

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

**Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці**

**ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА «КОМП'ЮТЕРНІ НАУКИ»
галузь знань 12 «Інформаційні технології»
спеціальність 122 «Комп'ютерні науки»**

Форма навчання: денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

**на тему: «Галузева торгівельна інформаційна система типу B2B з автоматизації
ІТ-послуг»**

здобувачки Зименок Анастасії Володимирівни

Науковий керівник:

К.Т.Н., доцент

_____ Маркіна О. М.

**Робота допущена до захисту перед
екзаменаційною комісією з
атестації здобувачів вищої освіти
завідувач кафедри:**

К.Е.Н., доцент

_____ Тішков Б.О.

Міністерство освіти і науки України
Київський національний економічний університет імені Вадима Гетьмана
Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА «КОМП'ЮТЕРНІ НАУКИ»

галузь знань 12 «Інформаційні технології»

спеціальність 122 «Комп'ютерні науки»

ПОГОДЖЕНО:

Керівник проектної групи (гарант) освітньо-професійної програми

_____ Помазун О.М.

“_____” _____ 2025 р.

ЗАТВЕРДЖЕНО:

Завідувач кафедри інформаційних систем в економіці

_____ Тішков Б.О.

“_____” _____ 2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувачки вищої освіти **Зименок Анастасії Володимирівни** *денної форми навчання*

на підготовку кваліфікаційної бакалаврської роботи
на тему: «Галузева торгівельна інформаційна система типу B2B з
автоматизації ІТ-послуг»

Тему затверджено наказом ректора Університету від «7» березня 2025р. № 466- ст.

Кваліфікаційна бакалаврська робота виконується на матеріалах отриманих під час навчання, здійснення студентських наукових досліджень, а також зібраного фактичного матеріалу з обраної теми дослідження під час практики

План кваліфікаційної бакалаврської роботи

Розділ I Характеристика та аналіз предметної галузі.

Розділ II Розробка вимог і моделювання галузевої торгівельної інформаційної системи типу B2B з автоматизації ІТ-послуг.

Розділ III Проектування та реалізація галузевої торгівельної інформаційної системи типу B2B з автоматизації ІТ-послуг.

Об'єкт дослідження: процеси автоматизації торгівлі ІТ-послугами в B2B-середовищі.

Предмет дослідження: методи та технології розробки інформаційної системи для автоматизації процесів купівлі-продажу ІТ-послуг у B2B-секторі.

Мета кваліфікаційного бакалаврського проекту: розробка галузевої торгівельної інформаційної системи типу B2B з автоматизації ІТ-послуг з використанням сучасних архітектурних рішень.

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

У розділі I АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ДОСЛІДЖЕННЯ ІСНУЮЧИХ

РІШЕНЬ виконати наступні завдання:

1.1. Аналіз предметної галузі

1.1.1 Визначити та описати предметну галузь, в якій буде використовуватися розроблювана система (B2B-торгівля ІТ-послугами).

1.1.2 Описати ключові процеси взаємодії між учасниками ринку ІТ-послуг (замовниками та постачальниками).

1.1.3 Визначити основні задачі та виклики, що існують у даній сфері (автоматизація вибору, замовлення та виконання ІТ-послуг).

1.1.4 Ідентифікувати потенційних користувачів (ІТ-компанії, фрілансери, замовники ІТ-послуг) та визначити їхні потреби та вимоги до системи.

1.2. Огляд існуючих рішень та наукових джерел

1.2.1 Провести аналіз наукових робіт, статей та інших джерел, що стосуються B2B-торгівлі та автоматизації ІТ-послуг.

1.2.2 Дослідити та проаналізувати існуючі програмні рішення в цій сфері (онлайн-біржі, маркетплейси ІТ-послуг).

1.2.3 Виявити переваги та недоліки існуючих рішень, визначити їхні обмеження та можливості для покращення.

У розділі II ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ виконати завдання

2.1. Вимоги до системи

2.1.1 Сформулювати функціональні вимоги (автоматизований пошук виконавців, укладання угод, управління замовленнями, виставлення рахунків тощо).

2.1.2 Сформулювати нефункціональні вимоги (продуктивність, безпека, масштабованість, інтеграція з платіжними системами тощо).

2.2. Основні задачі та алгоритми

2.2.1 Описати основні задачі, які має вирішувати система (автоматизоване управління ІТ-замовленнями, інтеграція з CRM-системами тощо).

2.2.2 Розробити алгоритми для реалізації цих задач (наприклад, алгоритм рейтингового відбору виконавців, автоматичне ціноутворення).

2.3. Діаграми та архітектура системи

2.3.1 Побудувати діаграми для візуалізації структури системи, її компонентів та взаємодій між ними.

2.3.2 Представити UML-діаграми (діаграми випадків використання, діаграми класів, діаграми послідовностей тощо).

2.3.3 Визначити загальну архітектуру системи та взаємозв'язок її компонентів.

У розділі III Реалізація системи та її тестування:

3.1. Опис даних та управління ними

3.1.1 Визначити набір даних, що будуть використовуватись у системі (замовлення, профілі користувачів, фінансові транзакції).

3.1.2 Описати джерела цих даних, способи збору та управління ними.

3.1.3 Продумати механізми захисту конфіденційної інформації.

3.2. Апаратні вимоги

3.2.1 Описати мінімальні та рекомендовані апаратні вимоги для запуску системи.

3.2.2 Визначити хмарні або локальні серверні рішення для розгортання системи.

3.3. Технологічний стек та інтерфейси

3.3.1 Описати вибрані технології (мови програмування, фреймворки, бази даних).

3.3.2 Охарактеризувати структуру API та інтерфейсів взаємодії між модулями.

3.4. Реалізація повноцінної інформаційної системи

3.4.1 Описати процес розробки повноцінної інформаційної системи.

3.4.2 Реалізувати всі необхідні функціональні модулі (реєстрація, створення замовлення, вибір виконавця, обробка платежів тощо).

3.4.3 Провести інтеграцію з зовнішніми сервісами (CRM-системи, платіжні шлюзи, API для автоматизації бізнес-процесів).

3.4.4 Оптимізувати продуктивність та безпеку системи.

3.5. Тестування та оцінка якості системи

3.5.1 Провести тестування основних сценаріїв використання.

3.5.2 Здійснити перевірку безпеки (аудит коду, тестування на уразливості).

Завдання підготував

науковий керівник

Завдання одержав

здобувач

Маркіна Ольга Миколаївна

“10” березня 2025 р.

Зименок Анастасія

Володимирівна

“10” березня 2025 р.

АНОТАЦІЯ

кваліфікаційної бакалаврської роботи
здобувачки першого (бакалаврського) рівня вищої освіти, 4 курсу,
Зименок Анастасії Володимирівни,
виконаної на тему: **Галузева торгівельна інформаційна система типу B2B з
автоматизації IT-послуг**
Київ: кафедра інформаційних систем в економіці, 2025р.

У кваліфікаційній бакалаврській роботі розглянуто процес розробки галузевої торговельної інформаційної системи типу B2B, орієнтованої на автоматизацію надання IT-послуг. Актуальність теми зумовлена зростанням потреби в ефективній взаємодії між компаніями у сфері цифрових послуг, що вимагає прозорості, швидкості та безпеки процесів.

У рамках роботи проаналізовано особливості предметної області — взаємодію замовників і виконавців IT-послуг, визначено основні вимоги до функціональності системи, побудовано алгоритми для автоматизованого підбору підрядників, формування замовлень, оцінювання якості виконання, інтеграції з платіжними сервісами та CRM-платформами. Застосовано сучасні методи структурного й процесного моделювання (UML, OPM), а також обґрунтовано вибір технологічного стеку: React Native, Node.js (Express), PostgreSQL.

Результатом дослідження стало архітектурне проектування системи з поділом на клієнтський, серверний та інтеграційний рівні. Встановлено вимоги до продуктивності, масштабованості та захисту даних, що забезпечують надійність у реальному середовищі експлуатації.

Отримані результати можуть бути використані для впровадження адаптивної B2B-платформи у компаніях, що спеціалізуються на IT-послугах, а також як основа для подальшого наукового або комерційного розвитку систем автоматизації в цій сфері.

РЕФЕРАТ

Кваліфікаційна бакалаврська робота 91 сторінки, 5 таблиць, 34 рисунків, список літератури з 27 найменувань, 7 додатків.

Галузева торгівельна інформаційна система типу B2B з автоматизації IT-послуг

Об'єктом дослідження виступає галузева інформаційна система для управління заявками, проектами та комунікацією між замовниками та виконавцями IT-послуг.

Предметом дослідження є процеси автоматизації взаємодії між компаніями у сфері IT-послуг за допомогою інформаційних систем типу B2B. Мета кваліфікаційної бакалаврської роботи полягає в розробці програмного забезпечення — галузевої інформаційної системи типу B2B, що автоматизує ключові бізнес-процеси між компаніями у сфері надання IT-послуг.

Завданнями кваліфікаційної бакалаврської роботи є наліз предметної області автоматизації IT-послуг у B2B-сегменті, постановка функціональних і нефункціональних вимог до системи, проектування архітектури інформаційної системи, розробка серверної частини з використанням Node.js, створення REST API, забезпечення авторизації та контролю доступу, підготовка до інтеграції з CRM і платіжними шлюзами, тестування функціональності. Апаратні та програмні засоби, що використовувались на при проектуванні: ноутбук із процесором не нижче Intel Core i5, 8 ГБ ОЗУ, ОС Windows 10, Apple MacBook Pro, середовище розробки Visual Studio Code, Node.js (Express), TypeScript, PostgreSQL, React Native, Git, Postman, Swagger.

Результати досягнуті в процесі роботи: реалізовано архітектуру серверної частини B2B-системи, виконано розробку бізнес-логіки (оголошення, проекти, виконавці), реалізовано автентифікацію та захист API, підготовлено можливість інтеграції зі сторонніми сервісами.

Одержані результати можуть бути використані для запуску MVP-платформи для IT-компаній, у навчальному процесі, для автоматизації внутрішніх бізнес-процесів компаній, у практичній діяльності з метою підвищення ефективності проєктного менеджменту. Рік виконання кваліфікаційної бакалаврської роботи 2025.Рік захисту кваліфікаційної бакалаврської роботи 2025.

Перелік ключових слів: інформаційна система, автоматизація, B2B-платформа, IT-послуги, бізнес-процеси, веб-розробка, Node.js, Express, PostgreSQL, REST API, CRM-система, платіжний шлюз, інтеграція, мобільний застосунок, React Native.

Відгук
про кваліфікаційну бакалаврську роботу
здобувача навчально-наукового інституту
«Інститут інформаційних технологій в економіці»
освітньо-професійної програми
«Комп'ютерні науки»

Зименок Анастасії Володимирівни
на тему «Галузева торгівельна інформаційна система типу B2B з автоматизації
ІТ-послуг»

Актуальність теми. У сучасному цифровому бізнес-середовищі автоматизація B2B-процесів має вирішальне значення для підвищення ефективності взаємодії між компаніями. Розробка галузевої інформаційної системи, яка враховує специфіку ІТ-послуг, є надзвичайно актуальною та перспективною, оскільки дозволяє оптимізувати бізнес-процеси, покращити обслуговування клієнтів та забезпечити конкурентні переваги.

Позитивні риси кваліфікаційної бакалаврської роботи. Робота має чітку структуру, логічну послідовність викладення матеріалу, вмотивований вибір технологій і рішень. У роботі враховано специфіку B2B-ринку, виконано системний аналіз предметної області, реалізовано важливі функціональні модулі (реєстрація компаній, каталог послуг, заявки на послуги, автоматизоване формування договорів тощо). Графічний інтерфейс зручний і демонструє гарний рівень опрацювання UI/UX.

Наявність самостійних розробок автора. У роботі реалізовано унікальну структуру системи з урахуванням ролей користувачів, власноруч спроектовано базу даних, створено адміністрування послуг та налаштування шаблонів договорів. Архітектура системи та логіка бізнес-процесів свідчать про високий рівень самостійної роботи здобувачки.

Цінність теоретичних висновків та практичних рекомендацій. Теоретичні положення узагальнюють принципи побудови інформаційних B2B-систем, підкріплені прикладами з практики. Практичні рекомендації охоплюють питання масштабування, захисту даних, подальшого розвитку сервісів. Розробка має потенціал для використання в реальному комерційному середовищі або в якості прототипу стартапу.

Наявність недоліків. Можна детальніше розглянути питання резервного копіювання даних і обробки помилок при збої сервера. Але це не знижує якість виконаної роботи.

Загальна оцінка кваліфікаційної бакалаврської роботи та її допущення до захисту перед ЕК. Кваліфікаційна бакалаврська робота Зименок Анастасії Олександрівни виконана на високому рівні, відповідає вимогам освітньої програми та демонструє здатність здобувача до комплексного вирішення прикладних завдань у сфері комп'ютерних наук.

РЕЦЕНЗІЯ

на кваліфікаційну бакалаврську роботу
здобувача вищої освіти

Зименок Анастасії Володимирівни

Тема «Галузева торгівельна інформаційна система типу B2B з
автоматизації IT-послуг»

Актуальність теми кваліфікаційної бакалаврської роботи і доцільність її розроблення. Тема роботи є актуальною в умовах зростаючого попиту на автоматизацію бізнес-процесів у сфері IT-послуг. Створення спеціалізованої інформаційної системи типу B2B дозволяє підвищити ефективність взаємодії між компаніями, оптимізувати управління замовленнями, контрактами та комунікаціями. Робота має практичну спрямованість і орієнтована на вирішення конкретних задач сучасного ринку.

Якість проведеного дослідження. У роботі простежується системний підхід до аналізу предметної області. Авторка провела детальний аналіз потреб цільової аудиторії, розробила архітектуру системи, базу даних та функціональні модулі. Застосовано сучасні інструменти програмування, продемонстровано вміння інтегрувати окремі компоненти у цілісне програмне рішення. Дослідження є завершеним і добре аргументованим.

Позитивні риси кваліфікаційної бакалаврської роботи. Повнота і логічна структура викладеного матеріалу.

- Високий рівень самостійності виконання технічної частини.
- Гарно структурована база даних із урахуванням бізнес-ролей.
- Реалізовано реєстрацію користувачів, облік послуг, генерацію договорів.
- Вдало продумана логіка взаємодії між компонентами системи.
- Актуальність і практична спрямованість результатів.

Зауваження:

- Опис процесів безпеки, зокрема захисту даних користувачів, міг би бути розширеним.
- Інтерфейс системи можна було б доповнити додатковими елементами зворотного зв'язку для користувачів.

Практична значимість висновків і рекомендацій. Розроблений веб-застосунок є цінним прототипом, який можна адаптувати до реальних потреб малого й середнього бізнесу. Він може стати основою для подальшого розвитку стартапу або бути використаним як кейс у професійній діяльності. Теоретичні висновки роботи узагальнюють важливі принципи побудови B2B-систем.

Кваліфікаційна робота Зименок Анастасії Володимирівни відповідає вимогам до робіт бакалаврського рівня, демонструє необхідний рівень самостійності, практичної реалізації та аналітичного опрацювання, заслуговує на оцінку «відмінно».

Місце роботи та посада рецензента

Доцент кафедри інформаційних систем та технологій
КПІ ім Ігоря Сікорського к.т.н., доцент

Підпис засвідчую

Декан ФІСТ



/Батрак С.О./

Винюк С.А.

ЗМІСТ

АНОТАЦІЯ.....	6
РЕФЕРАТ	7
ЗМІСТ.....	8
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ ВЕЛИЧИН І ТЕРМІНІВ	9
ВСТУП	11
РОЗДІЛ 1 ХАРАКТЕРИСТИКА ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ... 12	
1.1. Характеристика предметної області.....	12
1.2 Огляд існуючих рішень та наукових джерел.....	14
1.3 Висновок до розділу 1	15
РОЗДІЛ 2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ	17
2.1. Вимоги до системи.....	17
2.2 Основні задачі та алгоритми.....	21
2.3 Діаграми та архітектура системи.....	23
2.4 Висновок до розділу 2.....	48
РОЗДІЛ 3 РЕАЛІЗАЦІЯ СИСТЕМИ ТА ЇЇ ТЕСТУВАННЯ.....	50
3.1 Опис даних та управління ними	50
3.2 Апаратні вимоги	52
3.3 Технологічний стек та інтерфейси	54
3.4 Реалізація повноцінної інформаційної системи	56
3.5. Тестування та оцінка якості системи	76
3.6 Висновок до розділу 3	81
ВИСНОВКИ	82
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	83
ДОДАТКИ.....	85
ДОДАТОК А	85
ДОДАТОК Б	86
ДОДАТОК В	87
ДОДАТОК Г	88
ДОДАТОК Д.....	89
ДОДАТОК Е	90
ДОДАТОК Ж	91

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ ВЕЛИЧИН І ТЕРМІНІВ

API — інтерфейс прикладного програмування (англ. Application Programming Interface), набір визначених способів взаємодії між програмними компонентами.

B2B — бізнес для бізнесу (англ. Business-to-Business), модель взаємодії, коли одна компанія надає товари або послуги іншій компанії.

CRM — система управління взаємовідносинами з клієнтами (англ. Customer Relationship Management).

JWT — вебтокен JSON (англ. JSON Web Token), метод безпечної передачі інформації між сторонами у форматі JSON.

REST API — архітектурний стиль вебсервісів, який використовує стандартні HTTP-методи.

UML — уніфікована мова моделювання (англ. Unified Modeling Language), стандартизована мова для візуалізації, специфікації, конструювання та документування програмних систем.

IDEF0 — метод моделювання функцій, що використовується для опису рішень, операцій, організаційних процесів.

BPMN — нотація моделювання бізнес-процесів (англ. Business Process Model and Notation).

OPM — об'єктно-процесне моделювання (англ. Object-Process Methodology), підхід до моделювання систем, який об'єднує об'єкти та процеси в єдину діаграму.

Node.js — програмна платформа, що дозволяє запускати JavaScript-код на сервері.

React Native — фреймворк для створення кросплатформних мобільних додатків.

Express — мінімалістичний вебфреймворк для Node.js, що дозволяє швидко створювати серверні вебзастосунки.

PostgreSQL — об'єктно-реляційна система управління базами даних з

відкритим вихідним кодом.

JSON — формат обміну даними, що легко читається людиною і зручно парситься комп'ютером (JavaScript Object Notation).

Token — маркер доступу, який використовується для авторизації користувача в системі.

UI (User Interface) — користувацький інтерфейс, сукупність засобів для взаємодії користувача з системою.

UX (User Experience) — досвід користувача, якість взаємодії користувача з продуктом або послугою.

CRUD — акронім, який позначає базові операції з даними: Create, Read, Update, Delete.

PDF — формат електронних документів (Portable Document Format), який зберігає форматування та макет незалежно від програмного забезпечення.

CI/CD — безперервна інтеграція та доставка (Continuous Integration / Continuous Delivery), підхід до автоматизації розробки програмного забезпечення.

ВСТУП

У тенденціях цифрової трансформації бізнесу підприємства стали частіше стикатися з необхідністю оптимізації процесів взаємодії при закупівлі ІТ-послуг. Великий вплив на швидкість реалізації проєктів, якість кінцевого продукту та конкурентно спроможність компанії - це ефективність. Тому створення спеціалізованих інформаційних систем, які будуть спрощувати й автоматизовувати B2B-комунікації є актуальним розв'язання проблеми, яка забезпечить прозорість, масштабованість та надійність процесів.

Розробка галузевої торгівельної системи типу B2B для замовлення ІТ-послуг дозволяє не просто структурувати процес підбору підрядника, а й забезпечити контроль якості виконання, аналітику результатів та зменшення адміністративного навантаження. Ця система сприятиме зростанню довірливих відносин між компаніями завдяки автоматизації підтверджень, фіксації взаємодії та формуванню цифрового сліду процесу.

Однією з головних проблем у цій сфері є складність стандартизації вимог та процесів між різними компаніями. Для адаптування до різних структур замовлень, специфікацій проєктів та типів контрактів - систему слід зробити доволі гнучкою. При цьому слід забезпечити високу продуктивність та високий рівень захисту даних.

Стосовно вимог до сучасних B2B-систем - важливо зробити можливим масштабування, адаптації до нових регламентів і швидкого впровадження змін. Використання таких технологій як React Native для кросплатформеного інтерфейсу, Node.js (Express) для обробки бізнес-логіки та PostgreSQL для управління даними дає можливість збудувати надійний та гнучкий фундамент для розвитку ідеї.

Головною метою даної дипломної роботи є спроектувати, описати та частково реалізувати універсальну інформаційну систему, що автоматизує процеси взаємодії між компаніями в контексті замовлення ІТ-послуг. Буде розглянуто архітектуру системи, функціональні компоненти, технічні вимоги, та потенціал впровадження в реальні бізнес-процеси.

РОЗДІЛ 1 ХАРАКТЕРИСТИКА ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1. Характеристика предметної області

1.1.1 Визначення предметної галузі

Предметною галуззю, у якій застосовується дана система, є B2B-торгівля IT-послугами. Компанії-замовники укладають контракти з іншими компаніями на виконання IT-послуг. Це дозволяє бізнесам делегувати завдання професіоналам, зменшуючи витрати на утримання внутрішнього IT-відділу.

Сфера B2B IT-послуг включає:

- розробку програмного забезпечення,
- UX/UI-дизайн,
- тестування системи,
- супровід та технічну підтримку,
- IT-консалтинг і проєкний менеджмент.

Однією з особливостей B2B-моделі є складність узгодження, багатокрокова взаємодія та потреба в аналітиці. Отже, B2B-платформи мають бути надійними, безпечними, масштабованими та гнучкими.

1.1.2 Ключові процеси взаємодії між учасниками ринку

У межах B2B-моделі є взаємодія між двома основними сторонами:

- замовник - компанія, яка формує завдання, обирає виконавця та оплачує роботу,
- постачальник послуг - компанія, яка виконує завдання, комунікує зі замовником, робить звіти та отримує оплату.

Основними процесами у цьому випадку будуть: створення проєкту, подання заявки, вибір виконавця, виконання завдання, приймання результату, оплата та завершення проєкту.

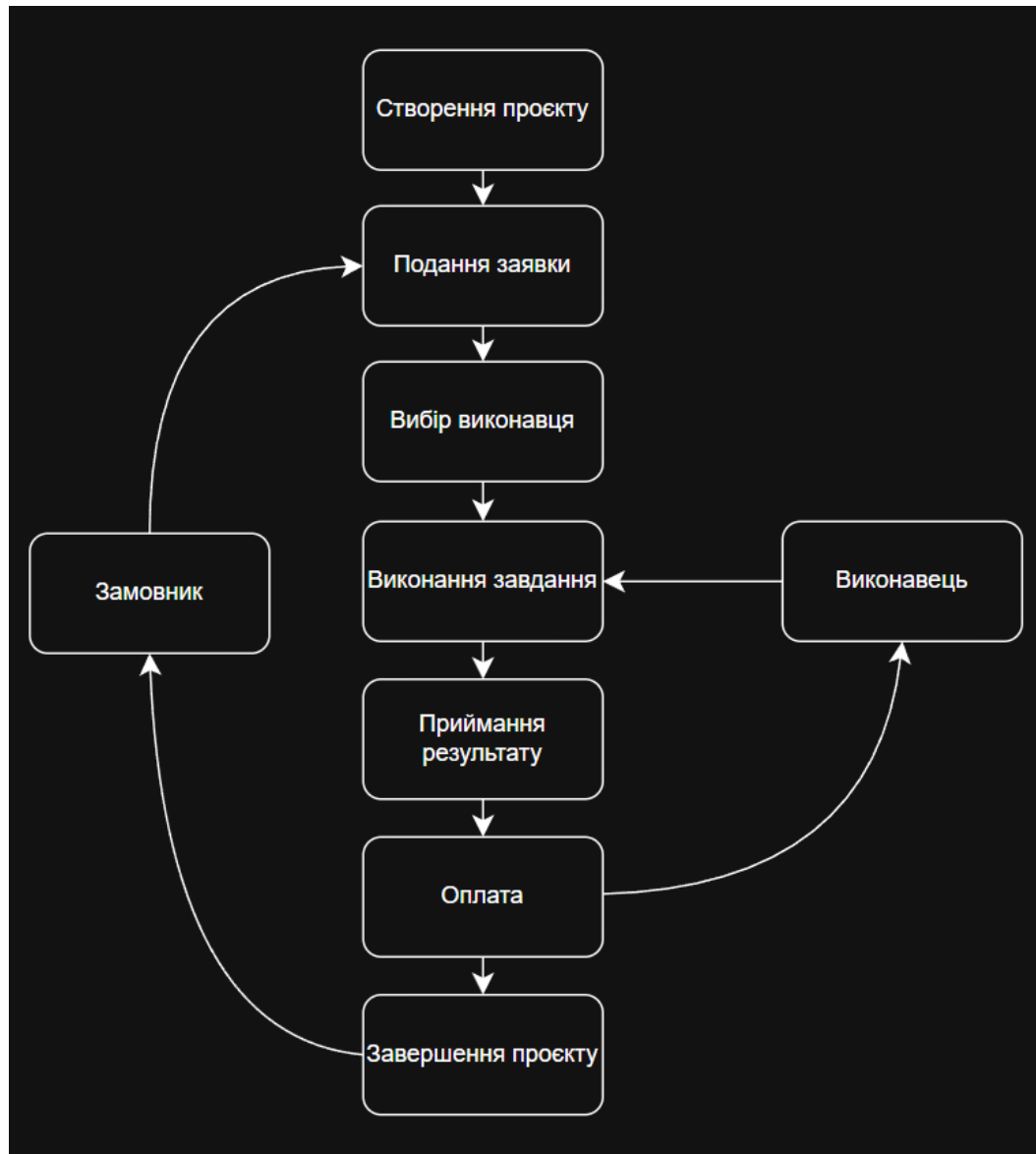


Рисунок 1.1 Схема ключових процесів взаємодії між замовником та постачальником послуг у B2B-моделі

На рисунку зображено послідовність основних етапів взаємодії між замовником і виконавцем у межах B2B-моделі замовлення IT-послуг. Процес починається зі створення проєкту замовником, який формує технічне завдання, бюджет та терміни виконання.

