми координації з партнерами та безпеку даних. Розв'язання цих проблем вимагає впровадження сучасних технологій та стратегій управління, що робить цю тематику актуальною та важливою для подальшого розвитку галузі.

1.2 Аналіз публікацій останніх років

У розвитку будь-якої галузі знань важливу роль відіграють дослідження та публікації, які висвітлюють останні досягнення, нові методики та актуальні тенденції. В контексті розробки багатоплатформової системи управління замовленнями послуг, аналіз сучасних наукових та професійних публікацій набуває особливого значення. Він дозволяє не тільки збагнути сучасний стан розробок у даній сфері, але й визначити ключові напрямки подальших досліджень та розробок. Цей аналіз стає невід'ємною частиною процесу проектування, адже дозволяє базувати рішення на перевірених даних та інноваційних рішеннях, що з'явилися на ринку.

Одним із ключових підходів у дослідженні систем керування послугами є використання концепції *ITIL* (*Information Technology Infrastructure Library*). У роботах українських дослідників, таких як Лисенко (2020) та Петрова (2021), наголошується на важливості *ITIL* для покращення управління IT-послугами в організаціях різних секторів.

Лисенко у своїй роботі «Аналіз впровадження *ITIL* у вітчизняних *IT*-компаніях» розглядає практичні аспекти застосування *ITIL* в українських *IT*-компаніях. Автор підкреслює, що застосування *ITIL* дозволяє підвищити

ефективність та якість надання *IT*-послуг за рахунок стандартизації процесів та впровадження найкращих практик.

Українські дослідники також активно досліджують інші моделі та методології управління послугами, такі як *LSM* (*Lean Service Management*) та *Six Sigma*. У роботі Коваленко (2022) «Використання *Lean Service Management* в українських підприємствах» автор аналізує вплив Lean підходу на ефективність надання послуг у сфері обслуговування.

Коваленко зазначає, що *LSM* допомагає знижувати витрати та підвищувати задоволеність клієнтів за рахунок усунення втрат і оптимізації процесів. Автор наводить приклади успішного впровадження *Lean* у вітчизняних компаніях, що дозволяє значно покращити якість послуг.

Однією з важливих сучасних тенденцій у сфері систем керування послугами є цифрова трансформація. У роботі Бойко (2023) «Цифрова трансформація в системах керування послугами» досліджуються новітні технології та їхній вплив на управління послугами.

Бойко аналізує, як цифрові інструменти, такі як штучний інтелект та великі дані, впливають на ефективність процесів надання послуг. Автор зазначає, що впровадження таких технологій дозволяє автоматизувати рутинні завдання, покращувати прогнозування попиту та оптимізувати управління ресурсами.

Українські дослідники також звертають увагу на виклики, що виникають у процесі впровадження сучасних систем керування послугами. У роботі Сидоренко (2021) «Проблеми та перспективи впровадження систем керування послугами в Україні» розглядаються основні бар’єри та можливості в цій сфері.

Сидоренко підкреслює, що однією з головних проблем є нестача кваліфікованих кадрів, здатних впроваджувати та обслуговувати сучасні системи керування. Також важливою проблемою є недостатнє фінансування та підтримка з боку держави. Водночас, автор зазначає, що зростання попиту на якісні послуги та розвиток технологій створюють нові можливості для впровадження ефективних систем керування.

Системи керування послугами є важливим елементом сучасного бізнес-середовища, що дозволяє підвищити ефективність та якість надання послуг. Українські дослідники активно вивчають різні аспекти цих систем, зокрема впровадження *ITIL, LSM* та цифрових технологій. Попри існуючі виклики, такі як нестача кваліфікованих кадрів та недостатнє фінансування, перспективи розвитку систем керування послугами в Україні залишаються позитивними завдяки зростанню попиту та технологічному прогресу. Результати цього аналізу були використані для формування відповідного теоретичного підґрунтя для практичної реалізації проекту, а також для аргументації вибору конкретних технологій, методів та підходів у розробці системи. Це дозволило не тільки підвищити якість кінцевого продукту, але й забезпечити його відповідність сучасним стандартам та вимогам ринку.

1.2.1 Методологія пошуку

У контексті розробки багатоплатформової системи управління замовленнями послуг ціль полягає у знаходженні даних про останні досягнення у сфері технологій, архітектурних рішень, інноваційних підходів до взаємодії з користувачами та забезпечення безпеки систем. Важливо виявити актуальні тренди, що можуть вплинути на проектування та реалізацію проекту, а також з'ясувати потенційні прогалини у дослідженнях, які можна було б заповнити власними розробками.

Ефективний пошук вимагає використання різноманітних джерел інформації, серед яких:

1. Наукові бази даних. *IEEE Xplore, ACM Digital Library, ScienceDirect, JSTOR* та інші ресурси, які забезпечують доступ до великої кількості наукових статей, доповідей конференцій та патентів.
2. Бібліотеки та архіви. Використання університетських та національних бібліотек, що надають доступ до друкованих та електронних видань.

3. Інтернет-пошукові системи. *Google Scholar* та інші спеціалізовані пошукові системи, що дозволяють швидко знаходити актуальні дослідницькі матеріали.
4. Професійні асоціації та організації. Веб-сайти та публікації від асоціацій, таких як *ACM* або *IEEE*, які часто публікують результати досліджень та аналітичні звіти за спеціалізованими напрямками.

Для ефективного пошуку необхідно скласти список ключових слів, що пов'язані з темою проекту. Наприклад, для проекту системи управління замовленнями послуг ключові слова включають "*multi-platform service management*", "*service ordering system*", "*user interaction in service platforms*", "*data security in service applications*" тощо.

Пошук був організуваний за етапами:

1. Попередній огляд. Швидкий пошук за ключовими словами для визначення загальної кількості доступних матеріалів і виявлення найбільш цитованих джерел.
2. Глибинний аналіз. Детальне вивчення вибраних публікацій, аналіз змісту, методологій досліджень та висновків.
3. Систематизація знайденої інформації. Класифікація джерел за темами та напрямками, які вони охоплюють, для легшого доступу до інформації у подальшій роботі над проектом.

1.2.2 Аналіз знайдених публікацій

Аналіз зібраних публікацій починається з розгляду основних тем, які зустрічаються в наукових роботах, пов'язаних із багатоплатформовими системами управління замовленнями послуг. Більшість досліджень зосереджена на кількох ключових аспектах: інновації в архітектурі систем, підходи до забезпечення безпеки даних, удосконалення користувацького досвіду та оптимізація взаємодії з користувачами через різні платформи.

У публікаціях, які аналізують архітектурні рішення для подібних систем, часто акцентується на використанні мікросервісної архітектури. Цей підхід

дозволяє забезпечити високу масштабованість та гнучкість систем, що є критично важливим для обслуговування великої кількості замовлень і користувачів з різних платформ. Мікросервісна архітектура також сприяє поліпшенню процесів неперервної інтеграції та неперервного розгортання (*CI/CD*), що підвищує швидкість впровадження нових функцій та забезпечення високої доступності сервісу.

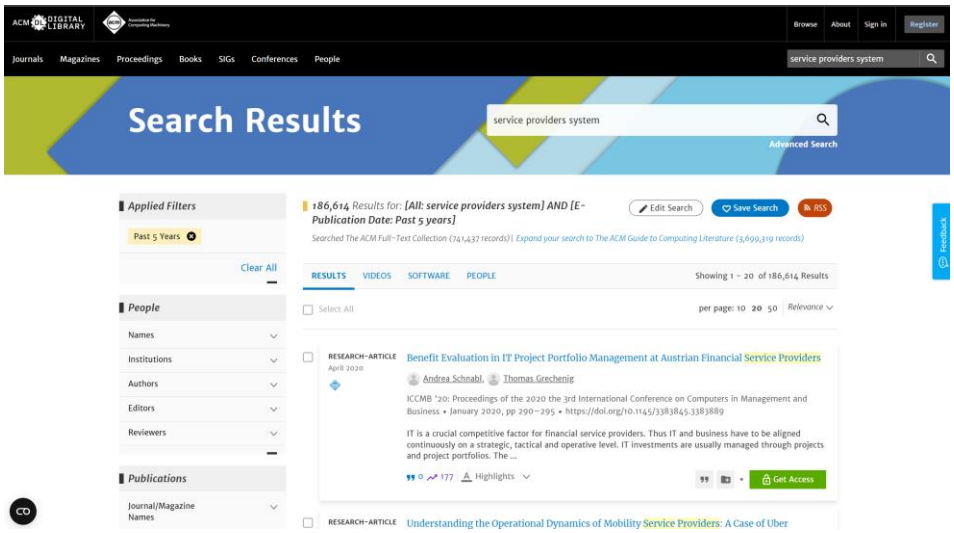


Рис. 1.1 – Результати пошуку в науковій базі даних *ACM Digital Library*

Значна кількість досліджень присвячена питанням забезпечення безпеки в багатоплатформових системах. Аналіз публікацій виявляє широке використання технологій шифрування, аутентифікації та авторизації, зокрема, впровадження двофакторної аутентифікації та використання токенів *JWT* для керування сесіями користувачів. Також обговорюються методи захисту від *SQL*-ін'єкцій та *XSS*-атак, які є важливими для забезпечення цілісності та конфіденційності користувацьких даних.

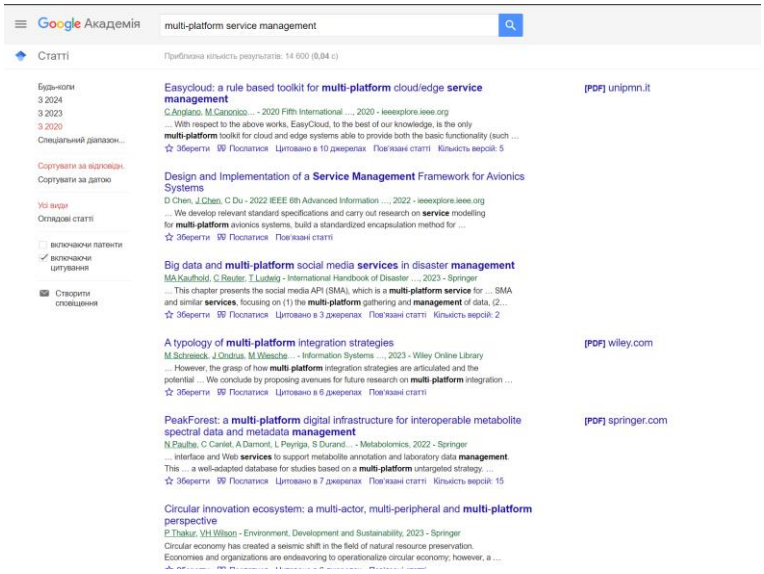


Рис. 1.2 – Результати пошуку в пошуковій системі *Google Scholar*

Статті, що зосереджені на користувацькому досвіді та інтерфейсах, часто обговорюють необхідність створення інтуїтивно зрозумілих та зручних інтерфейсів для користувачів різних платформ. Важливість адаптивного дизайну, що забезпечує ефективну взаємодію як на мобільних, так і на стаціонарних пристроях, є однією з ключових тем. Також велика увага приділяється персоналізації користувацького досвіду через використання штучного інтелекту та машинного навчання для аналізу поведінки користувачів та оптимізації інтерфейсів на основі зібраних даних.

Вивчення та аналіз знайдених публікацій виявили важливі напрямки розвитку у сфері багатоплатформових систем управління замовленнями послуг, однак також вказали на деякі проблеми та недоліки існуючих рішень. Критичний огляд цих матеріалів дозволяє не тільки зрозуміти поточний стан галузі, але й сформулювати рекомендації для подальшого розвитку власного проекту. Поглиблений аналіз наявних публікацій дозволяє виділити декілька ключових аспектів, що потребують уваги:

1. Масштабованість та гнучкість. Багато досліджень акцентують увагу на важливості мікросервісної архітектури для масштабованості системи. Проте, часто не враховуються складнощі управління такою архітектурою, включаючи збільшення витрат на технічну підтримку та складність оркестрації сервісів.

2. Безпека. Хоча багато систем інкорпорують сучасні підходи до забезпечення безпеки, як-от двофакторна автентифікація та шифрування, існує стійка потреба у вдосконаленні механізмів захисту даних, особливо у контексті зростаючих загроз кібербезпеці.

3. Користувацький досвід. Хоча публікації підкреслюють значення інтуїтивно зрозумілих інтерфейсів, недостатньо уваги приділяється індивідуалізації користувацьких взаємодій на основі аналітики поведінки користувачів.

Критичний огляд та синтез наукових публікацій надає можливість не лише збагнути існуючі тенденції та виклики в розробці багатоплатформових систем, але й визначити напрямки для інновацій та покращення власного проекту. Це сприяє формуванню обґрунтованої та актуальної розробки, що відповідає сучасним вимогам і очікуванням користувачів.

1.3 Дослідження існуючих аналогів

Було проведено дослідження існуючих систем керування замовленнями послуг. Мета цього дослідження – вивчити функціональні можливості, особливості та переваги різних систем керування замовленнями послуг, щоб визначити, які з них можуть бути найкращим вибором для розробки власної системи.

Для дослідження існуючих аналогів буде використано наступні методи:

1. Пошук інформації. Було проведено пошук інформації про системи керування замовленнями послуг в Інтернеті, в наукових журналах, на веб-сайтах вендорів та в інших джерелах.

2. Аналіз функціональних можливостей. Було проведено аналіз функціональних можливостей різних систем керування замовленнями послуг, щоб визначити, які з них пропонують необхідні функції для розробки власної системи.

3. Аналіз функціональних можливостей. Було проведено аналіз функціональних можливостей різних систем керування замовленнями послуг,

щоб визначити, які з них пропонують необхідні функції для розробки власної системи.

4. Порівняльний аналіз. Було проведено порівняльний аналіз різних систем керування замовленнями послуг за такими критеріями, як:

- функціональні можливості;
- особливості;
- переваги;
- вартість;
- відгуки.

На основі результатів дослідження було обрано найкращий аналог для розробки власної системи.

1.3.1 *Salesforce Service Cloud*

Функціональні можливості. *Salesforce Service Cloud* – це комплексна платформа керування обслуговуванням клієнтів (*CRM*), яка включає в себе систему керування замовленнями послуг.

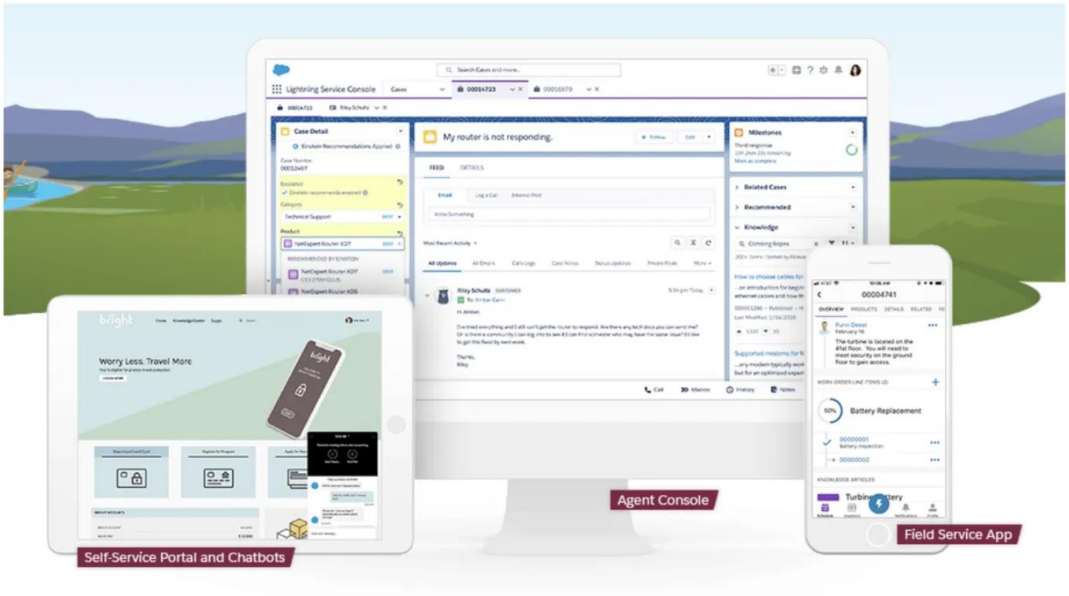


Рис. 1.3 – *Salesforce Service Cloud*

Система пропонує широкий спектр функцій, таких як:

- створення та відстеження замовлення;
- управління ресурсами;

- спілкування з клієнтами;
- аналітика та звітність.

Особливості. *Salesforce Service Cloud* має ряд унікальних особливостей, таких як:

- штучний інтелект та машинне навчання (*ML*) для автоматизації завдань;
- мобільний додаток для зручного доступу до системи з будь-якого місця;
- інтеграція з іншими системами *Salesforce*, такими як *Sales Cloud* та *Marketing Cloud*.

Переваги. *Salesforce Service Cloud* має ряд переваг, таких як:

1. Масштабованість. Система може бути масштабована для потреб будь-якого розміру бізнесу.
2. Надійність. Система має високу надійність та доступність.
3. Безпека. Система забезпечує високий рівень безпеки даних.
4. Вартість. *Salesforce Service Cloud* – це платна система, ціна якої залежить від кількості користувачів та функцій.
5. Відгуки. *Salesforce Service Cloud* має переважно позитивні відгуки від користувачів.

1.3.2 *Zendesk Sunshine*

Функціональні можливості. *Zendesk Sunshine* – це платформа керування обслуговуванням клієнтів (*CRM*), яка включає в себе систему керування замовленнями послуг.

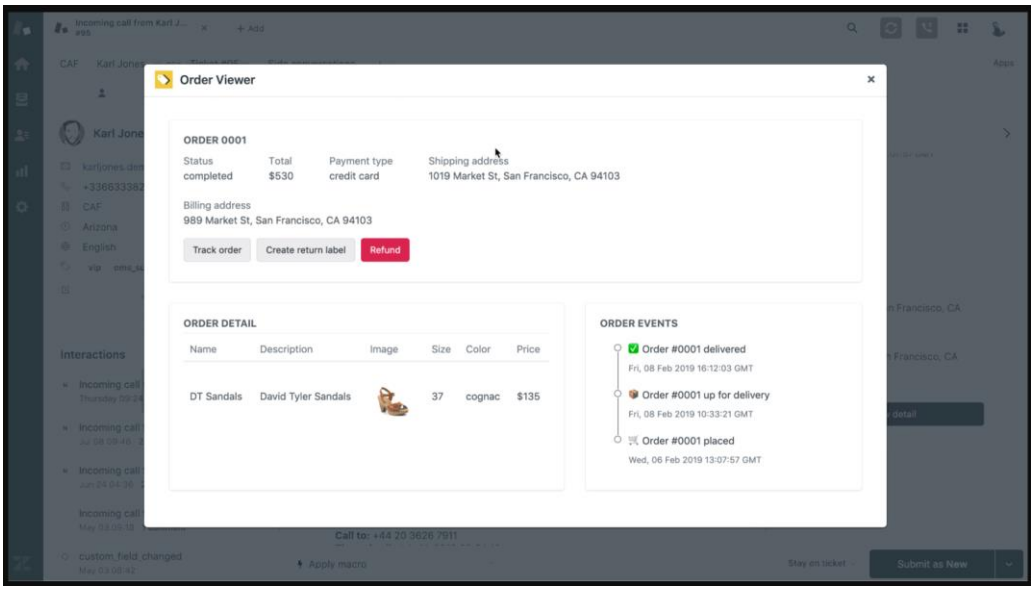


Рис. 1.4 – *Zendesk Sunshine*

Система пропонує широкий спектр функцій, таких як:

- створення та відстеження замовлення;
- управління ресурсами;
- спілкування з клієнтами;
- аналітика та звітність.

Особливості. *Zendesk Sunshine* має ряд унікальних особливостей, таких як:

- відкрита платформа, яка може бути інтегрована з іншими системами;
- підтримка кількох каналів зв'язку з клієнтами;
- штучний інтелект та машинне навчання (*ML*) для автоматизації завдань.

Переваги. *Salesforce Service Cloud* має ряд переваг, таких як:

1. Масштабованість. Система може бути масштабована для потреб будь-якого розміру бізнесу.
2. Гнучкість. Система може бути налаштована відповідно до потреб вашого бізнесу.
3. Доступність. Система має доступні ціни.

1.3.3 *Zoho Desk*

Функціональні можливості. *Zoho Desk* – це комплексна платформа керування обслуговуванням клієнтів (*CRM*), яка включає в себе систему керування замовленнями послуг.

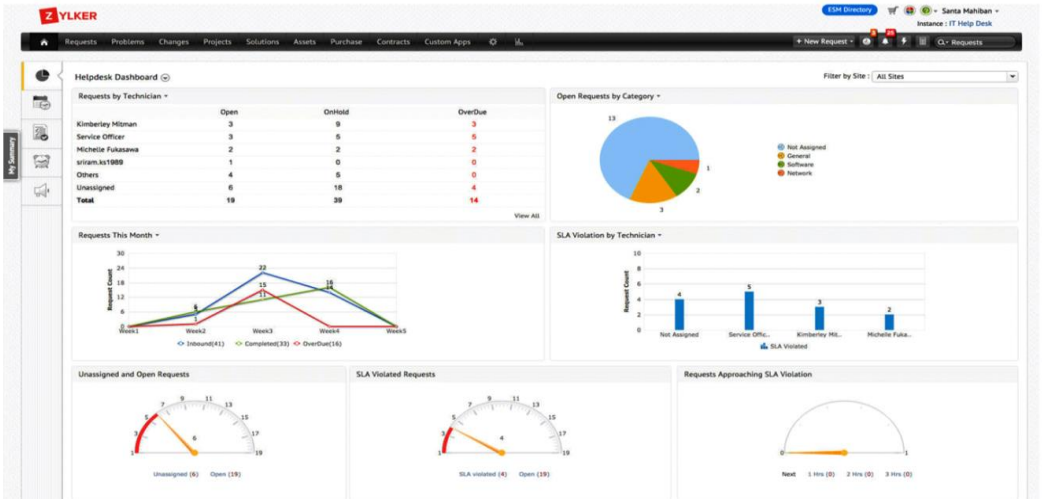


Рис. 1.5 – *Zoho Desk*

Система пропонує широкий спектр функцій, таких як:

- створення та відстеження замовлення;
- управління ресурсами;
- спілкування з клієнтами;
- аналітика та звітність.

Особливості. *Zoho Desk* має ряд унікальних особливостей, таких як:

- вбудована система *CRM*;
- підтримка кількох каналів зв'язку з клієнтами;
- штучний інтелект (ШІ) та машинне навчання (*ML*) для автоматизації завдань.

Переваги. *Zoho Desk* має ряд переваг, таких як:

1. Доступність. Система має доступні ціни.
2. Простота використання. Система проста у використанні та налаштуванні.
3. Інтеграція з іншими системами *Zoho*.
4. Вартість. *Zoho Desk* пропонує кілька тарифних планів, ціна яких залежить від кількості користувачів та функцій.
5. Відгуки. *Zoho Desk* має переважно позитивні відгуки від користувачів.

1.3.4 *Freshservice*

Функціональні можливості. *Freshservice* – це комплексна платформа керування обслуговуванням клієнтів (*CRM*), яка включає в себе систему керування замовленнями послуг.

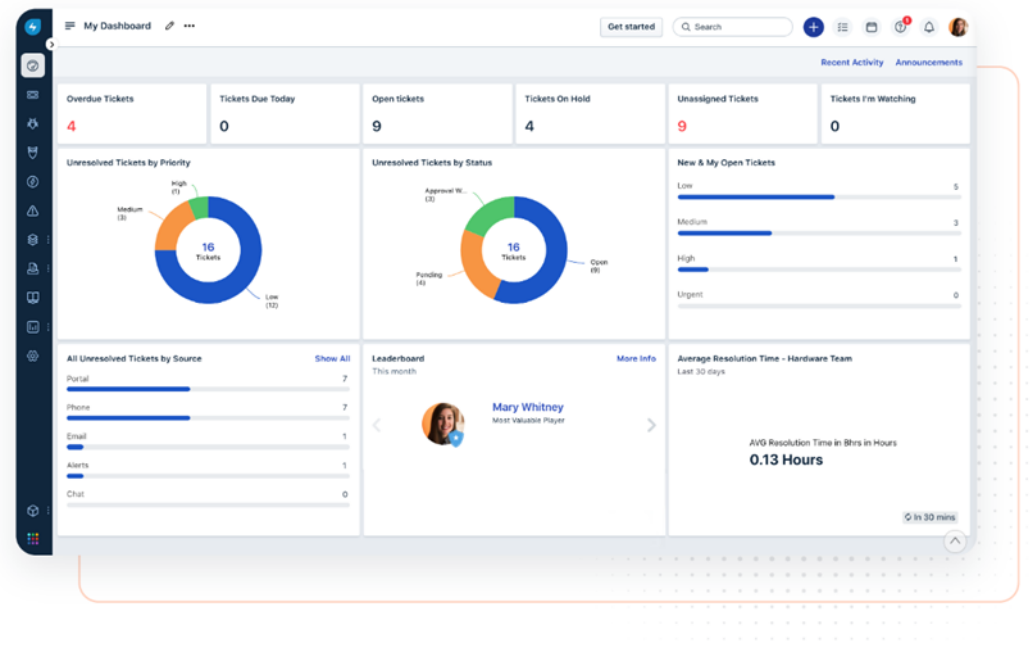


Рис. 1.6 – *Freshservice*

Система пропонує широкий спектр функцій, таких як:

- створення та відстеження замовлення;
- управління ресурсами;
- спілкування з клієнтами;
- рівняння по лівому боку:
- аналітика та звітність.

Особливості. *Freshservice* має ряд унікальних особливостей, таких як:

- вбудована система *CRM*;
- підтримка кількох каналів зв'язку з клієнтами;
- штучний інтелект (ШІ) та машинне навчання (*ML*) для автоматизації завдань.

Переваги. *Freshservice* має ряд переваг, таких як:

1. Доступність. Система має доступні ціни.

2. Простота використання. Система проста у використанні та налаштуванні.
3. Інтеграція з іншими системами
4. Вартість. *Freshservice* пропонує кілька тарифних планів, ціна яких залежить від кількості користувачів та функцій.
5. Відгуки. *Freshservice* має переважно позитивні відгуки від користувачів.

