

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
ВЗАЄМОДІЇ ЗАМОВНИКІВ ТА ПОСТАЧАЛЬНИКІВ ПОСЛУГ
RESEARCH AND SOFTWARE IMPLEMENTATION OF METHODS FOR THE
INTERACTION OF CUSTOMERS AND SERVICE PROVIDERS

Виконав: студент 2 курсу, групи КНТ-222м
Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

ФЕДЬКОВ І.С.

(ПРИЗВИЩЕ та ініціали)

Керівник ПАРХОМЕНКО А. В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЗЕЛЕНЬОВА І. Я.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
« » 2023 року

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ФЕДЬКОВА Ігоря Станіславовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи): Дослідження та програмна реалізація методів взаємодії замовників та постачальників послуг. Research and Software Implementation of Methods for the Interaction of Customers and Service Providers
керівник проєкту (роботи): Пархоменко Анжеліка Володимирівна, к.т.н., доц.
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові,)

затверджені наказом закладу вищої освіти від «14» листопада 2023 р. № 437

2. Строк подання студентом проєкту (роботи): 1 грудня 2023 року

3. Вихідні дані до проєкту (роботи): завдання, література

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): 1. Аналіз методів та засобів взаємодії замовників та постачальників послуг. 2. Методика програмної реалізації SRM-системи. 3. Проєктування програмного забезпечення SRM-системи. 4. Конструювання програмного забезпечення SRM-системи.

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

6. Консультанти розділів проєкту (роботи):

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ПАРХОМЕНКО А.В., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання «08» _____ вересня _____ 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання
2	Аналіз предметної області та методів вирішення проблеми	2-3 тиждень	Розділ 1
3	Розробка методики вирішення проблеми	4-5 тиждень	Розділ 2
4	Проектування програмного забезпечення	6 тиждень	Розділ 3
5	Конструювання програмного забезпечення. Тестування та експериментальне дослідження програмного забезпечення.	7-9 тижні	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	10-11 тиждень	ПЗ, Додатки
7	Захист роботи	12 тиждень	

Студент

_____ Ігор ФЕДЬКОВ
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Анжеліка ПАРХОМЕНКО
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
125 с., 23 табл., 33 рис., 2 додатки, 139 джерел.

ПОСТАЧАЛЬНИКИ ПОСЛУГ, АВТОМАТИЗАЦІЯ БІЗНЕС-ПРОЦЕСУ, SRM-СИСТЕМА, СПЕЦІАЛІЗОВАНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВЕБЗАСТОСУНОК.

Об'єкт дослідження – процес взаємодії замовників та постачальників послуг.

Предмет дослідження – методи та засоби автоматизації процесу взаємодії замовників та постачальників послуг.

Мета роботи – дослідження та програмна реалізація спеціалізованого програмного забезпечення автоматизації взаємодії замовників та постачальників послуг для підвищення економічної ефективності бізнес-практик.

Матеріали, методи та технічні засоби: програмна платформа Node.js версія 20.10.0 LTS; програмне забезпечення для розгортання бази даних Docker, образ postgres 14; менеджер пакетів Yarn 4+; персональний комп'ютер з процесором Intel i5 4-го покоління або Apple M1, оперативною пам'яттю 8 Гб; 50 Гб місця на жорсткому диску, операційна система Windows 10+.

Результати. Створено вебзастосунок, який надає можливість керувати взаємодією постачальників та замовників послуг та є спеціалізованим програмним забезпеченням SRM-системи.

Висновки. Застосування програмного застосунку дозволить підвищити ефективність взаємодії замовників та постачальників послуг та рівень послуг, а також контролювати часові та матеріальні витрати бізнес-процесів.

Галузь використання – бізнес-діяльність у сфері надання послуг.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 125 pages, 23 tables, 33 figures, 2 appendixes, 139 resources.

SERVICE PROVIDERS, BUSINESS PROCESS AUTOMATION, SRM SYSTEM, SPECIALIZED SOFTWARE, WEB APPLICATION.

The object of research is the process of interaction between customers and service providers.

The subject of research is methods and tools of automating the process of interaction between customers and service providers.

The purpose of the work is research and software implementation of specialized software for automating the interaction of customers and service providers to increase the economic efficiency of business practices.