Далі потенційні виконавці можуть подавати заявки на участь у проєкті. Після цього замовник здійснює вибір виконавця, спираючись на рейтинг, портфоліо та умови заявки. Обраний виконавець переходить до виконання завдання згідно з погодженими умовами.

Після завершення робіт замовник здійснює приймання результату. Якщо результат відповідає очікуванням, виконується оплата, яка може бути реалізована

через інтеграцію з платіжними системами. Завершальним етапом є закриття проєкту, що супроводжується залишенням відгуку та можливим оновленням рейтингу виконавця.

1.1.3 Основні задачі та виклики сфери

Серед основних задач, які вирішує автоматизація в B2B IT-секторі є: централізація управління проєктами, швидкий пошук виконавців, прозоре відстеження статусу проєктів, оцінка якості виконання, автоматичне формування аналітики й звітів.

З іншого боку, постають наступні виклики: складність уніфікації вимог до різних проєктів, підтримка гнучкої моделі оплати, високі вимоги до захисту даних, різний рівень цифрової грамотності користувачів.

1.1.4 Потенційні користувачі та їх потреби

В у мовах даного проєкту я виділила три типи користувачів: замовник, виконавець та адміністратор платформи.

Основними потребами та очікуванням від замовника буде швидкий пошук виконавців, аналітика та безпека даних.

Щодо виконавця - зрозумілий інтерфейс, рейтингова система та гарантовані виплати за роботу.

В цей час адміністратори матимуть можливість модерування, контроль доступів та звітність.

Система має задовільняти потреби всіх користувачів одночасно, забезпечуючи зрозумілий та привабливий інтерфейс, прозорі механізми вибору та оплати та адаптивну логіку для кожної з ролей.

1.2 Огляд існуючих рішень та наукових джерел

1.2.1 Аналіз наукових джерел

Питання необхідності цифрової трансформації процесів управління IT-послугами підіймається в таких наукових роботах, як Nuseibeh, Eramo, а також в дослідженнях Gartner. Акцент робиться на важливості автоматизованого підбору підрядника, цифрових каналів комунікації та гнучкої інтеграції з CRM-системами.

Фокусом досліджень були маркетплейси для цифрової торгівлі IT-послугами та автоматизовані системи оцінки ризиків.

1.2.2 Існуючі програмні рішення

Аналізуючи ринок, було виявлено декілька наближених платформ-аналогів: Upwork, Toptal, Fiverr, Clutch, freelancer. [29]

Таблиця 1 Порівняння існуючих платформ

Платформа	Аудиторія	Автоматизація	API	Гнучкість	Рейтинг	Локалізація
Fiverr	Масова	Середня	Так	Низька	Є	Англ.
Upwork	Масова	Висока	Так	Середня	Є	Англ.
Clutch	B2B-компанії	Низька	Ні	Висока	Є	Англ.
Freelancer	Україна	Середня	Так	Середня	Є	Укр.
Toptal	Профі	Висока	Частково	Низька	Є	Англ.

Розглянуті існуючі системи на орієнтовані на B2B-взаємодію в контексті довгострокових контрактів із поділом на етапи, бюджетний контроль, правову відповідальність й аналітику. обмежувальними факторами для цих платформ було відсутністю локалізації, складна інтеграція з внутрішніми системами компаній, відсутність кастомізації ролей та прав доступу.

Це дає можливість створити вузькоспеціалізовану, модульну та адаптовану систему, яка стане цифровою платформою для надійної B2B-взаємодії у сфері IT-послуг.

1.3 Висновок до розділу 1

У цьому розділі було здійснено всебічний аналіз предметної галузі — B2B-торгівлі IT-послугами. Визначено ключові характеристики сфери, зокрема специфіку взаємодії між замовниками та постачальниками послуг, основні етапи співпраці,

потреби користувачів, а також проблеми, які виникають у процесі організації та реалізації таких проєктів.

Розглянуто та узагальнено основні задачі, які потребують автоматизації: пошук виконавців, управління замовленнями, укладання угод, моніторинг прогресу, аналітика та звітність. Особливу увагу приділено викликам, серед яких — необхідність гнучкості системи, безпеки даних та адаптації до різного рівня цифрової грамотності користувачів.

У ході аналізу існуючих рішень було виявлено, що більшість популярних платформ не орієнтовані на вузькоспеціалізовану B2B-взаємодію в сфері IT-послуг. Це створює передумови для розробки нової інформаційної системи, яка враховуватиме недоліки наявних сервісів і краще відповідатиме потребам користувачів.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Вимоги до системи

Формулювання чітких вимог, які дозволяють проєктувати архітектуру, логіку та інтерфейс програмного продукту є невід’ємною частиною розробки інформаційної системи. У контексті нашого проєкту система повинна охопити повний цикл взаємодії між користувачами - від створення проєкту до його повного завершення. Це наводить на думку, що доцільно розділити вимоги на функціональні та нефункціональні.

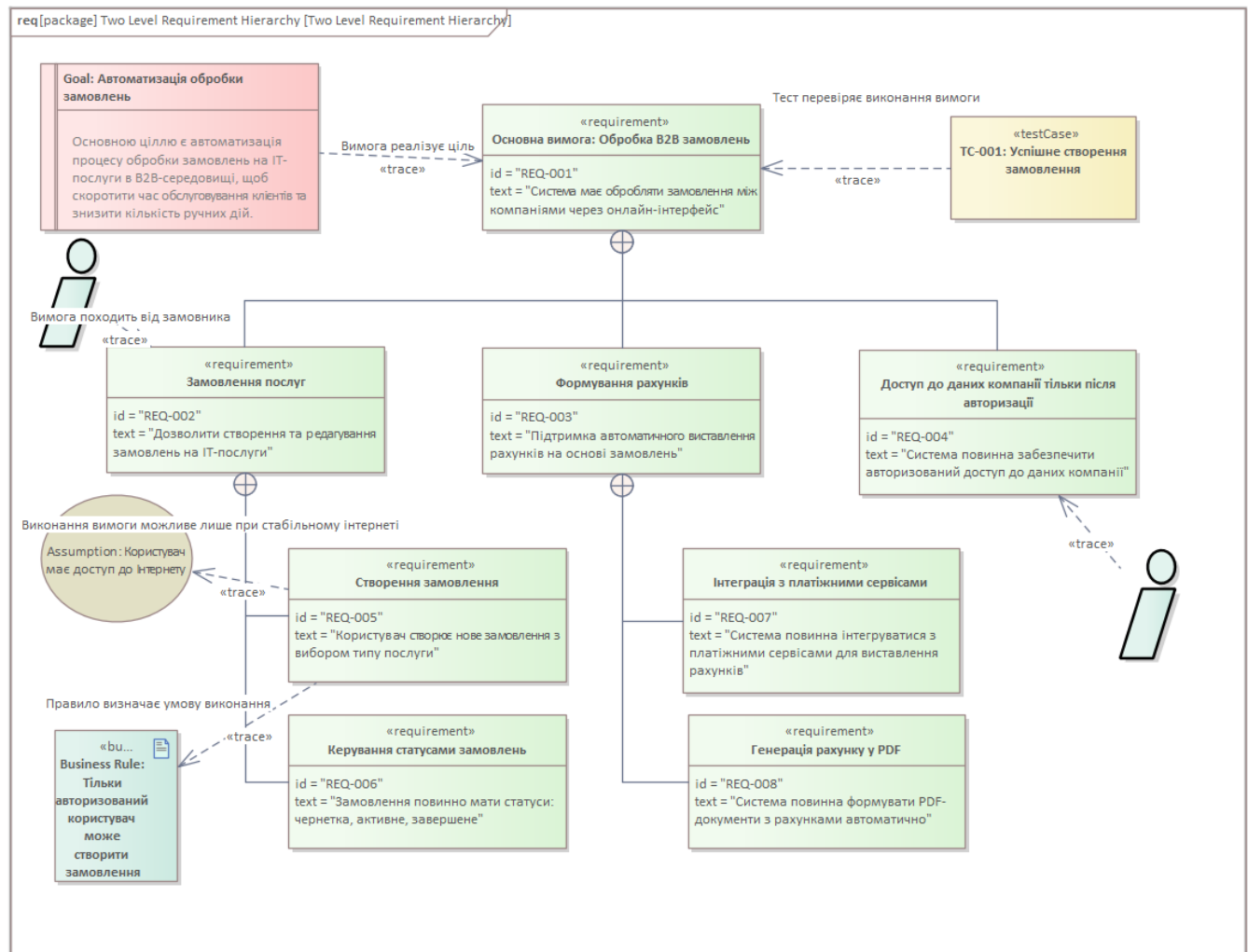


Рисунок 2.1 Діаграма вимог

У межах аналізу вимог до інформаційної системи типу B2B було побудовано дворівневу діаграму ієрархії вимог, яка відображає зв'язок між загальною метою, високорівневими функціональними очікуваннями та деталізованими вимогами до системи. Головна мета, закладена в основу системи, полягає в автоматизації обробки

замовлень на ІТ-послуги в В2В-середовищі. Це має забезпечити скорочення часу взаємодії між компаніями та зниження кількості ручних дій при обслуговуванні клієнтів.

На основі поставленої мети сформульовано основну системну вимогу — забезпечити обробку замовлень між компаніями через єдиний вебінтерфейс. Дана вимога деталізується через сукупність конкретних функціональних вимог, які сформульовані з урахуванням очікувань ключового стейкхолдера — замовника послуги. Зокрема, користувач має змогу створювати нові замовлення з вибором типу послуги, що реалізується через вимогу REQ-005. Для відображення життєвого циклу замовлення передбачено вимогу REQ-006, що встановлює можливість змінювати статус замовлення на «чернетка», «активне» або «завершене».

З метою забезпечення безпеки доступу до системи передбачено вимогу REQ-004 — система повинна допускати до перегляду й створення замовлень лише авторизованих користувачів. Це також підкріплюється окремим діловим правилом: «Тільки авторизований користувач може створити замовлення». Для зручності фінансових операцій додано вимоги REQ-003 та REQ-007, які стосуються автоматичного виставлення рахунків і їх інтеграції з платіжними сервісами. Завершальною функціональністю є REQ-008, що описує генерацію рахунків у форматі PDF.

Варто зазначити, що виконання зазначених вимог можливе лише за умови стабільного підключення до мережі Інтернет, що вказано як припущення в діаграмі. Окремо позначено тестовий сценарій TC-001, який перевіряє реалізацію ключового сценарію створення замовлення.

Таблиця 2 Таблиця вимог

ID	Тип вимоги	Опис вимоги	Складність	Пріоритет
1	2	3	4	5
REQ-001	Функціональна	Авторизація користувача	Середня	Високий

Таблиця 2 Таблиця вимог (продовження)

1	2	3	4	5
REQ-002	Функціональна	Створення нового замовлення	Висока	Високий
REQ-003	Функціональна	Відстеження статусу замовлення	Низька	Середній
REQ-004	Нефункціональна	Автоматичне створення рахунку	Висока	Високий
REQ-005	Нефункціональна	Інтеграція з платіжними сервісами	Середня	Високий
REQ-006	Бізнес-вимоги	Збереження логів активності користувача	Середня	Середній
REQ-007	Нефункціональна	Підтримка багатофакторної аутентифікації	Висока	Високий
REQ-008	Бізнес-вимога	Відображення звітності на дашборді	Низька	Середній

Таблиця вимог у форматі дозволяє швидко отримати уявлення про структуру вимог до системи. Завдяки такій структурі можна з легкістю відстежувати реалізацію кожної функції на етапі розробки та тестування.

2.1.1 Функціональні вимоги

Для визначення які саме задачі повинна вирішувати система - використовують функціональні вимоги. В рамках дипломного проєкту передбачено наступні функціональні вимоги:

По-перше, система має підтримувати автоматизований пошук виконавців. Користувач-замовник повинен мати змогу переглядати кандидатуру виконавців, фільтрувати їх за категоріями, а також отримувати рекомендованих підрядників спираючись на попередні взаємодії.

По-друге, важливим є управління замовленням. Це передбачає створення замовником нового проєкту з описом технічного завдання з можливістю поділити на етапи виконання, визначенням дедлайнів та бюджету. Усі ці етапи мають відображатися в інтерфейсі з чітким визначенням статусу.

Наступною, та не менш важливою, є функція укладання угод. Процес

прийняття заявки виконавця на конкретний проєкт повинно автоматично формувати цифрову угоду між сторонами, яка буде зберігатися в БД із фіксацією дати, умов та сторін.

Також, система повинна реалізовувати механізм подання заявок. Кожен виконавець матиме змогу залишити заявку на проєкт, який має опис пропозиції, вартість, дедлайн і тому подібне.

Виставлення рахунків і фіксація оплат - ще одна з важливих функціональних вимог. По закінченню проєкту, замовник матиме змогу сформувати рахунок, а виконавець - отримати сповіщення про оплату. У разі інтеграції з платіжними системами можлива автоматична верифікація проведених платежів.

Для керування платформою має бути реалізована адміністративна панель, яка дозволить модерувати користувачів, переглядати статистику, розв'язувати конфліктні ситуації та змінювати системні параметри.

Таким чином, така сукупність функціональних вимог формує набір інструментів, необхідних для ефективної реалізації платформи як цифрового середовища взаємодії між ІТ-компаніями.

2.1.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи, які безпосередньо впливають на зручність її використання, надійність, безпеку та масштабованість.

Перш за все, система має демонструвати високу продуктивність. Час відгуку клієнтської частини не повинен перевищувати 300 мілісекунд. Серверна частина має забезпечувати обробку до 500 запитів на секунду без втрати стабільності. Передбачається можливість одночасної підтримки щонайменше 500 активних користувачів.

Наступною важливою вимогою є масштабованість. Архітектура системи повинна передбачити можливість горизонтального масштабування, іншими словами додавання нових обчислювальних вузлів не змінюючи загальну структуру. Компоненти системи мають бути незалежними.

Також слід приділити увагу безпеці даних. Паролі користувачів мають

зберігатися в зашифрованому вигляді. Авторизація буде реалізована через токен JWT. Контроль доступу до функціоналу здійснюється згідно з ролевою моделлю. Має бути впроваджено обмеження запитів та логування критичних подій.

Для взаємодії системи з іншими сервісами, вона має мати високу інтеграційність. Передбачається наявність REST API для підключення CRM-системи, платіжні шлюзи, аналітичні сервіси.

Через високу кількість різних пристроїв, система має бути кросплатформенною. Клієнтський інтерфейс буде реалізовано з використанням фреймворку React Native, що забезпечить доступність для мобільних ОП.

На майбутнє планується інтернаціоналізація - підтримка багатомовного інтерфейсу, адаптація під часові зони формати дати та валюту.

Сумуючи вище перелічені нефункціональні вимоги дають змогу визначити базові характеристики системи, необхідні для її стабільної роботи в умовах навантаження.

2.2 Основні задачі та алгоритми

У рамках дипломного проєкту ключовими задачами є автоматизація створення та супроводу замовлення, пошук виконавців, інтеграції з CRM-системами, забезпечення фінансової взаємодії та рейтингування учасників. Для реалізації цих задач треба спроектувати відповідні алгоритми, які забезпечать нам точність, швидкодію та масштабованість системи.

2.2.1 Основні задачі інформаційної системи

На основі вимог до систем було виділено сі ключових задач: автоматизоване управління замовленнями, автоматизований відбір виконавців, рейтингування та аналіз ефективності, інтеграція з CRM-системами, автоматизоване ціноутворення, контроль дотримання дедлайнів, генерація фінансових документів.

Для того щоб забезпечити автоматизоване управління замовленнями, користувачі системи повинні мати змогу створювати проєкти, описувати технічні завдання, визначати етапи, редагувати параметри замовлень, контролювати процес виконання та фіксувати результати.

Автоматизований відбір виконавців буде проходити наступним чином: на

основі заданих фільтрів система повинна відбирати підходящих виконавців. Це має підвищити швидкість вибору та зменшити людський фактор.

Для рейтингування та аналізу ефективності буде введено систему оцінювання роботи, де користувачі зможуть залишати оцінки, які впливатимуть на загальний рейтинг профілю. Система буде обчислювати рейтинг за певною формулою з урахуванням ваги критеріїв.

Буде передбачено API для передачі даних про замовлення, клієнтів, платежі у стороні системи для подальшого бізнес-аналізу.

Для автоматизованого ціноутворення система буде надавати рекомендований діапазон бюджету спираючись на проєкти зі схожою складністю.

Для контролю дотримання дедлайнів платформа буде сповіщати виконавців про наближення термінів здачі завдання, та буде фіксувати статистику своєчасності.

Також, система буде створювати автоматично рахунки, звіти та зберігати їх в PDF-форматі з можливістю завантаження.

2.2.2 Алгоритми реалізації задач

Серед вище перерахованих задач є ті, котрі можна прорахувати за допомогою формул.

Алгоритм рейтингового відбору виконавців буде включати багатфакторну оцінку. Вона забезпечить швидкий та якісний відбір.

Формула загального рейтингу виконавця:

$$R=0,4*Q+0,3*T+0,2*C+0,1*A$$

Де:

Q – середня оцінка якості виконання(1-5)

T – коефіцієнт вчасно зданих проєктів

C – коефіцієнт комунікабельності (оцінка замовника)

A – активність користувача (співвідношення поданих і виграних заявок)

Вибір виконавця буде здійснюватись через сортування кандидатів за рейтингом R та релевантністю технологій до проєкту.

Алгоритм автоматизованого ціноутворення буде використовуватися в тому випадку, коли замовник не вкаже фіксований бюджет.

Цей алгоритм буде складатися з декількох етапів:

Перший, вибірка проєктів:

$P = \{p_1, p_2, \dots, p_i\}$ – множина завершених проєктів, які мають таку саму категорію, як новий проєкт, $i=50$.

Другий, вибірка цін:

$C = \{c_1, c_2, \dots, c_n\}$ – вартість кожного проєкту p_i

Переходимо до обчислення середньої ціни:

$$C_{\text{середня}} = (c_1 + c_2 + \dots + c_n) / n$$

Обираємо $C_{\min}$ та $C_{\max}$

У результаті буде запропоновано діапазон $[C_{\min}; C_{\max}]$, але вказано $C_{\text{середня}}$

Алгоритм сповіщення про дедлайни.

Метою цього алгоритму буде виявлення етап, строків виконання які закінчують протягом 48 годин або вже прострочені та надсилання відповідного повідомлення.

$$t_{\text{залишок}} = t_{\text{дедлайн}} - t_{\text{зараз}}$$

Де:

$t_{\text{залишок}}$ – залишок часу

$t_{\text{дедлайн}}$ – дедлайн етапу

$t_{\text{зараз}}$ – теперішній час

Далі буде розраховуватись умова для сповіщення

Якщо до дедлайну ≤ 48 год.:

$t_{\text{залишок}} \leq 48$ год. $\rightarrow$ надіслати нагадування.

Якщо дедлайн прострочено:

$t_{\text{залишок}} < 0 \rightarrow$ позначити як прострочений та надіслати про це повідомлення.

2.3 Діаграми та архітектура системи

Процес проєктування галузевої інформаційної системи типу B2B з автоматизації IT-послуг потребує чіткого уявлення про склад системи, взаємозв'язки її компонентів та способи взаємодії із зовнішнім середовищем. Побудова структурних діаграм дозволяє формалізувати архітектурну модель, що є основою для реалізації та подальшого масштабування системи.

Візуалізація архітектури допомагає не лише зрозуміти внутрішню організацію системи, але й забезпечує ефективну комунікацію між розробниками, аналітиками, бізнес-користувачами та іншими стейкхолдерами. Формалізовані діаграми сприяють зменшенню помилок у процесі розробки, покращують узгодженість рішень та пришвидшують розгортання програмних компонентів.

У процесі побудови моделей використовуються сучасні підходи до архітектурного моделювання, такі як функціонально-орієнтоване моделювання (IDEF0), об'єктно-орієнтоване (UML) та процесно-орієнтоване (BPMN). У межах даного розділу зосереджено увагу саме на структурному представленні системи у вигляді логічно взаємопов'язаних компонентів та потоків даних.

2.3.1 Діаграми структури системи

Структурне моделювання інформаційної системи дає змогу формалізувати її архітектуру, визначити взаємозв'язки між логічними модулями та описати компоненти, необхідні для реалізації функціональності. У межах розробки галузевої торговельної B2B-системи було застосовано кілька видів діаграм, кожна з яких відображає окремий аспект структури системи.

Контекстна діаграма – це діаграма самого верхнього рівня, що представляє систему загалом, у вигляді "чорного ящика", і зв'язує її із зовнішнім світом за допомогою інтерфейсних дуг. Контекстна діаграма складається з одного функціонального блоку, будь-якої кількості стрілок, мети моделювання й точки зору.



Рисунок 2.2 Контекстна діаграма

Дана контекстна діаграма відображає головну функцію дипломного проєкту — розробку галузевої інформаційної системи типу B2B для автоматизації IT-послуг. Суть системи полягає в організації взаємодії між компаніями, що шукають IT-виконавців, та тими, хто ці послуги надає. На вхід система отримує запити на послуги та пропозиції виконавців. Її робота регулюється умовами співпраці, бізнес-логікою та вимогами до безпеки. Для реалізації функцій система взаємодіє з інформаційними сервісами, CRM-платформами, платіжними системами та OAuth-сервісами автентифікації. Результатом є укладені угоди між компаніями та завершені IT-проєкти. Діаграма чітко визначає межі системи, її учасників і загальну логіку обміну даними, формуючи основу для подальшої декомпозиції та проєктування.

ОРМ діаграма - це графічне представлення системи, яке включає об'єкти та процеси. Об'єкти представляють елементи системи, такі як клієнти, продукти або бази даних, а процеси - дії, які виконуються над цими об'єктами.

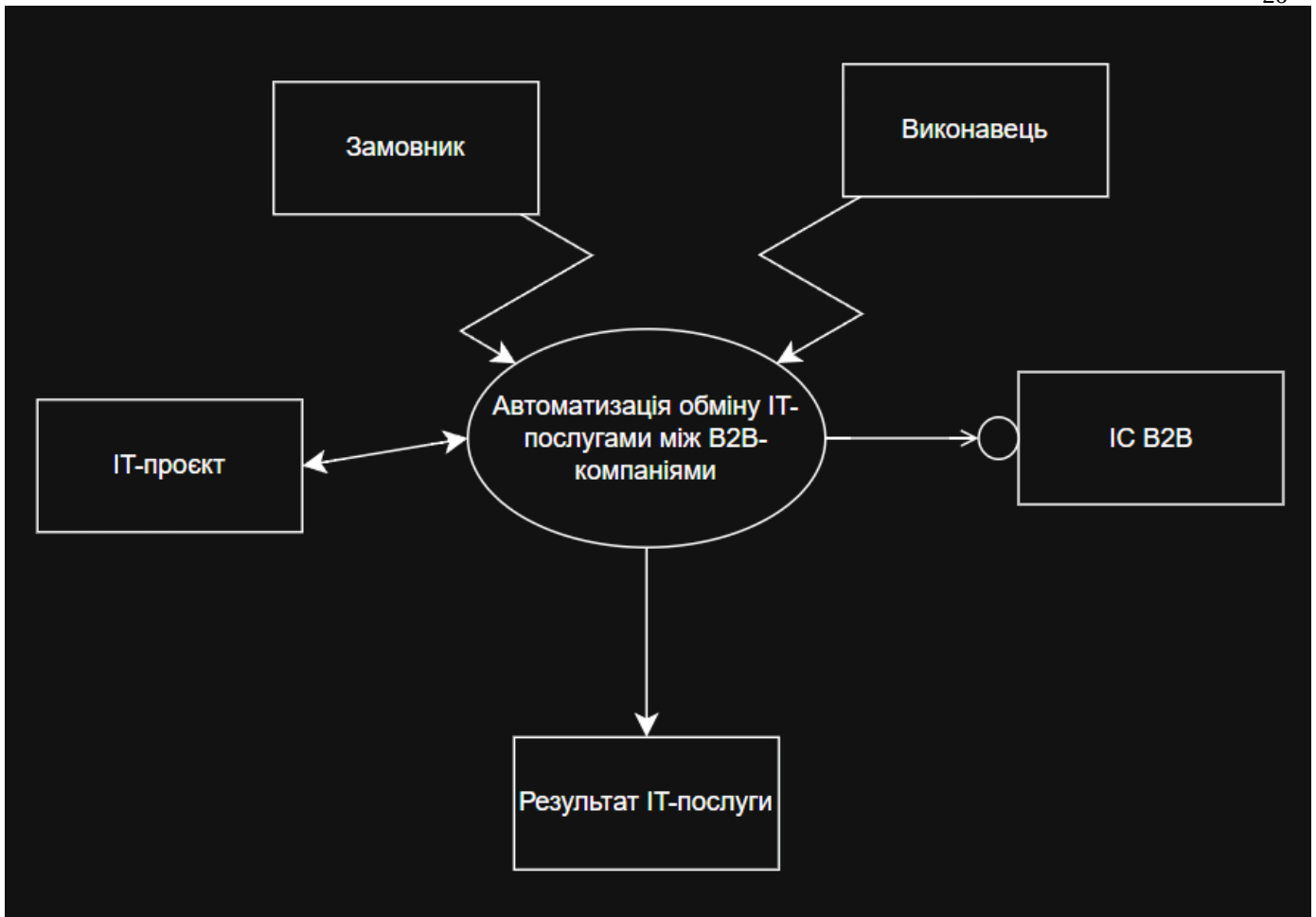


Рисунок 2.3 OPM-діаграма верхнього рівня

На рисунку ми бачимо OPM-діаграму верхнього рівня, яка моделює загальний процес автоматизації обміну ІТ-послугами між В2В-компаніями. Головними агентами процесу виступають Замовник і Виконавець, які ініціюють взаємодію в системі (зв'язки типу *Invocation*).

Вхідним об'єктом процесу є ІТ-проект, що є підставою для запуску роботи системи. Інструментом для реалізації процесу виступає ІС В2В, яка забезпечує технічне виконання дій. У результаті формується Результат ІТ-послуги, який відображає завершену взаємодію між сторонами.