1.3.5 *Monday.com*

Функціональні можливості. *Monday.com* – це платформа для управління проектами та робочими процесами, яка може бути використана для створення системи керування замовленнями послуг.

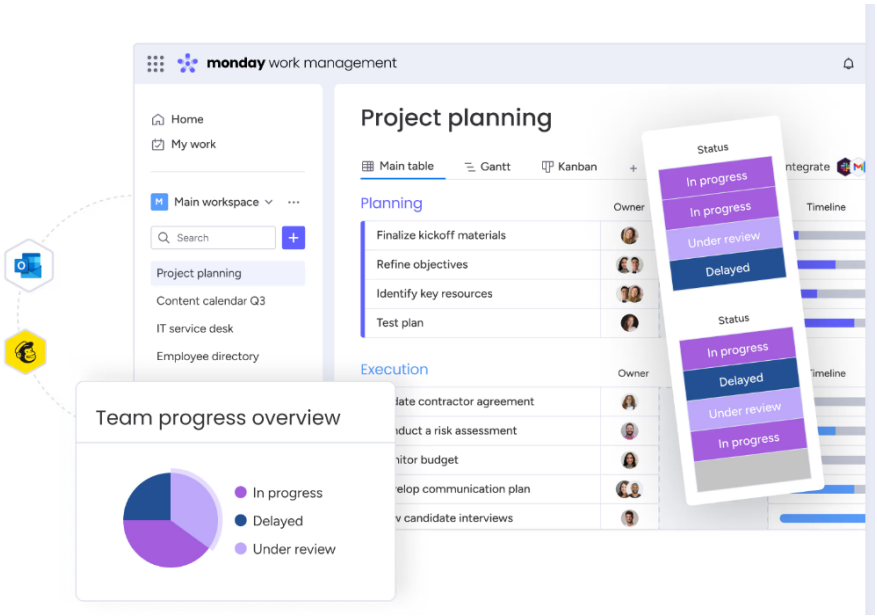


Рис. 1.7 – *Monday.com*

Система пропонує широкий спектр функцій, таких як:

- створення та відстеження замовлення;
- управління ресурсами;
- спілкування з клієнтами;
- автоматизація завдань.

Особливості. *Salesforce Service Cloud* має ряд унікальних особливостей, таких як:

- гнучкий інтерфейс, який можна налаштувати відповідно до ваших потреб;
- інтеграція з іншими системами;
- доступність мобільного додатку.

Переваги. *Monday.com* має ряд переваг, таких як:

1. Простота використання. Система проста у використанні та налаштуванні.
2. Доступність: Система має доступні ціни.
3. Масштабованість: Система може бути масштабована для потреб будь-якого розміру бізнесу.
4. Вартість. *Monday.com* пропонує кілька тарифних планів, ціна яких залежить від кількості користувачів та функцій.
5. Відгуки. M*onday.com* має переважно позитивні відгуки від користувачів.

На основі результатів дослідження найкращим аналогом для розробки власної системи керування замовленнями послуг є *Salesforce Service Cloud*. Це пов'язано з наступними причинами:

Широкий спектр функцій. *Salesforce Service Cloud* пропонує широкий спектр функцій, необхідних для розробки власної системи керування замовленнями послуг, таких як створення та відстеження замовлень, управління ресурсами, спілкування з клієнтами та аналітика та звітність.

Унікальні особливості: *Salesforce Service Cloud* має ряд унікальних особливостей, таких як штучний інтелект (ШІ) та машинне навчання (*ML*) для автоматизації завдань, мобільний додаток для зручного доступу до системи з будь-якого місця та інтеграція з іншими системами *Salesforce*.

Переваги. *Salesforce Service Cloud* має ряд переваг, таких як масштабованість, надійність, безпека та доступність позитивних відгуків від користувачів.

Однак, важливо зазначити, що інші аналоги, такі *як Zendesk Sunshine, Zoho Desk, Freshservice* та *monday.com*, також мають свої переваги та недоліки.

1.4 Розробка технічного завдання (ТЗ)

Розробка технічного завдання (ТЗ) для проекту є одним із ключових етапів планування та проектування, який забезпечує чітке розуміння цілей, вимог, функціоналу та обмежень системи. Технічне завдання формулює всі основні аспекти проекту, визначає основні завдання, які має вирішувати система, а також встановлює критерії для оцінки її успішності.

Технічне завдання виступає як основний документ, який використовують всі учасники проекту для забезпечення його правильного виконання. Воно служить дорожньою картою для розробників, дизайнерів, тестувальників та керівників проектів, дозволяючи всім зацікавленим сторонам мати однакове розуміння завдань та очікуваних результатів. Чітке та детальне ТЗ допомагає уникнути непорозумінь і помилок у процесі розробки, а також забезпечує високу якість кінцевого продукту.

Процес розробки ТЗ починається з аналізу потреб користувачів та бізнес-вимог. Він включає збір вхідних даних через інтерв'ю з зацікавленими сторонами, аналіз ринку та існуючих рішень, формування вимог до продукту, і, нарешті, документування всіх зібраних відомостей у вигляді структурованого документу.

1.4.1 Формулювання цілей проекту

Цілі проекту включають розробку багатоплатформової системи, яка дозволить користувачам ефективно управляти замовленнями послуг через веб-інтерфейс та мобільні додатки. Система має включати можливості для зручного введення, обробки, моніторингу та аналізу замовлень, що надходять від різних користувачів, з особливим акцентом на гнучкість, безпеку даних та високу доступність сервісу.

Конкретні цілі проекту:

1. Розробка єдиної системи управління. Інтеграція всіх процесів управління замовленнями в одну платформу, що дозволить уникнути фрагментації даних і поліпшити загальну продуктивність системи.

2. Мультіплатформеність. Забезпечення функціональності системи на різних типах пристроїв і платформах, зокрема настільних комп'ютерах і мобільних телефонах, для зручності всіх категорій користувачів.

3. Гнучка адаптація до потреб користувачів. Система має бути здатна адаптуватися до змінюваних вимог користувачів та ринкових умов, швидко впроваджуючи необхідні зміни у функціонал.

4. Забезпечення високої безпеки. Реалізація передових технологій захисту даних для запобігання несанкціонованому доступу та забезпечення конфіденційності інформації користувачів.

5. Інтеграція з іншими системами та сервісами. Налагодження взаємодії з іншими бізнес-системами, такими як *CRM*-системи, фінансові платформи, системи електронної комерції, що дозволить розширити можливості системи та підвищити її ефективність.

6. Розробка інтуїтивно зрозумілого користувацького інтерфейсу. Створення зручного і доступного інтерфейсу, який відповідає сучасним стандартам дизайну та забезпечує високий рівень користувацького досвіду.

Для забезпечення виконання встановлених цілей, проект передбачає регулярний моніторинг і аналіз ефективності різних компонентів системи. Це включає тестування на різних етапах розробки, збір зворотного зв'язку від користувачів та адаптацію системи відповідно до отриманих даних. Вимірювання ефективності системи буде проводитися на основі заздалегідь визначених критеріїв успішності, таких як швидкість обробки замовлень, кількість помилок у роботі системи, задоволеність користувачів, та інші індикатори продуктивності.

1.4.2 Визначення вимог до системи

Визначення вимог до системи є фундаментальним етапом у процесі розробки технічного завдання, який забезпечує детальне розуміння функціональних та нефункціональних потреб проекту. Цей процес включає аналіз та формулювання вимог, які повинні виконуватися системою для

досягнення визначених бізнес-цілей і забезпечення задоволеності кінцевих користувачів.

Функціональні вимоги описують конкретні задачі, які має виконувати система. Вони формулюються на основі потреб бізнесу та входять до складу основних операцій, які користувачі зможуть виконувати за допомогою системи:

1. Обробка замовлень. Система має автоматизувати прийом, обробку та відстеження замовлень в реальному часі.
2. Управління клієнтськими даними. Інтеграція з *CRM* для збору та аналізу інформації про клієнтів.
3. Розрахунок вартості послуг. Автоматизація процесів калькуляції вартості послуг з урахуванням різних факторів та знижок.
4. Звітність та аналітика. Генерація звітів про продажі, клієнтські запити та інші ключові показники ефективності.
5. Інтеграція з іншими системами. Сумісність з зовнішніми системами, такими як платіжні шлюзи та логістичні сервіси.

Нефункціональні вимоги стосуються якості та стандартів, які повинна відповідати система, і включають наступні аспекти:

1. Продуктивність. Система повинна обробляти замовлення в режимі реального часу.
2. Надійність. забезпечення стабільної роботи системи з мінімальним часом простою.
3. Масштабованість. Можливість розширення системи для обслуговування збільшеного обсягу користувачів та замовлень.
4. Безпека. Захист даних від несанкціонованого доступу, забезпечення конфіденційності та цілісності інформації.
5. Універсальність. Спроможність системи адаптуватися до різних правових та культурних умов використання.

1.4.3 Планування реалізації

Проект буде реалізований у чотири основні етапи:

1. Розробка архітектури та дизайну:
   - вивчення існуючих аналогів, збір вимог від зацікавлених сторін, аналіз ринкових потреб та визначення можливостей для інновацій;
   - визначення структурної організації системи, вибір між монолітною та мікросервісною архітектурою в залежності від вимог до масштабованості та легкості обслуговування;
   - розробка прототипів інтерфейсів для різних платформ, *UI/UX* дизайну для забезпечення зручності та інтуїтивності використання.
2. Реалізація функціоналу.
   - програмування основних модулів системи, таких як управління користувацькими профілями, обробка замовлень, адміністрування та звітність;
   - підключення платіжних систем, *CRM*, *ERP* та інших інструментів для забезпечення широкого спектру можливостей;
   - створення надійного та безпечного *API* для забезпечення взаємодії з мобільними застосунками та зовнішніми інтеграціями.
3. Тестування та виявлення помилок.
   - перевірка кожного компоненту системи на коректність роботи в ізоляції;
   - забезпечення того, що всі модулі системи працюють разом без помилок;
   - симуляція реального використання системи для перевірки її продуктивності та масштабованості;
   - виконання тестів за сценаріями, які імітують реальні умови використання системи кінцевими користувачами.
4. Запуск у продакшн та підтримка.
   - розгортання системи на серверах, налаштування робочого середовища;
   - встановлення інструментів для моніторингу системи, аналіз її роботи та оптимізація для підвищення ефективності;

– забезпечення технічної підтримки користувачів, внесення необхідних змін та додавання нового функціоналу.

1.4.4 Критерії приймання

Приймання системи до запуску визначається через комплексний підхід, що забезпечує відповідність всім зазначеним у технічному завданні вимогам. Система вважатиметься готовою до впровадження, коли будуть досягнуті всі основні етапи верифікації та валідації задуму проекту, що забезпечить бездоганну функціональність та надійність в роботі.

Першочерговим критерієм приймання системи є повне виконання функціональних вимог, викладених у технічному завданні. Це означає, що кожен описаний процес та функціонал має бути реалізованим та працювати згідно з задумом без помилок і збоїв.

Система повинна пройти ретельне комплексне тестування, яке охоплює не тільки функціональні, але й нефункціональні аспекти, такі як продуктивність, безпека, надійність та зручність користування. Ці тести мають виявити і усунути будь-які потенційні помилки та переконатися, що система може ефективно функціонувати під навантаженням та в різних умовах експлуатації.

Фінальним етапом критеріїв приймання є демонстрація системи стейкхолдерам. Цей захід має на меті показати, що система відповідає всім заявленим вимогам та готова до впровадження.

Завершення всіх цих етапів забезпечить, що система не тільки відповідає технічним вимогам, але й задовольняє бізнес-потреби замовників та користувачів, гарантуючи її успішне впровадження та експлуатацію в реальних умовах.

1.4.5 Ризики та стратегії їх мінімізації

У процесі розробки багатоплатформової системи управління замовленнями послуг важливо ідентифікувати потенційні ризики, які можуть негативно вплинути на успішність проекту. Це включає технічні, організаційні,

фінансові та ринкові ризики. Кожен з цих ризиків вимагає чіткої стратегії мінімізації для забезпечення стабільності проекту та досягнення запланованих результатів.

Технічні ризики пов'язані з можливими проблемами у технологічній реалізації проекту, включаючи:

1. Застаріле програмне забезпечення. Використання непідтримуваних бібліотек або платформ може призвести до втрати функціональності або безпеки.
2. Інтеграція систем. Труднощі з інтеграцією нової системи з існуючими бізнес-процесами або програмним забезпеченням.
3. Продуктивність та масштабованість. Потенційна нездатність системи ефективно обробляти велику кількість замовлень або користувачів.

Стратегії мінімізації: Проведення регулярних технічних аудитів, вибір надійних і підтримуваних технологій, розробка модульної та легко масштабуємої архітектури, раннє виявлення проблем інтеграції через прототипування та тестування.

Організаційні ризики стосуються управління проектом і командою:

1. Недостатнє управління ресурсами. Неадекватне планування ресурсів може призвести до затримок у виконанні проекту.
2. Комунікаційні бар'єри. Погана комунікація між членами команди та зацікавленими сторонами може ускладнити виконання проекту.

Стратегії мінімізації: Застосування передових практик проектного менеджменту, використання професійних інструментів для управління проектами, регулярні зустрічі та звітність, залучення досвідчених менеджерів.

Ринкові ризики стосуються змін у потребах користувачів та конкурентному середовищі:

1. Зміна трендів на ринку. Нові технологічні рішення або зміни у попиті, що можуть зробити продукт застарілим до моменту його запуску.
2. Конкуренція. Випуск подібних продуктів конкурентами, що може вплинути на успіх проекту.

Стратегії мінімізації: Постійне відстеження ринкових тенденцій, гнучкість у плануванні та розробці, прогнозування потенційних змін у попиті та технологіях.

Ретельне врахування цих ризиків та застосування стратегій їх мінімізації є критично важливим для забезпечення успішного запуску та експлуатації системи, що в кінцевому підсумку веде до забезпечення стабільності та досягнення запланованих цілей проекту.

Висновок до першого розділу

Було проведено детальний аналіз предметної області багатоплатформових систем управління замовленнями послуг, що виявив ключові аспекти для покращення оперативності та ефективності обслуговування клієнтів. Вивчено наукові публікації, що акцентують на автоматизації процесів, використанні штучного інтелекту, хмарних рішень та мобільних платформ. Проведено дослідження існуючих систем для виявлення їх обмежень і визначення можливостей для інновацій у власному проекті. На основі цього сформульовано технічне завдання, яке чітко окреслює цілі проекту, вимоги до системи, етапи реалізації та методи оцінки ефективності, а також стратегії реагування на потенційні ризики.

РОЗДІЛ 2
ПРОЕКТУВАННЯ

2.1 Функції системи

У процесі проектування комплексної багатоплатформової системи управління замовленнями послуг ключове значення має чітке визначення функцій системи. Це визначення формує основу для всіх наступних етапів розробки, від архітектури до імплементації та тестування. Функції системи описують основні можливості, які система надає своїм користувачам, визначають способи взаємодії користувачів з системою та інтеграцію з іншими системами та сервісами.

Визначення функцій системи є основоположним у процесі проектування, адже саме на їх базі розробляється архітектура системи та формується технічне завдання для подальших етапів розробки. Це також визначає напрямки для інтеграції з іншими системами, визначення необхідних зовнішніх і внутрішніх інтерфейсів, а також для забезпечення належної безпеки і надійності системи.

2.1.1 Визначення функцій системи

Розробка багатоплатформової системи управління замовленнями послуг має на меті створити універсальне рішення, яке оптимізує процеси замовлення, виконання та моніторингу послуг через різні платформи. Система повинна бути здатною автоматизувати та спрощувати численні задачі, забезпечуючи високий рівень обслуговування клієнтів та оперативне управління ресурсами.

Для багатоплатформової системи управління замовленнями послуг, основні функції можуть включати:

5. Автоматизація процесів замовлення:
   - автоматизований прийом та реєстрація замовлень з різних джерел (веб-сайт, мобільний додаток);

- управління статусами замовлень (нове, в обробці, виконане, скасоване);
- відстеження історії замовлень для аналізу та звітності.

6. Управління клієнтами:
   - реєстрація та авторизація користувачів;
   - управління профілями користувачів, включаючи персональні дані, історію замовлень, налаштування доступу.
7. Фінансове управління:
   - інтеграція з платіжними системами для обробки транзакцій;
   - управління ціноутворенням та акціями.
8. Інтеграція та *API*:
   - розробка API для інтеграції з зовнішніми системами (наприклад, *CRM*, *ERP*, логістичні сервіси).;
   - обмін даними у реальному часі для синхронізації інформації між різними платформами.
9. Аналітика та звітність:
   - автоматизація збору даних для формування звітів про ефективність бізнесу;
   - використання інструментів аналітики для оптимізації процесів і покращення якості сервісу.

Забезпечення взаємодії системи з зовнішніми сервісами, такими як платіжні шлюзи, логістичні компанії та інші сервісні платформи, є необхідним для розширення функціоналу і забезпечення безперебійної роботи. Інтеграція має включати:

1. Автоматизацію процесів оплати та обліку.
2. Взаємодію з системами доставки для оптимізації логістики замовлень.
3. Обмін даними з аналітичними та маркетинговими платформами для покращення бізнес-стратегій.

2.1.2 Представлення функціональної схеми

Для розробки детальної функціональної схеми системи керування замовленнями послуг слід використовувати *UML* діаграми, що допоможуть візуалізувати взаємодію між компонентами системи та основні потоки даних. Основні типи діаграм, які будуть використані, включають діаграму класів та діаграму послідовностей.

Діаграма послідовностей відображає процеси створення замовлення, обробки платежів і надання послуг. Вона показує взаємодію між об'єктами системи в хронологічному порядку.

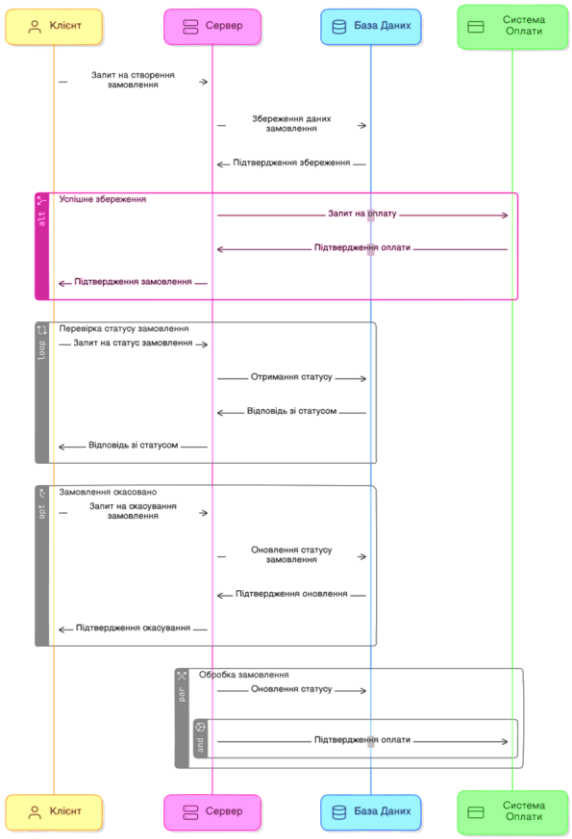


Рис. 2.1 – Функціональна схема

Розробка функціональної схеми системи за допомогою *UML* діаграм, таких як діаграма класів та діаграма послідовностей, допоможе чітко відобразити структуру бази даних, взаємозв'язки між різними елементами системи та основні процеси. Це сприятиме кращому розумінню системи розробниками та забезпечить основу для подальшої реалізації проекту.

2.2 Структура системи

У процесі розробки багатоплатформової системи управління замовленнями послуг критично важливим є визначення структури системи. Це дозволяє не тільки розуміти, як окремі компоненти системи будуть взаємодіяти між собою, але й яким чином ці компоненти будуть інтегровані з існуючою інфраструктурою та зовнішніми системами. Правильно спроектована структура є ключем до стабільності, ефективності, та легкості масштабування системи в майбутньому.

Визначення структури системи впливає на багато аспектів проекту, включаючи розробку, впровадження, тестування, та подальшу підтримку. Чітка архітектурна структура допомагає зменшити кількість помилок у процесі розробки, спрощує інтеграцію нових модулів та функцій, та покращує загальну прозорість проекту для всієї команди розробників.

2.2.1 Визначення структури системи

Структура системи визначає архітектурний склад системи, включаючи основні компоненти та їх взаємозв'язки, що необхідні для реалізації функціональних вимог системи. Основна мета структурної організації полягає у забезпеченні ефективної інтеграції різних модулів системи, оптимального розподілу відповідальностей між ними, та забезпечення масштабованості та легкості обслуговування.

Основні компоненти системи можуть включати:

1. Фронтенд модулі. Веб-інтерфейс та мобільний додаток, які забезпечують інтерактивний доступ до системи для кінцевих користувачів.
2. Бекенд модулі. Серверна логіка, яка обробляє запити від фронтенду, виконує бізнес-логіку та здійснює взаємодію з базою даних та зовнішніми інтеграціями.
3. База даних. Забезпечує зберігання даних, необхідних для функціонування системи.

4. Системи інтеграції. З'єднання з зовнішніми *API*, системами платежів, *CRM* і *ERP* системами.
5. Сервіси аналітики та звітності. Модулі для збору, обробки та представлення даних про активність у системі.

2.2.2 Представлення структурної схеми

Структурна схема системи ілюструє як основні компоненти системи взаємопов'язані та взаємодіють один з одним.

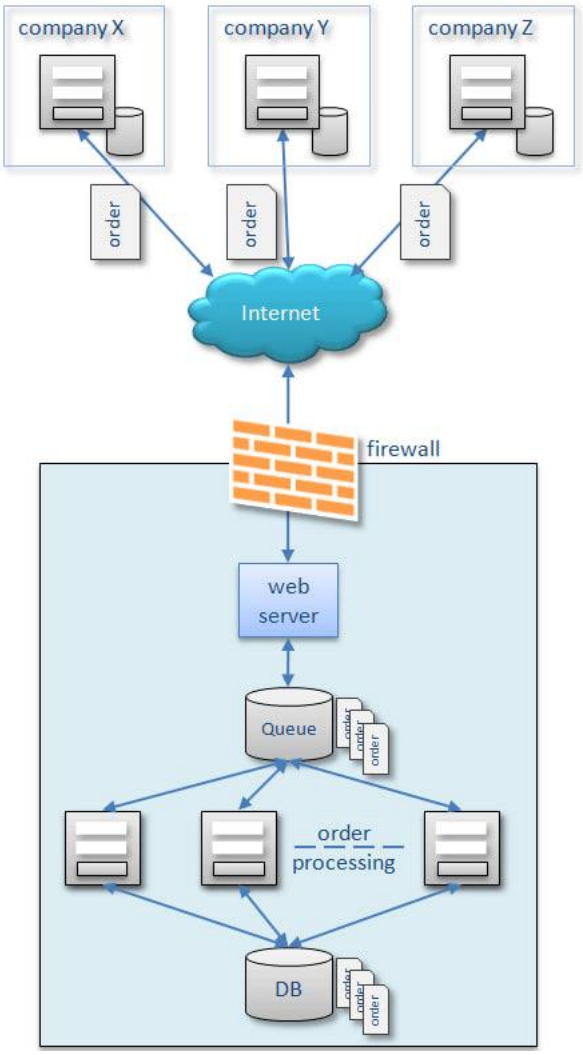


Рис. 2.2 – Структурна схема

Представлена діаграма демонструє, що система має чітко структуровану архітектуру з добре визначеними взаємозв'язками між компонентами. Це дозволяє ефективно управляти даними користувачів, замовленнями, послугами, платежами та інформацією про постачальників послуг.

Добре структурована схема системи дозволяє легко масштабувати додаток у разі збільшення кількості користувачів чи обсягу оброблюваних даних. Це забезпечує готовність системи до розширення та інтеграції нових функцій в майбутньому.

Таким чином, детальна функціональна схема системи, створена за допомогою UML діаграм, є важливим кроком у процесі розробки системи керування замовленнями послуг, що сприяє забезпеченню її надійності, ефективності та масштабованості.

2.3 Архітектура системи

Архітектура системи визначає загальну структуру програмного забезпечення, включаючи спосіб організації її компонентів, взаємодію між ними, а також з зовнішнім середовищем. Архітектура визначає структурну організацію системи, її компоненти та способи взаємодії між ними. Це каркас, на якому будується вся система, і від його міцності залежать стабільність, продуктивність та масштабованість кінцевого продукту.

Грамотно спроектована архітектура сприяє легкій модифікації та додаванню нових функцій без порушення існуючої експлуатації системи. Вона дозволяє системним архітекторам передбачити потенційні проблеми на ранніх етапах, забезпечуючи ефективні рішення для їх вирішення. Важливість архітектури полягає у її впливі на подальшу здатність системи адаптуватися до змінюваних умов використання, збільшення навантаження та інтеграції з іншими системами.

Основною ціллю архітектурного проектування є створення структури, яка не тільки відповідає всім вимогам зацікавлених сторін, але й забезпечує системі можливість ефективно розвиватися та модернізуватися протягом тривалого часу. Це включає забезпечення компонентів, які можна легко тестувати, налаштовувати та масштабувати в міру розширення або зміни функціоналу системи.

2.3.1 Визначення архітектури системи

Архітектура багатоплатформової системи управління замовленнями послуг спроектована так, щоб оптимально задовольняти потреби користувачів і бізнесу, забезпечуючи гнучкість, масштабованість і безпеку. Вибір архітектури базується на принципах модульності і здатності до інтеграції з іншими системами і технологіями.

Ключові аспекти архітектури:

1. Модульність. Система розробляється з використанням модульного підходу, де кожен компонент відповідає за певну функцію. Це дозволяє незалежно оновлювати та масштабувати кожен модуль, а також спрощує процес впровадження нових технологій і сервісів.

2. Мікросервісна архітектура. Для забезпечення високої доступності та легкості масштабування системи використовується мікросервісна архітектура. Кожен мікросервіс відповідає за виконання конкретної бізнес-функції і комунікує з іншими сервісами через визначені API.

3. Централізоване управління даними. Незважаючи на розподілену природу мікросервісів, важливо забезпечити централізоване управління даними та їхню консистентність. Використовуються технології, які дозволяють ефективно управляти транзакціями і забезпечувати інтегрітет даних.