Materials, methods, and technical tools: software platform Node.js version 20.10.0 LTS; Docker database deployment software, postgres 14 image; Yarn 4+ package manager; a personal computer with a 4th generation Intel i5 or Apple M1 processor, 8 GB RAM; 50 GB of hard disk space, Windows 10+ operating system.

Results. A web application has been created that provides an opportunity to manage the interaction of service providers and customers and is a specialized software of the SRM system.

Conclusions. The usage of the software application will improve the efficiency of interaction between customers and service providers and the level of services, as well as control the time and material costs of business processes.

The area of usage is business activity in the field of service provision.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	7
Вступ.....	8
1 Аналіз методів та засобів взаємодії замовників та постачальників послуг ...	9
1.1 Системи автоматизації взаємодії замовників та постачальників	9
1.2 Огляд методів управління відносинами з постачальниками	13
1.3 Переваги та виклики ПЗ SRM	15
1.4 Огляд існуючих SRM-систем.....	17
1.5 Постановка завдань дипломної роботи.....	19
2 Методика програмної реалізації SRM-системи.....	20
2.1 Структура SRM-системи	20
2.2 Етапи розробки SRM-системи	21
2.3 Висновки до розділу 2	32
3 Проєктування програмного забезпечення SRM-системи.....	33
3.1 Аналіз вимог	33
3.2 Архітектура системи.....	37
3.3 Аналіз та обґрунтування вибору інструментарію розробки.....	39
3.4 Висновки до розділу 3	66
4 Конструювання програмного забезпечення SRM-системи	67
4.1 Структура та алгоритми функціонування системи.....	67
4.2 Опис особливостей програмної реалізації	80
4.3 Опис інтерфейсу	94
4.4 Тестування програми	99
4.5 Висновки до розділу 4.....	100
Висновки	101
Перелік джерел посилань	102
Додаток А Текст програми.....	113
Додаток Б Слайди презентації	119

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ACID	– Atomicity, Consistency, Isolation, Durability;
API	– Application Programming Interface;
BOM	– Bill of Materials;
CRM	– Customer Relationship Management;
GDPR	– General Data Protection Regulation;
IDE	– Integrated Development Environment;
JSX	– JavaScript XML;
REST	– Representational State Transfer;
ROI	– Return on Investment;
SQL	– Structured Query Language;
SRM	– Supplier Relationship Management;
UI	– User Interface;
UX	– User Experience;
VDOM	– Virtual Domain;
БД	– база даних;
ІТ	– інформаційні технології;
ПЗ	– програмне забезпечення;
СКБД	– система керування базою даних.

ВСТУП

Ефективна організація співпраці постачальників та замовників дозволяє надати бізнес-відносинам більшої цінності. Основною метою керування взаємодією між замовниками та постачальниками товарів, матеріалів чи послуг є вибір постачальників, які є економічно вигідними та з якими легко працювати [1]-[2].

З точки зору організацій важливо розглядати цю взаємодію під різними кутами зору, оскільки з одного боку компанія може бути постачальником для клієнтів а з іншого бути клієнтом, який працює з певними постачальниками [3].

В більшості сучасних бізнес-практик, управління взаємодією замовників та постачальників складніше, ніж просто відбір контрагентів із найнижчою ціною. За останні кілька років управління взаємовідносинами з постачальниками зазнало значних змін через зміну технологій та зростання глобальних масштабів економіки. Постачальників багато, вибрати найбільш відповідного складно, але використання нових стратегій для оптимізації системи керування постачальниками допомагає вирішити цю задачу [1].

Сьогодні існує багато різних інструментів і застосунків для керування відносинами між постачальниками та клієнтами [2]. Але це програмне забезпечення (ПЗ) зазвичай досить складне з точки зору впровадження, а також не забезпечує урахування всіх особливостей бізнес-діяльності кожної окремої компанії. Отже, розробка власної спеціалізованої SRM-системи, яка дозволить ефективно працювати як з постачальниками так і з замовниками послуг та запропонує стратегії, які допоможуть ефективно керувати бізнес-процесами є доцільною, а тема роботи актуальною.