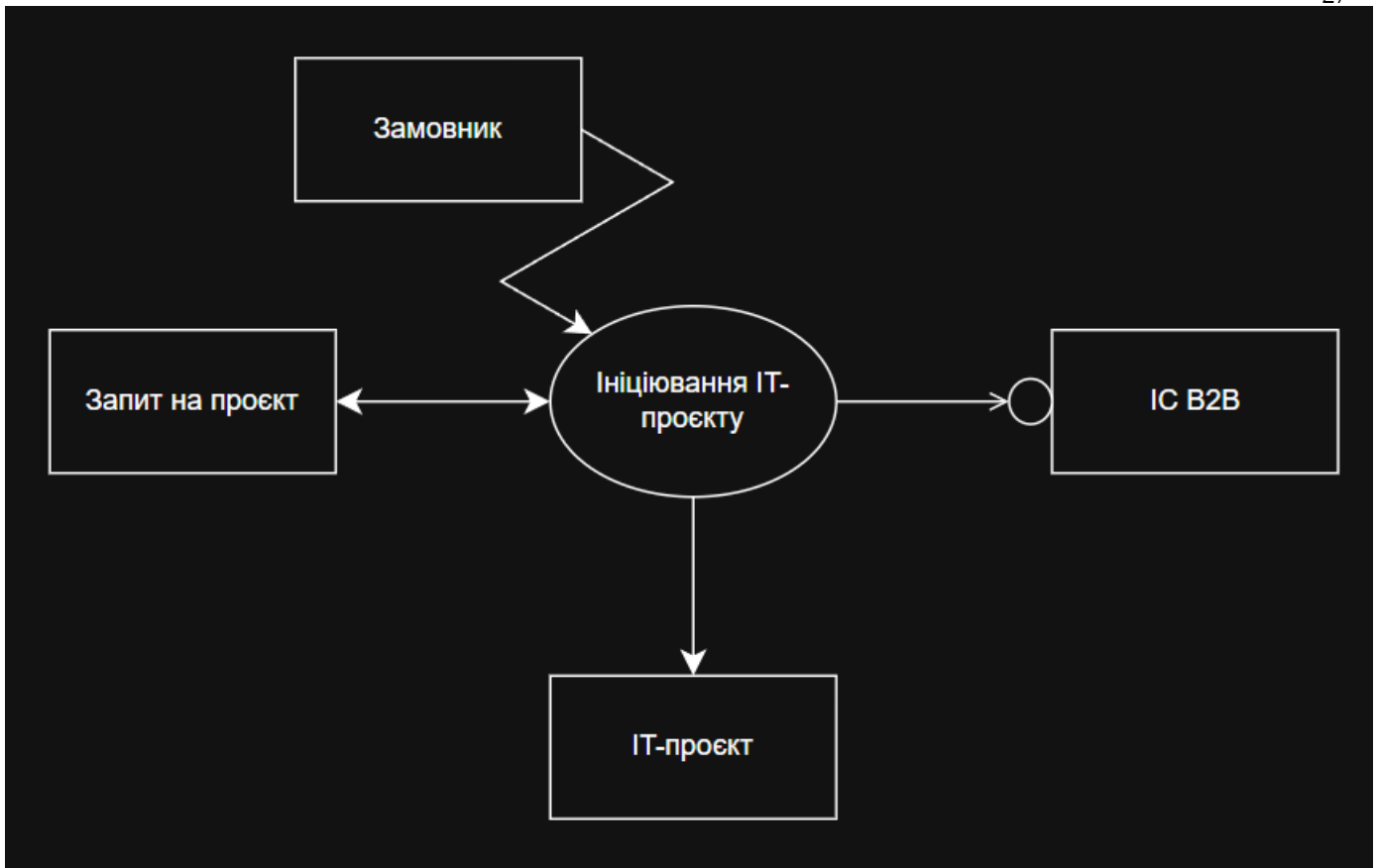


Рисунок 2.4 OPM-діаграма "Ініціювання ІТ-проєкту"

Ця діаграма ілюструє, як система переводить запит замовника в структуровану форму проєкту для подальшого виконання та відображає послідовність дій на ранній фазі взаємодії в рамках платформи.

Агентом процесу виступає Замовник, який ініціює створення проєкту (зв'язок типу *Invocation*). Вхідним об'єктом є Запит на проєкт, що формалізує потребу компанії у певній ІТ-послужі.

Процес реалізується із використанням інформаційної системи В2В як інструмента, який забезпечує введення даних, валідацію, збереження та подальшу обробку. Результатом виконання процесу є сформований ІТ-проєкт, що стає основою для подальшого призначення виконавців і виконання завдань.

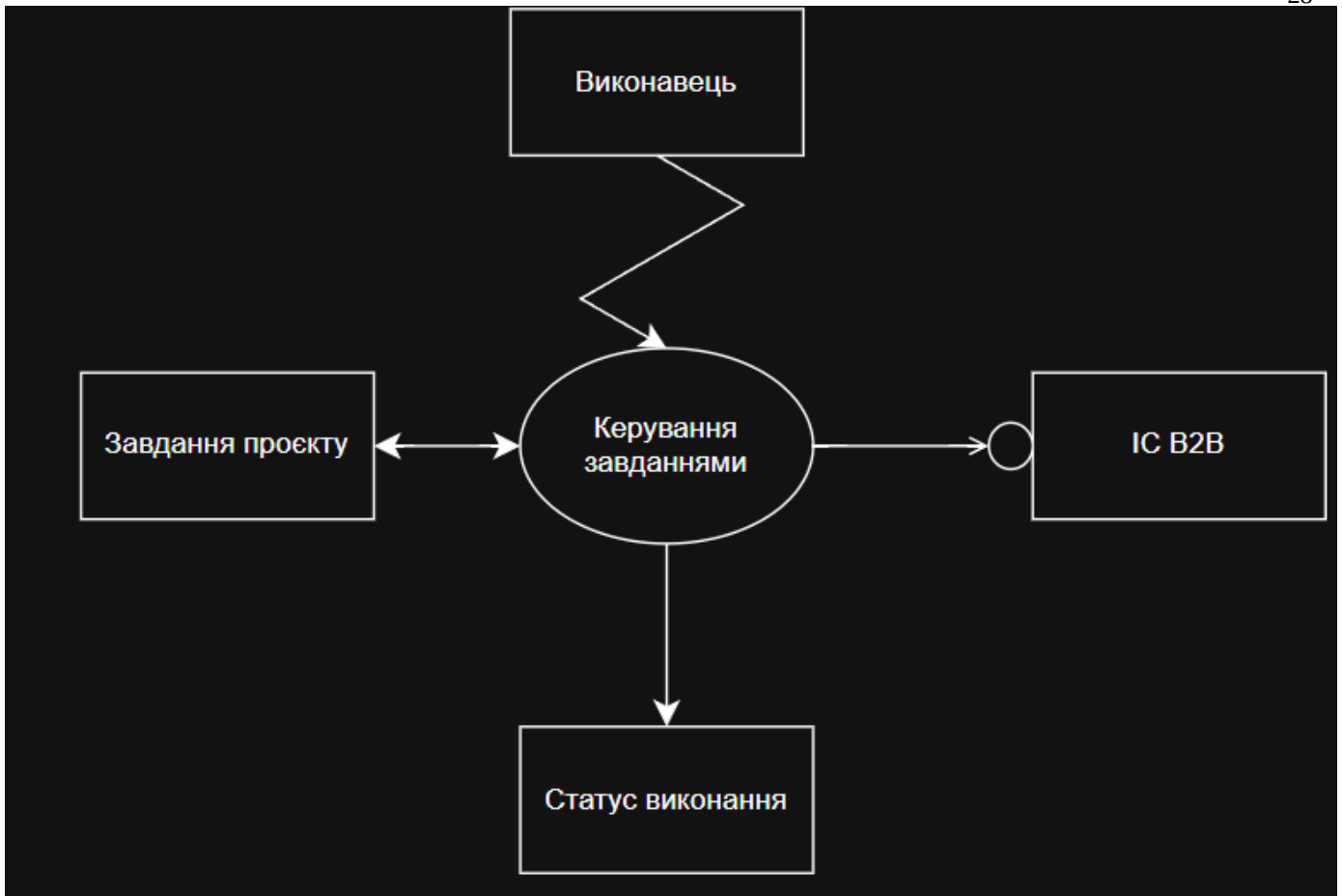


Рисунок 2.5 OPD-діаграма "Керування завданнями"

Діаграма демонструє, що процес ініціюється Виконавцем, який виступає агентом (*Invocation link*). Завдання проекту є вхідним об'єктом (*Effect link*), що потребує управління, контролю прогресу та звітності.

Процес здійснюється за допомогою інформаційної системи B2B, яка використовується як інструмент (*Instrument link*) для оновлення статусів, фіксації дій виконавця та відображення змін.

У результаті формується Статус виконання, що демонструє поточний стан завдання і є підставою для подальших дій з боку замовника або адміністратора.

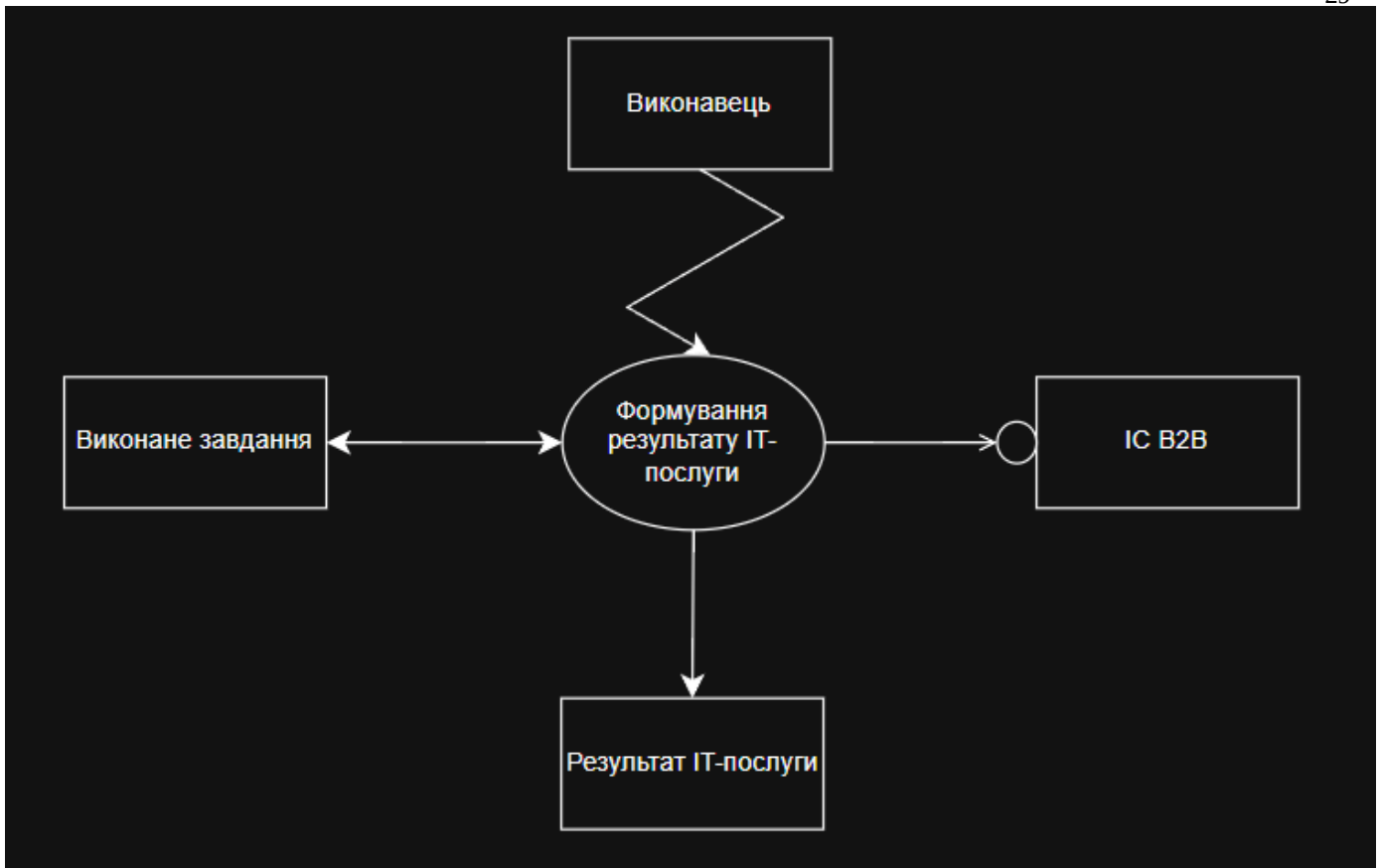


Рисунок 2.6 OPD-діаграма "Формування результату ІТ-послуги"

На діаграмі видно, що процес ініціюється Виконавцем, який виступає агентом (зв'язок типу *Invocation*). Виконане завдання є вхідним об'єктом (*Effect link*), що потребує структуризації та формалізації у вигляді результату. Для здійснення операцій використовується інформаційна система В2В, яка діє як інструмент (*Instrument link*) — вона забезпечує обробку, збереження та подання результатів послуги. Результатом процесу є Результат ІТ-послуги, який фіксується в системі, і пізніше буде переданий замовнику та долучений до аналітичної звітності.

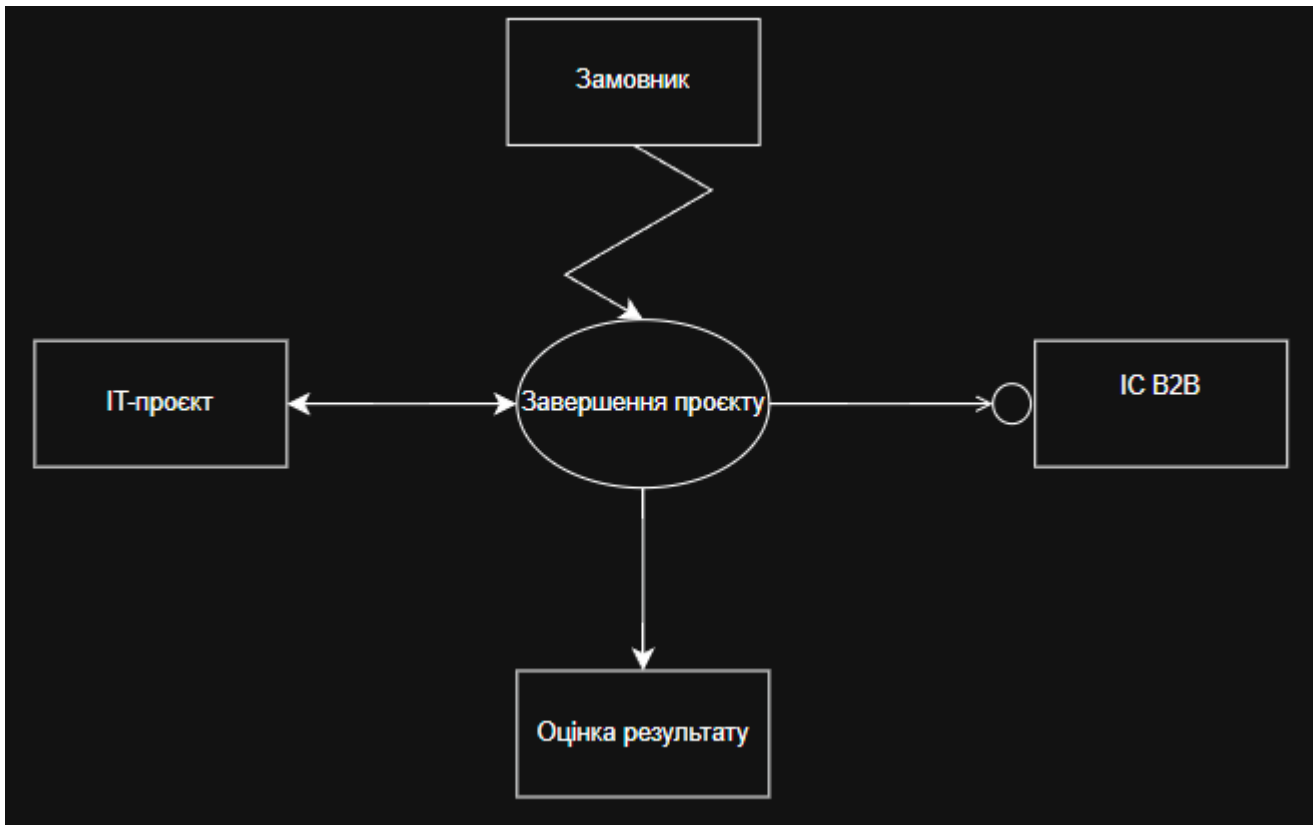


Рисунок 2.7 OPD-діаграма "Завершення проєкту"

Замовник виступає агентом, який ініціює процес (*Invocation link*), запускаючи завершальний етап взаємодії. ІТ-проєкт подається як вхідний об'єкт (*Effect link*), що потребує закриття з боку платформи.

Процес реалізується із використанням інформаційної системи В2В як інструмента (*Instrument link*) — для реєстрації статусу завершення, архівації даних і підготовки до звітності. У результаті формується Оцінка результату, що може включати рейтинг, фідбек та інші параметри ефективності виконання.

2.3.2. UML-діаграми

Для графічного моделювання взаємодії між користувачами та компонентами розробленої В2В-системи автоматизації ІТ-послуг було використано мову уніфікованого моделювання UML. Вона дозволяє чітко описати функціональні можливості, структуру та логіку роботи інформаційної системи на різних рівнях абстракції.

UML-діаграми слугують ефективним інструментом для виявлення взаємозв'язків між суб'єктами, об'єктами та їх поведінкою, що є важливим при проектуванні складних, розподілених систем із великою кількістю ролей [5], [8].

Діаграма прецедентів (або діаграма варіантів використання) — в UML, діаграма, на якій зображено відношення між акторами та прецедентами в системі. Діаграма прецедентів показує різні варіанти використання та різні типи користувачів системи і часто супроводжується іншими типами діаграм.

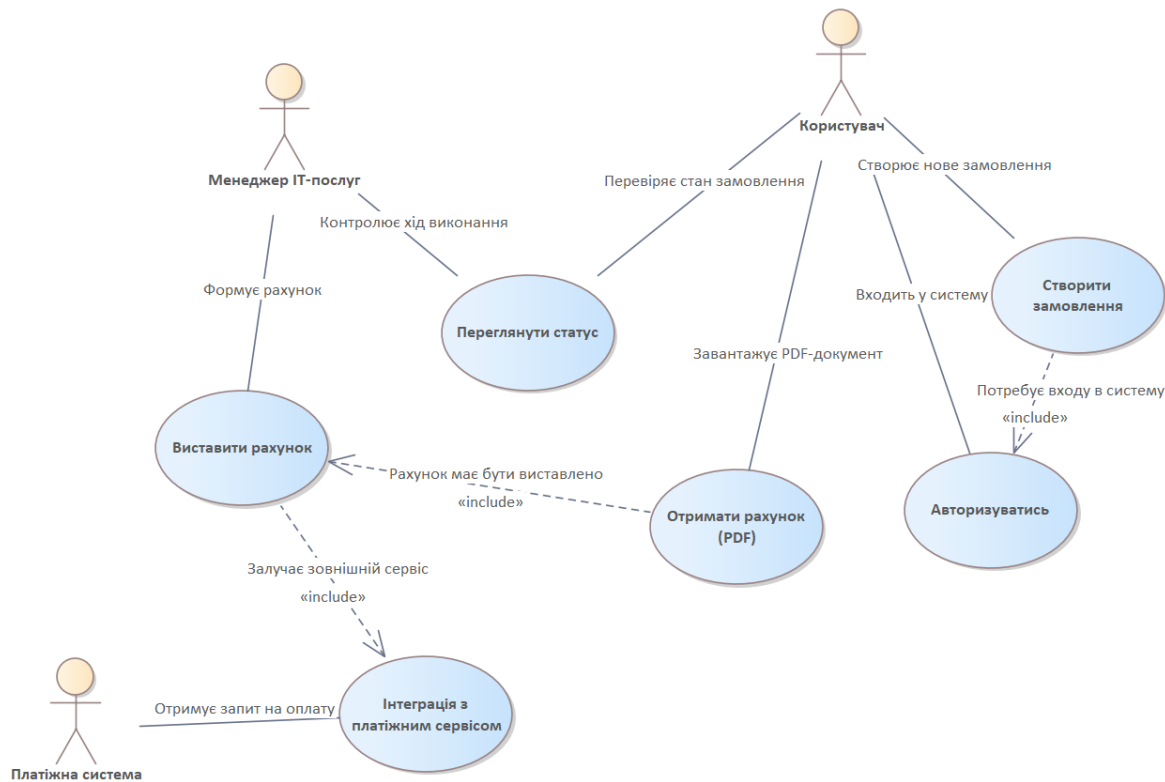


Рисунок 2.8 Діаграма прецедентів

На цій діаграмі проілюстровано основні сценарії використання ІС типу B2B для автоматизації IT-послуг. Акторами є: користувач, менеджер та платіжна система. Користувач взаємодіє із системою через авторизацію, створення замовлення, перегляд статусу та отримання рахунку. Авторизація – обов’язкова дія. Після входу користувач може створити замовлення, або перевірити статус поточного. Менеджер сформує рахунок після завершення проєкту, використовуючи дію «виставити рахунок», яка передбачає інтеграцію з платіжним сервісом для здійснення транзакцій. Отримавши рахунок, користувач зможе завантажити його у PDF форматі. Зовнішня платіжна система забезпечить обробку платежів, отримуючи відповідні запити від системи. Логіка взаємодії відбувається через зв’язки типу include, яка визначає обов’язковість виконання попередніх дій.

Діаграма класів — статичне представлення структури моделі в UML. Відображає статичні елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів може містити позначення для пакетів та може містити позначення для вкладених пакетів.

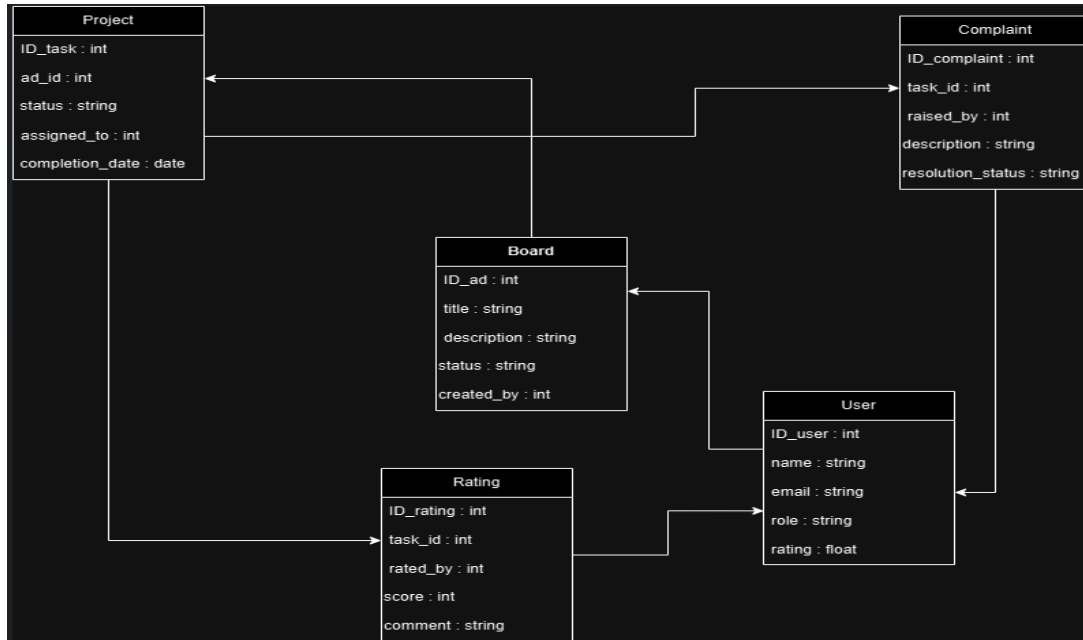


Рисунок 2.9 Діаграма класів

Діаграма класів UML, відображає ключові компоненти системи, їхні властивості, методи та взаємозв'язки. Основним елементом є клас Користувач (User), що представляє замовників і виконавців. Він містить атрибути userId, name, email, role та методи login(), logout() і updateProfile(). Клас Завдання (Task) відповідає за управління замовленнями, з атрибутами taskId, title, description, deadline, budget, status та методами для створення, оновлення й призначення завдань. Пропозиція (Proposal) зв'язана із завданнями через агрегацію, що відображає процес подання заявок виконавцями. Її атрибути proposalId, executorId, price, estimatedTime та методи забезпечують подання й оновлення пропозицій. Клас Оцінка (Rating) відповідає за оцінювання виконання завдань, а Скарга (Complaint) забезпечує механізм подання претензій. Діаграма коректно моделює зв'язки, такі як асоціації між користувачами й завданнями, та агрегації між завданнями і пропозиціями. Вона забезпечує чітку архітектуру системи, яка відповідає функціональним вимогам, зберігає модульність і дозволяє масштабувати проєкт.

Діаграма станів (фазова діаграма) — графічне зображення граничних ліній

співіснування фаз фізико-хімічної системи залежно від її складу та термодинамічних параметрів.