4. Безпека. В архітектурі передбачені комплексні механізми захисту даних, включаючи шифрування, аутентифікацію і авторизацію користувачів, а також засоби захисту від зовнішніх атак.

5. Інтеграція з зовнішніми системами. Система спроектована з можливістю легкої інтеграції з різними зовнішніми сервісами та платформами, включаючи фінансові системи, сервіси доставки та інші бізнес-системи, що дозволяє збільшити її функціональність і відкриває нові можливості для розвитку.

При виборі технологічного стеку для реалізації архітектури враховувалися такі критерії, як зрілість технологій, підтримка спільноти, можливості інтеграції, а також вартість впровадження та обслуговування. Особливу увагу приділялося

вибору таких технологій, які забезпечують високу продуктивність, надійність та безпеку системи.

2.3.2 Представлення архітектурних рівнів

Архітектура багатоплатформової системи керування замовленнями послуг базується на багаторівневій архітектурі, яка розділяє функціональні компоненти системи на окремі рівні. Це дозволяє підвищити масштабованість, гнучкість і підтримуваність системи. Основні архітектурні рівні включають:

1. Презентаційний шар (*Presentation Layer*). Інтерфейси користувача, включаючи веб-сайти та мобільні додатки, які забезпечують взаємодію користувачів з системою. До цього шару також входять засоби візуалізації даних та управління контентом.
2. Шар бізнес-логіки (*Business Logic Layer*). Містить всю логіку обробки даних, алгоритми, які використовуються для вирішення бізнес-завдань, такі як розрахунок цін, управління транзакціями, логіка звітності тощо. Цей шар часто реалізується у вигляді мікросервісів або модулів, які можуть бути розподілені на різних серверах чи платформах. Основні компоненти:
   - контролери – відповідають за обробку запитів від презентаційного рівня, викликають відповідні сервіси та повертають результати назад;
   - сервіси – інкапсулюють бізнес-логіку, забезпечують виконання бізнес-процесів (наприклад, створення замовлення, розрахунок вартості).
   - валідація даних – перевіряє коректність введених користувачами даних перед їх обробкою і збереженням.
3. Шар доступу до даних. Включає бази даних та механізми їх доступу, *ORM* системи, які використовуються для абстрагування і оптимізації запитів до даних. Цей шар також відповідає за забезпечення інтеграції з зовнішніми джерелами даних. Включає в себе:

- репозиторії – забезпечують абстракцію над базою даних, надаючи методи для виконання *CRUD*-операцій;
- *ORM* (*Object-Relational Mapping*) – інструменти, такі як *Entity Framework* або *Hibernate*, які забезпечують зручний доступ до реляційних баз даних через об'єктно-орієнтовані моделі;
- запити та трансакції – *SQL*-запити для взаємодії з базою даних, управління трансакціями для забезпечення цілісності даних.

4. Шар інтеграції (*Integration Layer*). Забезпечує зв'язок між різними системами та сервісами через *API*, веб-сервіси, а також управління обміном даними між різними частинами системи і зовнішніми компонентами. Основними компонентами є:
   - модулі інтеграції – компоненти, які реалізують логіку інтеграції з конкретними зовнішніми сервісами (наприклад, платіжними системами);

2.4 Визначення даних системи

У процесі розробки багатоплатформової системи управління замовленнями послуг одним із ключових аспектів є визначення даних, які буде обробляти та зберігати система. Цей етап є фундаментальним для забезпечення функціональності системи, ефективного управління замовленнями, а також для виконання аналітичних та звітних завдань.

Правильно структуровані та організовані дані є життєво необхідними для будь-якої інформаційної системи. Вони не тільки впливають на швидкість і якість обробки запитів, але й визначають можливості системи щодо масштабування, безпеки даних та інтеграції з іншими системами і платформами. Також, якісне управління даними забезпечує високу точність ведення замовлень та їх виконання, що безпосередньо впливає на задоволеність клієнтів.

Основні аспекти визначення даних:

1. Структурування даних. Важливим є розробка чіткої схеми бази даних, яка визначає таблиці, їхні поля, типи даних, ключі та зв'язки між таблицями.

Структурування даних повинно враховувати всі бізнес-процеси системи та вимоги до звітності та аналітики.

2. Забезпечення цілісності даних: Необхідно впровадити механізми, які забезпечують точність, повноту та надійність даних у системі. Це включає використання обмежень на рівні бази даних, тригерів та інших методів контролю цілісності.

3. Безпека даних. Забезпечення безпеки даних за допомогою шифрування, регулярних бекапів та реалізації політик доступу до даних є критично важливим для захисту інформації від несанкціонованого доступу або втрати.

4. Інтеграція даних. Особлива увага приділяється підтримці інтеграції з зовнішніми системами, що можуть вимагати обміну даними в реальному часі або періодичному синхронізації. Важливо передбачити можливості для легкої інтеграції та обміну даними.

Основною метою визначення даних є створення надійної, безпечної та ефективної системи, яка здатна адекватно відображати і обробляти всі процеси управління замовленнями послуг. Чітке розуміння структури даних дозволить не тільки ефективно розробити систему, але й забезпечити її гнучкість для подальших розширень та оптимізацій.

2.4.1 Визначення даних

Визначення даних у системі управління замовленнями послуг включає ідентифікацію всіх типів даних, які будуть оброблятися системою, а також встановлення відносин між ними. Цей процес є критично важливим для забезпечення адекватного збору, зберігання, обробки та використання даних у системі. Основні категорії даних включають:

1. Дані користувачів. Інформація про клієнтів та адміністраторів системи, включаючи ім'я, контактні дані, історію замовлень, налаштування доступу тощо.
2. Дані замовлень. Деталі замовлень, включаючи дату створення, статус, список послуг, загальну вартість, інформацію про платіжні транзакції.

3. Дані послуг. Інформація про послуги, які надає компанія, включаючи назву, опис, ціну, категорії та будь-які додаткові параметри, пов'язані з послугами.
4. Фінансові дані. Дані, пов'язані з фінансовими транзакціями, бухгалтерським обліком і звітністю.
5. Інтеграційні дані. Інформація, отримана від зовнішніх систем і сервісів, наприклад, статуси доставки від логістичних партнерів, дані з *CRM*-систем.

Забезпечення цілісності, доступності та конфіденційності даних є важливою частиною системи. Реалізація цих вимог передбачає:

1. Нормалізація даних. Застосування нормалізації для зменшення дублювання даних та покращення їх організації.
2. Шифрування даних. Використання сучасних методів шифрування для захисту персональних і фінансових даних.
3. Резервне копіювання та відновлення. Реалізація надійних механізмів бекапу та відновлення для захисту даних від втрати або пошкодження.

Для підтримки взаємодії з іншими системами та оптимізації внутрішніх процесів, система включає компоненти для інтеграції даних:

1. API для обміну даними. Розробка API для безпечного та ефективного обміну даними з зовнішніми системами, такими як фінансові установи, логістичні сервіси та маркетингові платформи.
2. Синхронізація даних: Механізми синхронізації для забезпечення актуальності даних у всіх компонентах системи.

2.4.2 Представлення схеми бази даних

База даних складається з різних об'єктів, таких як таблиці, збережені процедури, тригери. Об'єкти бази даних містять всю інформацію про її структуру й дані. Реляційні бази даних зберігають всі дані в таблицях. Таблиця це структура, що складається з безлічі неупорядкованих горизонтальних рядків (*rows*), кожна з яких містить однакову кількість вертикальних стовпців (*colums*). Перетинання окремого рядка й стовпця називається полем (*field*), що містить

специфічну інформацію. Багато принципів роботи реляційної бази даних узяті з визначень відносин (*relations*) між таблицями.

Схема бази даних для багатоплатформової системи управління замовленнями послуг розробляється з метою оптимізації зберігання та обробки даних. Ця схема є критично важливою для забезпечення ефективності, швидкості обробки запитів та інтеграції з іншими системами.

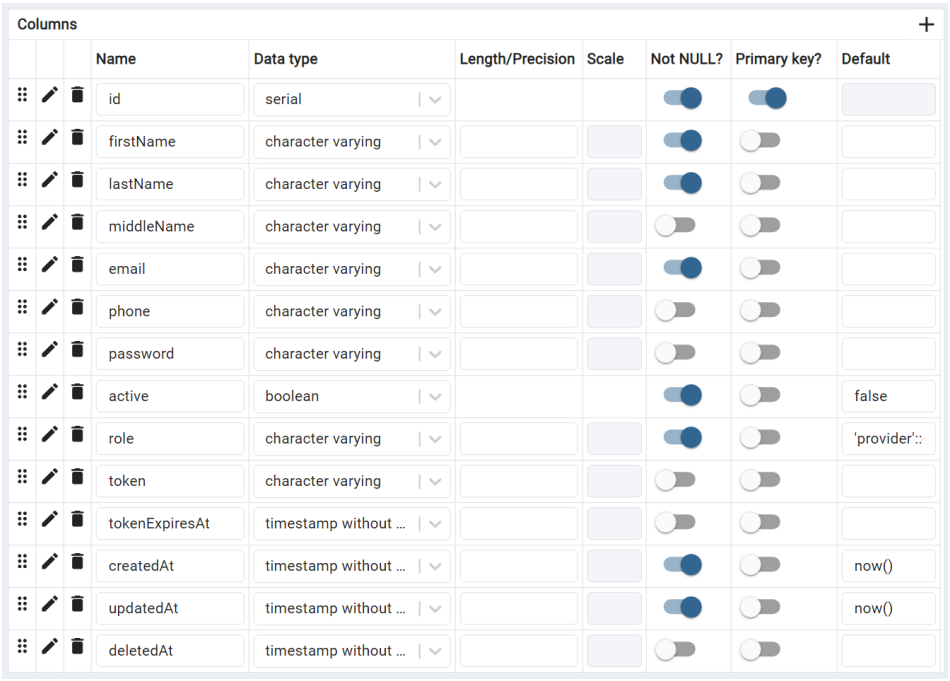
Основою схеми бази даних є сутності, що відображають ключові аспекти системи:

1. Користувачі (*Users*). Зберігають дані про користувачів системи, включаючи авторизаційні дані, контактну інформацію та роль в системі.
2. Замовлення (*Orders*). Включають інформацію про замовлення, таку як дата створення, статус, деталі послуги та пов’язані фінансові транзакції.
3. Послуги (*Services*). Описують доступні послуги, їхні характеристики та ціноутворення.
4. Транзакції (*Payments*). Відслідковують всі фінансові операції, пов’язані з замовленнями.
5. Постачальники (*Providers*). Зберігають дані про постачальника послуг включаючи контактну інформацію та керівника.
6. Категорії послуг (*Categories*). Описують доступні категорії, а також загальну комісію для унаслідуваної послуги.

Рисунок 2.3 представляє схему таблиці бази даних, використовується для зберігання інформації про користувачів у системі управління замовленнями послуг. Деталі структури таблиці:

- *id* (*serial*) – це первинний ключ таблиці, який автоматично інкрементується для кожного нового користувача;
- *firstName* (*character varying*) – зберігає ім'я користувача;
- *lastName* (*character varying*) – зберігає прізвище користувача;
- *middleName* (*character varying*) – зберігає по батькові користувача;
- *email* (*character varying*) – зберігає електронну пошту користувача;
- *phone* (*character varying*) – зберігає телефонний номер користувача;

– *password* (*character varying*) – зберігає хеш пароля користувача;
– *active* (*boolean*) – визначає, чи є обліковий запис користувача активним;
– *role* (*character varying*) – зберігає роль користувача в системі;
– *token* (*character varying*) – зберігає токен для сесії користувача;
– *tokenExpiresAt* (*timestamp without time zone*) – зберігає час, коли токен втрачає свою дію;
– *createdAt* (*timestamp without time zone*) – автоматично зберігає дату та час створення запису;
– *updatedAt* (*timestamp without time zone*) – автоматично зберігає дату та час оновлення запису;
– *deletedAt* (*timestamp without time zone*) – зберігає дату та час, коли запис був помічений як видалений, не видаляючи його фізично з бази.


Columns

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
id	serial			✓	✓	
firstName	character varying			✓		
lastName	character varying			✓		
middleName	character varying					
email	character varying			✓		
phone	character varying					
password	character varying					
active	boolean			✓		false
role	character varying			✓		'provider'::...
token	character varying					
tokenExpiresAt	timestamp without ...					
createdAt	timestamp without ...			✓		now()
updatedAt	timestamp without ...			✓		now()
deletedAt	timestamp without ...					

Рис. 2.3 – Сутність користувача

Таблиця розроблена таким чином, щоб оптимізувати зберігання інформації про користувачів та забезпечити легкий доступ до їх даних з можливістю швидкого оновлення та відстеження змін.

Рисунок 2.4 показує структуру таблиці, використовується для зберігання даних про сесії користувачів у системі управління замовленнями послуг. Деталі структури таблиці:

- *id* (*uuid*) – це первинний ключ таблиці, який використовує універсальний унікальний ідентифікатор (*UUID*) для кожної сесії;
- *token* (*character varying*) – токен сесії, який використовується для аутентифікації та авторизації запитів від користувача. Токен є ключем, що дозволяє системі впізнати користувача і надати доступ до ресурсів системи;
- *expiresIn* (*date*) – поле зберігає дату закінчення дії токена. Після цієї дати токен стає недійсним, і користувачу потрібно буде повторно аутентифікуватися для отримання нового токена;
- *middleName* (*character varying*) – зберігає ідентифікатор користувача, асоційованого з сесією.

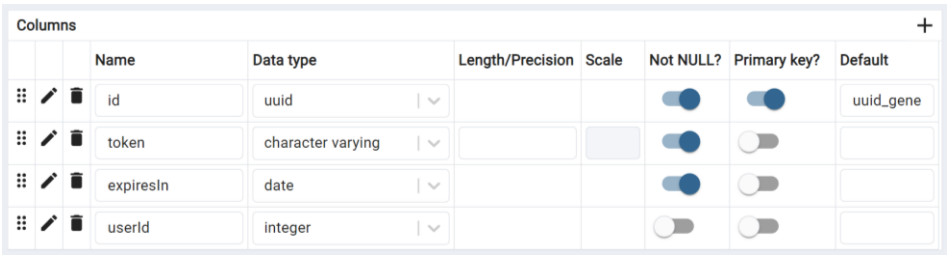


Рис. 2.4 – Сутність сесії користувача

Структура таблиці спроектована так, щоб оптимізувати процес управління сесіями користувачів. Використання *UUID* для ідентифікації сесій забезпечує глобальну унікальність ідентифікаторів, що важливо для систем з великою кількістю користувачів і паралельних сесій. Токен сесії служить для безпечного обміну даними між клієнтом та сервером, а поле *expiresIn* допомагає в обмеженні часу дії сесії для підвищення безпеки. Зв'язування сесії з конкретним користувачем через *userId* важливе для особистісної аутентифікації та авторизації.

Рисунок 2.5 показує структуру таблиці бази даних, яка призначена для зберігання інформації про категорії послуг у системі управління замовленнями послуг. Деталі структури таблиці:

- *id* (*serial*) – первинний ключ таблиці, що використовується для унікальної ідентифікації кожної категорії;
- *name* (*character varying*) – зберігає назву категорії послуг;

– *active* (*date*) – визначає, чи є категорія активною у системі;
– *fee* (*integer*) – комісія, асоційована з послугами в даній категорії;
– *description* (*text*) – поле для зберігання детального опису категорії.

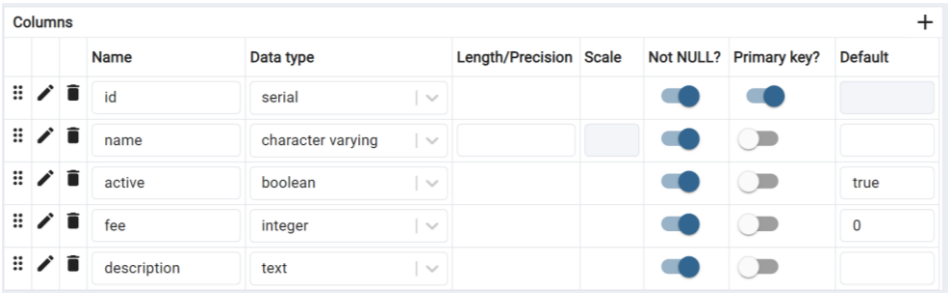


Рис. 2.5 – Сутність категорії послуг

Ця таблиця ефективно організує інформацію про категорії послуг, що полегшує управління, відстеження та відображення різних видів послуг, які можуть бути пропоновані користувачам системи. Наявність поля «active» дозволяє адміністраторам легко активувати або деактивувати категорії, залежно від потреб бізнесу, не вдаючись до фізичного видалення даних. Також, структура забезпечує чітке розуміння вартості послуг, асоційованих з кожною категорією, завдяки полю «fee».

Рисунок 2.6 показує структуру таблиці бази даних, яка, ймовірно, використовується для зберігання інформації про провайдерів послуг у системі управління замовленнями послуг. Ось детальний опис структури таблиці:

– *id* (*serial*) – первинний ключ таблиці, який автоматично інкрементується для кожного нового провайдера;
– *name* (*character varying*) – зберігає назву провайдера;
– *address* (*character varying*) – зберігає адресу провайдера;
– *phone* (*character varying*) – зберігає контактний телефонний номер провайдера;
– *active* (*integer*) – вказує, чи є провайдер активним у системі;
– *ownerId* (*integer*) – зовнішній ключ, який асоціює провайдера з користувачем, який є власником цього профілю провайдера.

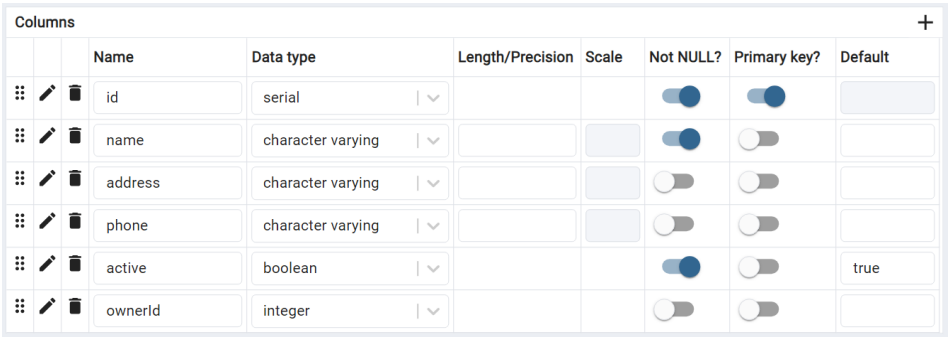


Рис. 2.6 – Сутність провайдера послуг

Ця таблиця важлива для управління інформацією про тих, хто надає послуги у системі. Інформація, збережена в цій таблиці, дозволяє системі не тільки відстежувати різних провайдерів і їхні дані, але й забезпечувати їхню активацію або деактивацію в залежності від потреб бізнесу. Використання зовнішнього ключа для зв'язку провайдера з конкретним користувачем покращує управління доступом та контроль за обліковими записами, що забезпечує додатковий рівень організації та безпеки.

Рисунок 2.7 показує структуру таблиці бази даних, використовується для зберігання інформації про послуги. Ось детальний опис структури таблиці:

- *id* (*serial*) – первинний ключ таблиці, який автоматично інкрементується для кожної нової послуги;
- *name* (*character varying*) – зберігає назву послуги;
- *code* (*character varying*) – унікальний код послуги, який може використовуватися для ідентифікації та пошуку послуги в системі;
- *type* (*character varying*) – категоризує тип послуги, наприклад, консультативна, технічна підтримка тощо;
- *active* (*boolean*) – вказує, чи є послуга активною для використання в системі;
- *autocomplete* (*boolean*) – вказує, чи потрібна додаткова валідація замовлення зі сторони співробітників;
- *price* (*integer*) – вартість послуги;
- *description* (*character varying*) – опис послуги, що може містити детальну інформацію про те, що включено в послугу;

– *images* (*text[]*) – масив текстових значень, які містять шляхи або посилання на зображення, асоційовані з послугою;
– *tags* (*text[]*) – масив текстових значень, які містять теги для категоризації та пошуку послуги;
– *providerId* (*integer*) – зовнішній ключ, що вказує на ідентифікатор провайдера, який надає цю послугу;
– *categoryId* (*integer*) – Зовнішній ключ, що вказує на ідентифікатор категорії, до якої належить послуга;
– *createdAt* (*timestamp without time zone*) – дата та час створення запису послуги;
– *updatedAt* (*timestamp without time zone*) – дата та час останнього оновлення запису послуги.

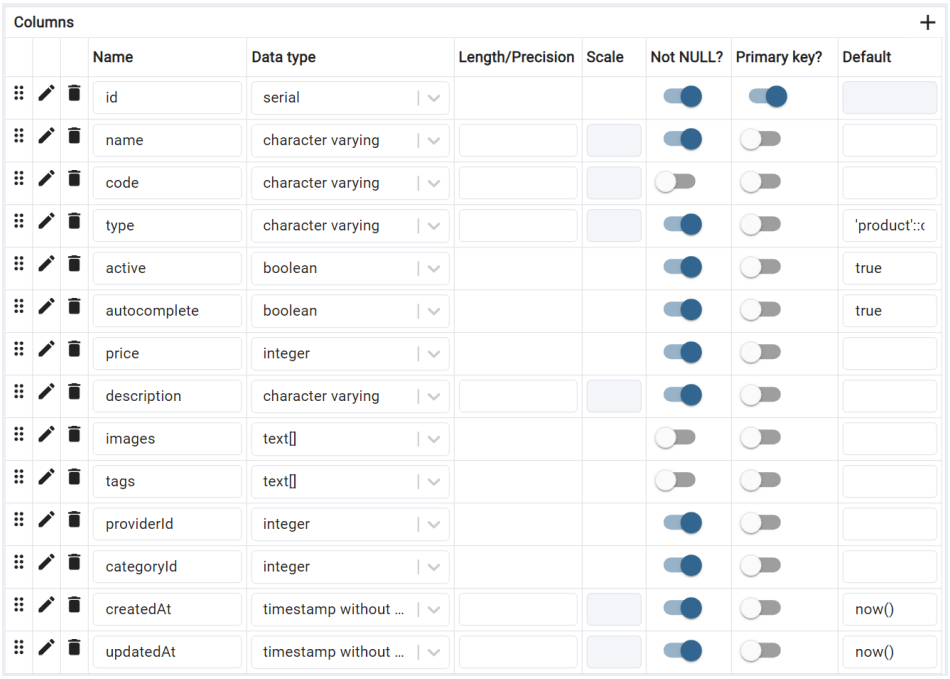


Рис. 2.7 – Сутність сервісу

Ця таблиця важлива для каталогізації та управління різними послугами, які надаються через систему. Вона дозволяє зберігати важливі деталі про послуги, що сприяє їх легкому пошуку, ідентифікації та категоризації за допомогою тегів і категорій.

Рисунок 2.8 показує структуру асоціативної таблиці бази даних, призначеної для встановлення взаємозв'язку між замовленнями та послугами в

системі управління замовленнями послуг. Таблиця дозволяє асоціювати одне замовлення з однією або декількома послугами, що забезпечує гнучкість у керуванні складними замовленнями. Ось деталі структури таблиці:

- *orderId* (*integer*) – зовнішній ключ, що вказує на ідентифікатор замовлення, це поле забезпечує зв'язок з таблицею замовлень, де зберігається основна інформація про кожне замовлення;
- *serviceId* (*integer*) – зовнішній ключ, що вказує на ідентифікатор послуги, це поле забезпечує зв'язок з таблицею послуг, де зберігається детальна інформація про послуги, які можуть бути частиною замовлення.

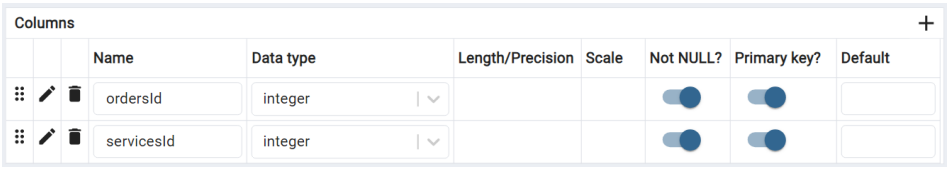


Рис. 2.8 – Асоціативна сутність між замовленням та сервісами

Таблиця використовується для створення звітів, оптимізації бізнес-процесів, нарахування коштів за послуги, і як основа для аналітики взаємодій між замовленнями та послугами в системі.

Рисунок 2.9 показує структуру таблиці бази даних, яка, ймовірно, використовується для зберігання інформації про платежі у системі управління замовленнями послуг. Ось детальний опис структури таблиці:

- *id* (*uuid*) – первинний ключ таблиці, використовує універсальний унікальний ідентифікатор (UUID) для кожної транзакції;
- *finSystemId* (*integer*) – ідентифікатор фінансової системи, через яку обробляється платіж;
- *price* (*integer*) – сума платежу;
- *fee* (*integer*) – вартість додаткових зборів, що можуть застосовуватися до транзакції;
- status (*payments_status_enum*) – перелічення, яке визначає статус платежу;

– *createdAt* (*timestamp without time zone*) – час і дата створення запису транзакції;
– *updatedAt* (*timestamp without time zone*) – час і дата останнього оновлення запису;
– *orderId* (*integer*) – категоризує тип послуги, наприклад, консультативна, технічна підтримка тощо.