Мета роботи – дослідження та програмна реалізація спеціалізованого програмного забезпечення автоматизації взаємодії замовників та постачальників послуг для підвищення економічної ефективності бізнес-практик.

1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ВЗАЄМОДІЇ ЗАМОВНИКІВ ТА ПОСТАЧАЛЬНИКІВ ПОСЛУГ

1.1 Системи автоматизації взаємодії замовників та постачальників

Як показали проведені дослідження, в останні два десятиліття експерти-теоретики та керівники компаній були зосереджені на управлінні взаємовідносинами з клієнтами (Customer Relationship Management, CRM). Суть цього підходу полягає в тому, щоб на основі оцінки поточного рівня взаємодії з окремими клієнтськими сегментами та їх майбутніх перспектив розвивати відносини з найбільш цінними клієнтами з використанням відповідних інструментів [4].

Але на думку авторів роботи [5], будь-яке підприємство має координувати управління взаємовідносинами не лише з замовниками (клієнтами), а й з постачальниками. Саме тому, на сьогоднішній день, бізнес використовує два типи систем для автоматизації взаємодії замовників та постачальників послуг: систему управління взаємовідносинами з клієнтами (CRM), яка допомагає керувати взаємодією клієнтів і бізнесу та систему управління взаємовідносинами з постачальниками (Supplier Relationship Management, SRM) [3].

Рис. 1.1 ілюструє взаємозв'язок між цими системами [6].

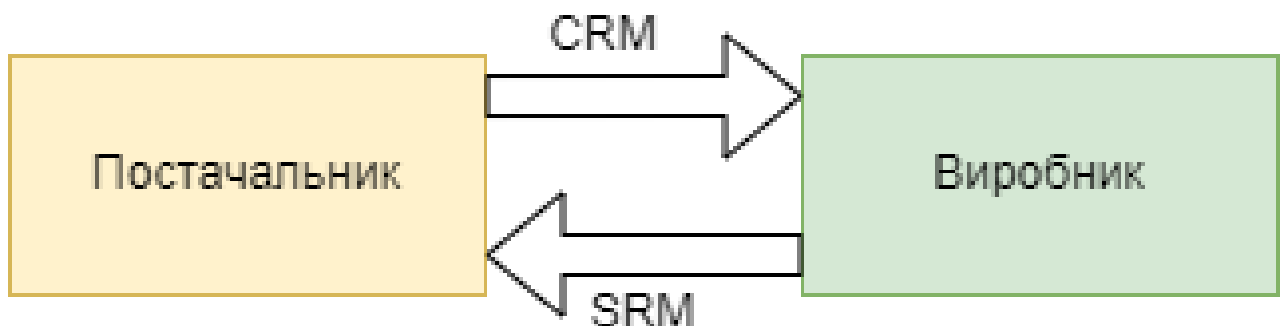


Рисунок 1.1 – Зв'язок між SRM і CRM в системі постачання [6]

Однак, згідно з [7], управління відносинами в системі постачання може бути найскладнішим аспектом управління системою. Компанія повинна прагнути перетворити постачальників на своїх партнерів і змусити їх допомогти підвищити продуктивність по відношенню до своїх клієнтів. Але це можливо лише за умови співпраці з ними та побудови партнерських відносин. Це партнерство може мати різні форми [8].

Кооперація в системі постачання розуміється як діяльність, у якій два або більше партнерів у системі постачання розвивають спільну діяльність для досягнення спільних цілей. Співпраця в системі постачання передбачає обмін інформацією, створення спільних цілей, синхронізацію рішень, спільне використання ресурсів і гармонізацію незалежних партнерів у системі постачання [9].

SRM – це розвиток відносин із найбільш перспективними постачальниками. За даними [6], необхідно розвивати тісніші відносини з ключовими постачальниками, які можуть надати досвід, необхідний для розробки нових інноваційних продуктів і допомогти в їх успішному виведенні на ринок. Постачальники та клієнти можуть співпрацювати в кількох функціональних сферах.