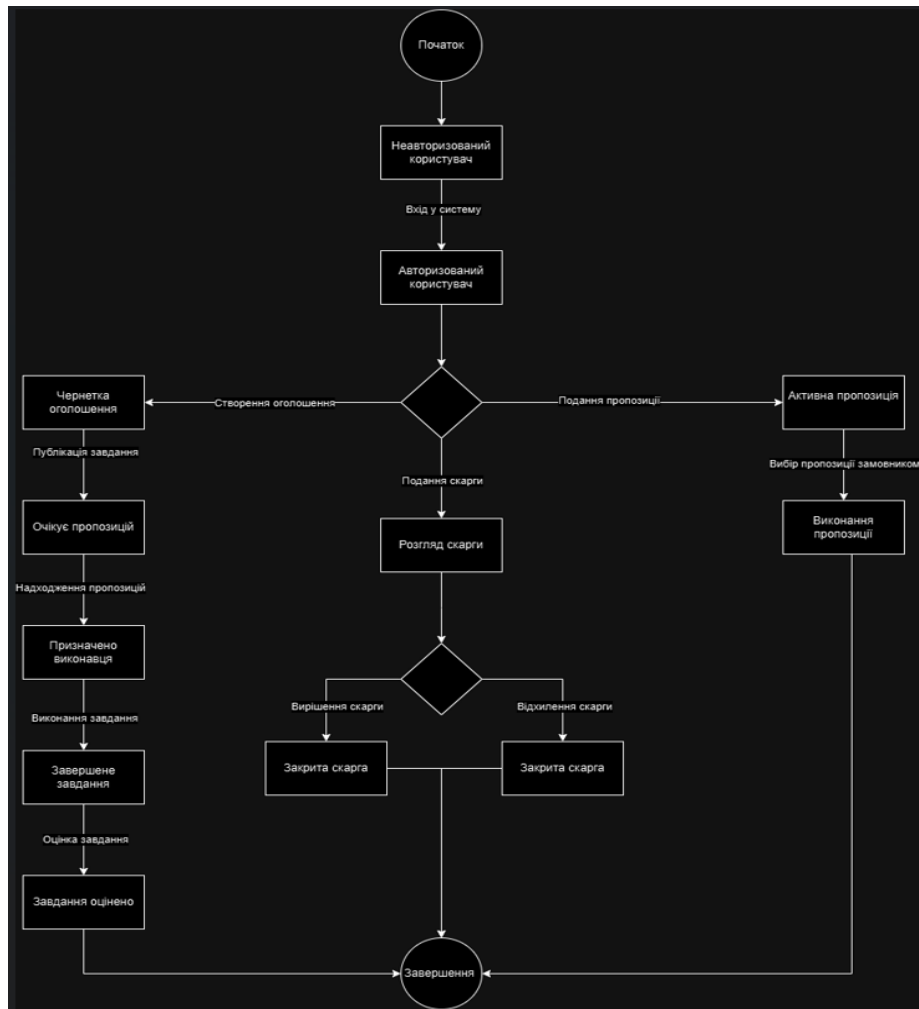


Рисунок 2.10 Діаграма станів

У системі визначено послідовну зміну станів користувача та об'єктів, що формують основні сценарії взаємодії. Початковим є стан «Неавторизований користувач», у якому перебуває користувач до моменту входу в систему. Після успішної авторизації він переходить у стан «Авторизований користувач», що надає доступ до базових функцій, зокрема створення оголошень, подання пропозицій і скарг.

При створенні оголошення воно спершу перебуває у стані «Чернетка оголошення», звідки після публікації переходить до «Очікує пропозицій». Після надходження відгуків і вибору одного з них, завдання переходить у стан «Призначено виконавця», а після завершення — у «Завершене завдання». Завершення підтверджується оцінкою від замовника.

Паралельно система передбачає окрему гілку для обробки скарг. Після подання скарги вона переходить у стан «Розгляд скарги», де адміністратор приймає рішення про її подальшу долю. Завершальним у цій гілці є стан «Закрита скарга».

Остаточне завершення взаємодії з системою відбувається після виходу користувача або завершення всіх активних процесів.

Діаграма дій є ключовим елементом моделювання процесів інформаційної системи, яка дозволяє наочно представити послідовність виконання дій і їхню взаємодію. Вона деталізує основні сценарії роботи системи, такі як створення завдань, подання пропозицій, вибір виконавця, виконання завдань і розгляд скарг. Завдяки діаграмі дій можна простежити, як користувачі (замовники та виконавці) взаємодіють із системою, а також зрозуміти, як виконуються бізнес-процеси на кожному етапі.

Діаграма охоплює як автоматизовані, так і ручні дії, що виконуються користувачами, адміністраторами або системою. Це дозволяє забезпечити повноту опису функціональності, зробити систему прозорою для розробників і користувачів, а також закласти основу для тестування й оптимізації процесів.

Для підтвердження логіки побудови діаграм дій і станів було змодельовано тестові сценарії, які відображають типові процеси взаємодії користувачів із системою. Ці сценарії дозволяють перевірити послідовність змін станів об'єктів і правильність обробки даних на всіх етапах бізнес-процесів.

Сценарій 1: Реєстрація нового користувача (замовник)

Мета: перевірити працездатність процесу реєстрації нового замовника

Передумови: Користувач не має акаунту, система працює в нормальному режимі

Кроки:

1. Користувач переходить на сторінку реєстрації
2. Обирає тип акаунту — "Замовник"
3. Вводить ім'я, email, пароль, назву компанії
4. Система перевіряє унікальність email
5. Надсилається код підтвердження на email

6. Користувач вводить код підтвердження
7. Система створює обліковий запис і авторизує користувача

Очікуваний результат: Користувач автоматично потрапляє до персонального кабінету, обліковий запис створено в базі даних.

Сценарій 2: Створення нового замовлення

Мета: перевірити працездатність процесу створення замовлення

Передумови: Користувач авторизований як замовник

Кроки:

1. Користувач переходить до розділу «Нові замовлення»
2. Натискає кнопку «Створити замовлення»
3. Вводить дані: назву, опис завдання, терміни, бюджет
4. Система перевіряє повноту та коректність введених даних
5. Замовлення зберігається в базі даних зі статусом «Очікує модерації»
6. Адміністратор отримує сповіщення про нове замовлення

Очікуваний результат: Запис замовлення з'являється у списку на модерацію в адміністратора, користувач бачить статус "Очікує підтвердження".

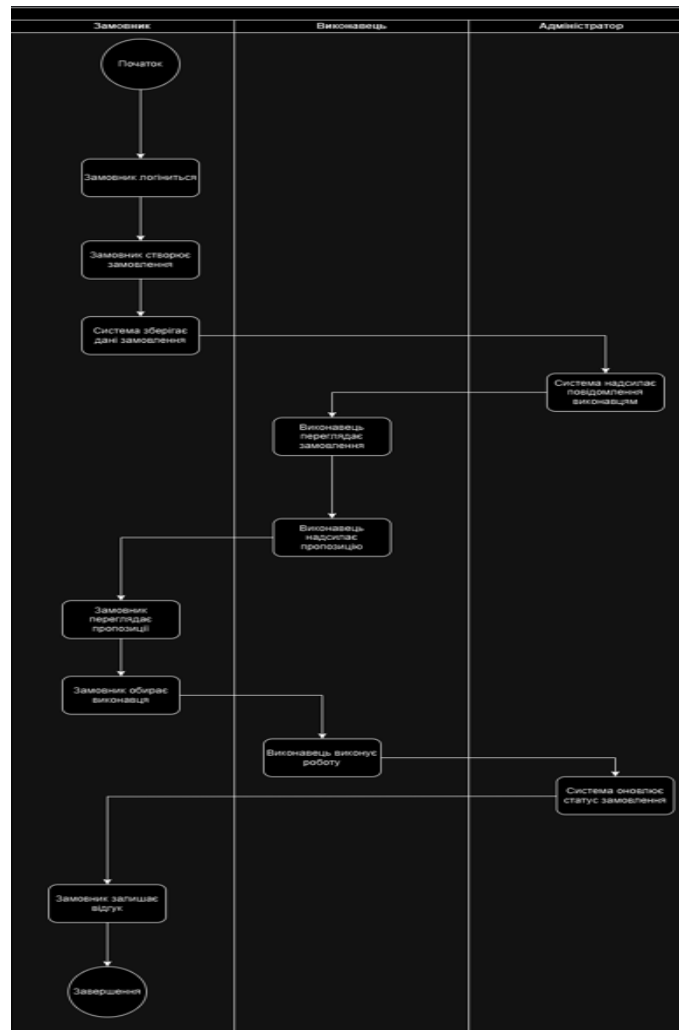


Рисунок 2.11 Діаграма дій

На рисунку показано діаграму дій, що моделює типовий процес взаємодії між основними учасниками B2B-системи під час виконання замовлення. Діаграма побудована з урахуванням ролей Замовника, Виконавця та Адміністратора, які розділено на відповідні «пули», що дозволяє відстежити відповідальність кожної сторони.

Процес починається з авторизації Замовника, після чого він створює замовлення. Система зберігає дані та надсилає сповіщення потенційним виконавцям. Далі Виконавець переглядає завдання та надсилає пропозицію. Замовник аналізує отримані варіанти, обирає виконавця, а той виконує роботу.

Після завершення роботи система оновлює статус замовлення. Останнім кроком є надання Замовником відгуку, що завершує бізнес-процес.

Діаграма послідовності відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність надісланих повідомлень.

В цій ІС було використано 3 класи UML : сутності, граничні та менеджери.

Сутності в цій системі представляють основні дані предметної області. Наприклад Order, Invoice, UserAccount та PaymentGatewayApi. Кожен з них містить відомості про клієнта, оплату або облікові дані. Тобто зберігають критично важливі дані, які використовує менеджер для обробки, або граничні класи для відображення в інтерфейсі.

Граничні класи забезпечують взаємодію користувачів із системою через інтерфейс. Прикладом є InvoicUI, LoginUI, OrderUI, PaymentUI та StatusUI. Вони надають інтерфейс для реєстрації, створення замовлення і т.д.

Менеджери реалізують логіку обробки даних, керуючи взаємодією між сутностями та граничними класами. Вони забезпечують виконання бізнес-правил та обробку користувацьких запитів. Прикладами є AuthController, InvoiceController, OrderController, PaymentController та StatusController. Вони керують замовленнями, перевіряють дані, здійснюють обробку платежів та контроль транзакцій.

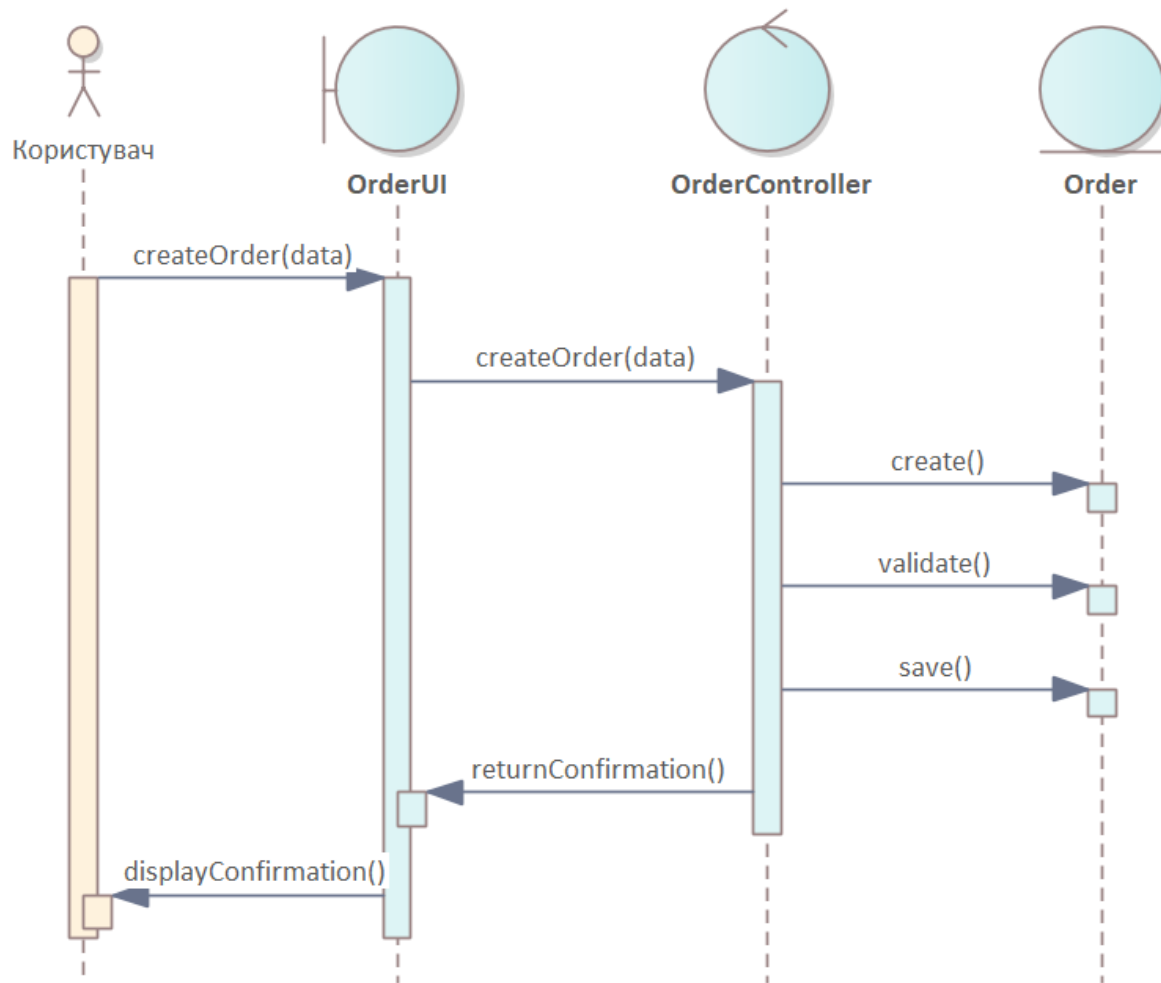


Рисунок 2.12 Діаграма послідовності для прецеденту «Створити замовлення»

Ця діаграма показує взаємодію між користувачем, інтерфейсом, контролером і сутністю. Користувач створює замовлення, дані перевіряються, замовлення створюється і система повертає повідомлення.

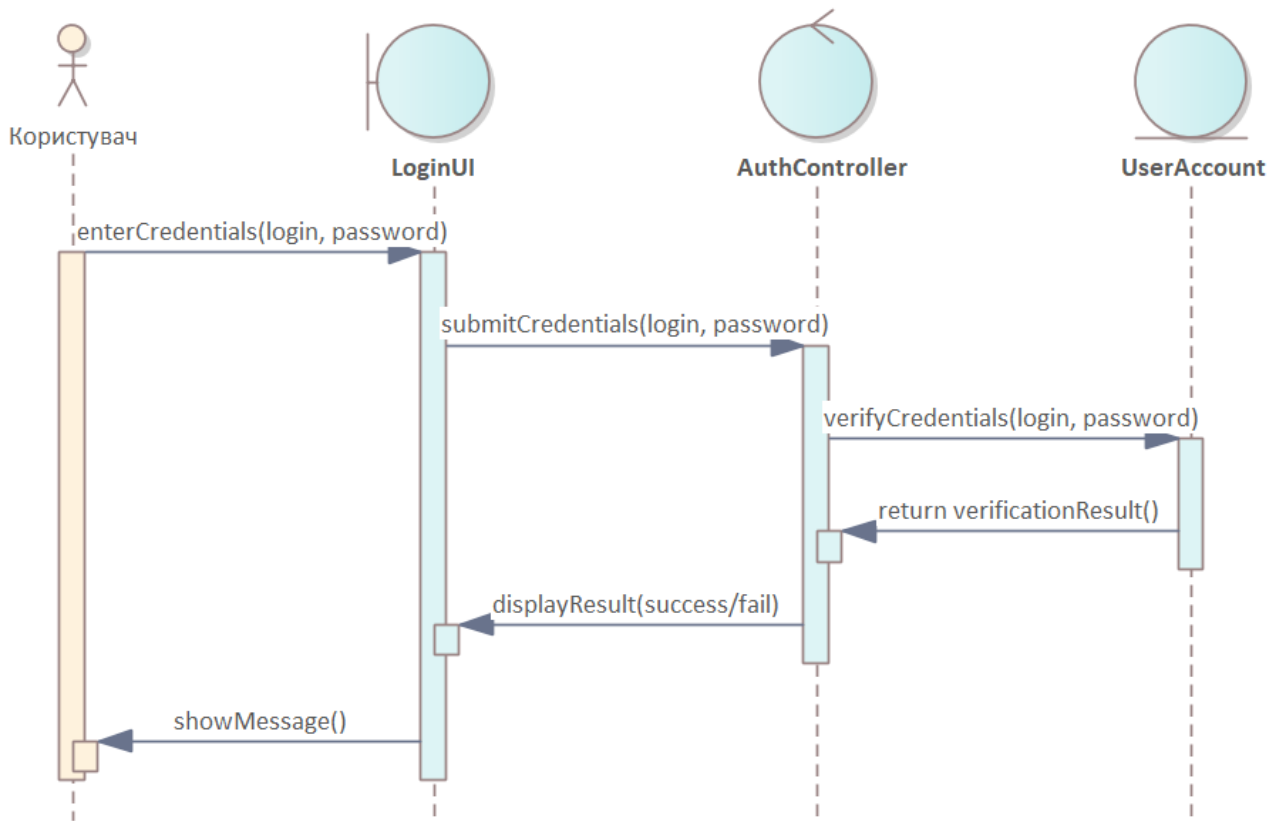


Рисунок 1.13 Діаграма послідовності для «Авторизація»

Діаграма показує взаємодію користувача із системою під час авторизації. Користувач вводить дані, система перевіряє їх та повідомляє про результат.

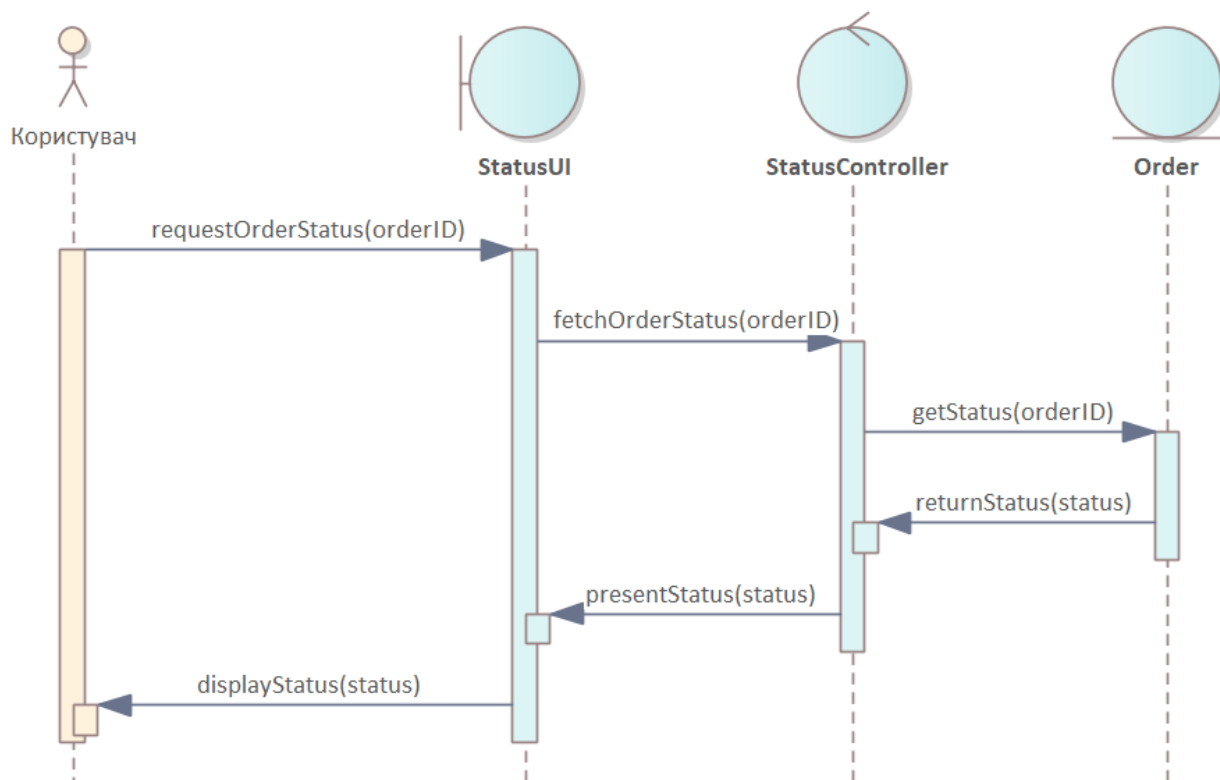


Рисунок 2.14 Діаграма послідовності для "Перегляду статусу замовлення"

Ця діаграма демонструє взаємодію між користувачем та системою під час перегляду статусу замовлення. Користувач надсилає запит, який обробляється для передачі результату.

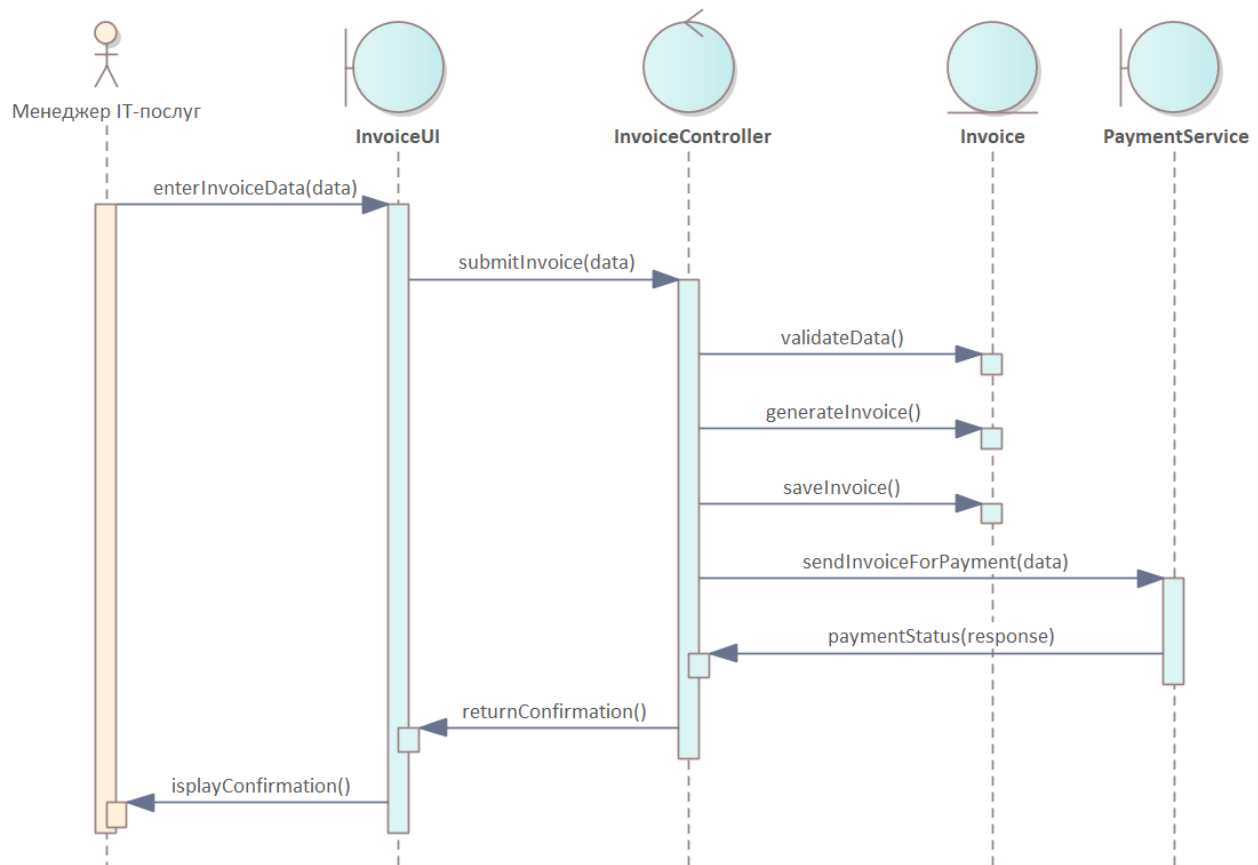


Рисунок 2.15 Діаграма послідовності для "Виставити рахунок"

Діаграма демонструє взаємодію між менеджером та сервісом під час виставлення рахунку. Менеджер вводить дані рахунку, контролер перевіряє їх, далі ініціює генерацію рахунку, викликає зовнішній сервіс, після цього надходить підтвердження про рахунок.

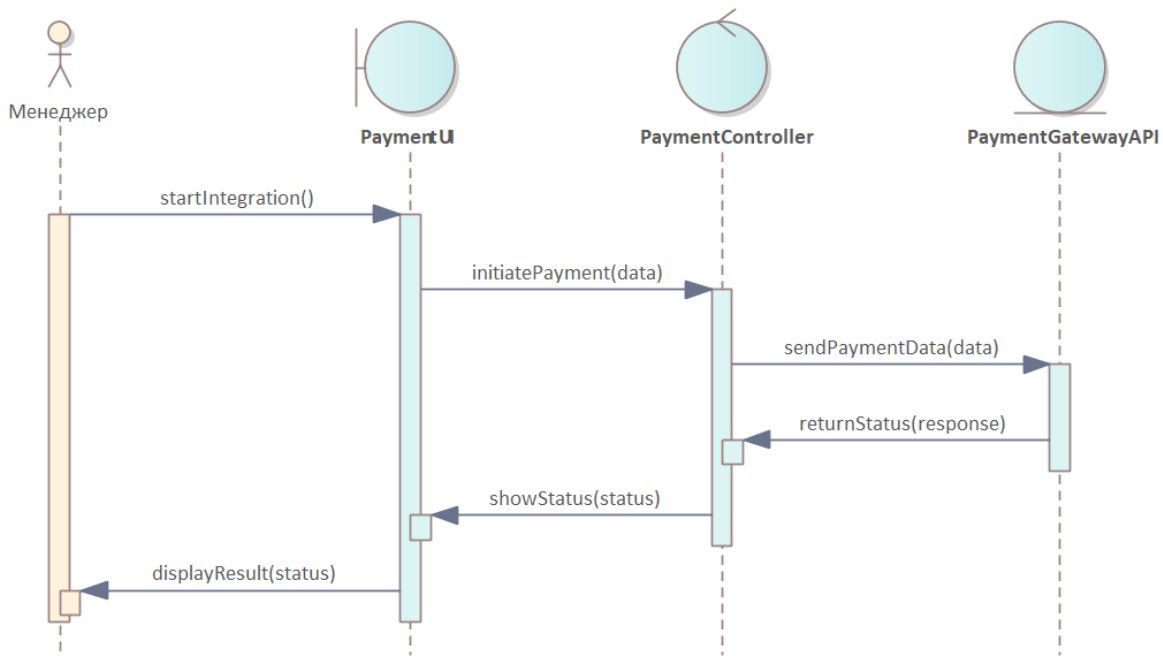


Рисунок 2.16 Діаграма послідовності для "Інтеграція з платіжним сервісом"

Діаграма показує як проходить процес інтеграції з платіжним сервісом. Менеджер починає інтеграцію, дані передаються до контролера, він взаємодіє із зовнішнім сервісом, потім система повертає статус інтеграції менеджеру.