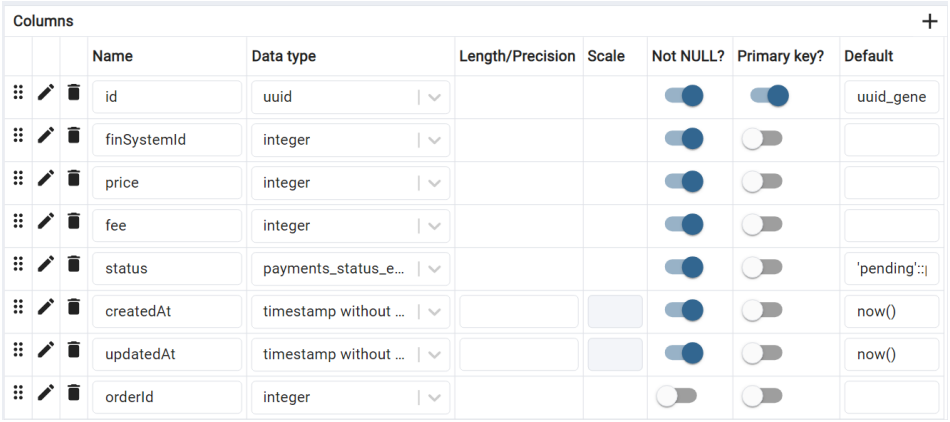


Рис. 2.9 – Сутність платежу

Ця таблиця критично важлива для обробки фінансових транзакцій у системі. Вона дозволяє відстежувати всі платежі, асоційовані з послугами або продуктами, що надаються через систему. Забезпечення точного обліку платежів є життєво важливим для фінансової стійкості та звітності в будь-якій комерційній системі. Використання *UUID* для ідентифікації транзакцій допомагає уникнути конфліктів ідентифікаторів при одночасній роботі з різними джерелами даних та при масштабуванні системи.

Рисунок 2.10 показує структуру таблиці бази даних, призначеної для зберігання інформації про замовлення в системі управління замовленнями послуг. Ось детальний опис структури таблиці:

– *id* (*serial*) – первинний ключ таблиці, який автоматично інкрементується для кожного нового замовлення;
– *externalId* (*uuid*) – унікальний ідентифікатор замовлення, створений як *UUID*, що забезпечує глобальну унікальність у розподілених системах;
– *address* (*character varying*) – зберігає адресу доставки або місцезнаходження, де має бути надана послуга або доставлений товар;

– *status* (*orders_status_enum*) – перелічення, яке допомагає відслідковувати поточний стан замовлення;
– completionAt (*date*) – вказує дату, коли замовлення має бути завершено;
– *userId* (*integer*) – зовнішній ключ, який пов'язує замовлення з користувачем, який його здійснив.

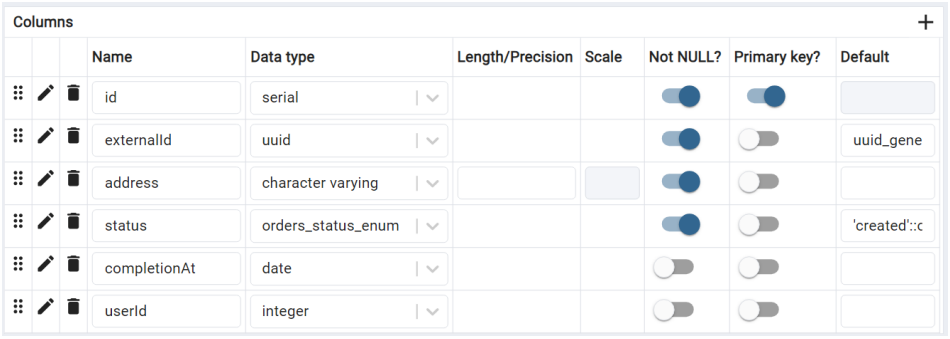


Рис. 2.10 – Сутність замовлення

Таблиця замовлень є ключовою компонентою системи, оскільки вона дозволяє централізовано зберігати всі дані, пов'язані з замовленнями. Наявність поля *externalId* дозволяє легко інтегруватися з іншими системами та сервісами, поле *status* забезпечує можливість відслідковувати прогрес виконання замовлень, а *completionAt* допомагає в плануванні та організації роботи. Використання поля *userId* підвищує персоналізацію обслуговування та сприяє покращенню взаємодії з клієнтом. Ця структура є важливою для ефективного управління процесами в системі і надає потужні інструменти для аналітики та звітності.

На представленій діаграмі (рис. 2.11) відображена схема бази даних для системи управління замовленнями послуг. Ця схема включає різноманітні таблиці, кожна з яких відіграє важливу роль у збереженні і обробці даних, що відносяться до різних аспектів операцій системи.

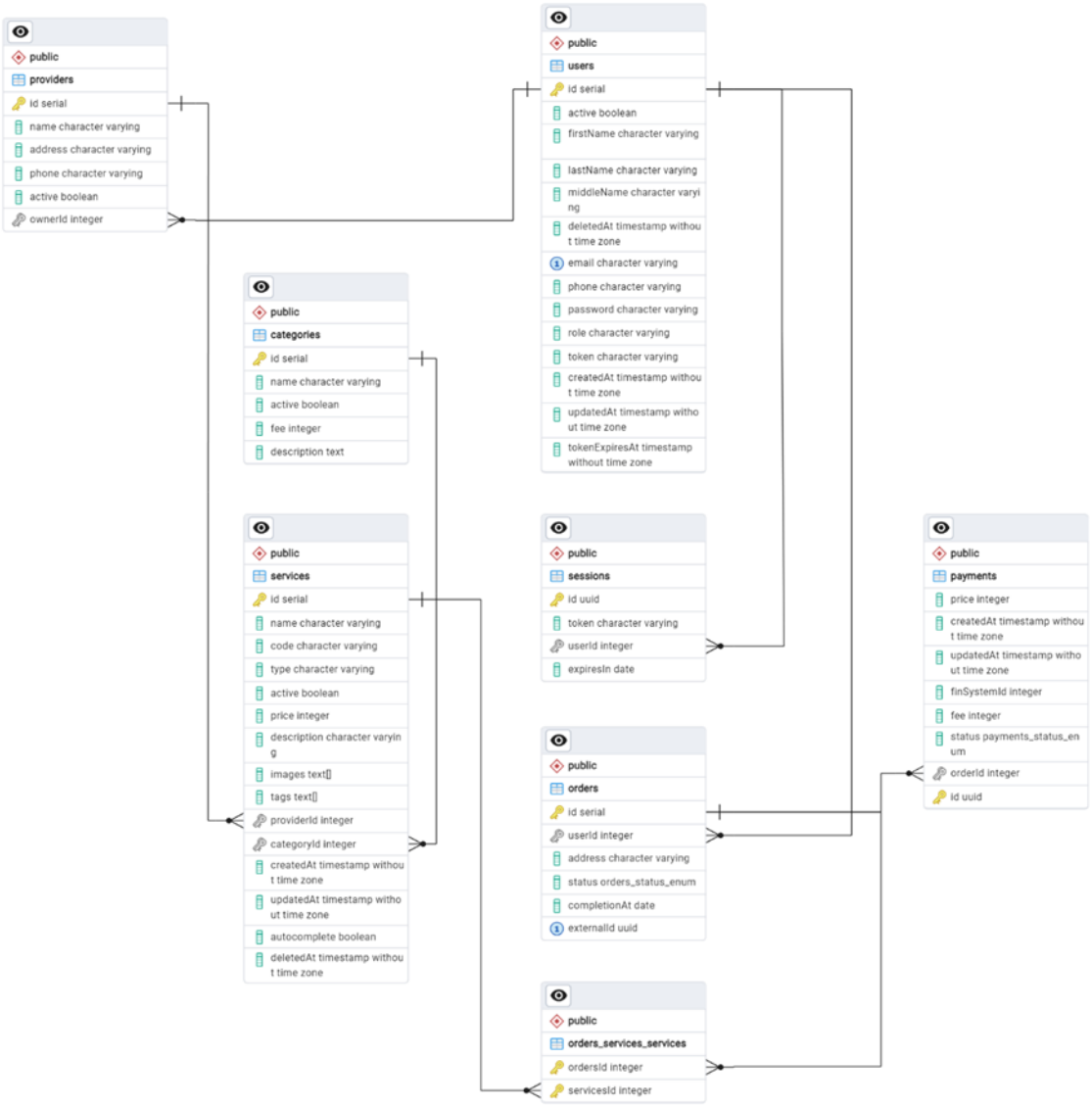


Рис. 2.11 – *ERD* за нотацією “гусяча лапка”

Система організована таким чином, що забезпечує ефективне збереження даних і їх легкий доступ для операційних і аналітичних потреб системи управління замовленнями послуг. Вона використовує сучасні практики баз даних, забезпечуючи гнучкість, масштабованість та високу продуктивність обробки запитів.

Висновок до другого розділу

Було здійснено комплексний аналіз та розроблено ключові аспекти багатоплатформової системи управління замовленнями послуг. Визначено функції системи, що дозволило сформувати вимоги та можливості, які вона повинна надавати. Архітектурне проектування, засноване на принципах

мікросервісної архітектури та шарової моделі, забезпечило розмежування відповідальностей, що важливо для підтримки гнучкості, масштабованості та надійності. Визначення та організація даних, включаючи створення схеми бази даних, забезпечило фундамент для ефективного зберігання та обробки інформації.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

3.1 Вибір і обґрунтування програмних засобів реалізації проекту

Під час реалізації будь-якого проекту, зокрема в галузі *IT*, критично важливим є адекватний вибір програмних засобів, які використовуватимуться в процесі розробки і подальшої експлуатації системи. Цей вибір має стратегічне значення, адже від технологічного стеку залежать ключові характеристики продукту, такі як його функціональність, надійність, масштабованість та легкість підтримки.

Правильно підібрані технологічні рішення сприяють не тільки ефективному вирішенню поставлених задач, але й забезпечують високу швидкість розробки, легкість інтеграції з іншими системами, зниження вартості власності, а також допомагають у досягненні високого рівня користувацького досвіду.

3.1.1 Фронтенд технології

Вибір фронтенд технологій для проекту багатоплатформової системи управління замовленнями послуг є критично важливим, оскільки це безпосередньо впливає на взаємодію з користувачем, швидкість розробки та майбутнє масштабування продукту.

Мови програмування для фронтенду:

4. *JavaScript*. Найпопулярніша мова програмування для розробки веб-додатків, що підтримується всіма сучасними браузерами.
5. *TypeScript*: Суперсет JavaScript, який додає строгу типізацію та інші можливості для покращення організації коду та підвищення його надійності.

Таблиця 3.1

Порівняння мов програмування

Мова програмування	Переваги	Недоліки
JavaScript	Широко використовується	Відсутність строгої типізації може призводити до помилок
	Велика спільнота та багато ресурсів	Може бути складно утримувати великі проекти
TypeScript	Строга типізація допомагає уникнути помилок	Вимагає компіляції
	Підтримка передових функцій програмування	Може додати складність у маленьких проектах

Фреймворки для фронтенду:

1. *React*. Бібліотека від *Facebook* для побудови інтерфейсів користувача. Використовується для створення масштабованих і швидких веб-додатків.
2. *Angular*. Потужний фреймворк від *Google*, який пропонує широкий набір інструментів для комплексної розробки веб-додатків.
3. *Vue.js*. Легкий і гнучкий фреймворк, який набуває популярності завдяки своїй простоті та інтуїтивно зрозумілій архітектурі.

Таблиця 3.2

Порівняння фронтенд фреймворків

Фреймворк	Переваги	Недоліки
React	Швидкість і гнучкість	Висока крива навчання
	Велика спільнота та екосистема	Не повний фреймворк, вимагає використання додаткових бібліотек
Angular	Повнофункціональний фреймворк	Складність і важкість фреймворку

Продовження таблиці 3.2

Vue.js	Простота і легкість використання	Менша спільнота порівняно з *React* та *Angular*

Враховуючи потреби проекту – швидкість розробки, масштабованість та надійність, вибір припав на *TypeScript* як мову програмування та *React* як фреймворк. Комбінація *TypeScript* та *React* дозволяє створювати міцну, ефективну та легко підтримувану кодову базу. *TypeScript* забезпечує додаткову безпеку за рахунок строгої типізації, покращуючи якість коду і знижуючи кількість помилок на ранніх етапах розробки. *React*, у свою чергу, є одним із найпопулярніших інструментів для побудови динамічних інтерфейсів, що ідеально підходить для створення реактивних веб-додатків з високим рівнем користувацького досвіду.

3.1.2 Розробка бекенду

При виборі технологій для розробки серверної частини проекту необхідно враховувати такі фактори, як швидкість розробки, підтримка, стабільність та можливості масштабування. Для даного проекту було розглянуто декілька мов програмування та фреймворків.

Мови програмування для бекенду:

1. *TypeScript* – це надбудова *JavaScript*, що додає строгу статичну типізацію. Це дозволяє виявляти помилки ще до виконання коду, що значно покращує якість коду і спрощує великі проекти.
2. *C#* – мова розробки, створена *Microsoft*, часто використовується з *.NET Framework* для створення різноманітних типів програм, включаючи веб-сервіси.
3. *Kotlin* – мова програмування від *JetBrains*, яка в основному використовується для розробки на платформі *Android* та серверних додатків.
4. *Python* – високорівнева мова програмування, відома своєю читабельністю та простотою.

Таблиця 3.2

Порівняння бекенд мов та фреймворків

Мова/Фреймворк	Переваги	Недоліки
TypeScript (*NestJS*)	Строга типізація для зниження помилок	Вимагає певного часу на вивчення *TypeScript*
	Сучасна архітектура, підтримка мікросервісів	Менша спільнота порівняно з іншими мовами
	Інтеграція з іншими *JS/TS* бібліотеками і фронтендом	
C# (*.NET Core*)	Висока продуктивність і підтримка корпоративних рішень	Вимагає використання *Windows* для деяких функцій *.NET*
	Сильна підтримка з боку *Microsoft*	Може бути важчим для новачків у порівнянні з *JavaScript*
Kotlin (*Spring Boot*)	Сучасна мова з високою експресивністю	Час запуску та споживання ресурсів можуть бути вищими
	Інтеграція з *Java* екосистемою	Складність у налаштуванні порівняно з *Node.js*
Python (*Django, Flask*)	Швидкий старт та легкість використання	Нижча продуктивність порівняно з компільованими мовами
	Велика спільнота та багато готових рішень	Динамічна типізація може призвести до помилок у великих проектах

TypeScript разом з *NestJS* були вибрані через їхню здатність забезпечувати розробку надійних і масштабованих бекенд-систем з використанням сучасних

підходів, таких як мікросервіси та вбудована підтримка *Dependency Injection* та модульності. Ця комбінація пропонує:

1. Строга типізація. Помітне зниження помилок на етапі компіляції, що є великою перевагою для комплексних систем.
2. Модульністю Завдяки архітектурі *NestJS*, можна легко створювати великі та зрозумілі системи, які легко підтримувати та розширювати.
3. Інтеграція. Легке злиття з фронтендом, розробленим на *TypeScript*, забезпечує безперебійну інтеграцію та спільне використання типів та інтерфейсів.

Цей вибір аргументується високою продуктивністю розробки і відносною простотою управління проектом на всіх етапах життєвого циклу, забезпечуючи швидке впровадження змін і легке масштабування системи.

3.2 Створення бази даних

Створення бази даних є одним із фундаментальних аспектів реалізації будь-якої інформаційної системи, особливо коли мова йде про систему управління замовленнями послуг. Ефективна база даних не лише забезпечує зберігання та організацію даних, але й відіграє критичну роль у забезпеченні продуктивності, масштабованості та безпеки системи. В цьому контексті, ключовим завданням є не тільки технічне втілення бази даних, але й глибоке розуміння бізнес-процесів, які ця база даних підтримує.

3.2.1 Вибір системи управління базами даних

Для ефективного функціонування будь-якої інформаційної системи, зокрема системи управління замовленнями послуг, вибір підходящої системи управління базами даних (СУБД) є вирішальним. Розглядаючи різні доступні СУБД, важливо аналізувати їх здатність задовольняти специфічні вимоги проекту, такі як производительність, надійність, масштабованість, підтримка транзакцій, інтеграційні можливості та вартість.

Критерії вибору СУБД:

1. Масштабованість: Спроможність системи ефективно масштабуватися відповідно до зростаючих обсягів даних і користувачів.
2. Транзакційна підтримка. Підтримка *ACID* для забезпечення надійності даних у багатокористувацькому середовищі.
3. Безпека. Вбудовані засоби для забезпечення безпеки даних, включаючи шифрування, управління доступом та аудит.
4. Вартість. Вартість ліцензування та підтримки СУБД.
5. Спільнота та підтримка. Наявність розширеної спільноти та документації, яка може допомогти при розгортанні та налаштуванні СУБД.

Аналіз популярних СУБД:

1. *MySQL*. Популярна реляційна СУБД, відома своєю легкістю використання та широким застосуванням у веб-розробці. Втім, має певні обмеження у складності транзакцій та масштабуванні.
2. *Microsoft SQL Server*. Надійна СУБД для корпоративних рішень з високим рівнем підтримки та інтеграції з іншими продуктами *Microsoft*. Може бути дорогою в обслуговуванні та ліцензуванні.
3. *MongoDB*. Документо-орієнтована нереляційна СУБД, оптимальна для проектів, яким необхідна велика гнучкість схеми даних. Втім, може мати проблеми з транзакційною надійністю.
4. *PostgreSQL*. Реляційна СУБД, яка вирізняється своєю розширюваністю, строгим дотриманням стандартів *SQL*, підтримкою складних запитів та транзакцій.

PostgreSQL є найкращим вибором для проекту з управління замовленнями послуг з кількох причин:

1. Масштабованість та висока продуктивності. *PostgreSQL* ефективно обробляє великі обсяги даних та складні запити, що є критично важливим для нашої системи.
2. Розширена транзакційна підтримка. Повна підтримка *ACID* та надійні механізми відновлення після збоїв забезпечують надійність і цілісність даних.

3. Відкритий код і гнучкість. *PostgreSQL* є відкритим програмним забезпеченням, що знижує загальні витрати на впровадження та підтримку.
4. Активна спільнота та багата документація. Велика спільнота розробників та наявність глибоких ресурсів для навчання та підтримки сприяють легкості впровадження та експлуатації.

3.2.2 Реалізація бази даних

Реалізація бази даних в рамках проекту системи управління замовленнями послуг включала використання *TypeORM* – потужного інструмента для взаємодії з базами даних у *TypeScript*. *TypeORM* дозволяє автоматизувати процес створення схем бази даних, забезпечуючи високий рівень абстракції та гнучкості. Ключові кроки та аспекти реалізації бази даних за допомогою цього інструменту включають:

1. Конфігурація *TypeORM*. Перед початком реалізації бази даних було налаштовано *TypeORM* з урахуванням специфіки проекту:
   - підключення до бази даних. Налаштування параметрів з'єднання з базою даних, включаючи тип СУБД (*PostgreSQL*), адресу сервера, порт, ім'я бази даних, ім'я користувача та пароль;
   - *synchronization*. Включення опції автоматичної синхронізації схеми, що дозволяє *TypeORM* автоматично оновлювати структуру бази даних у відповідності до моделей даних у коді.
2. Визначення сутностей. Для кожної таблиці бази даних були визначені класи-сутності в *TypeScript*, які описують структуру таблиць та відносини між ними:
   - моделі даних. Класи з декораторами *TypeORM*, такими як *@Entity*, *@Column*, *@PrimaryGeneratedColumn*, що відображають поля таблиць;
   - зв'язки між таблицями. Використання декораторів *@ManyToOne*, *@OneToMany*, *@ManyTo*Many для визначення відносин між

сущностями, таких як зв'язки "багато до одного", "один до багатьох" та "багато до багатьох".

3. Міграції. Для контролю версій бази даних та забезпечення безпеки даних при оновленні схеми було використано механізм міграцій:

- створення міграцій. Автоматичне генерування міграційних скриптів, які містять *SQL*-команди для модифікації структури бази даних;
- застосування міграцій. Виконання міграцій через командний інтерфейс *TypeORM* для застосування змін до бази даних.

Лістинг 3.1. Фрагмент коду сутності *User*

```
@Entity({ name: 'users' })
export class User extends BaseEntity {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ nullable: false })
  firstName: string;

  @Column({ nullable: false })
  lastName: string;

  @Column({ nullable: true })
  middleName?: string;

  @Column({ nullable: false, unique: true })
  email: string;

  @Column({ nullable: true })
  phone: string;

  @Column({ nullable: true, select: false })
  password: string;

  @Column({ nullable: false, default: false })
  active: boolean;

  @Column({ nullable: false, default: UserRole.PROVIDER })
```

```
  role: UserRole;

  @Column({ nullable: true })
  token: string;

  @Column({ nullable: true, default: null})
  tokenExpiresAt: Date;

  @OneToMany(() => Session, (session) => session.user)
  sessions: Session[];

  @OneToMany(() => Order, (order) => order.customer)
  orders: Order[];

  @CreateDateColumn({})
  createdAt: Date;

  @UpdateDateColumn()
  updatedAt: Date;

  @DeleteDateColumn()
  deletedAt: Date;
}
```

Цей підхід до реалізації бази даних з використанням *TypeORM* забезпечив ефективне впровадження схеми, що відповідає вимогам проекту, а також гнучкість у управлінні структурою даних та їхньої інтеграції з різними компонентами системи.

3.2.3 Оптимізація та налаштування

Оптимізація та налаштування бази даних – це ключові компоненти успішної імплементації інформаційної системи, особливо в контексті системи управління замовленнями послуг, де вимоги до швидкості обробки даних і надійності є високими. Налаштування бази даних включає ряд дій, спрямованих на підвищення її продуктивності, забезпечення безпеки даних та підтримання стабільності системи. Ось основні аспекти, бути розглянуті:

1. Параметри конфігурації СУБД:

- кешування та буферизація. Налаштування параметрів кешування дозволяє зменшити затримки при доступі до даних і знизити навантаження на дискову систему;
- розмір пулу з'єднань. Оптимальний розмір пулу з'єднань важливий для балансування між максимальним використанням ресурсів сервера і мінімізацією часу очікування відкликання.

2. Індексація:
   - створення індексів. Ефективна індексація критичних стовпців може значно покращити швидкість виконання запитів. Важливо аналізувати запити до бази даних для визначення, які стовпці потребують індексації;
3. Оптимізація запитів:
   - аналіз планів виконання запитів. Використання інструментів аналізу планів запитів дозволяє ідентифікувати "вузькі місця" у запитах і вносити зміни для покращення продуктивності;
   - оптимізація SQL-коду. Рефакторинг запитів може включати переписування SQL-операторів для використання більш ефективних конструкцій або зменшення кількості даних, що обробляються.
4. Безпека:
   - контроль доступу. Встановлення строгих правил доступу до даних, включаючи аутентифікацію, авторизацію та аудит;
   - шифрування. Забезпечення шифрування даних, що зберігаються та передаються, для захисту від несанкціонованого доступу.

3.3 Реалізація доступу до даних

У рамках розробки системи управління замовленнями послуг, реалізація доступу до даних відіграє критично важливу роль, адже вона безпосередньо впливає на продуктивність, безпеку та загальну ефективність системи. Надійний та ефективний доступ до даних забезпечує основу для всіх основних операцій системи, включаючи обробку замовлень, управління клієнтською базою, здійснення фінансових транзакцій, та аналітику.

Значущість цього процесу полягає в тому, що відповідно до вимог, кожен елемент системи має функціонувати в унісоні, забезпечуючи швидкий, надійний і безпечний доступ до даних для користувачів системи. Важливим є не тільки технічне втілення, але й дотримання стандартів захисту інформації, а також гарантування відповідності до регулятивних і бізнес-вимог.

3.3.1 Архітектура доступу до даних

Архітектура доступу до даних у проекті системи управління замовленнями послуг заснована на мікросервісній структурі, що використовує *RESTful API*, реалізоване за допомогою фреймворку *NestJS*. Цей підхід дозволяє забезпечити високу масштабованість, гнучкість та надійність системи, що є важливими аспектами для сучасних бізнес-додатків.

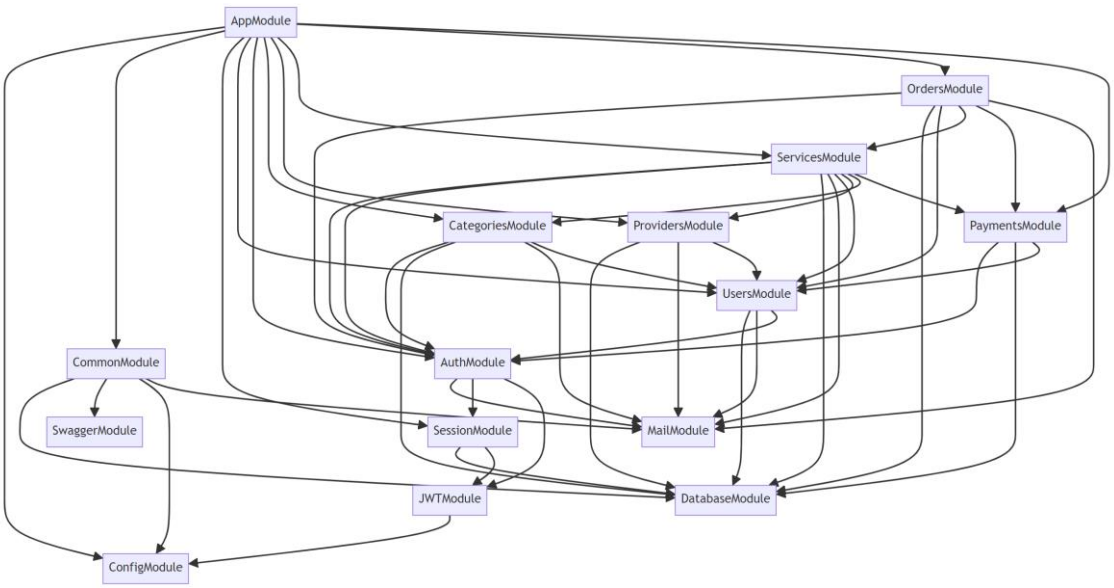


Рис. 3.1 – Схема взаємодії на модульному рівні

Мікросервісна архітектура передбачає розділення функціональності системи на незалежні сервіси, кожен з яких виконує певний набір завдань і спілкується з іншими сервісами через визначені *API*.

Для взаємодії між мікросервісами було обрано архітектурний стиль *REST*, який використовує стандартні *HTTP* методи для доступу та маніпуляції даними. *RESTful API* забезпечує:

1. Стандартизацію звернень. Легкість інтеграції з різними клієнтами та сервісами завдяки використанню стандартних *HTTP* запитів.
2. Безстановість. Кожен запит від клієнта містить всю необхідну інформацію для його обробки, що спрощує масштабування та забезпечення надійності.
3. Кешування відповідей. Підвищення продуктивності системи за рахунок можливості кешування відповідей.