Автори роботи [10] підкреслюють, що функціональна сфера закупівель має стратегічне значення, оскільки вона знаходиться в найбільш тісному контакті з постачальником. Автори роботи [6] також включають дослідження і розробки, маркетинг, продаж і фінанси в сферах кооперації в системі постачання. Робота [11] розширює та конкретизує цей перелік – серед сфер співпраці називає маркетингові дослідження ринку, контролінг або управління обслуговуванням клієнтів. Але автори роботи [12] зазначають, що чим більше функціональних сфер залучено до взаємодії в системі постачання, тим складнішим стає управління всією системою.

З іншого боку, інтеграція операцій з постачальниками може покращити ефективність бізнесу [6].

Ще одна перевага полягає в тому, що управління відносинами з постачальниками може одночасно створювати цінність і для постачальників.

Аналіз показав, що в загальному випадку існують 4 види закупівель:

- прямі товари – матеріальні товари, необхідні для виробництва товарів або надання послуг клієнтам. Ці товари, як правило, є частиною специфікації BOM (Bill of Materials), тому їх купують у постачальників і використовують безпосередньо у виробництві кінцевого продукту;
- прямі послуги – це можуть бути субпідрядні роботи, транспортування, стороннє виробництво тощо;
- непрямі товари – не є частиною специфікації, і вони не обов’язково потрібні для надання послуг або виробництва товарів для клієнтів. Прикладами непрямих товарів є канцелярське приладдя, комп’ютерне обладнання, транспортне обладнання, меблі тощо;
- непрямі послуги – різні додаткові послуги, які можуть знадобитися компаніям з різних причин. Прикладом непрямих послуг можуть бути: маркетингові послуги, ІТ, юридичний та управлінський консалтинг, дослідження та розробки, подорожі, транспорт, капітал, фінансові консультації тощо [13].

Але які інструменти використовувати в даній сфері і як їх реалізувати, наразі вирішено недостатньо на теоретичному рівні [14].

Побудова міцних стосунків із постачальниками полягає у вирішенні закупівель у спосіб, який виходить за рамки лише угод про купівлю та контрактів. Підтримуючи безперебійну роботу ланцюга постачання, компанії можуть гарантувати, що і вони самі, і їхні постачальники працюватимуть швидко над вирішенням проблем, постійно шукатимуть шляхи покращення операцій і допомагатимуть одна одній скористатися перевагами відносин [15].

Ефективне впровадження SRM-системи вимагає зміни підходу компанії до своїх постачальників і використання відповідних інструментів для розвитку взаємовідносин. Однак для того, щоб це було зроблено якомога

ефективніше, компаніям необхідно використовувати ПЗ для управління відносинами з постачальниками (рис.1.2) [13].