Діаграма взаємодії використовуються в UML для встановлення зв'язку між об'єктами. Він не маніпулює даними, пов'язаними з конкретним шляхом зв'язку. Діаграми взаємодії здебільшого зосереджені на передачі повідомлень і тому, як ці повідомлення складають одну функціональність системи. Діаграми взаємодії призначені для відображення того, як об'єкти реалізовуватимуть конкретні вимоги системи. Критичним компонентом діаграми взаємодії є лінія життя та повідомлення.

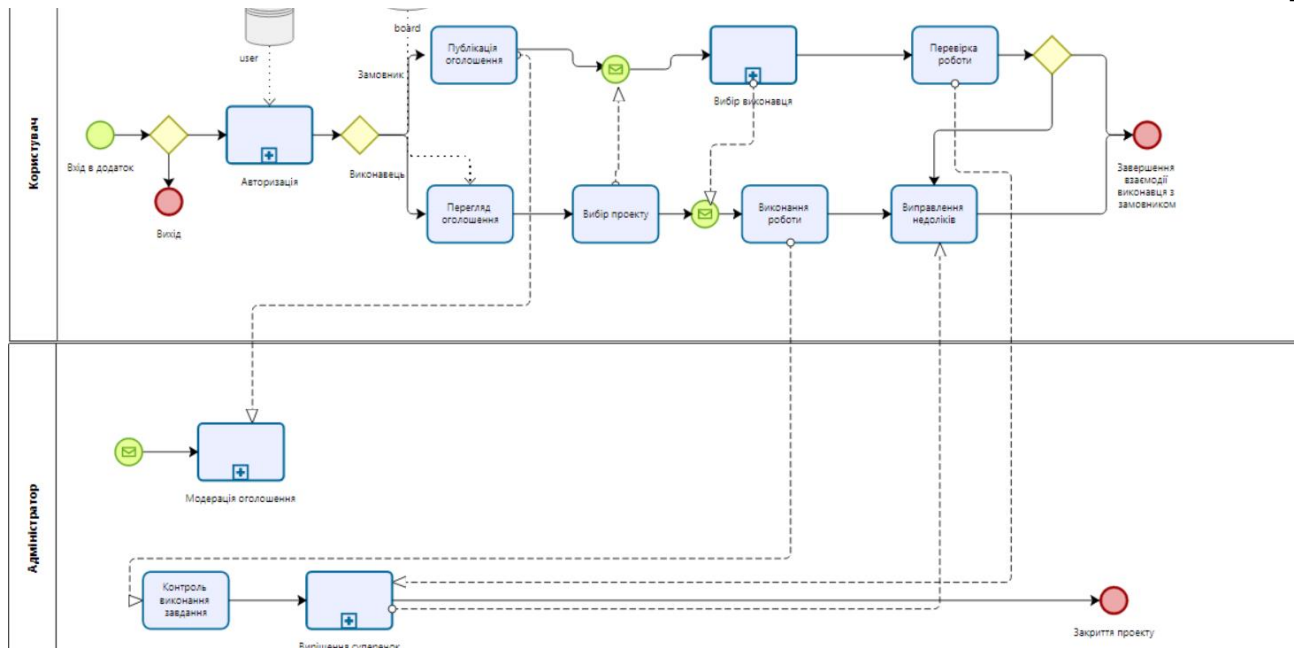


Рисунок 2.17 Діаграма взаємодії.

У поданій діаграмі зображено два пули — «Користувачі» та «Адміністратор». Пул користувачів не поділяється на окремі доріжки для замовника й виконавця, однак сценарії взаємодії обох ролей чітко простежуються в логіці процесу. Процес користувача починається з авторизації, під час якої визначається його роль у системі — замовник або виконавець. У випадку, якщо користувач є замовником, він створює оголошення, яке проходить модерацію з боку адміністратора. Якщо оголошення відповідає вимогам, воно схвалюється і стає доступним для перегляду виконавцями. Якщо ж ні — воно повертається замовнику на доопрацювання. Після схвалення замовник обирає виконавця, контролює виконання завдання, а також має можливість відправити результат на доопрацювання або завершити процес, якщо робота відповідає очікуванням.

У свою чергу, виконавець переглядає доступні оголошення, бере завдання в роботу, виконує його й, за потреби, вносить корективи на запит замовника. Адміністратор бере участь у модерації оголошень, контролі за процесом виконання завдань, а також вирішенні суперечок. Якщо скарга замовника підтверджується, завдання повертається виконавцю. Якщо ні — завдання вважається завершеним.

На діаграмі також відображені шлюзи, які регулюють ключові переходи: зокрема, після перевірки завдання замовником — для вибору між завершенням роботи або її доопрацюванням; після модерації адміністратором — для схвалення або

відхилення оголошення. Додатково на діаграмі показано інтеграцію з базами даних: база user використовується під час авторизації, а база board — для зберігання та обробки оголошень. Усі процеси логічно завершуються: для користувачів — із підтвердженням виконаної роботи, а для адміністратора — після завершення модерації, контролю чи врегулювання конфліктів.

Наведемо приклад діаграми процесу авторизації, яка дозволяє наочно показати, як реалізовано сценарії авторизації користувачів у системі.

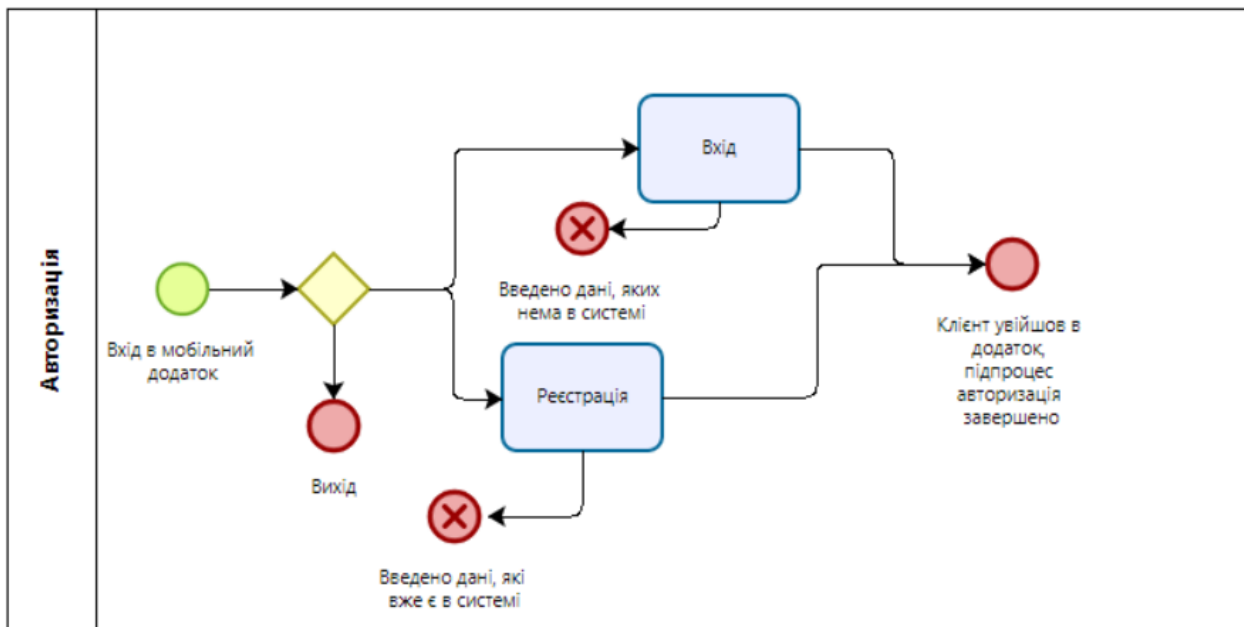


Рисунок 2.18 Процес «Авторизація»

Опис діаграми: Заходячи у додаток, користувач може Увійти або Зареєструватися. Помилкою при вході є введення даних, які не містяться в системі. Помилкою реєстрації є введення даних, які вже є в системі.

Для більшого розуміння процесу «Модерації оголошення», нижче наведено діаграму.

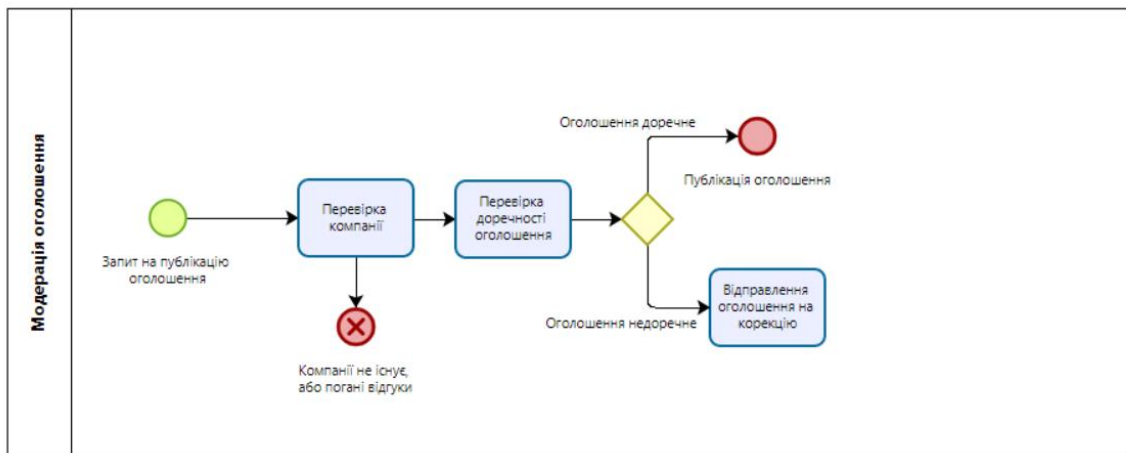


Рисунок 2.19 Процес «Модерації оголошення»

Опис діаграми: Після запиту на публікацію оголошення, адміністратор перевіряє спочатку компанію, потім оголошення. Помилкою є сумнівні дані про компанію-замовника. Також враховано повернення оголошення для корекції.

Вибір виконавця для замовника є важливою частиною для успішного проєкту, тому цей процес слід виділити та надати діаграму до нього.

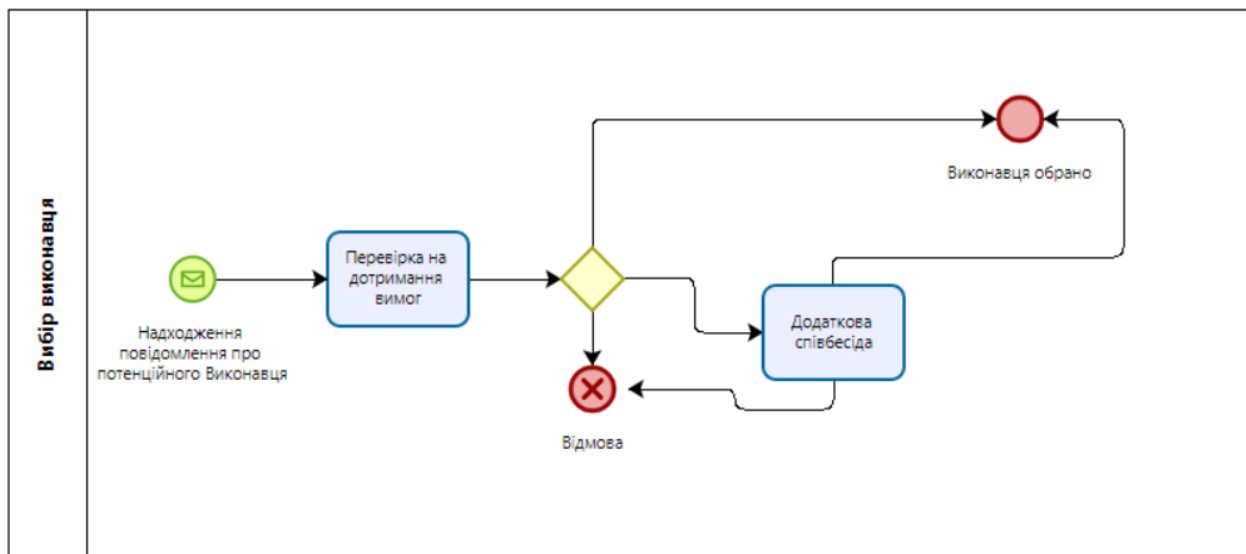


Рисунок 2.20 Процес «Вибір виконавця»

Опис діаграми: Компанія-замовник отримує лист про відгук на оголошення. Вони перевіряють дотримання вимог та якщо виникають питання, можуть компанію-виконавця запросити на додаткову співбесіду.

Також важливим процесом у створенні проєкту після його страту – є улагодження конфліктів. Адміністратор має чітко визначити проблему та діяти за встановленими правилами додатку. Приклад діаграми наведено нижче.

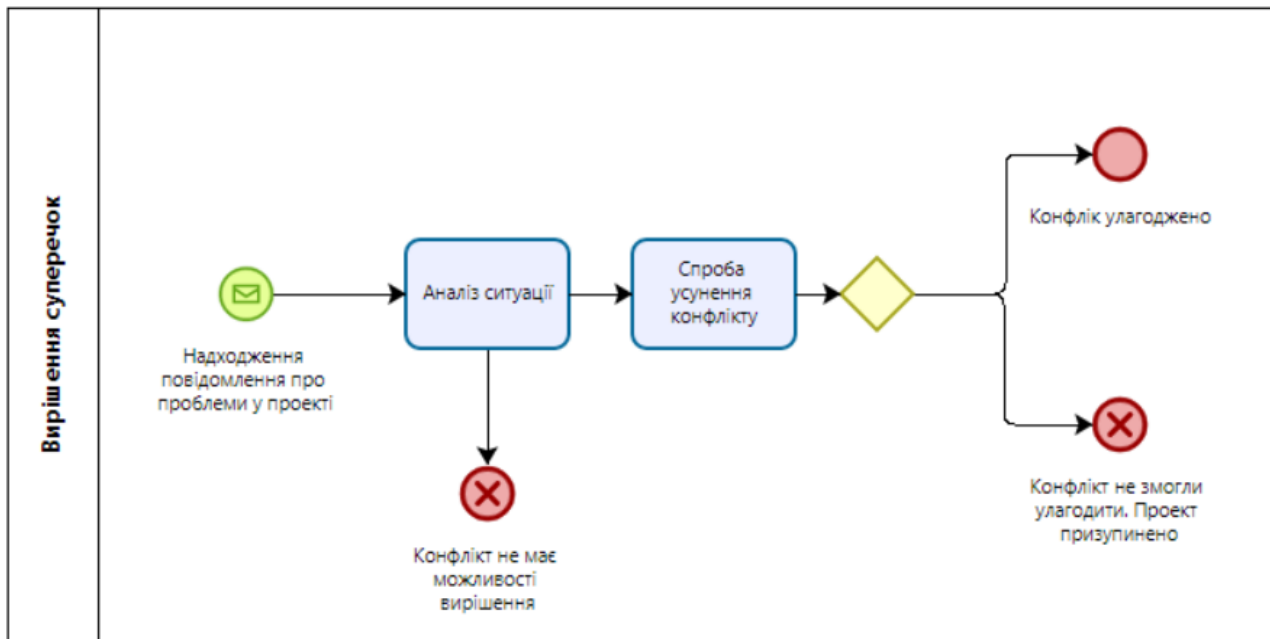


Рисунок 2.21 Процес «Вирішення суперечок»

Адміністратору надходить повідомлення про проблеми на проекті, він проводить аналіз суперечки та намагається усунути конфлікт. Якщо спроба неуспішна – співпраця між компаніями завершується.

2.3.3. Загальна архітектура системи та зв'язок її компонентів

Архітектура програмної системи визначає її загальну структуру, логічне компонування, а також розподіл функціональності між модулями та рівнями. В контексті розроблюваної B2B-системи для замовлення ІТ-послуг було застосовано **клієнт-серверну архітектуру** з розподіленням обов'язків між фронтенд-частиною, бекендом та сховищем даних.

Основними компонентами системи є:

- Мобільний клієнт — реалізований на базі React Native, відповідає за взаємодію з користувачем;
- Серверна частина — реалізована з використанням Node.js (Express), обробляє запити клієнтів, виконує бізнес-логіку;
- База даних — PostgreSQL, забезпечує надійне зберігання інформації про користувачів, замовлення, пропозиції, оцінки тощо;
- Зовнішні сервіси — інтеграції для автентифікації, надсилання електронних листів, платіжні шлюзи та аналітика.

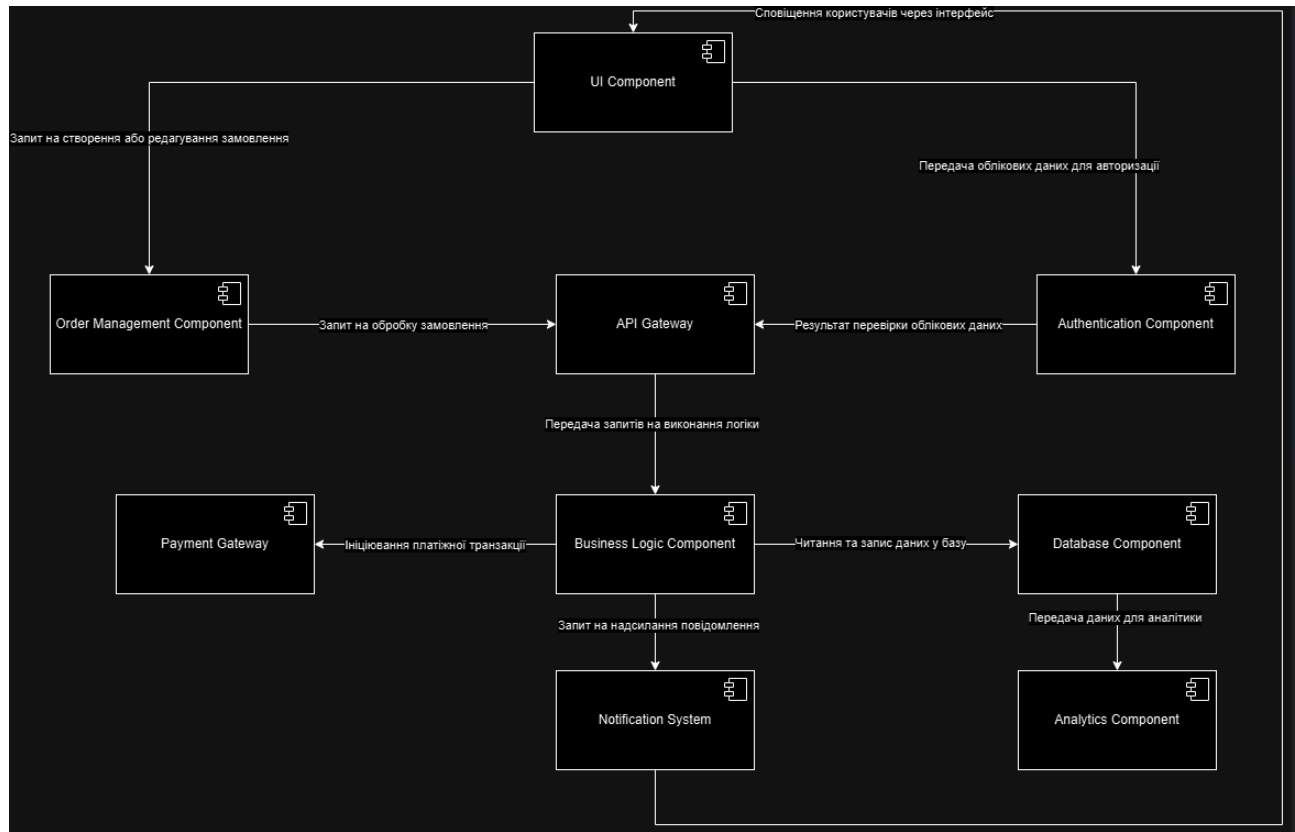


Рисунок 2.22 Діаграма компонентів

Для відображення логічної структури системи було побудовано діаграму компонентів, яка демонструє розділення функціональних модулів: модуль автентифікації, модуль замовлень, модуль пропозицій, модуль обробки скарг тощо. Зв'язки між компонентами відображають передачу даних і залежності між модулями.

Фізичне розміщення елементів системи представлено на діаграмі розгортання, яка моделює взаємодію між клієнтськими пристроями, серверним середовищем (наприклад, VPS або хмарна платформа), базою даних та зовнішніми API. Це дозволяє оцінити продуктивність, масштабованість та захищеність системи в умовах реальної експлуатації.

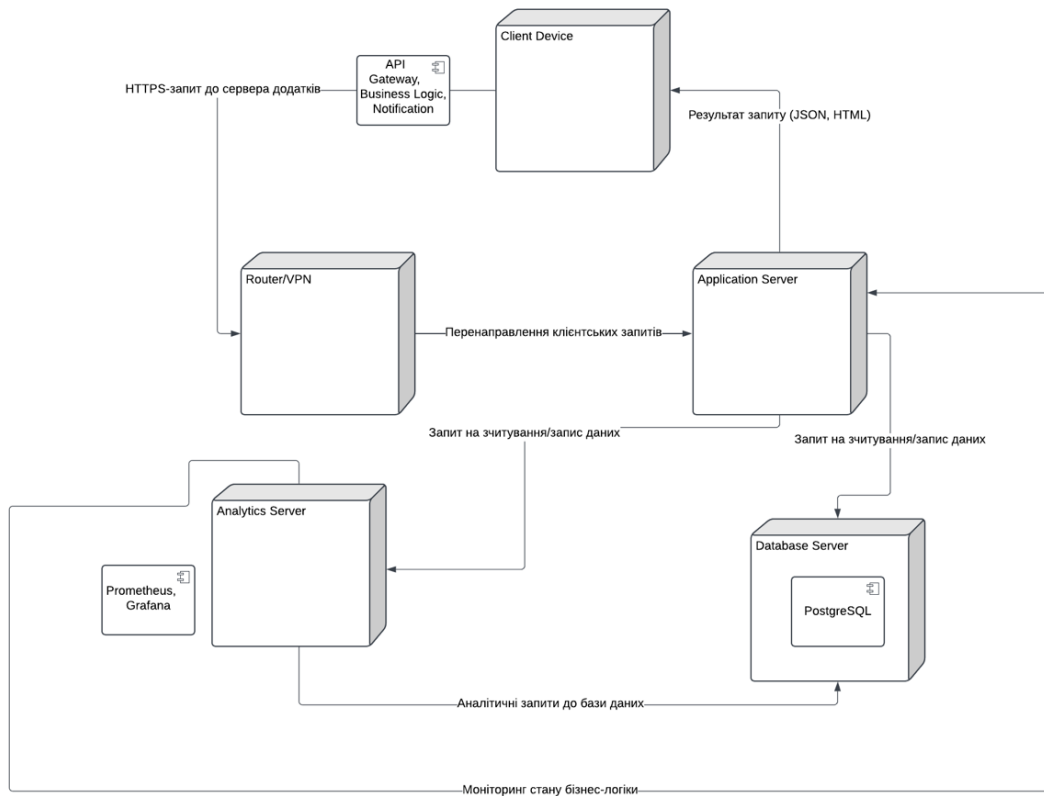


Рисунок 2.23 Діаграма розгортання

Діаграма розгортання UML ілюструє структуру технічних засобів системи замовлення IT-послуг, відображаючи взаємодію між клієнтськими пристроями, мережевим обладнанням, сервером додатків, сервером бази даних та сервером аналітики. Клієнтські пристрої (смартфони, планшети, ПК) підключаються до системи через мережеве обладнання (маршрутизатори з підтримкою VPN), яке забезпечує безпечне з'єднання із сервером додатків. Сервер додатків, розміщений у хмарному середовищі, обробляє запити користувачів, звертається до бази даних для зчитування чи збереження інформації та передає метрики на сервер аналітики для моніторингу продуктивності. Така структура забезпечує надійність, безпеку, масштабованість і централізовану обробку даних у системі.

2.4 Висновок до розділу 2

У другому розділі було здійснено комплексне проєктування галузевої B2B-системи, орієнтованої на автоматизацію надання IT-послуг. Під час роботи над розділом визначено та формалізовано функціональні й нефункціональні вимоги до системи, враховуючи потреби ключових ролей — замовника, виконавця та адміністратора. Здійснено класифікацію основних бізнес-задач та розроблено

алгоритми для їх реалізації, включно з рейтинговим відбором виконавців, автоматизованим ціноутворенням, генерацією звітів та сповіщенням про дедлайни.

Особливу увагу було приділено моделюванню структури та логіки системи: побудовано діаграми прецедентів, класів, станів, дій, послідовності, взаємодії, а також структурні OPD та OPM-діаграми, що дозволяють відстежити поведінку користувачів і внутрішню архітектуру рішень. Систему спроектовано за принципами клієнт-серверної архітектури з використанням сучасних технологій: React Native, Node.js (Express) та PostgreSQL, що забезпечує її масштабованість, гнучкість і безпеку.

Узагальнюючи, результати другого розділу створюють теоретичну й технічну основу для подальшої реалізації повноцінної інформаційної системи. Вони забезпечують відповідність вимогам замовника, підтримку критично важливих бізнес-функцій і закладають основу для надійного впровадження рішення в реальне B2B-середовище.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ СИСТЕМИ ТА ЇЇ ТЕСТУВАННЯ

3.1 Опис даних та управління ними

В даній системі ключову роль відіграють структуровані дані, які відображають процеси створення замовлень, вибору виконавців, виконання завдань, формування оплати та взаємодії між учасниками.