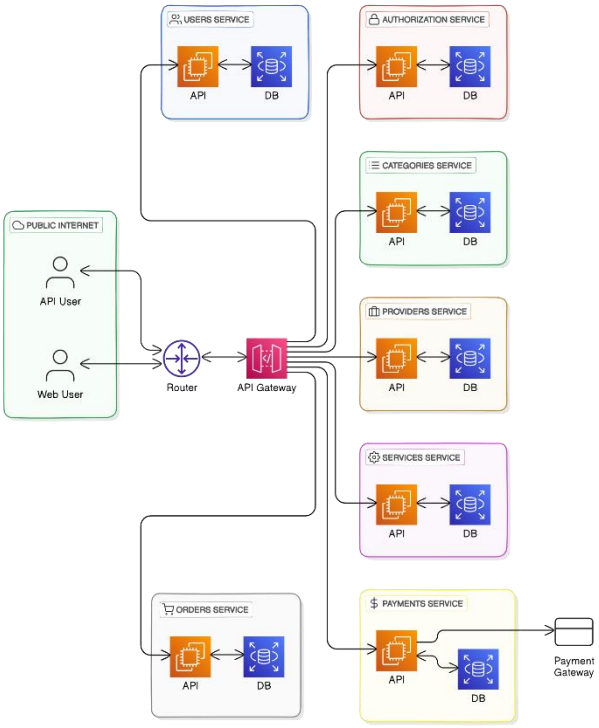


Рис. 3.2 – Схема взаємодії на сервісному рівні

Реалізація доступу до даних через мікросервісну архітектуру з використанням *RESTful API* на базі *NestJS* дозволяє системі управління замовленнями послуг ефективно працювати на великому масштабі, забезпечуючи високу продуктивність, надійність та легкість у додаванні нових функцій.

3.3.2 Реалізація API

Реалізація *API* в системі управління замовленнями послуг є ключовим елементом, який забезпечує інтерфейс між різними компонентами системи та її

користувачами. Використання *NestJS* як фреймворку для створення *RESTful API* дозволяє створити структуроване, надійне та легко масштабоване рішення.

Кожна кінцева точка *API* визначається з урахуванням бізнес-логіки, яку потрібно обробити, та даних, які потрібно відправити чи отримати. Наприклад:

- *POST /v1/crm/categories* для створення нової категорії;
- *GET /v1/crm/categories/{id}* для отримання інформації про конкретну категорію;
- *GET /v1/crm/categories* для отримання пагінованого переліку категорій;
- *PATCH /v1/crm/categories/{id}* для оновлення даних категорії;
- *DELETE /v1/crm/categories/{id}* для видалення категорії.

Для кожної кінцевої точки *API* реалізується відповідна бізнес-логіка, що обробляє запити та формує відповіді (рис. 3.3).

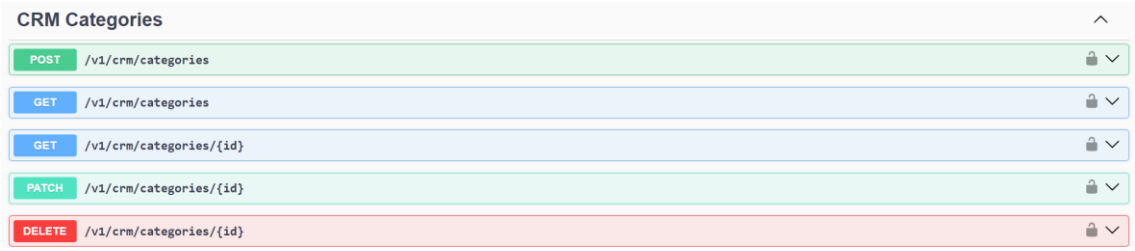


Рис. 3.3 – *CRUD* операції контролера *CRM Categories*

NestJS дозволяє використовувати декоратори та ін'єкцію залежностей для визначення сервісів, які обробляють логіку:

1. Сервіси. Створення сервісних класів, що включають методи для обробки бізнес-операцій (наприклад, створення замовлення, отримання деталей замовлення тощо), лістинг 3.2 демонструє метод створення замовлення.
2. Контролери. Визначення контролерів, які приймають *HTTP*-запити, викликають відповідні методи сервісів і повертають *HTTP*-відповіді, лістинг 3.3 демонструє метод отримання пагінованого переліку.

Лістинг 3.2. Фрагмент коду сервіса замовлень

```
async create(orderInputDto: DeepPartial<Order>): Promise<Order> {
  return this.ordersRepository.create({
    ...orderInputDto,
```

```
      status: orderInputDto
        .services
        .every(({ autocomplete }) => autocomplete === true) ?
OrderStatus.ACCEPTED : OrderStatus.WAITING
    }).save();
  }
```

Лістинг 3.3. Фрагмент коду контролера категорій

```
@Get()
  @Roles([UserRole.ADMIN, UserRole.PROVIDER])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiQuery({ name: 'limit', type: Number, required: false })
  @ApiQuery({ name: 'offset', type: Number, required: false })
  @ApiOkResponse({ type: PaginatedCategoryOutputDto })
  async findAll(@Query() queryParams: PaginatedRequestParamsDto):
Promise<PaginatedCategoryOutputDto> {
    const { limit, offset } = queryParams;
    const [categories, total]= await this.categoriesService.findAll(limit,
offset);
    return {
      status: ResponseStatus.SUCCESS,
      total,
      limit,
      offset,
      data: plainToInstance(CategoryOutputDto, categories)
    };
  }
```

Валідація вхідних даних критично важлива для забезпечення коректності даних, що обробляються. *NestJS* пропонує інтеграцію з класами-валідаторами, які можна використовувати для перевірки даних на відповідність встановленим правилам:

1. *DTO* (*Data Transfer Objects*). Використання *DTO* для визначення структур даних, які приймаються від користувачів (лістинг 3.4).
2. Декоратори валідації. Застосування декораторів, таких як *@IsNotEmpty*, *@IsInt*, *@IsEmail* для автоматичної валідації вхідних параметрів.

Лістинг 3.4. Фрагмент коду вхідної *DTO* моделі

```
export class PaginatedRequestParamsDto {
  @ApiPropertyOptional({
    default: DEFAULT_OFFSET
  })
  @IsNumber()
  @Transform(({ value }) => parseInt(value))
  public readonly offset?: number = DEFAULT_OFFSET;

  @ApiPropertyOptional({
    default: DEFAULT_LIMIT
  })
  @IsNumber()
  @Transform(({ value }) => parseInt(value))
  public readonly limit?: number = DEFAULT_LIMIT;

  @ApiPropertyOptional({
    default: DEFAULT_ORDER_BY
  })
  @IsString()
  public readonly orderBy?: string = DEFAULT_ORDER_BY;

  @ApiPropertyOptional({
    default: DEFAULT_ORDER,
    enum: ORDER
  })
  @IsString()
  @Transform((param) => param.value.toUpperCase())
  @IsEnum(ORDER)
  public readonly order?: ORDER = DEFAULT_ORDER;
}
```

Документування *API* є важливим для забезпечення зручності користування та підтримки.

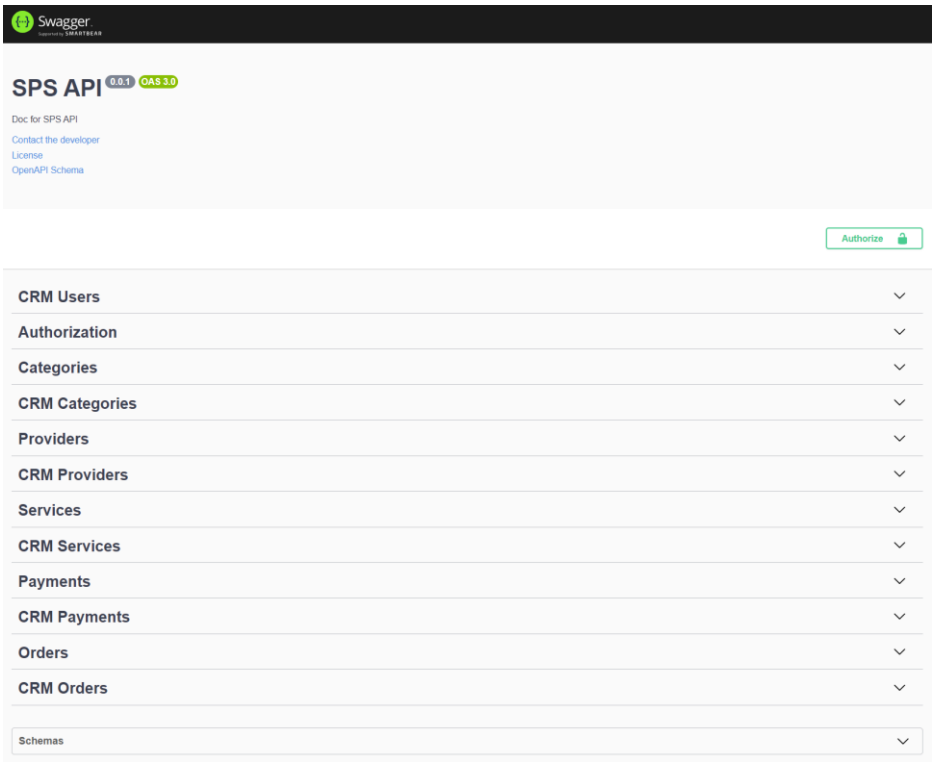


Рис. 3.4 – *Swagger* документація

NestJS підтримує інтеграцію з *Swagger*, яка дозволяє автоматично генерувати інтерактивну документацію *API*:

1. *Swagger*. Конфігурація *Swagger* для автоматичного створення документації, що включає опис кінцевих точок, параметрів, відповідей та моделей даних.

Реалізація *API* відіграє ключову роль у забезпеченні функціональності та інтеграційної спроможності системи управління замовленнями послуг, а використання *NestJS* сприяє створенню структурованого, ефективного та легко тестованого *API*.

3.4 Реалізація користувацького інтерфейсу

Реалізація користувацького інтерфейсу в системі управління замовленнями послуг є критичною складовою, що впливає на зручність та ефективність використання системи кінцевими користувачами. Цей процес вимагає глибокого розуміння потреб користувачів, технічних можливостей інструментів розробки, а також трендів у дизайні та взаємодії з користувачами. Інтерфейс має бути не тільки функціональним і зручним, але й естетично

привабливим, щоб спонукати користувачів повертатися до використання системи.

3.4.1 Розробка CRM

Сайт розроблений на мові *Typescript* за допомогою фреймворку *React*. Він має інтуїтивно зрозумілий та зручний інтерфейс користувача, адаптований для різних розмірів екранів мобільних пристроїв. Сайт доступний декількома мовами, що дозволяє використовувати його користувачам з різних країн.

Сторінка реєстрації (рис. 3.5). Відображає сторінку реєстрації для користувача.

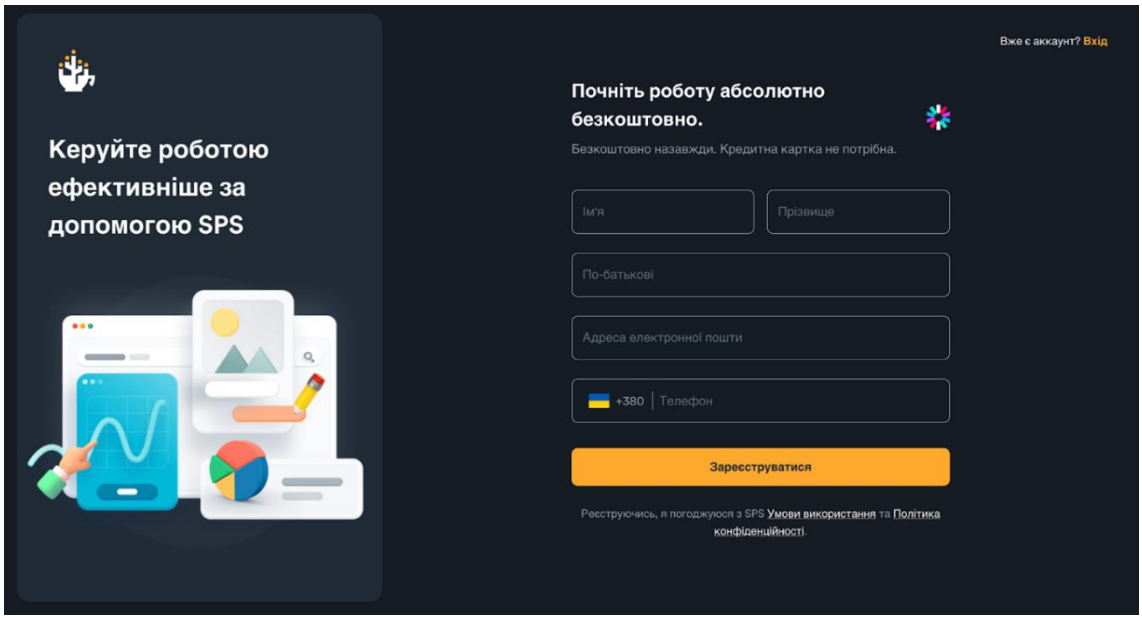


Рис. 3.5 – Сторінка реєстрації

Містить кнопки для:

- реєстрації;
- входу.

Має посилання на:

- умови використання;
- політику конфіденційності.

Має поля для вводу:

- імені;
- прізвища;

– по-батькові;
– адреси електронної пошти;
– номеру телефону.

На цій сторінки користувач може зареєструватися для подальшої роботи в *CRM*.

Сторінка входу (рис. 3.6). На цій сторінці відображена форма для входу користувача якщо в нього вже є акаунт.

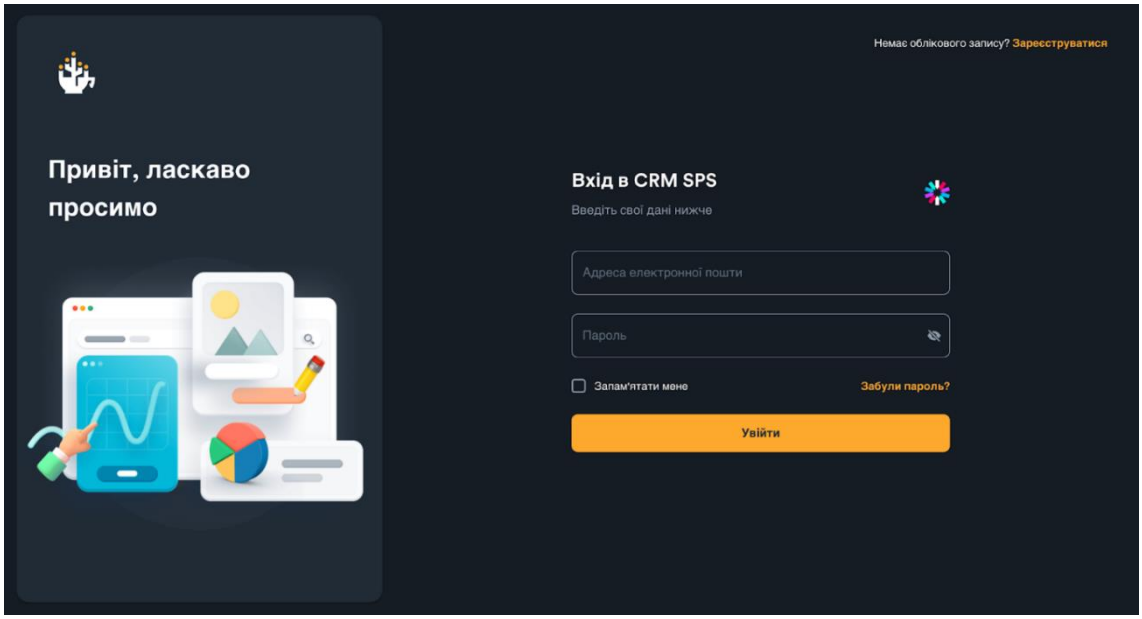


Рис. 3.6 – Сторінка входу

Містить кнопки:

– входу, дозволяє користувачу відправити дані і якщо такий користувач є у базі то відбудеться вхід;
– реєстрації, яка призначена для користувачів які ще не мають особистого кабінету;
– кнопка на полі вводу паролю, яка дозволяє побачити користувачу що він ввів;
– чек-бокс, якщо користувач не хоче кожен раз вводити дані, то він може використати цей чек-бокс і в майбутньому дані будуть поставлятися автоматично.

Має поля для вводу:

– адреси електронної пошти;
– пароль.

Якщо ви не ввели дані то отримаєте повідомлення як на рисунку 3.7, або якщо дані будуть невірно введені, повідомлення як на рисунку 3.8.

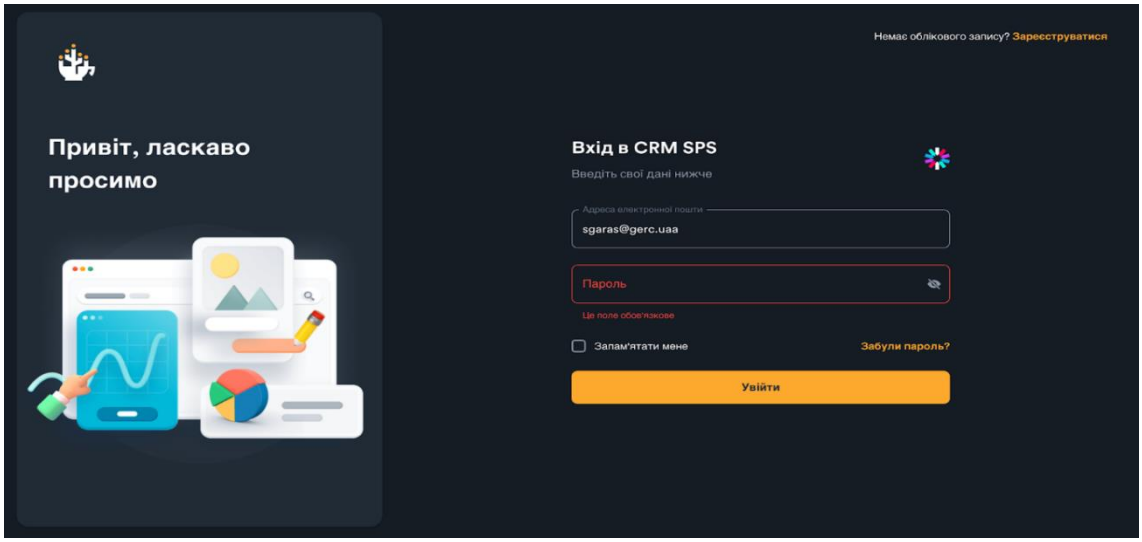


Рис. 3.7 – Реакція на не заповнене поле

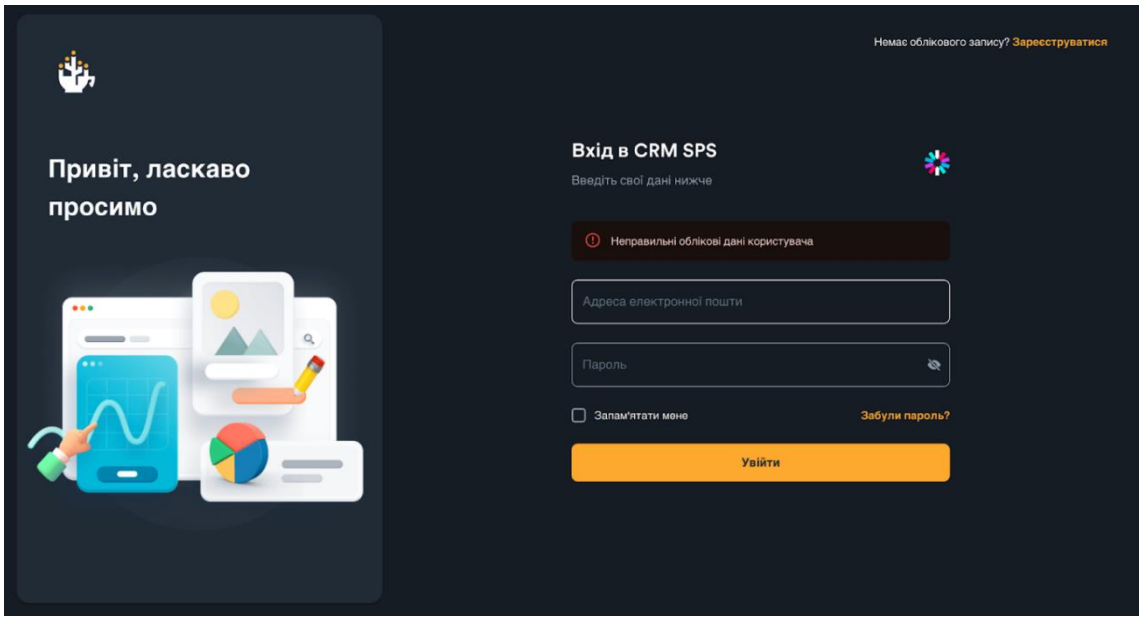


Рис. 3.8 – Реакція на невірно вказані дані

Сторінка поновлення паролю. Тут користувач якщо він забув пароль може його поновити.

Містить кнопки:

- для відправки електронної пошти по якому користувач хоче поновити пароль;
- кнопка для повернення на сторінку входу, якщо користувач передумає.

Має поля для вводу:

– адреси електронної пошти.

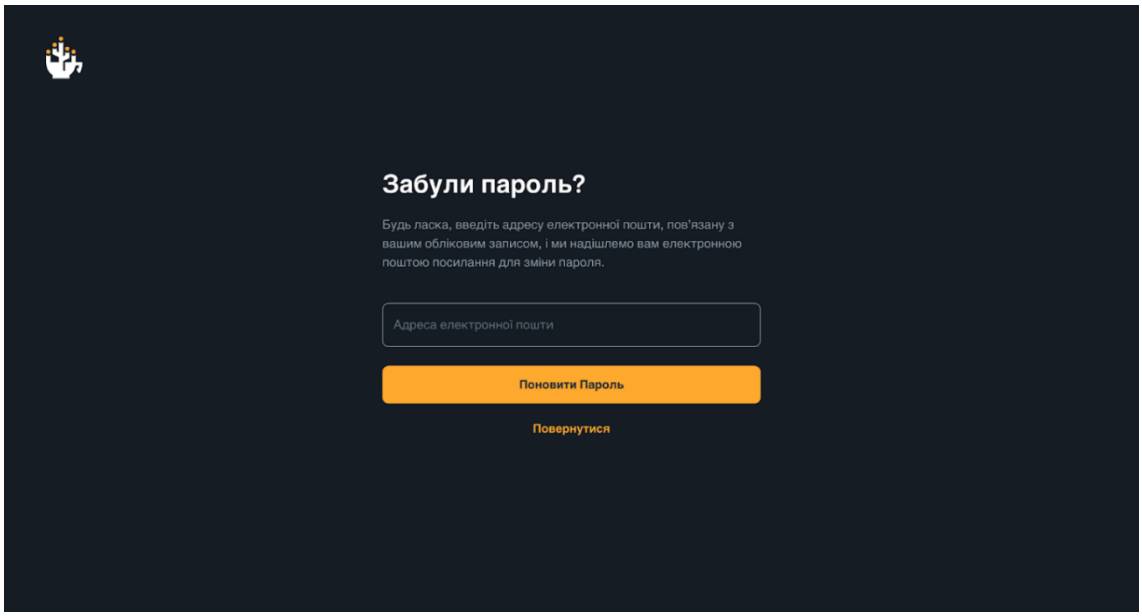


Рис. 3.9 – Сторінка для поновлення паролю

Після реєстрації чи поновлення пароля користувача перекидає на сторінку для підтвердження дій. Під час цього процесу користувачу на електронну пошту приходить код підтвердження, в залежності від дій це може бути лист як на рисунку 3.10 якщо ми реєструємося, або 3.11 якщо ми поновлюємо пароль.

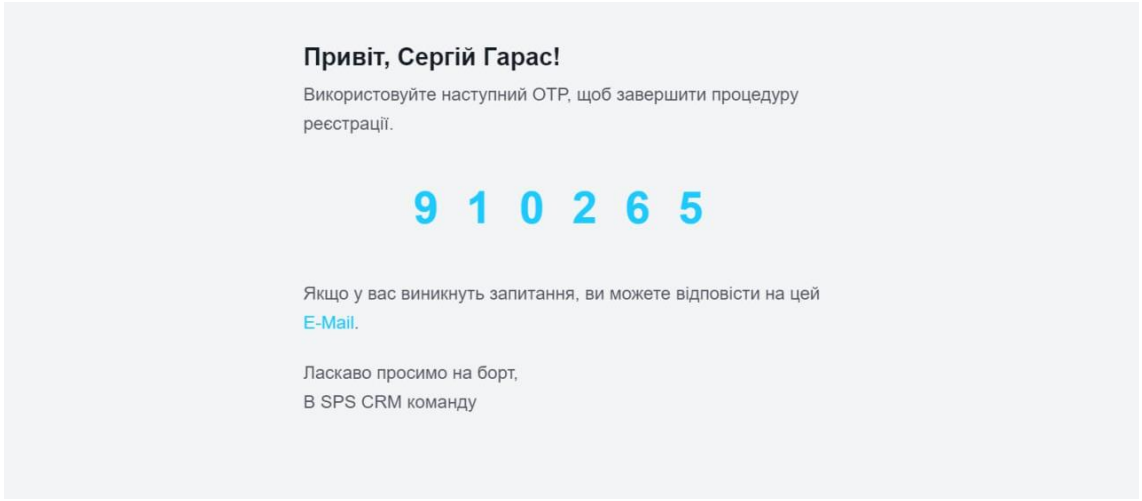


Рис. 3.10 – Сторінка для поновлення паролю

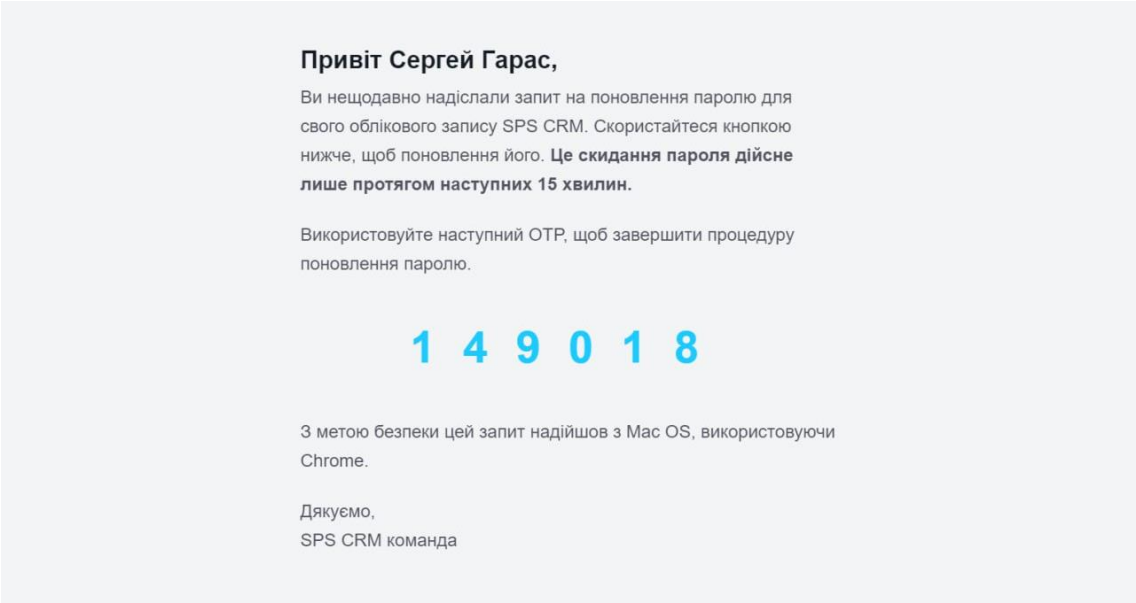


Рис. 3.11 – Сторінка для поновлення паролю

Код з отриманого листа ми вводимо на сторінку підтвердження дій (рис. 3.12).