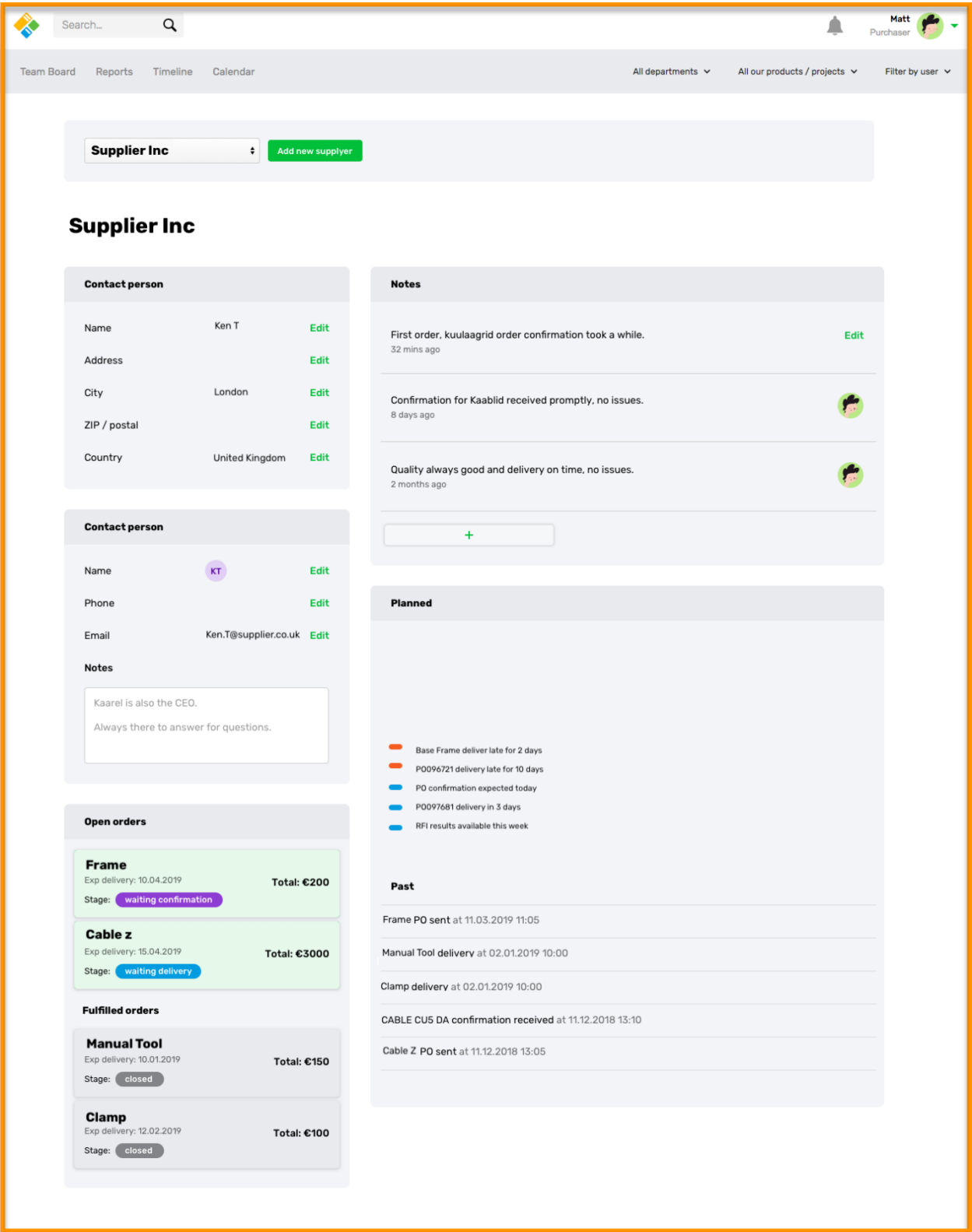


Рисунок 1.2 – Приклад екранної форми SRM-системи [13]

1.2 Огляд методів управління відносинами з постачальниками

Особливості використання SRM-системи можуть відрізнятися залежно від кожної окремої організації, але основний акцент на розвитку взаємовигідних відносин з постачальниками, особливо тими, хто є стратегічними партнерами, залишається постійним [15].

Управління відносинами з постачальниками складається з трьох важливих етапів, на яких реалізуються наступні методи (рис.1.3):

- сегментація постачальників: порівняння постачальників із прибутковістю та ризиком;
- розробка стратегії постачальника: розподіл внутрішніх ресурсів і планів для задоволення потреб бізнесу;
- виконання стратегії постачальника.