3.1.1 Набір даних що використовується у системі

Для реалізації ключових сценаріїв взаємодії замовника та виконавця формується та використовується широкий набір даних. Ці дані зберігаються в реляційні БД PostgreSQL. Система обробляє наступні дані:

Дані користувача – цей набір включає в себе ідентифікаційну інформацію, роль та рейтинг. Також, користувач може додати додаткову інформацію до профілю (опис досвіду або компанії)

Проекти та етапи – замовник має змогу створити проєкт, який буде містити опис проблеми, бюджет, дедлайн та перелік етапів. Етап матиме власні строки виконання, статус та зможе завершуватись окремо. Це забезпечить гнучке управління виконанням.

Заявки – ці дані подаються виконавцями у відповідь на відкритий проєкт. Заявка буде містити запропоновану вартість, термін, опис мотивації та за необхідністю портфолію. Також можливий статус заявки (очікує, прийнята та відхилена)

Фінансові транзакції – тут будуть міститися дані про рахунки за успішно виконані етапи, суми, дати та зв'язок із відповідним замовленням. Суми будуть зберігатися у гривнях.

Відгуки та оцінки – після завершення проєкту дві сторони зможуть надати оцінку один одного за шкалою від 1 до 5, та залишити коментар.

Службові дані – тут будуть зберігатися лог-файли, історія подій, статуси сповіщень та інформація про помилки.

Ця система збудована за принципом багаторівневої ієрархії сутностей, що дозволить масштабувати її при розширенні логіки.

3.1.2 Джерела, способи збору та управління даними

Дані в системі будуть формуватися з кількох джерел.

Перше – безпосереднє введення користувачем. Користувачі вводять дані при реєстрації, створенні проєкту, подачі заявки. Ці дії будуть виконуватись через інтерфейс мобільного застосунку, розробленого за допомогою React Native.

Друге – автоматичне формування на сервері. Прикладом цього є завершення етапу проєкту після чого сервер згенерує запис рахунку, або оновлення статусу – створення лог-події.

Третє – зовнішні інтеграції. При прив’язці до платіжних шлюзів, CRM-систем або API-аналітики, дані з цих джерел проходитимуть обробку через проміжні модулі і зберігатимуться лише у вигляді токенів або метаданих, без прямого зберігання платіжної або персональної інформації.

Управління цими даними буде реалізовано через REST-архітектуру. Кожен запит клієнта до API передасть дані у форматі JSON, оброблятися на сервері та вже потім вноситиметься до БД. Ця структура моделей забезпечить збереження цілісності даних.

3.1.3 Захист конфіденційної інформації

Дуже важливо мати захист даних у системі, оскільки вона опрацьовує персональні дані, комерційні пропозиції та фінансові транзакції. Для цього слід реалізувати багаторівневу модель захисту.

Паролі користувачів не будуть зберігатися у відкритому вигляді. Застосування алгоритму bcrypt для шифрування унеможливить обчислення зворотного значення. Мінімальна кількість ітерацій буде встановлена на рівні 12, що відповідає сучасним стандартам.

Для забезпечення безпечного обміну даними між клієнтом і сервером без постійного зберігання стану на сервері, система буде використовувати JSON Web Token. Кожен токен містить зашифровану інформацію про користувача, термін його дії та цифровий підпис.

Для контролю доступу використовується механізм ролевого контролю, який дозволяє обмежити доступ до функцій системи.

Усі файли будуть зберігатися у захищеному сховищі із обмеженим доступом. Кожне посилання на файл буде тимчасовим, що не дозволить пряме завантаження без

авторизації.

Система не буде зберігати карткові або банківські дані. Транзакції будуть здійснюватися лише через інтегровані платіжні сервіси. Зберігатися будуть лише токени платіжів, їх статус та метадані.

Резервне копіювання БД буде здійснюватись автоматично кожні 6 годин. При підозрілих діях система сповістить адміністратора.

3.2 Апаратні вимоги

Експлуатація інформаційної системи вимагає наявності певних ресурсів з боку клієнта та серверної частини. Апаратні вимоги залежать від архітектури системи, рівня навантаження, оброблення даних та швидкості реагування.

Для підвищення доступності передбачається підтримка мобільних пристроїв середнього рівня та хмарного серверного середовища.

3.2.1 Мінімальні та рекомендовані апаратні вимоги

Врахувавши архітектуру системи було визначено вимоги до ресурсів на різних рівнях.

Таблиця 3 Мінімальні та рекомендовані вимоги для клієнтського пристрою

Параметр	Мінімальні вимоги	Рекомендовані вимоги
Операційна система	Android 8.0/iOS 13.0	Android 10+/ iOS 15+
Оперативна пам'ять	2 ГБ	4 ГБ і більше
Процесор	ARM Cortex-A53/ Apple A9	ARM Cortex-A75/ Apple A13
Місце для зберігання	100 МБ	200+ МБ
Інтернет-з'єднання	LTE або Wi-Fi	LTE або Wi-Fi, або 5G

Дана таблиця демонструє, що застосунок оптимізований для роботи на бюджетних пристроях. Це дозволить охопити широку аудиторію.

Таблиця 4 Мінімальні та рекомендовані вимоги до серверної частини

Параметр	Мінімальні вимоги	Рекомендовані вимоги
Операційна система	Ubuntu 20/04 LTS	Ubuntu 22/04 LTS
Процесор	2 ядра (x86_64)	4 ядра і більше-x86_64, ЖЕО
Оперативна пам'ять	2 ГБ	4-8 ГБ
Місце на диску	20 ГБ SSD	40-80 ГБ SSD
Підключення	100 Мбіт/с	1 Гбіт/с

Ця таблиця показує, що наша система не потребує високопродуктивного серверного середовища й може функціонувати на мінімальній конфігурації. Рекомендовані параметри забезпечують стабільну роботу враховуючи зростання кількості користувачів, підвищення навантаження або інтеграції зі сторонніми сервісами.

3.2.2 Хмарні або локальні серверні рішення

Для хостингу бекенду та бази даних можуть бути використані як локальні сервери, так і хмарні платформи. Ці два підходи мають як плюси, так і мінуси.

Варіант локального розгортання підходить для невеликих команд або у випадках, коли потрібен повний контроль над інфраструктурою. Серед переваг є повний контроль над середовищем та незалежність від сторонніх постачальників. В той час недоліками є вища вартість обслуговування та необхідність самостійної побудови захисту та бекапів. У цьому випадку сервер може бути розгорнутий на власному обладнанні або орендованому VPS. Необхідно буде налаштувати систему моніторингу, оновлення та резервного копіювання.

Хмарне розгортання орієнтоване на системи, які мають швидкий запуск і доступність з будь-якої точки світу. Основними перевагами є мінімальні затрати на інфраструктуру, простота оновлення розгортання, масштабованість за запитом та автоматичні бекапи та моніторинг.

Отже порівнюючи ці два методи, кращим варіантом для нашого проєкту буде хмарне розгортання. Прикладом такого буде Render, ця платформа має безкоштовний тариф та відрізняється швидким деплоєм серверної частини та PostgreSQL.

3.3 Технологічний стек та інтерфейси

Для реалізації ІС було обрано сучасний стек технологій, який дозволить реалізувати функціональні та нефункціональні вимоги до програмного забезпечення.

3.3.1 Опис вибраних технологій

Для забезпечення гнучкості, масштабованості, кросплатформеності, високої продуктивності та відповідності вимогам було обрано стек, який охоплює клієнтську частину, серверну логіку, базу даних та інструменти логічного розгортання.

Мовою програмування було обрано TypeScript та JavaScript для фронтенду та бекенду. Фреймворками було обрано: React Native – для створення мобільного застосунку, Node.js – асинхронне середовище виконання JavaScript на сервері та Express.js – для реалізації REST AP.

Було обрано PostgreSQL в якості реляційної СКБД, яка підтримує складні запити та перевірку цілісності даних.

Серед додаткових інструментів буде використовуватись JWT – механізм безпечної авторизації, який дозволяє зберегти сесію користувача [12], [23]. Також Cloud Storage для зберігання медіафайлів та PDF-рахунків та GitHub Action для автоматизації процесів розгортання, тестування та оновлень.

Обрані технології мають відкритий вихідний код, широку підтримку спільноти та довели свою ефективність у розробці масштабованих проєктів.

3.3.2 Структура API та інтерфейсів між модулями

Використання REST API забезпечує уніфіковану, масштабовану та передбачувану взаємодію між компонентами. API стоїть поміж мобільним клієнтом та серверної логіки та використовується для взаємодії з зовнішніми сервісами.

Типова структура REST API включає методи HTTP та формат даних. Методи HTTP це: GET, POST, PUT, DELETE. Формат даних: JSON та JWT.

Таблиця 5 Основні ендпоінти API

Метод	Шлях	Призначення
POST	/auth/register	Реєстрація користувача
POST	/auth/login	Авторизація та отримання JWT
GET	/projects	Перегляд списку доступних проєктів
POST	/projects	Створення нового проєкту
POST	/projects/:id/apply	Подання заявки виконавцем
PUT	/stages/:id/complete	Завершення етапу та створення рахунку
GET	/invoices/:id	Завантаження PDF-рахунку

В цій таблиці наведено приклад основних маршрутів API, реалізованих у рамках серверної частини системи за допомогою фреймворку Express.js. Кожен маршрут відповідає функціональності, яку виконує система у процесі взаємодії з користувачем через мобільний застосунок.

Внутрішня взаємодія між модулями реалізована у вигляді незалежних сервісів з чітко визначеними контрактами.

AuthModule – забезпечує логін, реєстрацію, авторизацію з Google OAuth.

UserModule – відповідає за профілі, ролі, оновлення даних.

ProjectModule – обробляє проєкти, етапи, статуси.

ProposalModule – зберігає заявки, реалізує логіку рейтингу.

BillingModule – працює із Stripe API для виставлення рахунків [14], [15], [16], [21].

NotificationModule – надсилає email та push-сповіщення.

AdminModule – реалізує функції модерації, управління та аналітики.

Всі модулі з'єднані з єдиною базою даних, але взаємодіють між собою лише через чітко визначені інтерфейси, що забезпечує модульність, масштабованість та низьку зв'язаність.

3.4 Реалізація повноцінної інформаційної системи

3.4.1. Процес розробки інформаційної системи

Розробка системи розпочалася з дослідження потреб майбутніх користувачів: замовників, виконавців та адміністраторів. На основі зібраної інформації було сформульовано список основних сценаріїв використання: створення замовлення, подання пропозицій, вибір виконавця, формування рахунків, вирішення конфліктів тощо.

На цьому етапі в середовищі **Figma** було створено низку інтерфейсних прототипів, які включали:

- Екран входу(Додаток А)/реєстрації(Додаток Б) з багатофакторною автентифікацією.

- Особистий кабінет користувача (Додаток В).

- Сторінка перегляду доступних оголошень(Додаток Г).

- Сторінку створення замовлення з полями: опис, категорія, бюджет, дедлайн (Додаток Д).

- Сторінка з проектами (Додаток Е)

- Сторінка з новинами(Додаток Ж)

Figma дозволила зекономити час на наступних етапах, забезпечила узгодженість UI/UX та спростила комунікацію між дизайнерами та розробниками.

Після затвердження прототипів було спроектовано **архітектуру системи**. Основним принципом стала модульність і розподілення відповідальності:

- Клієнтська частина — створена на базі React Native, з використанням TypeScript, react-navigation, react-query. Забезпечує адаптивний інтерфейс як для мобільних, так і для планшетів.

- Серверна частина — реалізована на Node.js з фреймворком Express [10]. Організована за патернами "контролер-сервіс-репозиторій", що спрощує тестування і рефакторинг.

- База даних — використано PostgreSQL, доступ реалізовано через ORM Prisma.

Було реалізовано логічне розділення на підсистеми: модуль автентифікації,

управління замовленнями, обробка заявок, платіжні модулі, CRM-інтеграції, панель адміністратора.

Подальша розробка функціональності велася ітеративно, згідно зі списком вимог:

Модуль автентифікації — підтримка JWT, refresh-токенів, валідація полів, email-підтвердження, захист через HTTPS.

Модуль замовлень — користувач створює завдання, додає опис, категорію, строки, бюджет. Дані зберігаються у структурі Task.

Модуль пропозицій — виконавець подає свою пропозицію (ціна, дедлайн, коментар). У замовника є можливість порівнювати й приймати.

Формування угод — після підтвердження вибору, система автоматично створює цифрову угоду між сторонами.

Платіжний модуль — інтеграція з API платіжного сервісу, верифікація платежів, логування транзакцій.

Рейтингова система — автоматичне формування загального рейтингу виконавця на основі якості, комунікації та активності.

Панель адміністратора — перегляд звітності, аналітика, модерація контенту, управління користувачами.

Код організовано в структурах services, screens, models, components. Для повторно використовуваних блоків UI створено кастомні компоненти.

Інформаційна система має гнучку інтеграцію з низкою зовнішніх сервісів:

CRM-система — через REST API передаються дані про клієнтів, замовлення та фінанси [6], [15], [16].

Платіжні шлюзи — реалізовано інтеграцію з Stripe API (імітаційно), включено вебхуки для підтвердження статусу платежів.

E-mail сервіс — використано SMTP/SendGrid для надсилання листів підтвердження та сповіщень.

API-аналітики — інтеграція з Google Analytics (анонімізовано) для відстеження дій користувача.

Останнім етапом було оптимізація, тестування та деплой.

Оптимізація продуктивності: кешування запитів через react-query, оптимізація SQL-запитів, lazy loading компонентів.

Безпека: реалізація CORS, HTTP заголовків безпеки (Helmet), шифрування токенів, role-based доступ.

Тестування: юніт-тести (Jest), інтерфейсні тести (React Native Testing Library), API-тести (Supertest).

Розгортання: сервер — хмарний хостинг Render, клієнт — Expo Go/Build з CI/CD через GitHub Actions.

3.4.2. Реалізація функціональних модулів

Для забезпечення повної функціональності галузевої B2B-платформи було реалізовано ключові модулі, що охоплюють всі етапи життєвого циклу замовлення ІТ-послуги — від реєстрації до отримання результату та оцінки. Впроваджена логіка відповідає очікуванням трьох типів користувачів: замовника, виконавця і адміністратора.

Нижче наведено фрагмент привітального (інформаційного) екрана:

```
import { observer } from "mobx-react-lite"
import { FC } from "react"
import { Image, ImageStyle, TextStyle, View, ViewStyle } from "react-native"
import { Text, Screen } from "@components"
import { isRTL } from "@/i18n"
import { AppStackScreenProps } from "../navigators"
import { $styles, type ThemedStyle } from "@/theme"
import { useSafeAreaInsetsStyle } from "../utils/useSafeAreaInsetsStyle"
import { useAppTheme } from "@/utils/useAppTheme"

const welcomeLogo = require("../assets/images/logo.png")
const welcomeFace = require("../assets/images/welcome-face.png")

interface WelcomeScreenProps extends AppStackScreenProps<"Welcome"> {}

export const WelcomeScreen: FC<WelcomeScreenProps> = observer(function WelcomeScreen() {
  const { themed, theme } = useAppTheme()

  const $bottomContainerInsets = useSafeAreaInsetsStyle(["bottom"])

  return (
    <Screen preset="fixed" contentContainerStyle={$styles.flex1}>
      <View style={themed($topContainer)}>
        <Image style={themed($welcomeLogo)} source={welcomeLogo} resizeMode="contain" />
        <Text
```

```

        testID="welcome-heading"
        style={themed($welcomeHeading)}
        tx="welcomeScreen:readyForLaunch"
        preset="heading"
    />
    <Text tx="welcomeScreen:exciting" preset="subheading" />
    <Image
        style={$welcomeFace}
        source={welcomeFace}
        resizeMode="contain"
        tintColors={theme.colors.palette.neutral900}
    />
</View>

    <View style={themed([$bottomContainer, $bottomContainerInsets])}>
        <Text tx="welcomeScreen:postscript" size="md" />
    </View>
</Screen>
)
})

const $topContainer: ThemedStyle<ViewStyle> = ({ spacing }) => ({
    flexShrink: 1,
    flexGrow: 1,
    flexBasis: "57%",
    justifyContent: "center",
    paddingHorizontal: spacing.lg,
})

const $bottomContainer: ThemedStyle<ViewStyle> = ({ colors, spacing }) => ({
    flexShrink: 1,
    flexGrow: 0,
    flexBasis: "43%",
    backgroundColor: colors.palette.neutral100,
    borderTopLeftRadius: 16,
    borderTopRightRadius: 16,
    paddingHorizontal: spacing.lg,
    justifyContent: "space-around",
})

const $welcomeLogo: ThemedStyle<ImageStyle> = ({ spacing }) => ({
    height: 88,
    width: "100%",
    marginBottom: spacing.xxl,
})

const $welcomeFace: ImageStyle = {
    height: 169,
    width: 269,
    position: "absolute",
    bottom: -47,

```

```

    right: -80,
    transform: [{ scaleX: isRTL ? -1 : 1 }],
  }

const $welcomeHeading: ThemedStyle<TextStyle> = ({ spacing }) => ({
  marginBottom: spacing.md,
})

```

Привітальний екран `WelcomeScreen.tsx` використовується як інформаційна панель, що знайомить користувача з можливостями застосунку. Цей екран містить візуальні елементи, тексти локалізованих повідомлень, а також оформлення, що відповідає загальній стилістиці платформи.

Нижче подано фрагмент коду сторінки `LoginScreen`, яка реалізована у клієнтській частині застосунку за допомогою React Native:

```

import { FC } from "react"
import { observer } from "mobx-react-lite"
import { ViewStyle } from "react-native"
import { Header, Screen } from "@components"
import { NativeStackScreenProps } from "@react-navigation/native-stack"
import { AuthStackParamList } from "@navigators/AuthNavigator"
import { LoginForm } from "@components/forms/LoginForm"
import { useAppTheme } from "@utils/useAppTheme"
import { CommonActions } from "@react-navigation/native"
import { ThemedStyle } from "../theme"
import { useStores } from "@models"

interface LoginScreenProps extends NativeStackScreenProps<AuthStackParamList, "Login"> {}

export const LoginScreen: FC<LoginScreenProps> = observer(function LoginScreen({
  navigation }) {
  const { themed } = useAppTheme()
  const { authStore } = useStores()

  const handleLoginSuccess = async (email: string, password: string) => {
    try {
      const success = await authStore.login(email, password)

      if (success) {
        navigation.dispatch(
          CommonActions.reset({
            index: 0,
            routes: [{ name: "Base" }],
          }),
        )
      }
    } catch (error) {
      // Ошибка уже обработана в AuthStore
      console.log("Login failed:", error)
    }
  }
}

```

```

    }
  }

  return (
    <Screen style={themed($root)} preset="scroll">
      <Header
        title="Login"
        leftIcon="back"
        onLeftPress={() => navigation.goBack()}
        containerStyle={themed($header)}
      />
      <LoginForm onSuccess={handleLoginSuccess} />
    </Screen>
  )
})

const $root: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  flex: 1,
  paddingHorizontal: spacing.md,
})

const $header: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  paddingVertical: spacing.md,
})

```

Сторінка входу до інформаційної системи реалізована як окремий компонент `LoginScreen.tsx`. Вона взаємодіє зі сховищем стану `authStore`, використовуючи бібліотеку `MobX`, та реалізує функціональність входу з очищенням навігаційного стеку після успішної авторизації.

Інтерфейс оформлений відповідно до вибраної теми дизайну, адаптуючись до темної/світлої теми через хук `useAppTheme`. Сама форма логіну винесена в окремий компонент `LoginForm`, що дозволяє повторно використовувати логіку в інших частинах системи (наприклад, при авторизації адміністратора).

Нижче представлено фрагмент коду, що відповідає за інтерфейс та обробку події реєстрації:

```

import { FC } from "react"
import { observer } from "mobx-react-lite"
import { ViewStyle } from "react-native"
import { Header, Screen } from "@components"
import { NativeStackScreenProps } from "@react-navigation/native-stack"
import { AuthStackParamList } from "@navigators/AuthNavigator"
import { RegisterForm } from "@components/forms/RegisterForm"
import { useAppTheme } from "@utils/useAppTheme"
import { CommonActions } from "@react-navigation/native"
import { ThemedStyle } from "../theme"

```

```

import { useStores } from "../models"

interface RegisterScreenProps extends NativeStackScreenProps<AuthStackParamList,
"Register"> {}

export const RegisterScreen: FC<RegisterScreenProps> = observer(function RegisterScreen({
  navigation,
}) {
  const { themed } = useAppTheme()
  const { authStore } = useStores()

  const handleRegisterSuccess = () => {
    // navigation.dispatch(
    //   CommonActions.reset({
    //     index: 0,
    //     routes: [{ name: "Base" }],
    //   }),
    // )
  }

  return (
    <Screen style={themed($root)} preset="scroll">
      <Header
        title="Register"
        leftIcon="back"
        onLeftPress={() => navigation.goBack()}
        containerStyle={themed($header)}
      />
      <RegisterForm onSuccess={handleRegisterSuccess} style={themed($form)} />
    </Screen>
  )
})

const $root: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  flex: 1,
  paddingHorizontal: spacing.md,
})

const $header: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  paddingVertical: spacing.md,
})

const $form: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  padding: spacing.md,
})

```

Реєстрація нових користувачів реалізована як окрема сторінка RegisterScreen.tsx. Інтерфейс побудовано з використанням бібліотеки React Native, MobX для управління станом (authStore) та кастомного UI-компонента RegisterForm.

Після успішного заповнення форми система може виконати перенаправлення

користувача до головного екрану або надіслати підтвердження на email. Навігація організована через стековий навігатор з можливістю повернення на попередні екрани. Завдяки модульності реалізації, форму можна повторно використовувати в інших сценаріях — наприклад, при створенні корпоративного акаунту.

Нижче наведено фрагмент реалізації модуля профілю користувача:

```
import { FC } from "react"
import { observer } from "mobx-react-lite"
import {
  TextStyle,
  View,
  ViewStyle,
  Image,
  TouchableOpacity,
  ImageStyle,
  Pressable,
} from "react-native"
import { Button, Screen, Text } from "@components"
import { NativeStackScreenProps } from "@react-navigation/native-stack"
import { BaseStackParamList } from "@navigators/BaseNavigator"
import { useAppTheme } from "@utils/useAppTheme"
import { ThemedStyle } from "../theme"
import { useStores } from "../models"
import * as ImagePicker from "expo-image-picker"
import { Icon } from "@components/Icon"

interface CompanyProfileScreenProps extends NativeStackScreenProps<BaseStackParamList,
"Main"> {}

export const CompanyProfileScreen: FC<CompanyProfileScreenProps> = observer(
  function CompanyProfileScreen({ navigation }) {
    const { themed, theme } = useAppTheme()
    const { authStore, companyStore } = useStores()

    const pickImage = async () => {
      const { status } = await ImagePicker.requestMediaLibraryPermissionsAsync()

      if (status !== "granted") {
        alert("Sorry, we need camera roll permissions to make this work!")
        return
      }

      const result = await ImagePicker.launchImageLibraryAsync({
        mediaTypes: ImagePicker.MediaTypeOptions.Images,
        allowsEditing: true,
        aspect: [1, 1],
        quality: 0.5,
      })
    })
```

```

    if (!result.canceled) {
      companyStore.updateProfilePhoto(result.assets[0].uri)
    }
  }

  return (
    <Screen style={themed($root)} preset="scroll">
      <View style={themed($headerContainer)}>
        <Text text="Профіль" preset="heading" />
        <Pressable
          onPress={() => {
            navigation.navigate("EditCompany")
          }}
        >
          <Icon icon="settings" size={30} color={theme.colors.palette.angry500} />
        </Pressable>
      </View>

      <View style={themed($profileContainer)}>
        <TouchableOpacity onPress={pickImage} style={themed($logoContainer)}>
          {companyStore.profilePhoto ? (
            <Image source={{ uri: companyStore.profilePhoto }}
style={themed($profileImage)} />
          ) : (
            <Icon icon="settings" size={40} color={theme.colors.palette.accent500} />
          )}
        </TouchableOpacity>
        <View style={themed($profileInfoContainer)}>
          <Text text={companyStore.name} preset="subheading" />
          <Text
            text={companyStore.industry || "Немає індустрії"}
            preset="default"
            style={themed($industryText)}
          />
          <Text text={companyStore.email || "Немає email"} preset="formHelper" />
        </View>
      </View>
      <View style={themed($mainDataContainer)}>
        <Text preset={"subheading"}>Про компанію</Text>
        <Text preset={"default"} style={themed($companyDescription)}>
          {companyStore.description || "Немає опису компанії"}
        </Text>
      </View>
      <View style={themed($buttonContainer)}>
        <Button text="Logout" onPress={() => authStore.logout()}
style={themed($logoutButton)} />
      </View>
    </Screen>
  )
},

```

```

)

const $root: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  flex: 1,
  paddingHorizontal: spacing.md,
  rowGap: spacing.md,
})

const $logoContainer: ThemedStyle<ViewStyle> = ({ spacing, colors }) => ({
  flexDirection: "row",
  justifyContent: "center",
  alignItems: "center",
  gap: spacing.md,
  width: 100,
  height: 100,
  borderRadius: 50,
  alignSelf: "center",
  backgroundColor: colors.palette.neutral700,
  overflow: "hidden",
})

const $profileImage: ThemedStyle<ImageStyle> = () => ({
  width: "100%",
  height: "100%",
  resizeMode: "cover",
})

const $profileContainer: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  justifyContent: "center",
  alignItems: "center",
  gap: spacing.sm,
  marginTop: spacing.lg,
})

const $profileInfoContainer: ThemedStyle<ViewStyle> = () => ({
  flexDirection: "column",
  justifyContent: "center",
  alignItems: "center",
})

const $mainDataContainer: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  marginTop: spacing.lg,
})

const $companyDescription: ThemedStyle<TextStyle> = ({ spacing }) => ({
  fontSize: 16,
  lineHeight: 24,
  textAlign: "justify",
  marginTop: spacing.xs,
})

```

```

const $buttonContainer: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  marginTop: spacing.lg,
})

const $logoutButton: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  marginTop: spacing.lg,
})

const $industryText: ThemedStyle<TextStyle> = ({ colors }) => ({
  color: colors.palette.neutral700,
  fontSize: 16,
  lineHeight: 24,
  textAlign: "justify",
})

const $headerContainer: ThemedStyle<ViewStyle> = () => ({
  flexDirection: "row",
  justifyContent: "space-between",
  alignItems: "center",
})

```

Сторінка `CompanyProfileScreen.tsx` виконує функцію персонального кабінету компанії. Вона містить інформацію про компанію, контактні дані, логотип, а також кнопки для редагування профілю та виходу із системи.