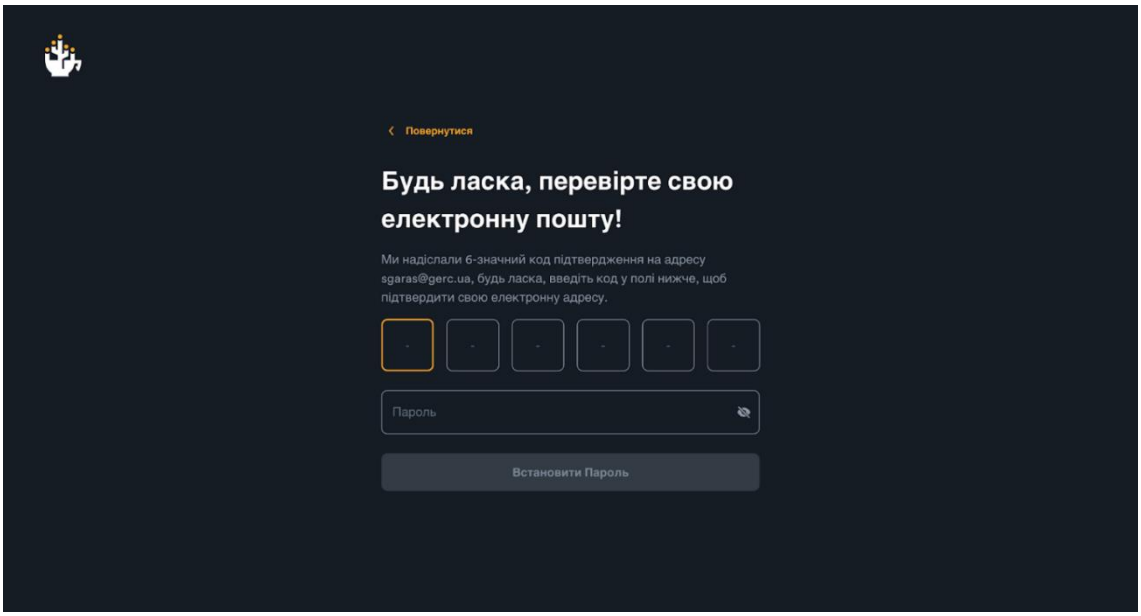


Рис. 3.12 – Сторінка для поновлення паролю

Містить кнопки для:

- для відновлення паролю;
- для повернення якщо користувач передумав виконувати свої дії далі.

Має поля для вводу:

- коду який прийде на пошту;
- паролю який користувач має придумати.

3.5 Збірка та розгортання функціональних одиниць

Збірка та розгортання функціональних одиниць системи управління замовленнями послуг здійснювалася за допомогою автоматизованих інструментів та процесів. Центральну роль в цьому процесі відіграли *GitHub Actions* та *SSH* деплой на віддалену машину. Для ефективної автоматизації було використано *self-hosted runner*, що дозволило вдосконалити контроль та гнучкість процесів збірки та розгортання.

3.5.1 Процес автоматизації за допомогою GitHub Actions

GitHub Actions дозволило автоматизувати процеси *CI/CD* (*Continuous Integration/Continuous Deployment*), що сприяло швидкій ітерації та випуску оновлень системи. Ось основні кроки, реалізовані у рамках *GitHub Actions*:

1. Тригери збірки. Збірка ініціювалася автоматично при здійсненні *push*-операцій до головної гілки (*main*) репозиторія, забезпечуючи неперервну інтеграцію коду.

2. Збірка проекту. Використання *self-hosted runner* дозволяло виконувати збірку безпосередньо на домашньому сервері, що забезпечувало вищу швидкість обробки та кращий контроль над збереженням і використанням ресурсів.

3. Тестування. Автоматичне виконання тестів для перевірки стабільності та відсутності помилок у нових версіях програмного забезпечення перед їх розгортанням.

3.5.2 Розгортання за допомогою SSH

SSH (*Secure Shell*) є стандартним методом для безпечного адміністрування віддалених серверів. Використання *SSH* для розгортання додатків включає кілька важливих кроків:

1. Настроювання безпечного з'єднання. Конфігурація *SSH* ключів для безпечного доступу до сервера без необхідності вводу пароля.

2. Передача файлів. Використання команд *rsync* через *SSH* для передачі зібраних файлів з *CI* сервера на віддалені сервери, де файлова структура і конфігурація системи вимагали точного розміщення для коректної роботи.

3. Автоматизація процесу розгортання. Налаштування скриптів на сервері для автоматичного розгортання оновлень та запуску або перезапуску відповідних сервісів, що забезпечує мінімальний простій системи.

4. Моніторинг стану після розгортання. Використання інструментів моніторингу для відстеження стану системи після розгортання, забезпечуючи можливість швидкого відгуку на будь-які проблеми, які можуть виникнути.

Лістинг 3.4. *Workflow file* з *Github*

```
name: Deploy React App
on:
  push:
    branches:
      - main
jobs:
  build:
    runs-on: self-hosted
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: Set up Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '20'
      - name: Cache Yarn cache
        uses: actions/cache@v4
        with:
          path: |
            ~/.yarn/cache
            node_modules
          key: ${{ runner.os }}-yarn-${{ hashFiles('**/yarn.lock') }}
          restore-keys: |
            ${{ runner.os }}-yarn-
      - name: Install dependencies
        run: yarn install --frozen-lockfile
```

```
      - name: Build React app
        run: yarn build
      - name: Upload production-ready build files
        uses: actions/upload-artifact@v4
        with:
          name: build
          path: build
  deploy:
    runs-on: self-hosted
    needs: build
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: Download build files
        uses: actions/download-artifact@v4
        with:
          name: build
          path: build
      - name: Add SSH key and set up known_hosts
        env:
          SSH_PRIVATE_KEY: ${{ secrets.SSH_PRIVATE_KEY }}
          SSH_HOST: ${{ secrets.SSH_HOST }}
        run: |
          mkdir -p ~/.ssh
          echo "${SSH_PRIVATE_KEY}" > ~/.ssh/private_key
          chmod 600 ~/.ssh/private_key
          ssh-keyscan ${SSH_HOST} >> ~/.ssh/known_hosts
      - name: Deploy to Server
        env:
          SSH_PRIVATE_KEY: ${{ secrets.SSH_PRIVATE_KEY }}
          SSH_HOST: ${{ secrets.SSH_HOST }}
          SSH_USER: ${{ secrets.SSH_USER }}
        run: |
          echo "${SSH_PRIVATE_KEY}" > private_key
          chmod 600 private_key
          rsync -avz -e "ssh -i private_key" build/
${SSH_USER}@${SSH_HOST}:/var/www/pay/
          rm private_key
```

Цей інтегрований підхід до збірки та розгортання дозволив забезпечити високий рівень автоматизації процесів, значно знижуючи ризики помилок при ручному розгортанні та збільшуючи загальну ефективність розробки. Використання сучасних інструментів та методик в рамках стратегії *CI/CD* стало фундаментом для підтримки постійного вдосконалення та оновлення системи управління замовленнями послуг.

Висновок до третього розділу

У цьому розділі було розглянуто ключові аспекти практичної реалізації проекту системи управління замовленнями послуг. Завдяки аналізу можливостей різних технологій, прийнято рішення використовувати NestJS для бекенду та React для фронтенду через їхню високу продуктивність, гнучкість та популярність у сучасній веб-розробці.

Основу даних системи спроектовано з використанням PostgreSQL, що забезпечує високу продуктивність, надійність та гнучкість у масштабуванні. Значна увага приділена забезпеченню безпеки, оптимізації запитів та інтеграції компонентів системи для стабільної роботи. Впроваджено автоматизацію процесів збірки та розгортання через GitHub Actions, що оптимізувало випуск оновлень. Особлива увага приділена розробці зручного та інтуїтивного користувацького інтерфейсу, що підвищує задоволеність користувачів.

РОЗДІЛ 4
ЕКОНОМІЧНІ РОЗРАХУНКИ

4.1 Техніко-економічний аналіз проекту

Темою даної роботи є “Розробка багатоплатформової системи керування замовленнями послуг”. Програмному продукту, що розробляється в даній дипломній роботі, характерні наступні властивості: ергономічність інтерфейсу програмного продукту, мультизадачність, мала ресурсозатратність.

Головною перевагою системи є те, що користувач може швидко оперувати інформацією в додатку. Це дозволяє користувачу значно економити свій час та дозволяє підвищити ефективність роботи. Правильна організація даних дозволяє швидко її обробляти та удосконалювати її.

Різноманітні функції дозволяють користувачеві ефективно оформляти замовлення в додатку.

При виконанні дипломної роботи база даних була створена за допомогою СУБД PostgreSQL, а додаток – VS Code.

4.1.1 Класифікаційна оцінка різновиду проекту

По масштабу проект, що розробляється, відноситься до малих проектів, які невеликі за масштабом, прості і обмежені об’ємами.

По термінах реалізації проект короткостроковий (термін реалізації до трьох років).

По складності проект, що розробляється, є проектом середньої складності – має велике сховище даних, використовується трьома або більше типами користувачів.

По характеру цільового завдання – комбінований проект (учбово-освітній і дослідницький).

По характеру проекту – галузевий проект.

По ступеню новизни – В (ПЗ, що мають аналоги).

4.1.2 Визначення мети і результатів проекту

Метою дослідження є розробка багатоплатформної системи управління замовленнями, що відповідає сучасним вимогам ринку та вирішує проблеми, пов'язані з існуючими системами.

Практичне значення отриманих результатів полягатиме в можливості використання цих розробок для покращення ефективності бізнес-процесів, оптимізації взаємодії з клієнтами та підвищення конкурентоспроможності підприємств на ринку послуг.

Таблиця 4.1

Етапи і стадії роботи

№	Стадії	Зміст	Тривалість виконання
1	Технічне завдання	Постановка задач, визначення складу ПП,техніко-економічне обґрунтування роботи, вибір мов програмування	лютий
2	Технічний проект	Розробка алгоритмів рішення задач. Розробка структури програми, програмної документації, схем БД	березень
3	Робочий проект	Програмування і налагодження програм. Комплексне налагодження задач і здача в експлуатацію для тестування	квітень – травень

Продовження таблиці 4.1

4	Впровадження	Підготовка і передача програмної документації для супроводу і виготовлення з оформленням акта. Перевірка алгоритмів	червень

4.2 Розрахунки ціни програмного продукту (ПП)

4.2.1 Визначення трудомісткості розробки ПП

Як вихідні дані для визначення трудомісткості розробки ПП використовується типовий склад етапів і укрупнені норми часу на розробку програмних засобів (ПЗ).

Розроблювальному ПП відповідає програма-аналог “ПЗ автоматизованих розрахунків” з обсягом 6000 умовних машинних команд і трудомісткістю $T_р$ = 330 люд.-год.

Трудомісткість розробки ПП включає розробку наступних етапів:

- технічного завдання – ТЗ;
- технічного проекту – ТП;
- робочого проекту – РП;
- упровадження – ВН.

Трудомісткість розроблювального ПП визначається по кожнім етапі окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня використання в розробці стандартних модулів на підставі формул:

$$T_{тз} = T_р \cdot L_1 \cdot K_н \quad (4.1)$$

$$T_{тп} = T_р \cdot L_2 \cdot K_н \quad (4.2)$$

$$T_{рп} = T_р \cdot L_3 \cdot K_н \cdot K_т \quad (4.3)$$

$$T_{вп} = T_р \cdot L_4 \cdot K_н \quad (4.4)$$

де: $T_р$ – укрупнена норма часу на розробку аналога ПЗ, люд.-години., що коректується поправочним коефіцієнтом, що враховує умови розробки ПЗ, тобто в умовах комп'ютера

$$К_к = 0,8$$

$$T_р = 330 \cdot 0,8 = 264 \text{ люд.-години}$$

L_i – питома вага і-го етапу розробки в залежності від ступеня новизни

$К_н$ – поправочний коефіцієнт, що враховує ступінь новизни

$К_т$ – поправочний коефіцієнт, враховує ступінь використання в розробці програм

$$T_{тз} = 264 \cdot 0,12 \cdot 0,7 = 22,18 \text{ люд.-години} \quad (4.1)$$

$$T_{тп} = 264 \cdot 0,11 \cdot 0,7 = 20,33 \text{ люд.-години} \quad (4.2)$$

$$T_{рп} = 264 \cdot 0,61 \cdot 0,7 \cdot 0,8 = 90,18 \text{ люд.-години} \quad (4.3)$$

$$T_{вн} = 264 \cdot 0,16 \cdot 0,7 = 29,60 \text{ люд.-години} \quad (4.4)$$

Розрахунок трудомісткості розробки ПП представлений у таблиці 4.2.

Таблиця 4.2

Розрахунок трудомісткості розробки ПП

Найменування етапів	Витрати розробника	Витрати керівника	Витрати нормоконтролю
1	2	3	4
1. Технічне завдання	$T_{ртз} = 22,18$	$T_{кк} = 0,7 \cdot N_{тз}$ $T_{кк} = 0,7 \cdot 5 = 3,5$	$T_{нк} = 0,15 \cdot N_{тз}$ $T_{нк} = 0,15 \cdot 5 = 0,75$
2. Розробка ТП (алгоритму і блок-схеми)	$T_{ртп} = 20,33$	$T_{кк} = 0,7 \cdot N_{тп}$ $T_{кк} = 0,7 \cdot 6 = 4,2$	$T_{нк}=0,15 \cdot N_{тп}$ $T_{нк}=0,15 \cdot 6 = 0,9$
3. Розробка робочого проекту	$T_{ррп} = 90,18$	$T_{кк} = 0,7 \cdot N_{рп}$ $T_{кк} = 0,7 \cdot 35 = 24,5$	$T_{нк} = 0,15 \cdot N_{рп}$ $T_{нк} = 0,15 \cdot 35 = 5,25$

Продовження таблиці 4.2

4. Налагодження і впровадження	$T_{рвн} = 29,60$	$T_{кк} = 0,7 \cdot N_{інстр}$ $T_{кк} = 0,7 \cdot 7 = 4,9$	$T_{нк} = 0,15 \cdot N_{інстр}$ $T_{нк} = 0,15 \cdot 7 = 1,05$
5. Пояснювальна записка	$T_{пз} = 1,5 \cdot N_{пз}$ $T_{пз} = 1.5 \cdot 45=67,5$	$T_{кк} = 0,7 \cdot N_{пз}$ $T_{кк} = 0,7 \cdot 45 = 31,5$	$T_{нк} = 0,15 \cdot N_{пз}$ $T_{нк} = 0,15 \cdot 45 = 6,75$
Усього по видах робіт:			
На розробку	$\sum T_р = 229,79=230$		
Контроль керівника		$\sum T_{кк} = 68,6=69$	
Нормоконтроль			$\sum T_{нк} = 14,7=15$
Загальна трудомісткість	$\sum T_i = 314$		

Примітка:

1. $T_{кк} = 0,7 \cdot N_{тз}$ – витрати праці керівника з розрахунку 0,7 години на 1 лист технічного завдання. $N_{тз}$ – кількість листів технічного завдання.
2. $T_{нк} = 0,15 \cdot Nтз$ – витрати праці по нормоконтролю з розрахунку 0,15 години на 1 лист технічного завдання.
3. $T_{кк} = 0,7 \cdot N_{рп}$ – витрати праці керівника з розрахунку 0,7 години на 1 лист лістингу програми.
4. $T_{кк} = 0,7 \cdot N_{інстр}$ – витрати праці керівника з розрахунку 0,7 години на 1 лист інструкції (по використанню програми). $N_{інстр}$ – кількість листів інструкцій.
5. $T_{пз} = 1,5 \cdot N_{пз}$ – витрати праці на пояснювальну записку з розрахунку 1,5 години на 1 лист записки без обліку аркушів, що містять етапи 1-4. $N_{пз}$ – кількість листів пояснювальної записки.

Тривалість розробки ПП у днях по кожнім етапі визначається по формулі:

$$T_{пп} = T_{ij} / 8 \qquad (4.5)$$

де: T_{ij} – трудомісткість j-го виду робіт з і-му етапу ($T_р$, $T_{кк}$, $T_{нк}$), години

8 – тривалість робочого дня, години

Розробка ПП $T_{ппр} = 230 / 8 = 28,75 = 29$ днів

Контроль керівника $T_{ппк} = 69 / 8 = 8$ днів

Нормоконтроль $T_{ппн} = 15 / 8 = 1,875 = 2$ дні

Загальна тривалість розробки ПП визначається по формулі:

$$\Sigma T_{ij} / 8, \text{ дні} \quad (4.6)$$

$$T_{пп} = 314 / 8 = 39,25 = 39 \text{ днів}$$

4.2.1 Розрахунок ціни ПП

У розділі “Розрахунок ціни ПП” для визначення ціни розраховуються матеріальні витрати, заробітна плата розробників, вартість машино-години та інші витрати на розробку ПП.

Розрахунок матеріальних витрат приведений у таблиці 4.3.

Таблиця 4.3

Розрахунок витрат на матеріали

Найменування матеріальних витрат	Од. вим.	Кількість	Ціна за од., грн.	Вартість, грн.
1	2	3	4	5
Диски	шт.	1	15	15
Папір	лист	200	0.67	134
Фарба (картридж)	шт.	1	400	400
Флеш-карта	шт.	1	110	110
Разом				659
Транспортно-заготівельні витрати – 10%				65,9
Усього				724,9

Розрахунок заробітної плати виконавців приведений у таблиці 4.4.

Таблиця 4.4

Розрахунок основної заробітної плати

Найменування робіт	Трудомісткість робіт у днях	Місячний оклад	Денна заробітна плата	Заробітна плата
1	2	3	4	5
Розробка ПП	29	8000	381	11049
Контроль керівника	8	8700	414	312
Нормоконтроль	2	9500	452	904
	39			15265

Додаткова заробітна плата враховує оплату чергових відпусток, премії, інші доплати. Приймається в розрахунках 15% від основної.

$$ЗП_{дод} = ЗП_{осн} \cdot 0,15 \quad (4.7)$$

ЗПдод = 15265 · 0,15 = 2289,75 грн

Єдиний соціальний внесок приймається у розмірі 22% від суми основної і додаткової заробітної плати.

$$ЄСВ = (ЗП_{осн} + ЗП_{доп}) \cdot 0,22 \quad (4.8)$$

ЄСВ = (15265 +2289,75) · 0,22 = 3862,05 грн

Витрати, зв'язані з використанням обчислювальної техніки, визначаються:

$$С_{еом} = t^{еом} \cdot К^{еом}_{в} \cdot Ц^{еом} \cdot К^{еом}_{бд} \cdot К^{еом}_{е} \quad (4.9)$$

де: $Т^{евм}$ – час використання ЕОМ для розробки даного ПП, години

$Т^{евм}$ = 90 годин

$К^{евм}_{и}$ – поправочний коефіцієнт обліку часу використання ЕОМ

$Ц^{евм}$ – ціна однієї години роботи на ЕОМ, грн.

$К^{евм}_{бд}$ – коефіцієнт обліку ступеня використання СУБД

$К^{евм}_{э}$ – коефіцієнт обліку швидкодії ЕОМ

$$С_{эвм} = 90 \cdot 1,08 \cdot 7 \cdot 1,1 \cdot 1,0 = 748,44 \text{ грн.}$$

Накладні витрати враховують адміністративні, загальновиробничі витрати, витрати на збут. Приймаються в розмірі 30% від основної заробітної плати.

$$Нв = 0,30 \cdot ЗП_{осн} \quad (4.10)$$

$$Нв = 0,3 \cdot 15265 = 4579,50 \text{ грн.}$$

На підставі здійснених розрахунків складається калькуляція планової собівартості ПП.

Таблиця 4.5

Калькуляція собівартості ПП

Найменування статей Витрат	Сума витрат (грн.)	Питома вага, %
Матеріали	724,9	3
Основна заробітна плата	15265	55
Додаткова заробітна плата	2289,75	8
Єдиний соціальний внесок	3862,05	14
Витрати, зв'язані с обчислювальною технікою	748,44	3
Накладні витрати	4579,50	17
Разом	27469,64	100

Ціна ПП визначається по формулі:

$$Ц = С + П_р \quad (4.11)$$

де: С – витрати на розробку програмної продукції (планова собівартість), грн.

$П_р$ – розмір прибутку, розрахований по формулі:

$$П_р = С \cdot \%Р_н / 100 \quad (4.12)$$

де: $Р_н$ – плановий рівень рентабельності (25%)

Прибуток у ціні ПП становить:

$$П = 27470 \cdot 0,25 = 6867,50 \text{ грн.}$$

Оптова ціна становить:

$$Ц = 27470 + 6867,50 = 34337,50 = 34338 \text{ грн.}$$

4.3 Розрахунок капітальних витрат

Для розрахунку економічної ефективності проекту визначаються капітальні і поточні витрати, зв'язані з використанням ІС.

Розрахунок капітальних витрат, зв'язаних із упровадженням ІС здійснюється по формулі:

$$К_2 = К_{пп} + К_п + К_{ко} + К_{во} \quad (4.13)$$

де: $К_{пп}$ – ціна програмного продукту

$К_п$ – попередвиробничі витрати

$К_{ко}$ – вартість комп'ютерного устаткування

$К_{во}$ – вартість допоміжного устаткування, необхідного для надійної роботи ІС

Ціна програмного продукту складає 34338 грн.

Попередвиробничі витрати містять у собі усі витрати, зв'язані з налагодженням і впровадженням ІС – постановка задач і їхня алгоритмізація, розробка, налагодження і впровадження програмного забезпечення (ПЗ), навчання обслуговуючого ІС персоналу і т. д.

Приймаються $К_п$ у розмірі 15% від вартості розробленого ПП.

$$К_п = 34338 \cdot 0,15 = 5150,70 \text{ грн.}$$

Вартість комп'ютера ($К_{ко}$) становить 15000 грн.

Вартість допоміжного устаткування визначається укрупнено в розмірі 10% від вартості комп'ютера.

$$К_{во} = 15000 \cdot 0,10 = 1500 \text{ грн.}$$

$$К_2 = 34338 + 5150,70 + 1500 + 1500 = 55988,70 \text{ грн.}$$

4.4 Розрахунок поточних (експлуатаційних) витрат

Розрахунок поточних (експлуатаційних) витрат, зв'язаних з використанням ПП (Сі), здійснюється по формулі:

$$C_i = C_{опл} + C_a + C_{ел} + C_р + C_{доп} + C_п \quad (4.14)$$

де: $C_{опл}$ – річний фонд основної і додаткової оплати праці персоналу, що обслуговує ІС з єдиним соціальним внеском (ЄСВ)

C_a – сума річних амортизаційних відрахувань від вартості основного і допоміжного устаткування ІС

$C_{ел}$ – вартість витрат на електроенергію в рік

$C_р$ – вартість річного ремонту основного і допоміжного устаткування

$C_{доп}$ – річна вартість допоміжних матеріалів, зв'язаних з експлуатацією КМ

$C_п$ – вартість річного утримання приміщень

4.4.1 Розрахунок поточних витрат до впровадження ПП

Під час проведення аналізу предметної області за базовий варіант було обрано діяльність двох співробітників без використання програмного продукту з заробітною платною 8500 грн і 9500 грн Розрахунок річного фонду основної і додаткової оплати праці персоналу з ЄСВ:

$$ЗП_{осн} = (ЗП_{окі} \cdot Ч_i) \cdot 12 \quad (4.15)$$

де: $Ч_i$ – чисельність, фахівців і-тої категорії, що обслуговують ІС

$ЗП_{окі}$ – місячний оклад фахівця і-тої категорії ІС

$$ЗП_{осн} = (8500 \cdot 1 + 9500 \cdot 1) \cdot 12 = 216000 \text{ грн.}$$

Фонд додаткової заробітної плати:

$$ЗП_{доп} = ЗП_{осн} \cdot К_{дод} \quad (4.16)$$

де: $К_{дод}$ – коефіцієнт додаткової заробітної плати

$$ЗП_{дод} = 216000 \cdot 0,2 = 43200 \text{ грн.}$$

Єдиний соціальний внесок приймається в розмірі 22% від суми основної і додаткової заробітної плати:

$$ЄСВ = (ЗП_{осн} + ЗП_{дод}) \cdot 0,22$$

$$ЄСВ = (216000 + 43200) \cdot 0,22 = 57024 \text{ грн.}$$

Загальні витрати на оплату праці:

$$С_{опл} = ЗП_{осн} + ЗП_{дод} + ЄСВ = 216000 + 43200 + 57024 = 316224 \text{ грн.}$$

4.4.2 Розрахунок поточних витрат, зв'язаних з використанням ПП

Після впровадження ПП роботу виконує одна людина – оператор з заробітною платою 10000 грн., а систему обслуговує програміст з погодинною ставкою 250 грн.

$$З_{посн} = 10000 \cdot 12 = 120000 \text{ грн.}$$

Фонд додаткової заробітної плати:

$$З_{доп} = 120000 \cdot 0,20 = 24\,000 \text{ грн.}$$

Річна заробітна плата програміста, що обслуговує систему 4 рази на місяць при годинній ставці 250 грн. складе:

$$ЗП_{осн} = С_т \cdot Т = 4 \cdot 12 \cdot 250 = 12000 \text{ грн}$$

Так як програміст працює погодинно, на нього не розраховується додаткова заробітна плата, що враховує оплату чергових відпусток, премії, інші доплати.