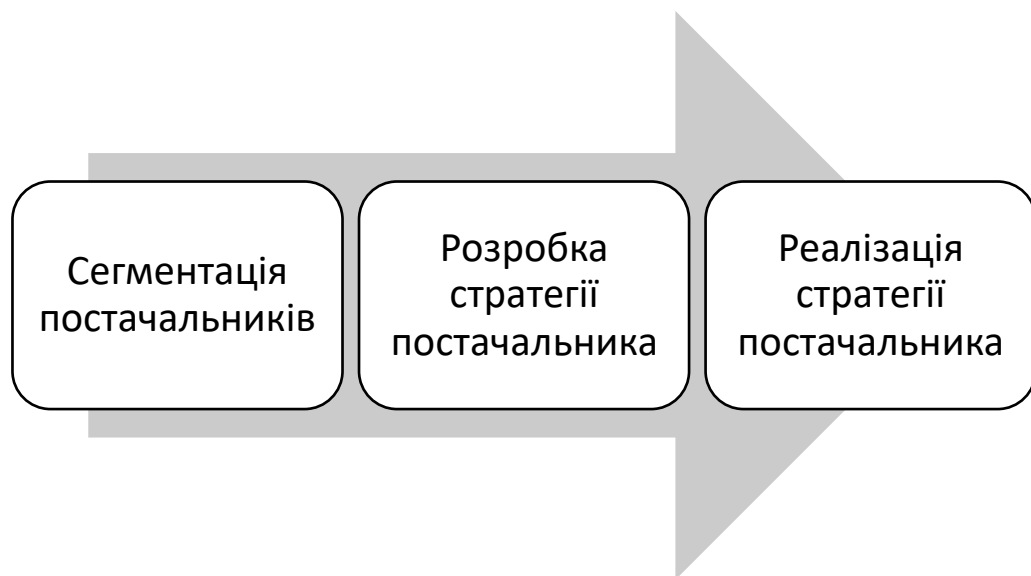


Рисунок 1.3 – Розвиток відносин з постачальниками [15]

Під час першого етапу компанії розглядають усіх своїх постачальників і класифікують їх на основі їх важливості. Різні постачальники мають різний рівень впливу на бізнес. Процес сегментації постачальників зосереджується на пошуку ключових постачальників з точки зору тих, хто надає товари та послуги, які є важливими для щоденної діяльності бізнесу, і тих, хто не є.

Роблячи це, компанії знатимуть, як визначити пріоритети у своїх відносинах. Однак це не означає, що слід нехтувати іншими постачальниками, оскільки всі вони важливі [15].

Під час другого етапу компанії аналізують своїх постачальників, за списком від найважливіших до найменш важливих, одночасно розробляючи план, як працювати з кожним із них окремо. Оскільки до постачальників слід ставитися як до партнерів, будь-яка розроблена стратегія повинна включати взаємовигідні методи, які можуть зробити всіх зацікавлених сторін задоволеними.

Під час третього етапу компанія виконує плани, розроблені разом із постачальниками. Якщо кожному ключовому постачальнику призначити менеджера, його робота буде зосереджена виключно на підтримці міцних стосунків із ним, зберігаючи відкриті лінії зв'язку з усіма з обох сторін відносин. Ці менеджери також мають узгоджувати мінливі потреби та плани вашої компанії з тим, що вони знають про постачальників, щоб було легше побачити, коли вам потрібно змінити постачальника для кращого підходу [15].

Автор роботи [16] розглядає шість ключових кроків у впровадженні програми управління взаємовідносинами з постачальниками:

- сегментація бази постачальників;
- встановлення цілей для ПЗ SRM;
- вимірювання ефективності роботи постачальника з цілями;
- складання плану залучення постачальників і управління;
- залучення постачальників та узгодження взаємодії з ними;
- співпраця та постійне вдосконалення.

Типові показники ефективності постачальника, які потрібно виміряти:

- продуктивність доставки (дані з ERP);
- показники якості (дані систем якості);
- ефективність обслуговування (дані опитувань);
- ефективність КСВ (дані самооцінки постачальника);

- статус ризику (дані самооцінки постачальника або зовнішніх джерел даних);
- інноваційні можливості (дані самооцінки постачальників або опитувань) [16].

1.3 Переваги та виклики ПЗ SRM

Компанії використовують управління взаємовідносинами з постачальниками та інші підходи для кращого ведення бізнесу, що допомагає їм підвищити ефективність і прибуток. SRM має численні переваги, і всі вони сприяють більшому прибутку [17].

Економія коштів. Фінансові стимули використовуються для встановлення відносин з постачальниками. Початкові витрати на встановлення партнерства з новими постачальниками можуть бути значними. Мета полягає в тому, щоб економія коштів і ефективність, отримані з часом, компенсували ці витрати. Впровадження SRM має призвести до зниження витрат на закупівлі, а також до зниження адміністративних витрат і витрат на зберігання запасів. Крім того, краще керовані постачальники повинні надавати вищу якість продукції та послуг за конкурентними цінами. Розвиток цих спільних зв'язків із постачальниками також може допомогти у вирішенні проблем із доступністю, із якістю матеріалів, затримками та ін.

Покращена якість. Управління ланцюгом поставок є складним процесом, який часто включає кілька етапів і багато різних гравців. Щоб оптимізувати процес і гарантувати, що готова продукція відповідає очікуванням клієнтів, якість має бути головним пріоритетом на кожному етапі. Співпраця з постачальниками для створення SRM може допомогти підвищити якість матеріалів і компонентів, що використовуються у виробництві. Це, у свою чергу, може призвести до покращення якості продукції та підвищення задоволеності споживачів.