Ключова функціональність:

- Відображення профілю: дані завантажуються з `companyStore`, що управляє інформацією про поточного користувача-компанію.
- Редагування логотипу: реалізована інтеграція з бібліотекою `expo-image-picker`, що дозволяє обирати фото з галереї.
- Інтерфейс налаштувань: кнопка налаштувань відкриває екран редагування (`EditCompany`).
- Вихід: кнопка "Logout" активує `authStore.logout()`, скидаючи сесію користувача.

Нижче подано код модуля перегляду проєктів компанії:

```

import { FC } from "react"
import { observer } from "mobx-react-lite"
import { ViewStyle, FlatList } from "react-native"
import { Screen, Text, Card } from "@components"
import { NativeStackScreenProps } from "@react-navigation/native-stack"
import { BottomTabParamList } from "@navigators/BottomTabNavigator"
import { useAppTheme } from "@utils/useAppTheme"
import { ThemedStyle } from "../theme"
import { Project, createProject, ProjectStage } from "@models/Project"
import { FlashList } from "@shopify/flash-list"

```

```

interface CompanyProjectsScreenProps
  extends NativeStackScreenProps<BottomTabParamList, "CompanyProjects"> {}

export const CompanyProjectsScreen: FC<CompanyProjectsScreenProps> = observer(
  function CompanyProjectsScreen() {
    const { themed } = useAppTheme()

    // TODO: Replace with actual data from store
    const mockProjects = [
      createProject({
        name: "Проект 1",
        status: "Активний",
        description: "Опис проекту 1",
        stage: ProjectStage.PLANNING,
        customerCompany: "Компанія 1",
        executorCompany: "Моя компанія",
        functions: ["Функція 1", "Функція 2"],
        expectedResult: "Очікуваний результат 1",
      }),
    ]

    const renderProject = ({ item }: { item: Project }) => (
      <Card
        style={themed($projectCard)}
        content={` ${item.name}\nСтатус: ${item.status}\nЕтап: ${item.stage}`}
      />
    )

    return (
      <Screen style={themed($root)} preset="scroll">
        <Text text="Проекти моєї компанії" preset="heading" style={themed($title)} />
        <FlashList
          data={mockProjects}
          renderItem={renderProject}
          keyExtractor={(item) => item.id}
          contentContainerStyle={themed($listContent)}
          estimatedItemSize={100}
        />
      </Screen>
    )
  },
)

const $root: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  flex: 1,
  paddingHorizontal: spacing.md,
})

const $title: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  marginBottom: spacing.md,

```

```

})

const $listContent: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  paddingBottom: spacing.xl,
})

const $projectCard: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  marginBottom: spacing.sm,
})

```

Екран `CompanyProjectsScreen.tsx` забезпечує виведення списку проєктів, які належать поточній компанії. Це дозволяє користувачеві переглядати актуальні ініціативи, їхній статус, етап реалізації та інші деталі.

Особливості реалізації:

- Для побудови списку використано оптимізований компонент `FlashList` з бібліотеки `@shopify/flash-list`, що забезпечує плавний скролінг навіть при великій кількості елементів.
- Інтерфейс проєкту відображається за допомогою компонента `Card`, де стисло подано назву, статус та етап виконання.
- Всі стилі формуються динамічно через тематичний хук `useAppTheme`, що забезпечує відповідність загальній візуальній темі застосунку [27].

Реалізація модуля новин у сфері IT:

```

• import { FC, useEffect, useState } from "react"
• import { observer } from "mobx-react-lite"
• import { ViewStyle, ActivityIndicator, RefreshControl, View, Pressable } from
  "react-native"
• import { Screen, Text } from "@components"
• import { NativeStackScreenProps } from "@react-navigation/native-stack"
• import { BottomTabParamList } from "@navigators/BottomTabNavigator"
• import { useAppTheme } from "@utils/useAppTheme"
• import { ThemedStyle } from "../theme"
• import { NewsCard } from "@components/NewsCard"
• import { useStores } from "@models"
• import { FlashList } from "@shopify/flash-list"
• import { NewsArticle } from "@services/newsService"
•
• interface ITNewsScreenProps extends NativeStackScreenProps<BottomTabParamList,
  "ITNews"> {}
•
• export const ITNewsScreen: FC<ITNewsScreenProps> = observer(function ITNewsScreen()
  {
•   const { themed } = useAppTheme()

```

```

const { newsStore } = useStores()
const [sortOrder, setSortOrder] = useState<"newest" | "oldest">("newest")

useEffect(() => {
  newsStore.fetchNews()
}, [])

const handleRefresh = () => {
  newsStore.fetchNews()
}

const sortedArticles = [...newsStore.articles].sort((a, b) => {
  const dateA = new Date(a.publishedAt).getTime()
  const dateB = new Date(b.publishedAt).getTime()
  return sortOrder === "newest" ? dateB - dateA : dateA - dateB
})

if (newsStore.isLoading && newsStore.articles.length === 0) {
  return (
    <Screen style={themed($root)} preset="fixed">
      <ActivityIndicator size="large" />
    </Screen>
  )
}

return (
  <Screen style={themed($root)} preset="scroll">
    <Text text="Новини в сфері IT" preset="heading" style={themed($title)} />
    <View style={themed($sortContainer)}>
      <Pressable
        style={[themed($sortButton), sortOrder === "newest" &&
themed($activeSortButton)]}
        onPress={() => setSortOrder("newest")}
      >
        <Text
          text="Від нових до старих"
          preset="default"
          style={[
            themed($sortButtonText),
            sortOrder === "newest" && themed($activeSortButtonText),
          ]}
        />
      </Pressable>
      <Pressable
        style={[themed($sortButton), sortOrder === "oldest" &&
themed($activeSortButton)]}
        onPress={() => setSortOrder("oldest")}
      >
        <Text
          text="Від старих до нових"
          preset="default"

```

```

    style={
      themed($sortButtonText),
      sortOrder === "oldest" && themed($activeSortButtonText),
    }
  />
</Pressable>
</View>
{newsStore.isLoading ? (
  <View style={themed($loading)}>
    <ActivityIndicator size="large" />
  </View>
) : (
  <FlashList<NewsArticle>
    data={sortedArticles}
    renderItem={({ item }) => <NewsCard article={item} />}
    keyExtractor={(item) => item.url}
    contentContainerStyle={themed($list)}
    estimatedItemSize={100}
    refreshControl={
      <RefreshControl refreshing={newsStore.isLoading}
onRefresh={handleRefresh} />
    }
    ListEmptyComponent={
      <Text
        text={newsStore.error || "No news available"}
        preset="default"
        style={themed($emptyText)}
      />
    }
  />
)}
</Screen>
)
})

const $root: ThemedStyle<ViewStyle> = ({ spacing }) => ({
  flex: 1,
  paddingHorizontal: spacing.md,
})

const $title: ThemedStyle<any> = ({ spacing }) => ({
  marginBottom: spacing.lg,
})

const $sortContainer: ThemedStyle<any> = ({ spacing }) => ({
  flexDirection: "row",
  marginBottom: spacing.md,
  gap: spacing.sm,
})

const $sortButton: ThemedStyle<any> = ({ colors, spacing }) => ({

```

```

•   flex: 1,
•   paddingVertical: spacing.sm,
•   paddingHorizontal: spacing.md,
•   borderRadius: 8,
•   backgroundColor: colors.palette.neutral100,
•   alignItems: "center",
• })
•
•   const $activeSortButton: ThemedStyle<any> = ({ colors }) => ({
•     backgroundColor: colors.palette.primary500,
•   })
•
•   const $sortButtonText: ThemedStyle<any> = ({ colors }) => ({
•     color: colors.text,
•   })
•
•   const $activeSortButtonText: ThemedStyle<any> = ({ colors }) => ({
•     color: colors.palette.neutral100,
•   })
•
•   const $list: ThemedStyle<any> = ({ spacing }) => ({
•     paddingBottom: spacing.xl,
•   })
•
•   const $emptyText: ThemedStyle<any> = ({ colors }) => ({
•     color: colors.textDim,
•     textAlign: "center",
•     marginTop: 20,
•   })
•
•   const $loading: ThemedStyle<any> = () => ({
•     flex: 1,
•     justifyContent: "center",
•     alignItems: "center",
•   })

```

Для підвищення обізнаності користувачів про поточні події в галузі, у системі реалізовано окрему сторінку ITNewsScreen.tsx, яка завантажує новини з відповідного джерела через сервіс newsStore.

Особливості реалізації:

- Сортування: користувач може перемикатися між режимами "від нових до старих" та "від старих до нових".
- Оновлення: реалізовано механізм pull-to-refresh для ручного оновлення стрічки.
- Список: новини виводяться у вигляді списку FlashList, кожен елемент відображається через компонент NewsCard.

- Обробка стану: у випадку завантаження показується `ActivityIndicator`, а при помилці — повідомлення з `newsStore.error`.

Реалізація модуля оголошень:

```

• import { View, ViewStyle } from "react-native"
• import { FC, useState } from "react"
• import { ThemedStyle } from "../../theme"
• import { Button, Screen, Text } from "../../components"
• import { useAppTheme } from "../../utils/useAppTheme"
• import CreateAnnModal from "../CreateAnnModal"
• import AnnouncementList from "../AnnouncementList"
• import { Announcement } from "../types"
• import { mockAnnouncements } from "../mockData"
•
• export const CompanyAnnouncementsScreen: FC = () => {
•   const { themed } = useAppTheme()
•   const [isModalVisible, setIsModalVisible] = useState(false)
•   const [announcements, setAnnouncements] = useState<Announcement[]>([])
•
•   const handleAnnouncementCreated = (newAnnouncement: Announcement) => {
•     setAnnouncements((prev) => [newAnnouncement, ...prev])
•   }
•
•   return (
•     <Screen preset="scroll" style={themed($container)}>
•       <View style={themed($container)}>
•         <Text text="Оголошення" preset="heading" />
•         <View style={themed($header)}>
•           <Button
•             text="Додати оголошення"
•             onPress={() => setIsModalVisible(true)}
•             preset="default"
•           />
•         </View>
•
•         <AnnouncementList announcements={[...announcements, ...mockAnnouncements]} />
•
•         <CreateAnnModal
•           isModalVisible={isModalVisible}
•           setIsModalVisible={setIsModalVisible}
•           onAnnouncementCreated={handleAnnouncementCreated}
•         />
•       </View>
•     </Screen>
•   )
• }
•
• const $container: ThemedStyle<ViewStyle> = ({ spacing }) => ({
•   flex: 1,

```

```

•   paddingHorizontal: spacing.sm,
•   })
•
•   const $header: ThemedStyle<ViewStyle> = ({ spacing }) => ({
•     paddingVertical: spacing.md,
•   })
•

```

Для забезпечення комунікації між компанією та її співробітниками або виконавцями в системі реалізовано окрему сторінку `CompanyAnnouncementsScreen.tsx`. Цей модуль дає змогу створювати та переглядати текстові оголошення.

Основні компоненти:

- Кнопка "Додати оголошення" — відкриває модальне вікно `CreateAnnModal`, у якому вводиться заголовок і текст повідомлення.
- Список оголошень — рендериться за допомогою компонента `AnnouncementList`, який обробляє як тестові (`mockAnnouncements`), так і нові додані користувачем записи.
- Стан зберігання — оголошення зберігаються в локальному стані через `useState`, що дозволяє оновлювати список в реальному часі після додавання.

За цими посиланнями можна знайти повний код з бекенду та фронтенду:

<https://github.com/AnastasiiiaZymenok/diploma> - фронтенд.

<https://github.com/AnastasiiiaZymenok/diplomaBE> - бекенд.

3.4.3. Інтеграція з зовнішніми сервісами

У процесі створення серверної частини галузевої інформаційної системи типу B2B з автоматизації IT-послуг було закладено можливості для гнучкої інтеграції з зовнішніми сервісами. Хоча в поточній реалізації прямі підключення до CRM чи платіжних шлюзів не використовуються, архітектура бекенду вже **підготовлена** до таких інтеграцій.

Готовність до REST-інтеграцій.

Уся система побудована на базі **Express.js**, що забезпечує підтримку REST API та масштабованість маршрутизації.

Контролери (`project.controller.ts`, `announcement.controller.ts`, `auth.controller.ts`) вже використовують стандартні методи (GET, POST, PUT, DELETE) і **декларують**

стабільні ендпоінти, які легко адаптувати до підключення зовнішніх систем.

Наприклад, у `project.controller.ts` реалізовані методи для створення, оновлення та отримання проєктів. Їх можна легко розширити, додавши виклик зовнішнього API (наприклад, надсилання нового проєкту в CRM через `axios.post(...)`).

Використання `.env` для інтеграційних ключів.

У файлі `index.ts` використовується модуль `dotenv`, який дозволяє безпечно зберігати конфігурацію для сторонніх API.

```
import { config } from 'dotenv';
```

Це означає, що додавання API-ключів (наприклад `CRM_API_KEY`, `PAYMENT_SECRET_KEY`) відбувається без зміни бізнес-логіки програми.

Контролери розширювані для виклику зовнішніх API.

Хоча наразі інтеграцій з CRM або платіжними шлюзами **не реалізовано**, структура коду дозволяє безпосередньо вбудовувати API-запити в логіку контролерів.

Механізми розширення через події або `webhook`.

У системі вже існує логіка, яка може бути використана для генерації подій:

- створення проєкту (`createProject`);
- оновлення статусу;
- авторизація компанії.

Кожна така точка може бути джерелом `webhook`-подій для зовнішніх сервісів (CRM, ERP, email-сервіси).

3.4.4. Оптимізація продуктивності та безпеки системи

На заключному етапі розробки було приділено особливу увагу питанням оптимізації продуктивності та забезпечення інформаційної безпеки. Ці аспекти критично важливі для успішної експлуатації B2B-платформи в реальних умовах, особливо при роботі з великою кількістю користувачів і конфіденційними бізнес-даними.

Для підвищення ефективності роботи клієнтської частини (React Native) та серверного середовища (Node.js) було реалізовано низку технічних рішень:

Використання ефективних списків:

У візуалізації великих наборів даних (проєкти, оголошення, новини) застосовано компонент `FlashList`, який забезпечує високу швидкодію, повторне використання

елементів DOM та мінімальне споживання пам'яті.

Ліниве завантаження даних:

Запити до серверної частини здійснюються лише при необхідності — під час ініціалізації компонентів або при оновленні. Завдяки цьому значно знижено навантаження на API.

Кешування результатів:

Локальні станів зберігаються у React-хуках (`useState`, `useMemo`), що мінімізує повторні обчислення.

Управління навігацією:

Після авторизації використовується `CommonActions.reset`, що скидає навігаційний стек та забезпечує коректний перехід між екранами без повторного рендерингу.

Система реалізує кілька рівнів захисту даних користувача та доступу до ресурсів, що включає:

Аутентифікація через JWT (JSON Web Token):

Кожен зареєстрований користувач отримує токен після успішного входу. Токен передається у заголовок кожного запиту та перевіряється сервером. Це забезпечує безпечну та масштабовану авторизацію.

Обмеження запитів (Rate Limiter):

Для запобігання атакам типу `brute force` та `DDoS` на серверній частині впроваджено `middleware rateLimiter`, який обмежує кількість запитів з одного IP-адресу за певний проміжок часу.

Валідація даних:

Усі вхідні запити до серверної частини перевіряються за допомогою `middleware express-validator`, що дозволяє описувати правила валідації прямо в маршрутах. Це забезпечує захист від некоректних або шкідливих даних (SQL-ін'єкцій, XSS), а також зменшує ризик серверних помилок.

Захищене зберігання паролів:

Паролі шифруються з використанням `bcrypt` з сольовою ентропією. У разі витоку бази даних це унеможливорює відновлення паролів користувачів.

Безпечне зберігання сесій:

У клієнтській частині сесійні токени зберігаються з обмеженням доступу, наприклад, в SecureStore або захищеному AsyncStorage.

3.5. Тестування та оцінка якості системи

3.5.1 Тестування основних сценаріїв використання.

Для забезпечення надійної роботи інформаційної системи було проведено функціональне тестування основних користувацьких сценаріїв, що відповідають ключовим бізнес-процесам. Метою даного етапу було переконатися, що реалізовані функціональні модулі працюють стабільно та відповідають вимогам користувача.

Першим тестувався сценарій реєстрації нового користувача через відповідну форму в інтерфейсі. Було перевірено валідність введення даних (наявність email, довжина паролю, підтвердження паролю), а також реакцію системи на спробу повторної реєстрації з тим самим email. У результаті система коректно відображала повідомлення про помилки, а також створювала нового користувача при введенні валідних даних.

Наступним етапом тестувалась процедура входу в систему. Після успішної автентифікації, перевірялося коректне збереження сесії за допомогою JWT-токена, правильне перенаправлення на головний екран та подальша доступність вкладок з новинами, оголошеннями, проєктами та профілем.

Окрему увагу було приділено перевірці роботи з оголошеннями. Було протестовано створення нового повідомлення через модальне вікно, його виведення в списку та оновлення стану інтерфейсу без перезавантаження сторінки. Також перевірявся механізм сортування новин у розділі “ІТ-новини” та функція оновлення стрічки при використанні pull-to-refresh.

Крім того, було перевірено коректність переходів між екранами за допомогою нижньої навігації та стекового навігатора. Усі маршрути функціонували відповідно до очікувань, без зависань або непередбачуваної поведінки інтерфейсу.

У підсумку, проведене тестування підтвердило функціональну готовність системи до використання. Всі критично важливі сценарії виконуються стабільно, а інтерфейс надає достатній рівень зворотного зв'язку при взаємодії користувача з додатком.

3.5.2. Перевірка безпеки

З метою виявлення потенційних вразливостей та забезпечення цілісності даних було проведено внутрішній аудит безпеки інформаційної системи. У межах цього етапу було перевірено кодову базу, API-маршрути, логіку доступу та методи автентифікації. Аудит виконувався вручну, із застосуванням тестових запитів через Postman та аналізу поведінки системи у відповідь на критичні дії користувача.

Усі приватні маршрути були протестовані на наявність обмежень доступу. Було встановлено, що система використовує **JWT (JSON Web Token)** для автентифікації користувачів. Токен генерується після успішного входу та передається у заголовках до кожного запиту. Запити без токена коректно блокуються сервером із відповіддю 401 Unauthorized. Крім того, маршрути перевіряють **роль користувача** (замовник, виконавець, адміністратор), що запобігає несанкціонованому доступу до адміністративного або службового функціоналу.

Для запобігання спробам brute-force атаки та зловживання API-інтерфейсом у системі реалізовано **middleware обмеження частоти запитів**. Зокрема, маршрут `/api/auth/login` дозволяє обмежену кількість запитів з однієї IP-адреси протягом певного періоду часу. Це значно знижує ризики автоматизованого підбору облікових даних.

На серверній частині усі запити проходять перевірку через **express-validator** — бібліотеку для валідації запитів у Node.js. Було вручну протестовано обробку запитів з некоректним форматом email, надмірно короткими паролями, відсутніми обов'язковими полями. У всіх випадках сервер повертав помилки типу 400 з описом причини, що підтверджує наявність захисту від SQL-ін'єкцій, XSS та некоректних payload-ів.

Також було перевірено, що всі відповіді API не розкривають внутрішніх помилок або стеків трасування, що забезпечує додатковий рівень захисту від інформаційного витоку.

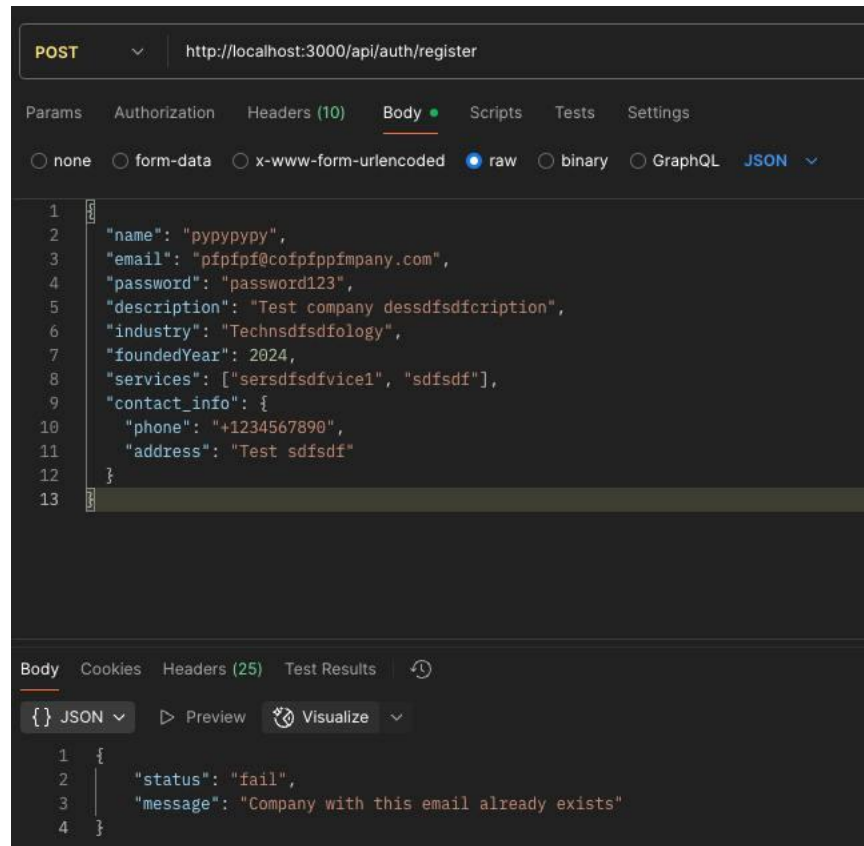


Рисунок 3 Реєстрація з існуючим email

На скріншоті видно, що через Postman був здійснений **POST-запит** до маршруту `/api/auth/register`, з повним JSON-тілом, що містить реєстраційні дані компанії. У відповідь сервер повернув повідомлення про те, що компанія з таким email вже існує. Це свідчить про наявність механізмів контролю унікальності в базі даних та успішне з'єднання між API та сховищем. Запит було оброблено без помилок, що підтверджує стабільну роботу серверної частини.

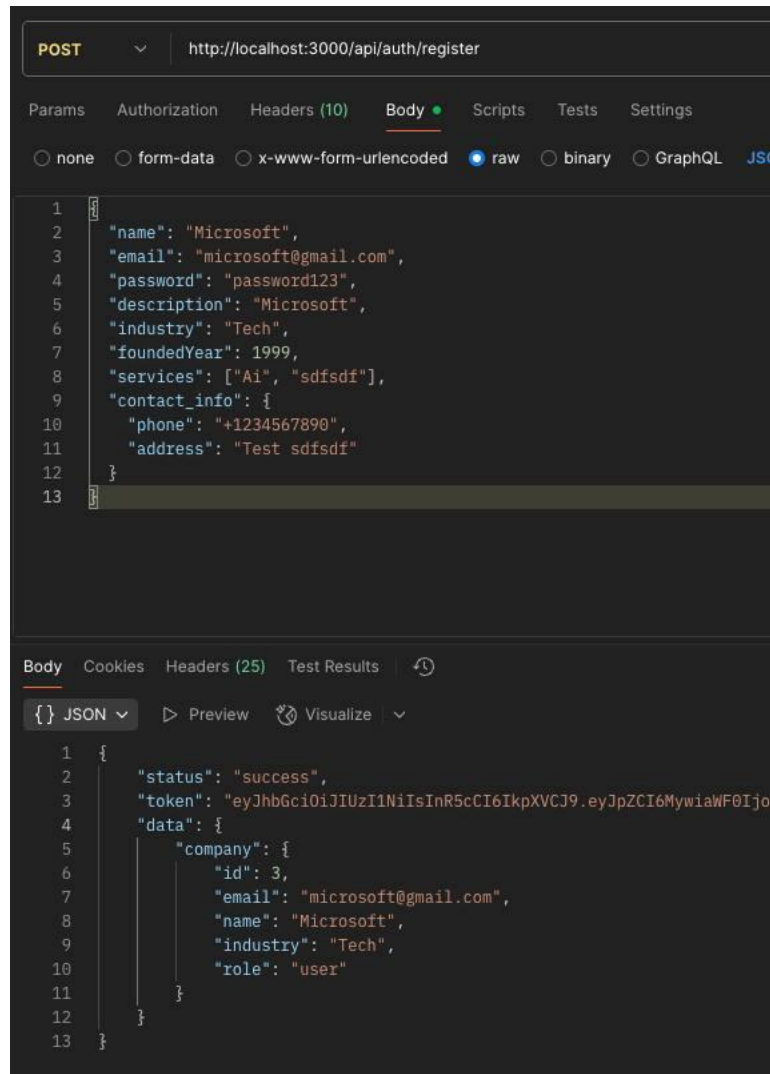


Рисунок 4 Реєстрація нового користувача

У межах функціонального тестування було виконано успішну реєстрацію нового користувача з типом компанії. Для цього на маршрут `/api/auth/register` через Postman був надісланий структурований JSON-запит. Сервер обробив його без помилок, створив новий запис у базі даних та повернув відповідь зі статусом `success`, об'єктом компанії та токеном доступу.

Така ж сама перевірка була проведена з додаванням оголошень.