Єдиний соціальний внесок:

$$ЄСВ = (120\,000 + 24\,000 + 12\,000) \cdot 0,22 = 34\,320 \text{ грн.}$$

Загальні витрати на оплату праці:

$$С_{опл2} = ЗП_{осн} + ЗП_{дод} + ЄСВ = 120000+24000+12000+34320=190320 \text{ грн.}$$

Розрахунок амортизаційних відрахувань визначається по формулі:

$$С_а = (К_{ко} + К_{во}) \cdot Н_а / 100 \qquad (4.17)$$

$$С_а = (15000 + 1500) \cdot 0,50 = 8250 \text{ грн.}$$

де: $Н_а$ – норма амортизаційних відрахувань.

Річна вартість споживаної електроенергії $С_{ел}$, визначається по формулі:

$$С_{ел} = М_у \cdot Т_{ко} \cdot Ц_е \cdot К_в \qquad (4.18)$$

де: $М_у$ – установлена сумарна потужність комп'ютерного устаткування

$Т_{ко}$ – річний фонд часу роботи ЕОМ, який визначається виходячи з кількості робочих днів у році ($Д_р$), тривалості робочого дня (Т) з урахуванням часу на профілактичні огляди за рік ($Т_{огл}$)

$$Тко = Др \cdot Т – Тогл$$

$$Тко = 305 \cdot 8 – 400 = 2040 \text{ годин}$$

$Ц_е$ – вартість 1 квт-години ел. енергії

$С_р$ – коефіцієнт інтенсивного використання потужності

$$С_{ел} = 0,45 \cdot 2040 \cdot 0,9 \cdot 4,32 = 3843,76 = 3844 \text{ грн.}$$

Витрати на ремонт ($С_р$) приймаються в розмірі 6% від вартості комп'ютерного устаткування:

$$С_р = (15000 + 1500) \cdot 0,06 = 990 \text{ грн.}$$

Витрати на допоміжні матеріали ($С_{доп}$) приймаються в розмірі 2.5% від вартості комп'ютерного устаткування:

$$С_{доп} = (15000 + 1500) \cdot 0,025 = 412,50 \text{ грн.}$$

Загальні витрати:

$$С_2 = 190320 + 8250 + 3844 + 990 + 412,50 = 2038163,50 \text{ грн.}$$

4.5 Розрахунок показників економічної ефективності проекту

Очікуваний економічний ефект визначається по формулі:

$$Е_о = (С_1 – С_2) – Е_н \cdot (К_2 – К_1) \qquad (4.19)$$

де: $С_1, С_2$ – поточні витрати відповідно до і після впровадження проекту

$К_2$ – капітальні витрати на впровадження ПП, грн.

$К_1$ – капітальні витрати до впровадження ПП, грн.

$Е_н$ – нормативний коефіцієнт ефективності одноразових витрат

$$Е_о = (316224 – 203816,50) – 0,25 \cdot (55988,70 – 0) = 98\,410,33 \text{ грн.}$$

Потім розраховується коефіцієнт ефективності капітальних витрат по формулі:

$$Е = (С_1 – С_2) / К_2 – К_1 \qquad (4.20)$$

$$Е = 112407,50 / 55988,70 = 2,01$$

Так як $Е > Е_н$, то проект ефективний.

Розраховується строк окупності капітальних витрат на впровадження проекту:

$$T = 1 / E = 1 / 2{,}01 = 0{,}4975 = 0{,}50 \text{ року} \quad (4.21)$$

Результати економічних розрахунків відображаються в підсумковій таблиці 4.6.

Таблиця 4.6

Техніко-економічні показники проекту

№	Найменування показників	Одиниця виміру	Значення показника	
			до впровадження проекту	після впровадження проекту
1	Трудомісткість розробки проекту	люд-год.	3	314
2	Ціна ПП	грн.		34338
3	Капітальні витрати	грн.		55988,70
4	Поточні витрати	грн/рік	316224	203 816,50
5	Економічний ефект від реалізації проекту	грн/рік		98410,33
6	Строк окупності	років		0,50
7	Економічна ефективність			2,01

Висновок до четвертого розділу

Результати проведених економічних розрахунків показують, що використання даної розробки є економічно вигідним та ефективним, так як значення коефіцієнту ефективності капітальних витрат більше значення нормативного коефіцієнту ефективності одноразових витрат ($E > E_н$).

В процесі виконання економічних розрахунків я визначив ціну ПП та економічну ефективність його використання. Ціна ПП становить

34338 грн. Капітальні витрати на впровадження ПП складають 55988,70 грн.

Економічний ефект полягає у зменшенні часу на виконання операцій по плануванню бюджету, що у грошовому еквіваленті складає економію у 98410,33 грн.

Витрати на впровадження досить швидко окупляться (0,5 року), що становить 6 місяців. Економічна ефективність значно більше нормативу (0,25) і становить 2,01 грн. на гривню капітальних вкладень, прибуток становить дві гривні одна копійка.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ

5.1 Шкідливі та небезпечні фактори при роботі користувача

Мобільний телефон – це джерело надвисокочастотного випромінювання, за допомогою якого і здійснюється зв’язок. Це випромінювання пригнічує тонкі електромагнітні імпульси клітин живих організмів. Найнебезпечнішою частиною смартфонів є антена, саме вона продукує хвилі надвисоких частот. Вони шкідливі тим, що нагрівають організм «зсередини» на клітинному рівні. Особливо від цього потерпають ті частини тіла, які не омиваються кров’ю, а відтак залишаються поза системою терморегуляції організму. Зокрема, кришталик ока. Від внутрішнього перегрівання він руйнується і мутніє. Це проявляється різзю в очах і шумом у голові.

Мозок людини, на щастя, захищений черепною коробкою і добре постачається кров’ю, тому перегрівання йому не загрожує. Але вчені застерігають від іншого. Зокрема, під впливом потужних електромагнітних хвиль мобільного телефону може відбутися збій у продукуванні електроімпульсів, через які мозок керує роботою організму.

Ці хвилі набагато слабкіші, ніж іонізуюче випромінювання, подібне до рентгенівських променів, ультрафіолетового та гамма-випромінювання, здатних проникати через тканини організму і завдавати шкоди клітинам, змінюючи структуру ДНК. Однак повністю вплив цього типу випромінювання на людський організм досі не вивчили.

Світ навколо пронизаний різноманітними радіохвилями: ультракороткі хвилі, на яких працюють місцеві радіостанції, мікрохвильове випромінювання, яке виробляється надвисокочастотними печами, теплове випромінювання і видиме світло.

Відомо, що неіонізуюче випромінювання не володіє достатньою енергією, щоб безпосередньо заподіяти шкоду структурі ДНК на клітинному рівні.

Кілька років тому в мюнхенській клініці «Гросхадерн» за допомогою спеціального тестування довели, що постійні сигнали мобільних апаратів змінюють електроімпульси головного мозку. Медики запропонували учасникам експерименту провести тривалі переговори телефоном. До їхніх голів приєднали електроди, які записували імпульси мозку під час розмови. З'ясувалося, що у двох третин піддослідних енцефалограма засвідчила підвищену активність мозку. Річ у тім, що, прикладаючи телефон до вуха, людина опромінює себе з потужністю 25 000 мВт / см^2, тоді як максимально допустима є значно меншою – 10 мВт / см^2. Організм людини ж працює зі значно меншою потужністю – 0,001 мВт / см^2.

5.2 Методи зниження впливу шкідливих та небезпечних факторів при роботі користувача

Найпотужніший радіосигнал – у передавальної антени, яка у сучасних смартфонів прихована всередині корпусу.

Якщо вам доводиться багато спілкуватися мобільним телефоном, не тримайте його постійно біля вуха. Купіть собі навушники. Це частково зменшить вплив надпотужного випромінювання.

Проте чим ближче антена до голови, тим вище очікуваний результат випромінюваної енергії, згідно *ACS*.

Не обирайте маленькі моделі мобільних телефонів, вони мають потужніше випромінювання порівняно з більшими.

Набравши потрібний номер, не притискайте відразу телефон до вуха – саме під час з'єднання відбувається найпотужніше випромінювання. Тому стежте за процесом виклику, дивлячись на екран смартфону, і лише після того, як з'єднання відбулося, підносьте його до вуха.

Тримайте телефон не ближче 2 м від ліжка, щоб віддалити себе від випромінюваного ним поля.

Намагайтеся не розмовляти довше трьох хвилин. Між розмовами робіть перерви не менш як на 15 хвилин. Стежте, щоб загальна кількість розмов за добу не перевищувала однієї години.

Майте на увазі, що не всі апарати мобільних телефонів мають однакове за силою випромінювання: одні більше, другі – менше. Тому, купуючи телефон, обов'язково попросіть показати копію сертифіката на обрану вами модель, де буде зазначено, що вона відповідає вимогам стандарту *FCC*. А це означає, що не перевищує і так високі показники випромінювання.

Обираючи оператора зв'язку, віддайте перевагу тому, який має най розгалуженішу мережу ретрансляторів. Бо що більше телефон напружується в пошуках базової станції (щоб здійснити зв'язок), то більшою стає випромінювана ним доза електромагнітних хвиль.

Якщо на екрані вашого смартфону кількість «антенок» зменшилася, це означає що ви потрапили в зону слабкої дії сигналу. Таке трапляється в приміщеннях вокзалів, аеропортів, метро, в підвалах. Намагайтеся уникати користування мобільним телефоном у таких умовах, бо інтенсивність його електромагнітного випромінювання збільшується в кілька разів.

5.3 Техніка безпеки при використанні системи керування замовленнями послуг на ПК та ноутбуці

Загальні правила:

1. Дотримуйтесь інструкцій з експлуатації системи керування замовленнями послуг – перед використанням системи керування замовленнями послуг уважно прочитайте інструкцію з експлуатації та чітко дотримуйтесь її вказівок.

2. Використовуйте лише ліцензійне програмне забезпечення: Не використовуйте піратське програмне забезпечення, оскільки воно може містити шкідливий код, який може пошкодити ваш комп'ютер або дані.

3. Регулярно оновлюйте програмне забезпечення системи керування замовленнями послуг. Оновлення програмного забезпечення системи керування

замовленнями послуг випускаються для усунення помилок та вразливостей безпеки. Регулярно встановлюйте всі доступні оновлення.

4. Використовуйте надійний пароль.

Техніка безпеки при роботі з системою керування замовленнями послуг на ПК:

1. Не залишайте ПК без нагляду. Якщо ви відходите від ПК, на якому запущено систему керування замовленнями послуг, вийдіть з системи або заблокуйте комп’ютер.

2. Не використовуйте ПК в небезпечних умовах. Не використовуйте ПК в місцях з підвищеною вологістю, пилом або вібрацією.

3. Не допускайте попадання рідини на ПК. Не пийте і не їжте над ПК, щоб уникнути попадання рідини на клавіатуру або інші компоненти.

4. Не перевантажуйте ПК. Не запускайте одночасно багато програм, щоб не перевантажувати процесор та пам'ять ПК.

Техніка безпеки при роботі з системою керування замовленнями послуг на ноутбуці:

1. Використовуйте ноутбук на твердій, стійкій поверхні. Не використовуйте ноутбук на м'якій поверхні, наприклад, на ліжку або на подушці, оскільки це може призвести до перегріву.

2. Не закривайте вентиляційні отвори ноутбука, щоб не перешкоджати циркуляції повітря.

3. Не використовуйте ноутбук на сонці. Не використовуйте ноутбук на прямому сонячному світлі, оскільки це може призвести до перегріву та пошкодження компонентів.

4. Не залишайте ноутбук у зарядженому стані на тривалий час. Не залишайте ноутбук у зарядженому стані на тривалий час, якщо ви ним не користуєтесь.

5. Використовуйте оригінальний блок живлення. Використовуйте лише оригінальний блок живлення, який йде в комплекті з ноутбуком.

Висновок до п’ятого розділу

Розробка та впровадження системи керування замовленнями послуг (СКЗП) може значно полегшити процес замовлення послуг для користувачів, а також підвищити ефективність роботи для постачальників послуг. При розробці СКЗП важливо враховувати потреби користувачів, забезпечуючи зручність та відповідність їхнім потребам. Система повинна мати всі необхідні функції для замовлення та відстеження послуг, бути безпечною та захищати дані користувачів. Ергономіка також відіграє важливу роль: система має бути розроблена таким чином, щоб не шкодити здоров'ю користувачів.

Крім того, дотримання техніки безпеки є необхідним при роботі з СКЗП, що допоможе уникнути травм, пошкодження обладнання та витоку даних. Постійне вдосконалення СКЗП забезпечить її відповідність актуальним потребам користувачів та ринку. Дотримання вищезазначених принципів дозволить створити систему, корисну як для користувачів, так і для постачальників послуг.

ЗАГАЛЬНІ ВИСНОВКИ

Розробка та впровадження системи керування замовленнями послуг (СКЗП) виявилися успішними та перспективними з економічної та функціональної точок зору, сприяючи покращенню якості обслуговування та оптимізації бізнес-процесів. Основні висновки такі:

1. Результати економічних розрахунків підтвердили ефективність та економічну вигідність використання розробленої системи керування замовленнями послуг. Капітальні витрати на впровадження системи виявилися значно меншими за очікувані економічні вигоди, що свідчить про швидку окупність і вигідність інвестицій.

2. В процесі розробки СКЗП були враховані потреби користувачів та постачальників послуг, що сприяло створенню зручного та ефективного інструменту для замовлення та відстеження послуг. Ергономіка системи була вдосконалена для забезпечення комфортного користувацького досвіду.

3. Застосування техніки безпеки та дотримання правил техніки безпеки при роботі з СКЗП гарантує захист даних користувачів та уникнення травм або пошкоджень обладнання.

4. Постійне вдосконалення СКЗП забезпечить його відповідність актуальним потребам ринку та користувачів, що є ключовим чинником для успішної експлуатації та конкурентоздатності системи у майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України "Про електронну комерцію". URL: https://zakon.rada.gov.ua/laws/show/675-19 (дата звернення: 20.03.2024).
2. Закон України "Про захист персональних даних". URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 20.03.2024).
3. ДСТУ ISO/IEC/IEEE 16326:2015 Розроблення систем та програмного забезпечення. Процеси життєвого циклу. Керування проектами. URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=67052 (дата звернення: 20.03.2024).
4. ДСТУ ISO/IEC 27001:2014 "Системи менеджменту інформаційної безпеки – Вимоги". URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=66910 (дата звернення: 20.03.2024).
5. Аврамов С.А., Аврамова О.В. Мобільні додатки для бізнесу: можливості та перспективи розвитку. Вісник Національного технічного університету "Київський політехнічний інститут" імені Ігоря Сікорського. Серія "Інформатика та комп'ютерна наука", 2022, № 83, с. 5-12.
6. Аникина Н.А., Аникин А.А. Розробка багатоплатформенної системи управління замовленнями такси. Доповіді Академії наук Республіки Татарстан, 2020, т. 62, № 2, с. 124-127.
7. Волкова О.В., Петров А.С. Розробка багатоплатформенної системи управління замовленнями у сфері послуг. Матеріали Міжнародної науково-практичної конференції "Інформаційні технології та системи", 2018, с. 134-137.
8. Гончаров А.Л., Петров А.Ю. Мобільні додатки для управління замовленнями: проблеми та перспективи розвитку. Вісник Хмельницького національного університету, 2019, № 11, с. 123-128.

9. Гусев В.А., Сидорова А.А. Розробка мобільного додатку для управління замовленнями на ремонт побутової техніки. Матеріали Міжнародної науково-технічної конференції "Інформаційні системи та технології", 2017, с. 145-148.
10. Дубцов А.В. Мобільні додатки для управління замовленнями: функціональні можливості та методи розробки. Вісник Національного університету "Львівська політехніка", 2018, № 837, с. 135-140.
11. Кравченко О.В., Головко О.М. React Native та Flutter: порівняльний аналіз та можливості використання для розробки багатоплатформенних систем управління замовленнями. Матеріали Міжнародної науково-практичної конференції "Інформаційні системи та технології", 2021, с. 135-140.
12. Петренко А.С., Сидоренко І.В. Мобільні додатки для управління замовленнями: сучасні тенденції та перспективи розвитку. Матеріали Міжнародної науково-практичної конференції "Інноваційні технології в освіті, науці та виробництві", 2022, с. 123-126.
13. Савченко О.В. Системи управління замовленнями: проектування, розробка, впровадження. - Х.: Фенікс, 2020.
14. Савченко О.В., Аврамов С.А. Мобільні додатки для управління замовленнями: проблеми впровадження та шляхи їх вирішення. Матеріали Міжнародної науково-практичної конференції "Інформаційні технології в освіті, науці та виробництві", 2020, с. 127-130.
15. Сидоренко І.В. Xamarin.Forms. Розробка кросплатформних мобільних додатків. - К.: НТУУ "КПІ" ім. Ігоря Сікорського, 2020.
16. "A Comprehensive Review of Service Order Management Systems: Challenges, Opportunities, and Future Directions" by Muhammad Umair, Muhammad Asif, and Muhammad Usman (2019). URL: https://ieeexplore.ieee.org/document/9768823 (дата звернення: 15.02.2024)
17. "A Survey of Service Order Management Systems" by Ahmed El Sayed, Mohamed Shawky, and Mohamed Abdel-Mottaleb (2021) URL: https://ieeexplore.ieee.org/document/9461196 (дата звернення: 01.05.2024)

18."An Empirical Study of the Factors Affecting the Adoption of Service Order Management Systems" by Xiaohong Liu, Xiangyu Li, and Yuhong Wang (2020) URL: https://www.researchgate.net/publication/333683897_An_empirical_study_of_the_factors_affecting_the_adoption_of_mobile_enterprise_applications (дата звернення: 18.04.2024)
19."How to Choose the Right Service Order Management System for Your Business" by Software Advice URL: https://www.softwareadvice.com/vendors/ (дата звернення: 20.04.2024)
20."Service Order Management: A Complete Guide for Beginners" by HubSpot URL: https://blog.hubspot.com/marketing/inventory-management (дата звернення: 01.05.2024)
21."Service Order Management: A Comprehensive Guide" by Salesforce URL: https://help.salesforce.com/s/articleView?id=sf.om_order_management.htm&language=en_US&type=5 (дата звернення: 12.03.2024)
22."The Americas Conference on Information Systems" (AMCIS) URL: https://aisel.aisnet.org/amcis/ (дата звернення: 19.03.2024)
23."The International Conference on Service Systems and Engineering" (SSSE) URL: https://www.sawe.org/events/conferences/ (дата звернення: 30.03.2024)
24."The Top 10 Service Order Management Software Solutions" by G2 URL: https://www.g2.com/ (дата звернення: 08.04.2024)
25."The Ultimate Guide to Service Order Management" by Zendesk URL: https://online-help.zendesk.com/hc/en-us/sections/360002078293-Order-management (дата звернення: 19.03.2024)

ДОДАТКИ

Додаток А Лістинг класів

Клас *AuthController:*

```
@ApiTags('Authorization')
@Controller('v1/auth')
export class AuthController {
  constructor(
    private readonly authService: AuthService,
    private usersService: UsersService
  ) { }

  @Get('account')
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard)
  @ApiOkResponse({ status: 200, type: BaseAccountOutputDto })
  async getAccount(@Request() request: ExRequest):
Promise<BaseAccountOutputDto> {
    const { id } = request.user;
    const user = await this.usersService.findById(id);

    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(UserOutputDto, user)
    };
  }

  @Patch('account')
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard)
  @ApiOkResponse({ status: 200, type: BaseAccountOutputDto })
  async updateAccount(@Request() request: ExRequest, @Body() body:
UserUpdateDto): Promise<BaseAccountOutputDto> {
    const { id } = request.user;
    const user = await this.usersService.findById(id);

    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(UserOutputDto, user)
    };
  }


  @Post('register')
  @ApiOkResponse({ status: 200, type: BaseRegisterOutputDto })
  async register(@Body() body: RegisterInputDto):
Promise<BaseRegisterOutputDto> {
    const user = await this.authService.register(body);

    return {
      status: ResponseStatus.SUCCESS,
```

```
      data: {
        email: user.email,
        message: 'Token sent to email!'
      }
    };
  }

  @Post('login')
  @ApiOkResponse({ status: 200, type: BaseLoginOutputDto })
  async login(@Body() body: LoginInputDto, @Res({ passthrough: true })
response: Response): Promise<BaseLoginOutputDto> {
    const { email, password } = body;

    const { accessToken, user, session } = await
this.authService.login(email, password);

    response.cookie('refreshToken', session.token, {
      domain: process.env.DOMAIN,
      path: '/',
      secure: true,
      expires: new Date(Date.now() + 2 * 24 * 60 * 60 * 1000),
      httpOnly: true,
      sameSite: 'none'
    });

    return {
      status: ResponseStatus.SUCCESS,
      data: {
        accessToken,
        user: plainToInstance(UserOutputDto, user)
      }
    };
  }

  @Post('logout')
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard)
  @ApiOkResponse({ status: 200, type: EmptyResponseDto })
  async logout(@Request() request: ExRequest, @Res({ passthrough: true })
response: Response): Promise<EmptyResponseDto> {
    const token = request.cookies.refreshToken;

    if (token) await this.authService.logout(token);

    response.clearCookie('refreshToken', {
      domain: process.env.DOMAIN,
      path: '/',
      secure: true,
      expires: new Date(Date.now()),
      httpOnly: true,
      sameSite: 'none'
    });

    return { status: ResponseStatus.SUCCESS };
  }

  @Post('setupPassword')
  @ApiOkResponse({ status: 200, type: BaseSetupPasswordOutputDto })
```

```
  async setupPassword(@Body() body: SetupPasswordInputDto, @Res({
passthrough: true }) response: Response):
Promise<BaseSetupPasswordOutputDto> {
    const { token, password } = body;

    const { accessToken, user, session } = await
this.authService.setupPassword(password, token);

    //DOTO: create Cookie service
    response.cookie('refreshToken', session.token, {
      domain: process.env.DOMAIN,
      path: '/',
      secure: true,
      expires: new Date(Date.now() + 2 * 24 * 60 * 60 * 1000),
      httpOnly: true,
      sameSite: 'none'
    });

    return {
      status: ResponseStatus.SUCCESS,
      data: {
        accessToken,
        user: plainToInstance(UserOutputDto, user)
      }
    };
  }

  @Post('forgotPassword')
  @ApiOkResponse({ status: 200, type: BaseForgotPasswordOutputDto })
  async forgotPassword(@Request() request: ExRequest, @Body() body:
ForgotPasswordInputDto): Promise<BaseForgotPasswordOutputDto> {
    const { email } = body;

    const parser = new UAParser(request.headers['user-agent']);
    const { browser: { name: browserName }, os: { name: operatingSystem }
} = parser.getResult();

    await this.authService.forgotPassword(email, browserName,
operatingSystem);

    return {
      status: ResponseStatus.SUCCESS,
      data: {
        email,
        message: 'Token sent to email!'
      }
    };
  }

  @Patch('updatePassword')
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard)
  @ApiOkResponse({ status: 200, type: EmptyResponseDto })
  async updatePassword(@Body() body: UpdatePasswordInputDto, @Request()
request: ExRequest): Promise<EmptyResponseDto> {
    const { password } = body;

    await this.authService.updatePassword(password, request.user.id);
```

```
    return { status: ResponseStatus.SUCCESS };
  }
}
```

Клас *CategoriesController:*

```
@ApiTags('Categories')
@Controller('v1/categories')
export class CategoriesController {
  constructor(private readonly categoriesService: CategoriesService) {}

  @Get()
  @ApiQuery({ name: 'limit', type: Number, required: false })
  @ApiQuery({ name: 'offset', type: Number, required: false })
  @ApiOkResponse({ type: PaginatedCategoryOutputDto })
  async findAll(@Query() queryParams: PaginatedRequestParamsDto):
Promise<PaginatedCategoryOutputDto> {
    const { limit, offset } = queryParams;

    const categories = await this.categoriesService.findAll(limit,
offset);
    const total = await this.categoriesService.count();

    return {
      status: ResponseStatus.SUCCESS,
      total: total,
      limit: limit,
      offset: offset,
      data: plainToInstance(CategoryOutputDto, categories)
    };
  }

  @Get(':id')
  @ApiOkResponse({ type: BaseCategoryOutputDto })
  async findOne(@Param('id') id: number): Promise<BaseCategoryOutputDto>
{
    const category = await this.categoriesService.findById(id);
    if (!category) {
      throw new NotFoundException('Cat not found');
    }
    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(CategoryOutputDto, category)
    };
  }
}
```