Більша ефективність. У будь-якому виробничому процесі важлива ефективність. SRM може допомогти досягти більшої ефективності у виробничому процесі, зменшивши потребу в переробці, покращивши комунікацію та впорядкувавши потік інформації.

Покращена гнучкість. Гнучкість є ще однією важливою характеристикою успішного виробництва. Коли компанія зможе швидко адаптуватися до змін попиту клієнтів, з'являться кращі можливості для досягнення успіху на сучасному ринку, що постійно змінюється. SRM може допомогти досягти більшої гнучкості шляхом покращення зв'язку та співпраці з постачальниками. Це дозволить своєчасно отримати необхідні матеріали та комплектуючі, що дозволить легше реагувати на зміни попиту клієнтів [17].

ПЗ SRM має на меті сприяти розвитку відносин з постачальниками, особливо тими, які вважаються стратегічними для бренду та прибутковості організації. Система SRM також зосереджена на якості, продуктивності та інноваціях для підвищення ефективності та отримання конкурентної переваги [18].

Переваги використання ПЗ SRM:

- краще використання можливостей постачальника;
- зниження витрат;
- забезпечення безперервності ланцюжка постачань;
- зниження ризиків ланцюжка поставок;
- покращена реакція постачальників;
- покращена якість;
- бачення майбутніх цін [18].

Виклики використання ПЗ SRM:

- управління ризиками та безперервністю під час пандемій, екологічних катастроф чи воєн;
- відсутність видимості постачальників та їхньої стратегічної цінності для організації;

- використовується лише для підвищення цінності за рахунок зниження витрат замість використання стратегічних відносин;
- відсутність узгодженості з бізнес-цілями та чіткими контрольними показниками;
- відсутність широкої бази постачальників;
- відсутність зобов'язань щодо програми через ігнорування принципів партнерства з постачальниками [18].

1.4 Огляд існуючих SRM-систем

Аналіз показав, що серед найдільш популярних і високо оцінених прикладів SRM-систем можна відзначити наступні, що представлені в табл. 1.1 [19]-[20].

Таблиця 1.1 – Огляд SRM-систем [19]-[20]

Назва	Опис
1	2
Kodiak Hub	Деякі з ключових переваг: спеціалізований набір функцій, простота використання, коротший час впровадження та нижча вартість
SAP Ariba	Охоплює всі аспекти управління постачальниками – від пошуку джерел і закупівель до оцінки ефективності постачальників. Завдяки своїй хмарній архітектурі забезпечує співпрацю з постачальниками в режимі реального часу, оптимізуючи процеси закупівель і сприяючи кращим відносинам з постачальниками.
Zycus	Зручна платформа, яка інтегрується з існуючими системами для оптимізації керування даними постачальників, зниження ризиків і відстеження продуктивності. Забезпечує видимість діяльності постачальника в реальному часі, для моніторингу його ефективності.

Продовження табл. 1.1

1	2
Epicor SRM	Пропонує надійні можливості керування взаємовідносинами з постачальниками, допомагаючи компаніям зменшити витрати, підвищити ефективність ланцюжка поставок і покращити співпрацю між постачальниками. Вбудована аналітика надає практичну інформацію для кращого прийняття рішень.
Oracle SRM	Це комплексна платформа SRM, розроблена для керування складними мережами постачальників. Це полегшує все, від управління контрактами до оцінки ризиків постачальників. Пакет дозволяє компаніям будувати ефективні стратегічні відносини зі своїми постачальниками.
Jaggaer	Забезпечує центральне сховище даних постачальників і використовує інтелектуальне ПЗ для управління ризиками та відкриття нових можливостей. Дані представлені на зручній інформаційній панелі, що дозволяє швидко створювати насичені інформацією звіти про продуктивність.
Medius	Містить централізований хаб, у який постачальники можуть завантажувати свою інформацію через портал самообслуговування, звільняючи клієнтів від адміністративного тягаря та підвищуючи точність даних постачальників. Є інструменти, які допомагають перевірити облікові дані постачальника, а також конструктор звітів, який надає дані про продуктивність постачальника за різними ключовими показниками.
Coupa	Пропонує широкий набір програмних рішень, які допомагають клієнтам оптимізувати витрати, керувати ланцюгом постачання та ліквідністю. Основна увага приділяється управлінню ризиками третьої сторони, а перспектива розглядається з точки зору зменшення ризиків і контролю, а управлінські звіти також зосереджені на управлінні ризиками, показниках і засобах контролю.
Ivalua	Надає широкий спектр рішень, що охоплюють не лише ризики та оцінку постачальників, але й пошук джерел, управління контрактами, виставлення рахунків, платежі та аналітику, об'єднуючи їх через загальну платформу для покращеної видимості, чіткішого розуміння, кращої ефективності та більш стратегічного прийняття рішень.