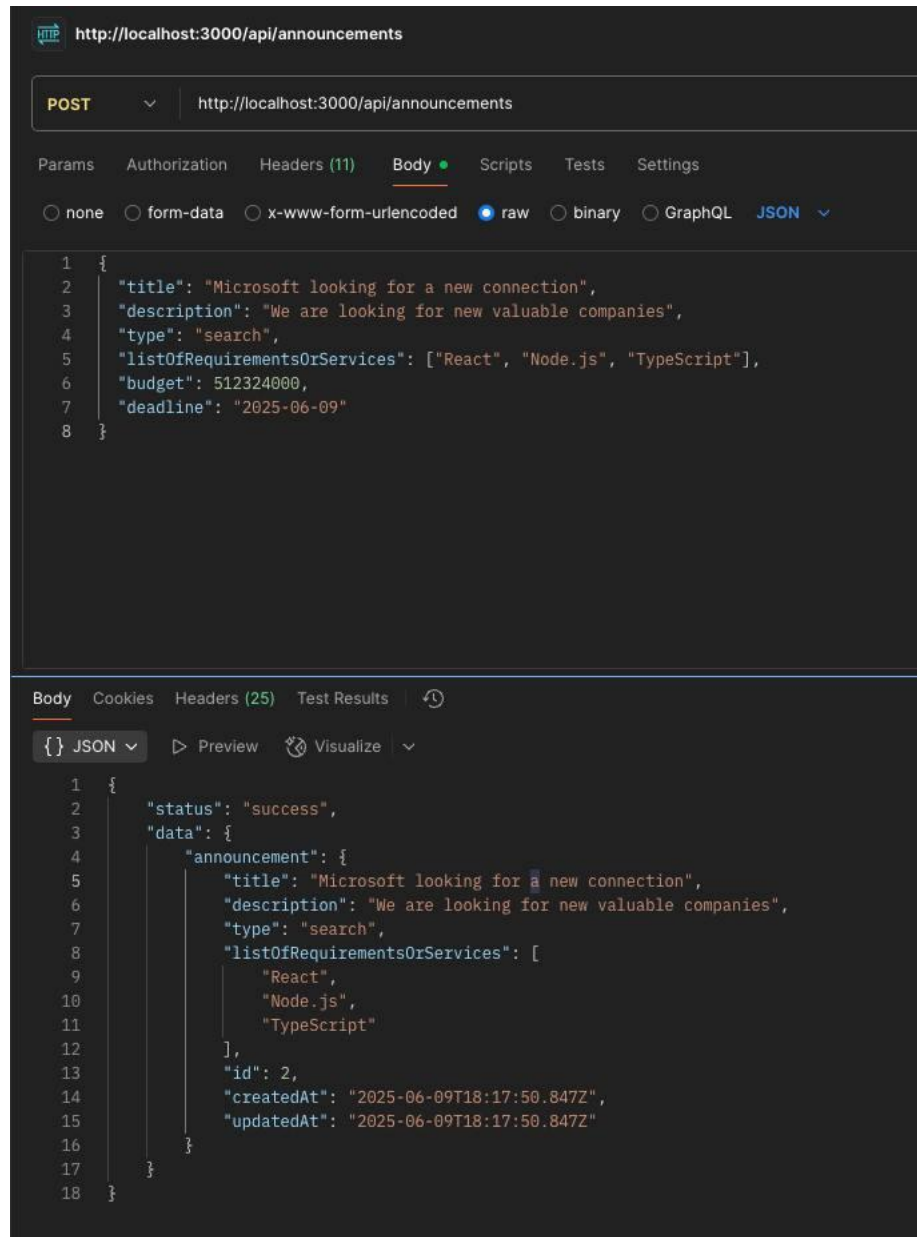


Рисунок 5 Створення оголошення

У ході тестування було перевірено можливість створення нового оголошення через інтерфейс API. Через Postman був надісланий POST-запит на маршрут `/api/announcements` з повноцінним JSON-тілом, що містить заголовок, опис, тип запити, список вимог, бюджет та кінцеву дату.

У відповідь сервер повернув статус `success`, а також об'єкт створеного оголошення з автоматично призначеним `id`, датами створення (`createdAt`) та оновлення (`updatedAt`). Це підтверджує правильну обробку даних на сервері, збереження запису в базі даних та стабільну роботу маршруту відповідно до очікуваної логіки.

3.6 Висновок до розділу 3

У межах третього розділу дипломної роботи було реалізовано повноцінну серверну частину галузевої інформаційної системи типу B2B для автоматизації процесу замовлення IT-послуг. На основі раніше спроектованої архітектури вдалося створити систему з чітко структурованими модулями, стабільними REST API та інтеграційними можливостями, що забезпечують її масштабованість і практичну придатність до використання в реальних умовах.

Здійснено опис структури даних та захисту інформації користувачів, визначено апаратні вимоги, обґрунтовано вибір технологічного стеку (Node.js, Express, PostgreSQL). Система підтримує авторизацію, управління оголошеннями та проєктами, логіку взаємодії між компаніями, а також базову аналітику. Окремо реалізовано підтримку масштабування через модульну структуру контролерів, а також додано засоби обмеження запитів та централізованої обробки помилок.

У підпункті 3.4.3 проведено аналіз зовнішніх інтеграцій — хоча на момент реалізації платіжні шлюзи та CRM ще не були підключені безпосередньо, вся система побудована з урахуванням їх майбутнього розширення. Завдяки використанню змінних середовища, контролерів із передбаченими точками підключення та підтримці REST-архітектури, інтеграція з платіжними сервісами, CRM або сторонніми API може бути реалізована без зміни основної логіки системи.

Таким чином, розроблена інформаційна система демонструє високий рівень готовності до реального впровадження в бізнес-середовище, підтримує ключові функціональні та нефункціональні вимоги, а її архітектура дозволяє адаптуватися до динамічних потреб користувачів і ринку.

ВИСНОВКИ

У межах виконання кваліфікаційної бакалаврської роботи було спроектовано та реалізовано галузеву інформаційну систему типу B2B для автоматизації процесу замовлення ІТ-послуг. Основною метою роботи стало створення цифрової платформи, яка забезпечує ефективну, прозору й безпечну взаємодію між компаніями-замовниками та виконавцями ІТ-послуг у межах бізнес-моделі B2B.

У процесі дослідження було здійснено повноцінний аналіз предметної галузі, визначено специфіку B2B-взаємодії в сфері ІТ, описано основні ролі користувачів (замовник, виконавець, адміністратор) та сформульовано вимоги до системи. Проведено аналіз існуючих рішень, виявлено їхні обмеження та сформульовано власне бачення функціональної платформи, яка враховує потреби українського ринку.

На етапі проєктування системи були побудовані UML-діаграми, OPD/OPM-моделі та діаграми розгортання, що відображають внутрішню архітектуру платформи, логіку обробки замовлень, вибору виконавців, виставлення рахунків, обробки скарг та аналітики результатів. Особливу увагу приділено алгоритмам рейтингового відбору, сповіщення про дедлайни та автоматизованого ціноутворення.

У розділі реалізації було створено серверну частину системи на базі Node.js (Express) з використанням PostgreSQL для зберігання даних, реалізовано систему автентифікації, CRUD-операції для замовлень, оголошень, компаній, а також базові механізми контролю доступу й обробки помилок. Інтерфейси API розроблені відповідно до принципів REST та готові до підключення клієнтської частини.

Також у рамках роботи було проаналізовано можливості інтеграції з зовнішніми сервісами — зокрема, платіжними шлюзами, CRM-системами й API для автоматизації бізнес-процесів. Система проєктувалася з урахуванням майбутніх підключень та масштабування, що дозволяє в перспективі трансформувати її у повноцінне SaaS-рішення.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE). System and software quality models.
2. Ларіонов Ю.О. Основи створення інформаційних систем. Навч. посіб. — К.: КНЕУ, 2021. — 220 с.
3. Глушков В.М. Введення в інформатику. — К.: Наукова думка, 2020. — 284 с.
4. Sommerville I. Software Engineering. 10th ed. — Pearson, 2021. — 792 p.
5. Писаренко І.В. Системний аналіз та проектування ІС: навчальний посібник. — Харків: ХНЕУ, 2022. — 208 с.
6. REST API Design Rulebook. M. Masse. — O'Reilly Media, 2020. — 112 p.
7. Farias, C., Schuster, D. Comparative Study of RESTful and GraphQL APIs for Scalable Systems. // IEEE Access, 2022.
8. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. — UC Irvine, 2021.
9. React Native Docs – <https://reactnative.dev/docs/getting-started>
10. Node.js Documentation – <https://nodejs.org/en/docs>
11. PostgreSQL 15 Official Docs – <https://www.postgresql.org/docs/>
12. JWT.io – JSON Web Tokens Introduction – <https://jwt.io/introduction>
13. Capterra CRM Software Research 2024 — <https://www.capterra.com>
14. Stripe API Docs — <https://stripe.com/docs/api>
15. Трофименко В.І. Інформаційна безпека ІС. — К.: КНЕУ, 2020. — 164 с.
16. Мясоєдов А.І. Проектування баз даних: навчальний посібник. — К.: КНЕУ, 2022. — 176 с.
17. Жежуха В.Я. Веб-сервіси та інтеграція систем. — Львів: ЛНУ, 2021. — 204 с.
18. Гармаш Л.Ю. Сучасні платформи програмування: посібник. — Харків: НТУ «ХПІ», 2023. — 264 с.
19. Gartner Magic Quadrant for CRM Systems 2023 — <https://www.gartner.com>
20. IEEE Standard for Software Verification and Validation — IEEE Std 1012-2022.
21. Романчук С.І. Захист даних у веб-додатках. — Тернопіль: ТНТУ, 2023. — 188 с.

- 22.W3C Web Security Guidelines — <https://www.w3.org/Security/>
- 23.Martin R.C. Clean Architecture: A Craftsman's Guide. — Prentice Hall, 2020.
- 24.ІТ в економіці. Підручник / за ред. Яцура О.П. — К.: КНЕУ, 2022.
- 25.ISO/IEC 27001:2022 Information security, cybersecurity and privacy protection.
- 26.Огляд платформ фрилансу: Upwork, Fiverr, Toptal. // DOU.ua, 2023. — <https://dou.ua/lenta/articles/freelance-platforms-review/>
- 27.Microsoft Azure Integration Guide — <https://learn.microsoft.com/en-us/azure/integration/>

ДОДАТКИ

ДОДАТОК А



Рисунок 6 Сторінка входу

ДОДАТОК Б

Реєстрація

← Login

Step 1: Create Account

Email

Password

Password

Next

Реєстрація

← Login

Step 2: Company Information

Company Name

Industry

Company Description

Complete Registration

Back

Рисунок 7 Сторінка реєстрації

ДОДАТОК В

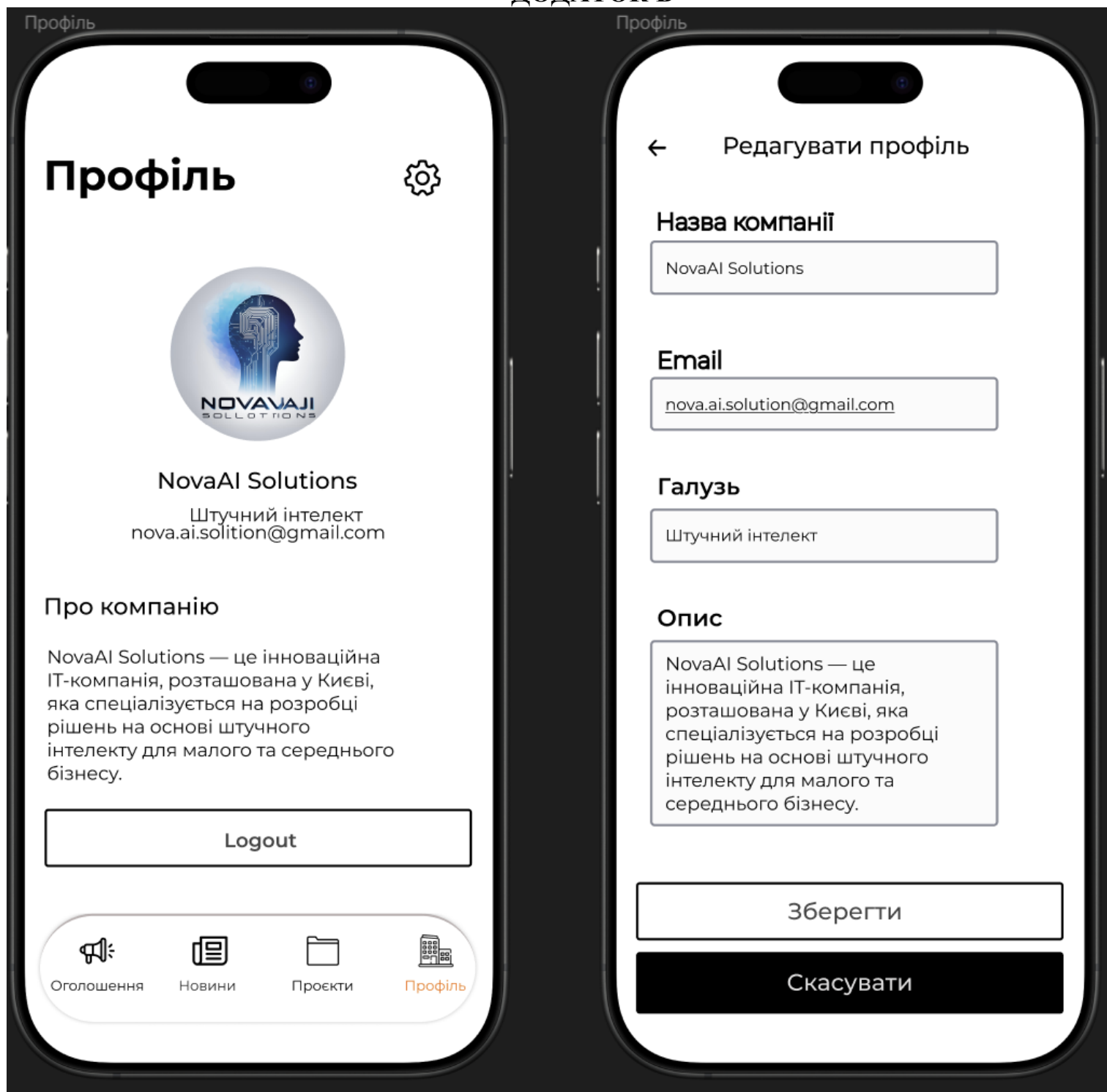


Рисунок 8 Сторінка профілю

ДОДАТОК Г

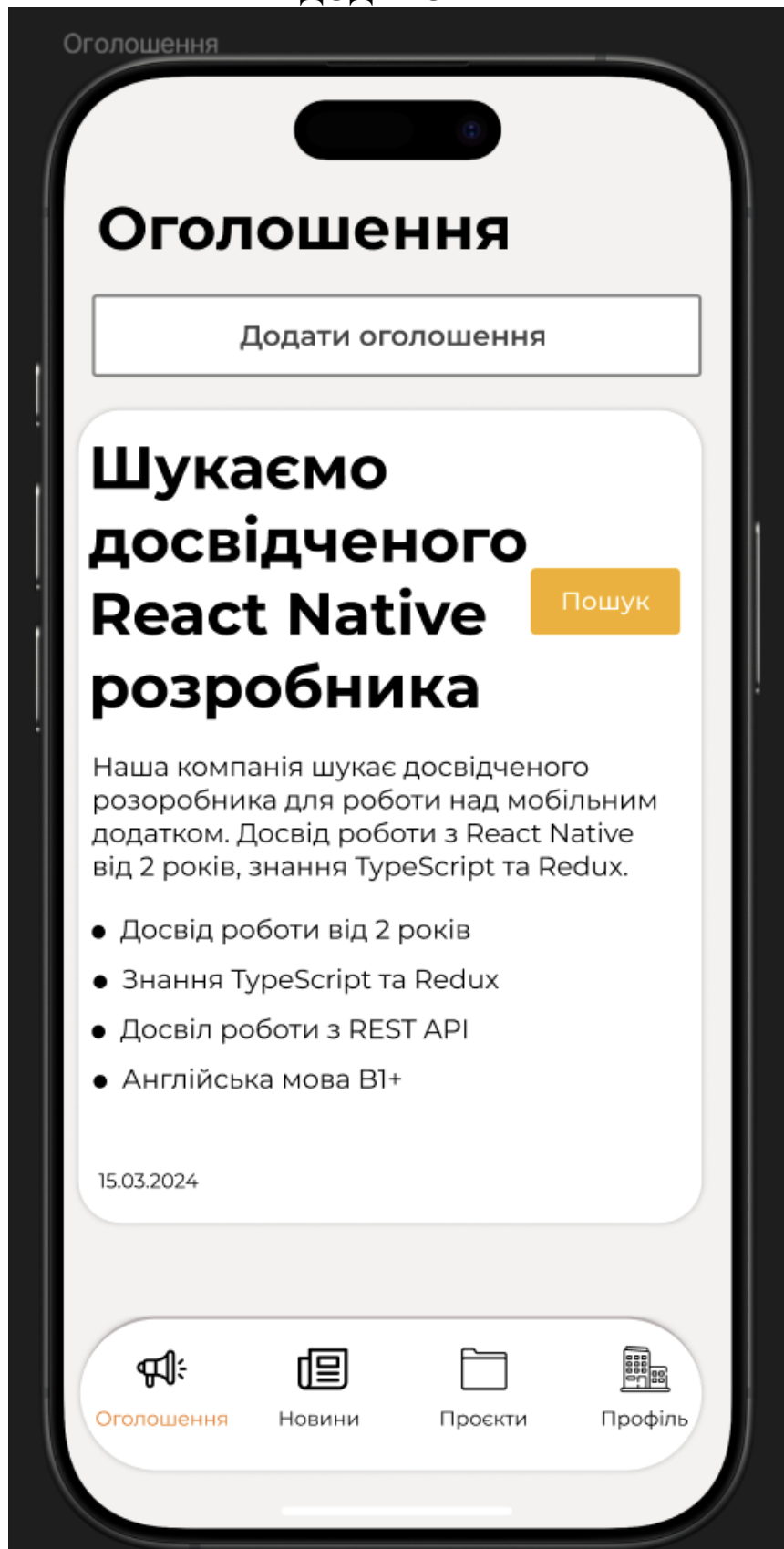


Рисунок 9 Екран перегляду оголошень

ДОДАТОК Д

Оголошення

Додати оголошення

Заголовок

Опис

Тип оголошення

☒ Пошук ☐ Пропозиція

Вимоги/Послуги

Рисунок 10 Екран створення оголошень

ДОДАТОК Е

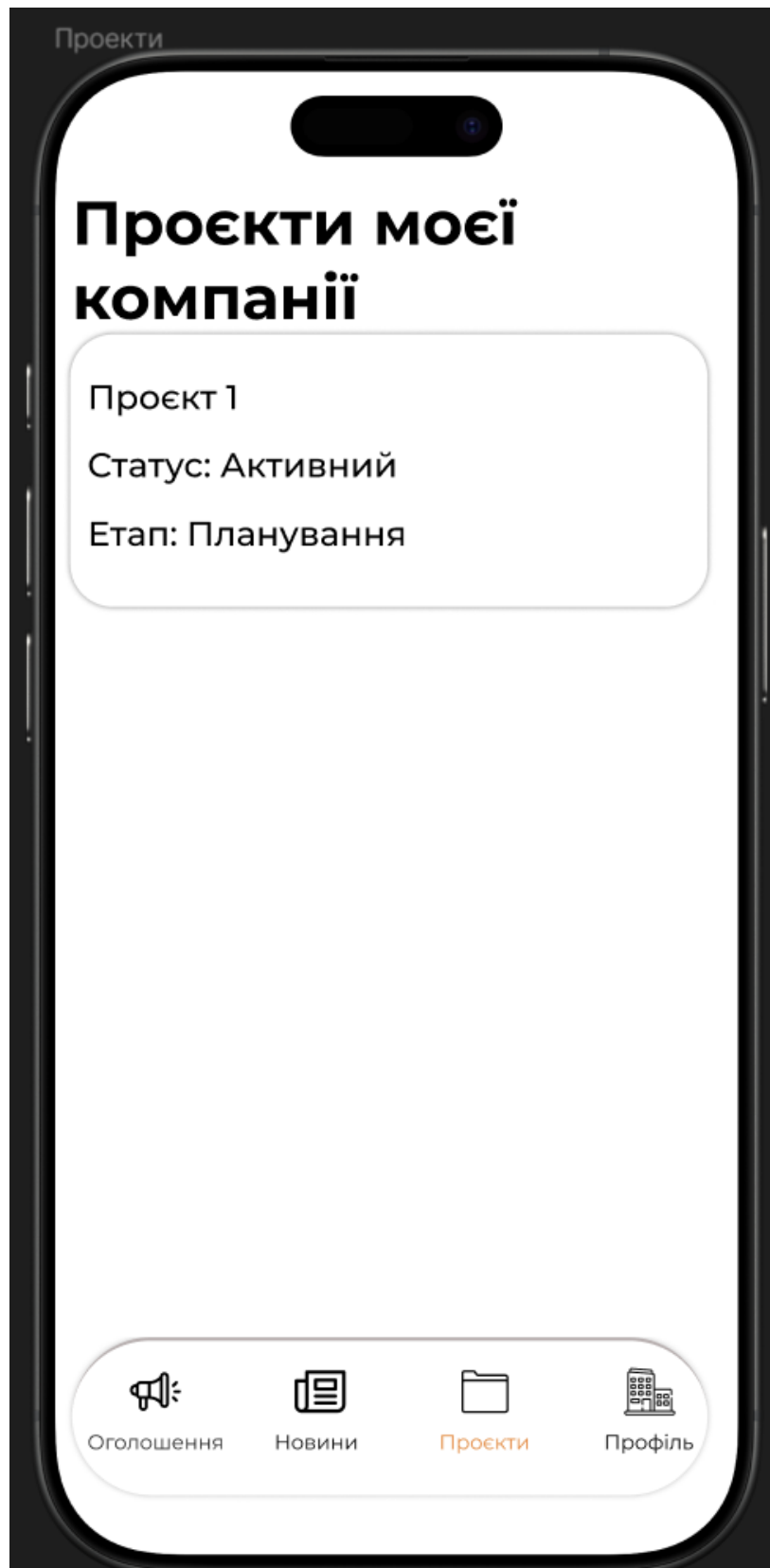


Рисунок 11 Екран проєктів

ДОДАТОК Ж

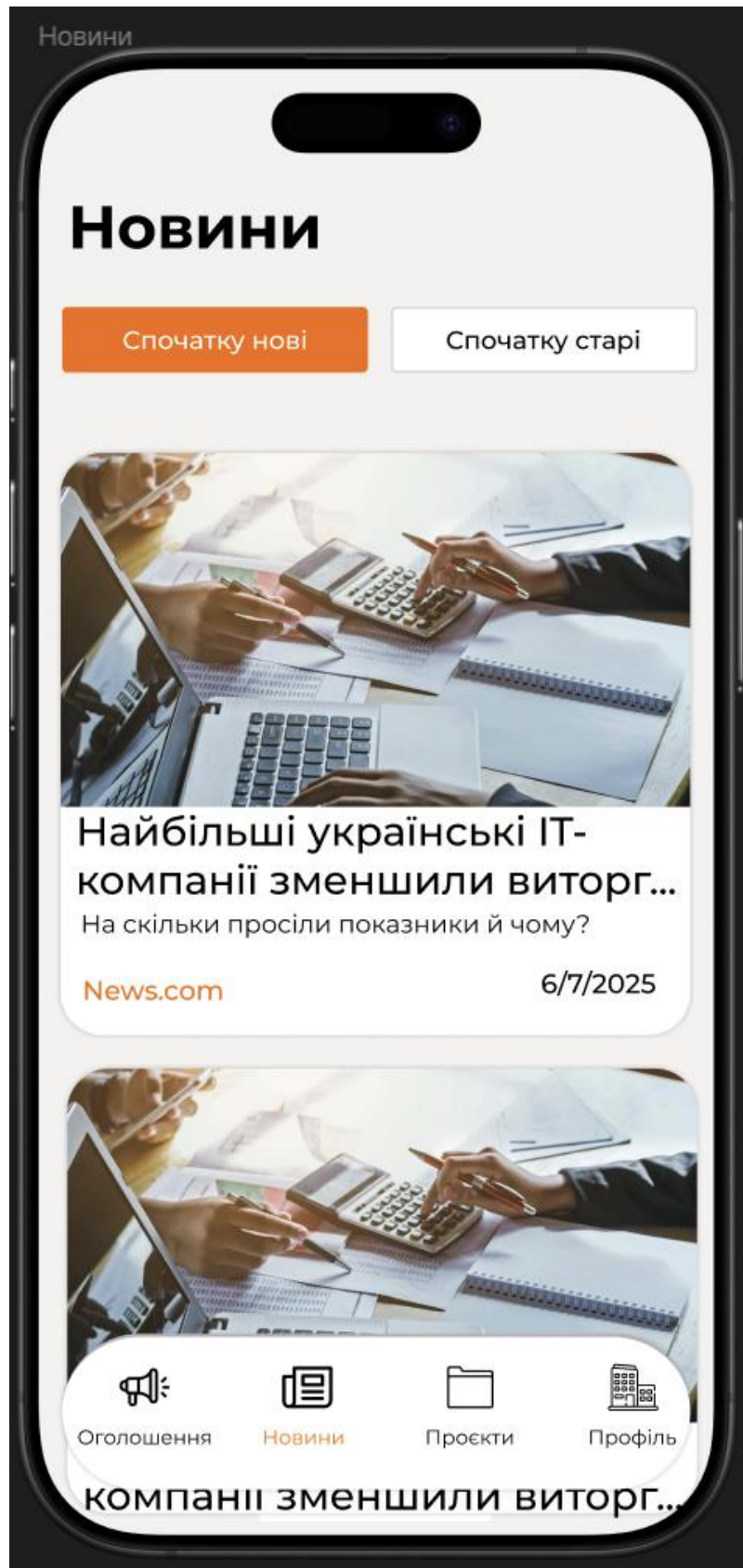


Рисунок 12 Екран новин