Клас *OrdersController:*

```
@ApiTags('Orders')
@Controller('v1/orders')
export class OrdersController {

  constructor(
    private readonly servicesService: ServicesService,
    private readonly paymentsService: PaymentsService,
    private readonly ordersService: OrdersService) { }

  @Post()
```

```
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard)
  @ApiOperation({ summary: 'Create new Order' })
  @ApiOkResponse({ status: 200, type: BaseOrderOutputDto })
  async invite(@Request() request: ExRequest, @Body() { serviceIds,
address }: OrderInputDto): Promise<BaseOrderOutputDto> {
    const { user } = request;
    const order = await this.ordersService.create({
      address,
      services: await this.servicesService.find({
        where: {
          id: In(serviceIds),
          active: true
        },
      }),
      customer: user
    });

    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(OrderOutputDto, order)
    };
  }

  @Get()
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard)
  @ApiOperation({ summary: 'Get All Orders' })
  @ApiOkResponse({ type: PaginatedOrderOutputDto })
  async findAll(@Request() request: ExRequest, @Query() queryParams:
PaginatedRequestParamsDto): Promise<PaginatedOrderOutputDto> {
    const { limit, offset } = queryParams;
    const { user } = request;
    const [orders, total] = await this.ordersService.findAll(queryParams,
user, false);

    return {
      status: ResponseStatus.SUCCESS,
      total,
      limit,
      offset,
      data: plainToInstance(OrderOutputDto, orders
        , { excludeExtraneousValues: true })
    };
  }

  @Get(':id')
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard)
  @ApiOperation({ summary: 'Get Order by Id' })
  @ApiOkResponse({ type: BaseOrderOutputDto })
  async findOne(@Request() request: ExRequest, @Param('id') id: number):
Promise<BaseOrderOutputDto> {
    const { user } = request;
    const order = await this.ordersService.findById(id, user, false);

    if (!order) throw new NotFoundException('Order not found');
    return {
```

```
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(OrderOutputDto, order)
    };
  }

  @Get(':externalId/info')
  @ApiOperation({ summary: 'Get Order by externalId' })
  @ApiOkResponse({ type: BasePayOrderOutputDto })
  async getOrderInfo(@Param('externalId') externalId: string):
Promise<BasePayOrderOutputDto> {
    const order = await this.ordersService.find({
      relations: ['customer', 'services', 'services.provider'],
      where: { externalId, status: OrderStatus.ACCEPTED }
    });

    if (!order) throw new NotFoundException('Order not found');
    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(OrderPayOutputDto, order)
    };
  }

  @Post(':externalId/pay')
  @ApiOperation({ summary: 'Payment for the Order' })
  @ApiOkResponse({ status: 200, type: BasePaymentOutputDto })
  async pay(
    @Body() { colorDepth, screenHeight, screenWidth, userAccept,
userAgent, cardNumber, cvv, exp, expYear }: OrderPayInputDto,
    @Param('externalId') externalId: string
  ): Promise<BasePaymentOutputDto> {
    const order = await this.ordersService.find({
      relations: ['customer', 'services'],
      where: {
        externalId
      }
    });

    if (!order) throw new NotFoundException('Order not found');
    if (order.status === OrderStatus.PAYED) throw new
BadRequestException('Order has been paid');

    const payment = await this.paymentsService.create({ order });
    const { data } = await this.paymentsService.pay({
      order,
      payment,
      card: {
        cardNumber, cvv, exp, expYear
      },
      threeDs: {
        colorDepth,
        screenHeight,
        screenWidth,
        userAccept,
        userAgent
      }
    });

    if (data.status === 'success') {
```

```
      order.status = OrderStatus.PAYED;
      payment.status = PaymentStatus.COMPLETED;
      await order.save();
      await payment.save();
    }

    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(PaymentData, data)
    };
  }

}
```

Клас *CrmProvidersController*:

```
@ApiTags('CRM Providers')
@Controller('v1/crm/providers')
export class CrmProvidersController {
  // eslint-disable-next-line no-unused-vars
  constructor(private readonly providersService: ProvidersService) {}

  @Post()
  @Roles([UserRole.ADMIN])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOkResponse({ status: 201, type: BaseProviderOutputDto })
  async create(@Body() providerInputDto: ProviderInputDto):
Promise<BaseProviderOutputDto> {
    const provider = await
this.providersService.create(providerInputDto);
    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(ProviderOutputDto, provider)
    };
  }

  @Get()
  @Roles([UserRole.ADMIN])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiQuery({ name: 'limit', type: Number, required: false })
  @ApiQuery({ name: 'offset', type: Number, required: false })
  @ApiOkResponse({ type: PaginatedProviderOutputDto })
  async findAll(@Query() queryParams: PaginatedRequestParamsDto):
Promise<PaginatedProviderOutputDto> {
    const { limit, offset } = queryParams;

    const providers = await this.providersService.findAll(limit, offset);
    const total = await this.providersService.count();

    return {
      status: ResponseStatus.SUCCESS,
      total: total,
      limit: limit,
      offset: offset,
      data: plainToInstance(ProviderOutputDto, providers)
    };
  }
```

```
  @Get(':id')
  @Roles([UserRole.ADMIN])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOkResponse({ type: BaseProviderOutputDto })
  async findOne(@Param('id') id: number): Promise<BaseProviderOutputDto>
{
    const category = await this.providersService.findById(id);
    if (!category) {
      throw new NotFoundException('Cat not found');
    }
    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(ProviderOutputDto, category)
    };
  }

  @Patch(':id')
  @Roles([UserRole.ADMIN])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOkResponse({ status: 200, type: BaseProviderOutputDto })
  async update(@Param('id') id: number, @Body() providerUpdateDto:
ProviderUpdateDto): Promise<BaseProviderOutputDto> {
    const category = await this.providersService.update(id,
providerUpdateDto);
    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(ProviderOutputDto, category)
    };
  }

  @Delete(':id')
  @Roles([UserRole.ADMIN])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOkResponse({ status: 200, type: EmptyResponseDto })
  async delete(@Param('id') id: number): Promise<EmptyResponseDto> {
    const deletedRows = await this.providersService.delete(id);
    if (!deletedRows) {
      throw new NotFoundException('Category not found');
    }
    return {
      status: ResponseStatus.SUCCESS
    };
  }
}
```

Клас *CrmServicesController:*

```
@ApiTags('CRM Services')
@Controller('v1/crm/services')
export class CrmServicesController {
  constructor(private readonly servicesService: ServicesService) { }

  @Post()
  @Roles([UserRole.ADMIN, UserRole.PROVIDER])
  @ApiBearerAuth('Bearer JWT Token')
```

```
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOperation({ summary: 'Create Service' })
  @ApiOkResponse({ status: 201, type: BaseServiceOutputDto })
  async create(@Request() request: ExRequest, @Body() serviceInputDto:
ServiceInputDto): Promise<BaseServiceOutputDto> {
    const { user } = request;
    const service = await this.servicesService.create(serviceInputDto,
user);
    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(ServiceOutputDto, service)
    };
  }

  @Get()
  @Roles([UserRole.ADMIN, UserRole.PROVIDER])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOperation({ summary: 'Get All Services' })
  @ApiOkResponse({ type: PaginatedServiceOutputDto })
  async findAll(@Request() request: ExRequest, @Query() queryParams:
PaginatedRequestParamsDto): Promise<PaginatedServiceOutputDto> {
    const { limit, offset } = queryParams;
    const { user } = request;
    const [services, total] = await
this.servicesService.findAll(queryParams, user, true);
    return {
      status: ResponseStatus.SUCCESS,
      total,
      limit,
      offset,
      data: plainToInstance(ServiceOutputDto, services)
    };
  }

  @Get(':id')
  @Roles([UserRole.ADMIN, UserRole.PROVIDER])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOperation({ summary: 'Get Service by Id' })
  @ApiOkResponse({ type: BaseServiceOutputDto })
  async findOne(@Request() request: ExRequest, @Param('id') id: number):
Promise<BaseServiceOutputDto> {
    const { user } = request;
    const service = await this.servicesService.findById(id, user, true);
    if (!service) throw new NotFoundException('Service not found');
    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(ServiceOutputDto, service)
    };
  }

  @Patch(':id')
  @Roles([UserRole.ADMIN, UserRole.PROVIDER])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOperation({ summary: 'Update Service' })
  @ApiOkResponse({ status: 200, type: BaseServiceOutputDto })
```

```
  async update(@Param('id') id: number, @Body() providerUpdateDto:
ServiceUpdateDto): Promise<BaseServiceOutputDto> {
    const category = await this.servicesService.update(id,
providerUpdateDto);
    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(ServiceOutputDto, category)
    };
  }

  @Delete(':id')
  @Roles([UserRole.ADMIN, UserRole.PROVIDER])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOperation({ summary: 'Delete Service' })
  @ApiOkResponse({ status: 200, type: EmptyResponseDto })
  async delete(@Param('id') id: number): Promise<EmptyResponseDto> {
    const deletedRows = await this.servicesService.delete(id);
    if (!deletedRows) {
      throw new NotFoundException('Category not found');
    }
    return {
      status: ResponseStatus.SUCCESS
    };
  }
}
```

Клас *CrmUsersController:*

```
@ApiTags('CRM Users')
@Controller('v1/crm/users')
export class CrmUsersController {
  // eslint-disable-next-line no-unused-vars
  constructor(private readonly userService: UsersService) { }

  @Get()
  @Roles([UserRole.ADMIN])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOkResponse({ type: PaginatedUserOutputDto })
  async findAll(@Query() queryParams: PaginatedRequestParamsDto):
Promise<PaginatedUserOutputDto> {
    const { limit, offset } = queryParams;

    const users = await this.userService.findAll(queryParams);
    const total = await this.userService.count();
    return {
      status: ResponseStatus.SUCCESS,
      total: total,
      limit: limit ? parseInt(`${limit}`) : undefined,
      offset: offset ? parseInt(`${offset}`) : undefined,
      data: plainToInstance(UserOutputDto, users)
    };
  }

  @Get(':id')
  @Roles([UserRole.ADMIN])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
```

```
  @ApiOkResponse({ status: 200, type: BaseUserOutputDto })
  async findOne(@Param('id') id: number): Promise<BaseUserOutputDto> {
    const user = await this.userService.findById(id);
    if (!user) {
      throw new NotFoundException('User not found');
    }
    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(UserOutputDto, user)
    };
  }

  @Patch(':id')
  @Roles([UserRole.ADMIN])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOkResponse({ status: 200, type: BaseUserOutputDto })
  async update(@Param('id') id: number, @Body() updateUserDto:
UserUpdateDto): Promise<BaseUserOutputDto> {
    const user = await this.userService.update(id, updateUserDto);
    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(UserOutputDto, user)
    };
  }

  @Delete(':id')
  @Roles([UserRole.ADMIN])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOkResponse({ status: 200, type: EmptyResponseDto })
  async delete(@Param('id') id: number): Promise<EmptyResponseDto> {
    const deletedRows = await this.userService.delete(id);
    if (!deletedRows) {
      throw new NotFoundException('User not found');
    }

    return {
      status: ResponseStatus.SUCCESS
    };
  }

  @Post('invite')
  @Roles([UserRole.ADMIN])
  @ApiBearerAuth('Bearer JWT Token')
  @UseGuards(AuthGuard, RolesGuard)
  @ApiOkResponse({ status: 200, type: BaseUserOutputDto })
  async invite(@Body() body: UserInputDto): Promise<BaseUserOutputDto> {
    const user = await this.userService.invite(body);

    return {
      status: ResponseStatus.SUCCESS,
      data: plainToInstance(UserOutputDto, user)
    };
  }
}
```

Додаток Б Графічний матеріал

Рис. Б.1 – Тема роботи

Рис. Б.2 – Об’єкт та предмет дослідження

Слайд 3

Актуальність

Актуальність тематики дослідження полягає в тому, що сучасний світ нерозривно пов'язаний з інформаційними технологіями, які стають ключовим фактором у розвитку різних сфер діяльності

Існуючі проблеми

- Управління замовленнями у великих компаніях. Збільшення обсягів та різноманітності послуг ускладнює управління замовленнями, що може призвести до затримок та невдоволення клієнтів.
- Нестабільність попиту. Непередбачуваність ринкових умов робить управління замовленнями складним через необхідність швидкої адаптації до змін.
- Якість обслуговування. Високі вимоги клієнтів до швидкості, зручності та персоналізації вимагають ефективного управління замовленнями для підтримки задоволеності та лояльності.
- Координація з партнерами. Співпраця з великою кількістю партнерів може призвести до затримок і помилок у виконанні замовлень.
- Безпека даних. Зростання цифрових даних вимагає посиленої уваги до захисту інформації від несанкціонованого доступу та кібератак.

Рис. Б.3 – Актуальність

Слайд 4

Цілі проекту

Розробити систему, яка забезпечувала би:

- Мультіплатформеність. Забезпечення роботи системи на різних пристроях, включаючи настільні комп'ютери та мобільні телефони.
- Гнучку адаптацію до потреб користувачів. Система повинна швидко адаптуватися до змінюваних вимог користувачів та ринкових умов.
- Високу безпеку. Використання передових технологій для захисту даних від несанкціонованого доступу.
- Інтеграція з іншими системами та сервісами. Налагодження взаємодії з CRM, фінансовими платформами та системами електронної комерції.
- Інтуїтивно зрозумілий користувацький інтерфейс. Створення зручного і доступного інтерфейсу відповідно до сучасних стандартів дизайну.

Рис. Б.4 – Цілі проекту

Слайд 5

Проектування

Ключові аспекти архітектури:

1. Модульність

- Система побудована за модульним принципом, де кожен компонент відповідає за конкретну функцію.
- Це забезпечує незалежне оновлення та масштабування модулів, спрощуючи впровадження нових технологій і сервісів.

2. Централізоване управління даними

- Незважаючи на розподілену природу мікросервісів, дані управляються централізовано для забезпечення їхньої консистентності.

3. Безпека

- Архітектура включає комплексні механізми захисту даних, такі як шифрування, аутентифікація, авторизація користувачів та захист від зовнішніх атак.

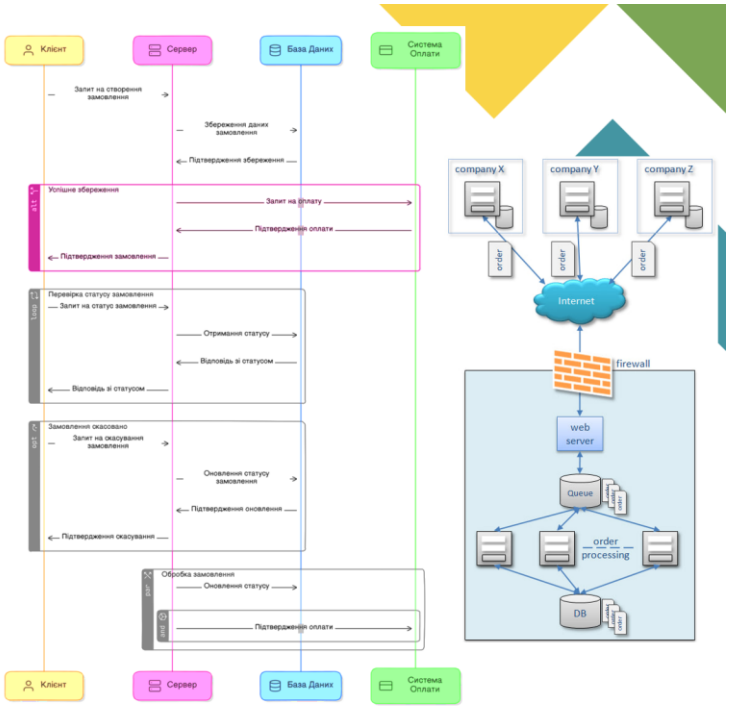


Рис. Б.5 – Проектування

Слайд 6

Обґрунтування програмних засобів

1. TypeScript та React для фронтенд-розробки.
   - Швидкість розробки та надійність
   - Популярність та ефективність
   - Комбінація з Nest.JS
2. Nest.JS для бекенд-розробки.
   - Строга типізація
   - Модульність
3. PostgreSQL для управління даними.
   - Масштабованість та висока продуктивність
   - Розширена транзакційна підтримка
   - Відкритий код та гнучкість
   - Активна спільнота та багата документація

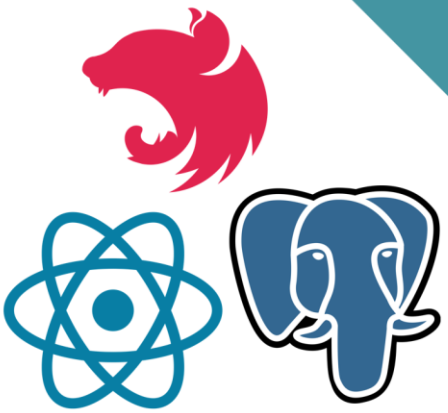


Рис. Б.6 – Обґрунтування програмних засобів

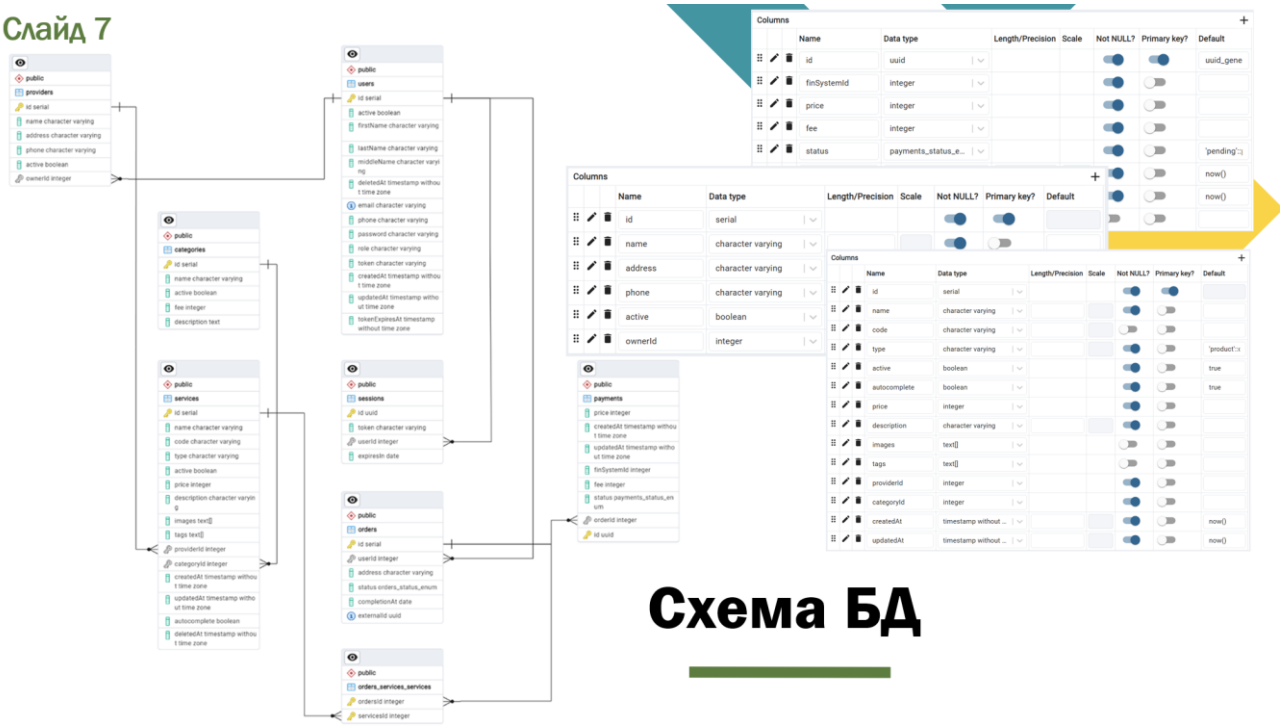


Рис. Б.7 – Схема БД

Рис. Б.8 – Реалізація доступу до даних

Слайд 9

Користувацький інтерфейс

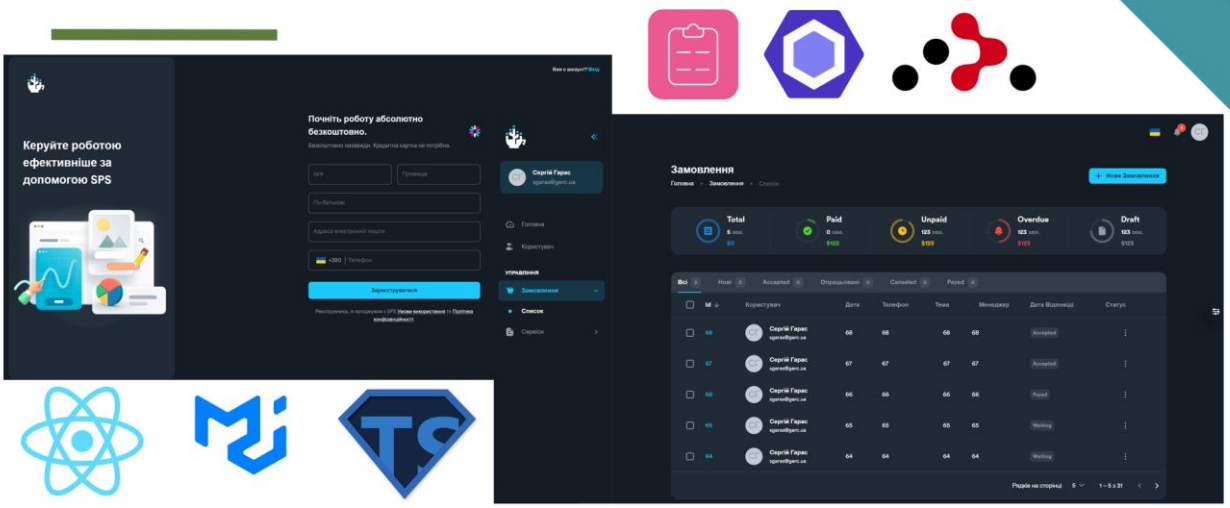


Рис. Б.9 – Користувацький інтерфейс

Слайд 10

Розгортання функціональних одиниць

GitHub Actions дозволило автоматизувати процеси *CI/CD*, що сприяло швидкій ітерації та випуску оновлень системи

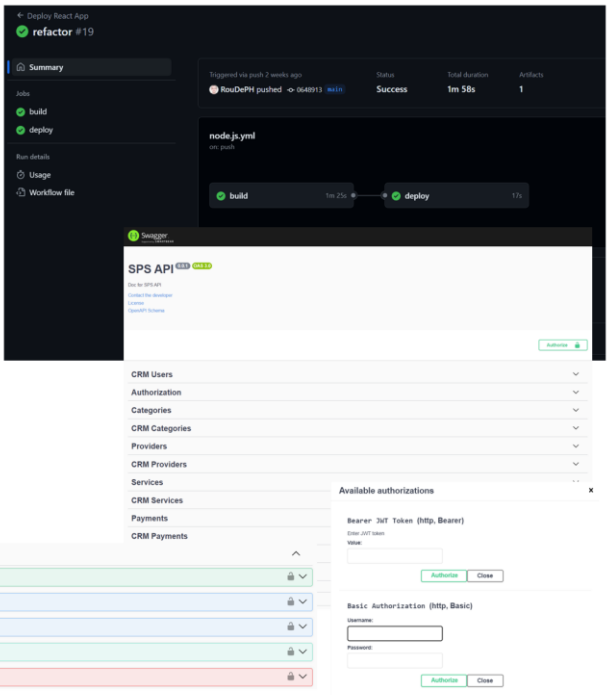


Рис. Б.10 – Розгортання функціональних одиниць

Слайд 11

Техніко-економічні показники проекту

Показники	Одиниці виміру	Значення
Капітальні витрати	грн	55 988,70
Ціна ПП	грн	34 338, 00
Економічний ефект	грн/рік	98 410,33
Термін окупності	рік	0,50
Економічна ефективність		2,01

Рис. Б.11 – Техніко-економічні показники проекту

Слайд 12

Загальні висновки

- Економічні розрахунки підтвердили, що **впровадження системи керування замовленнями послуг є вигідним**, з окупністю інвестицій за дуже **короткий термін**
- Система керування замовленнями послуг розроблена з урахуванням **потреб користувачів та постачальників**, забезпечуючи **зручний інтерфейс** та **високу функціональність**
- Застосування передових технік безпеки гарантує **захист даних** та **безпечне використання системи** в умовах **реальної** експлуатації

Рис. Б.12 – Загальні висновки

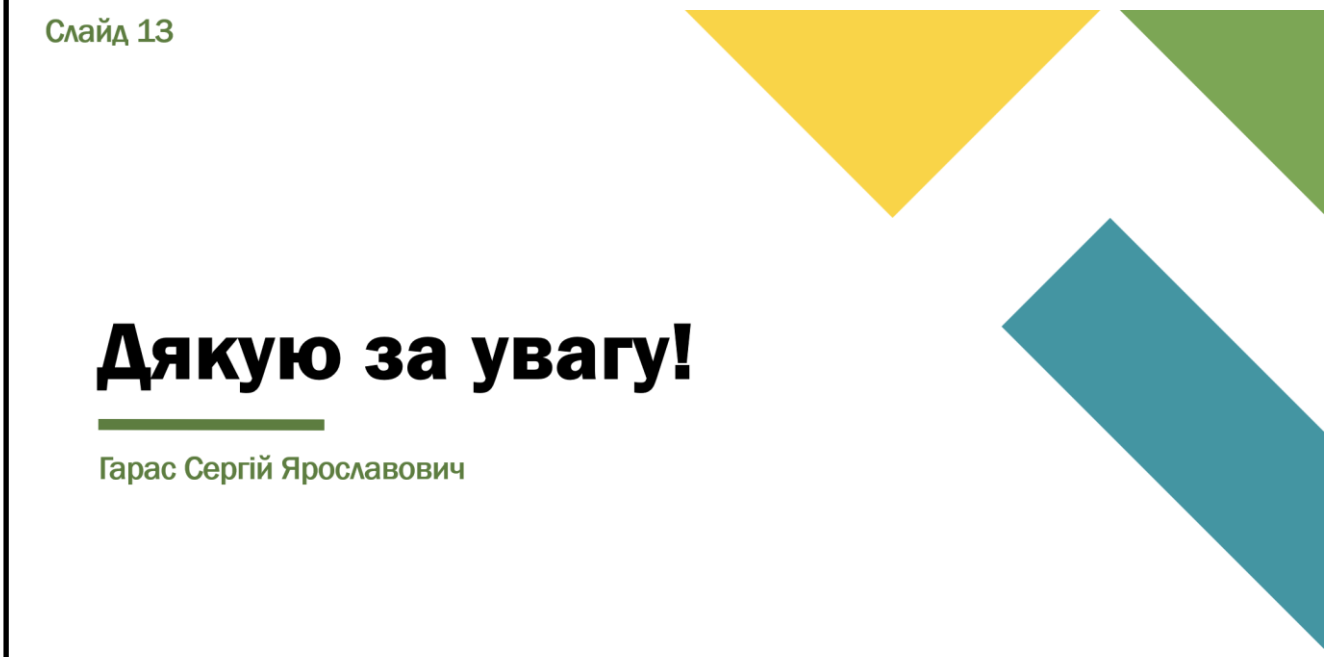


Рис. Б.13 – Дякую за увагу