1.5 Постановка завдань дипломної роботи

Як показали проведені дослідження, на сьогоднішній день існує багато різних систем і застосунків для керування відносинами між постачальниками та клієнтами [21].

Але це ПЗ зазвичай досить складне з точки зору впровадження, а також не забезпечує урахування всіх особливостей бізнес-діяльності кожної окремої компанії.

Тому, розробка власної спеціалізованої SRM-системи, яка дозволить ефективно працювати як з постачальниками так і з замовниками послуг та допоможе ефективно керувати бізнес-процесами є доцільною, а тема роботи актуальною.

Мета роботи – дослідження та програмна реалізація спеціалізованого програмного забезпечення автоматизації взаємодії замовників та постачальників послуг для підвищення економічної ефективності бізнес-практик.

Для досягнення поставленої мети необхідно розв'язати наступні завдання:

- розробити методику програмної реалізації спеціалізованої SRM-системи;
- виконати проєктування ПЗ SRM-системи;
- виконати розробку та тестування спеціалізованого ПЗ SRM-системи.

2 МЕТОДИКА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ SRM-СИСТЕМИ

2.1 Структура SRM-системи

ПЗ SRM – це інструмент зв'язку для виробників, дистриб'юторів і роздрібних торговців у межах ланцюжка поставок. Його основна мета – покращити бізнес-процеси між організацією та її постачальниками. Розробляючи спеціалізоване ПЗ SRM, важливо враховувати ключові компоненти, які сприяють ефективному управлінню відносинами з постачальниками [22].

Узагальнену структуру ПЗ SRM-системи наведено на рис. 2.1.



Рисунок 2.1 – Структура SRM-системи

Модуль 1. Сегментація постачальників. Дозволяє підприємствам визначити пріоритети своїх відносин і відповідно розподілити ресурси. Класифікуючи постачальників на основі їх ефективності та стратегічної важливості, підприємства можуть зосередити свої зусилля на побудові міцних відносин із ключовими постачальниками.

Модуль 2. Управління продуктивністю постачальника. Надає функції відстеження та оцінки ефективності роботи постачальника за попередньо визначеними показниками. Моніторинг роботи постачальника є важливим для підтримки високих стандартів і визначення областей для покращення. Відстежуючи ключові показники ефективності, компанії можуть об'єктивно оцінювати ефективність роботи постачальників і надавати зворотний зв'язок для постійного вдосконалення.

Модуль 3. Співпраця з постачальниками. Передбачає забезпечення безперебійного спілкування та співпраці з постачальниками. Ефективне спілкування та співпраця є життєво важливими для успішного управління відносинами з постачальниками. Забезпечуючи платформу для безперебійного спілкування, компанії можуть сприяти співпраці, обмінюватися інформацією та співпрацювати з постачальниками для досягнення спільних цілей.

Модуль 4. Управління ризиками постачальників. Спрямований на виявлення, оцінку та пом'якшення ризиків, пов'язаних з постачальниками. Управління ризиками є невід'ємною частиною управління взаємовідносинами з постачальниками. Виявляючи й оцінюючи ризики, пов'язані з постачальниками, підприємства можуть розробляти стратегії пом'якшення цих ризиків і забезпечення безперебійної роботи ланцюга постачання.

Модуль 5. Управління контрактами з постачальниками. Відповідає за управління контрактами з постачальниками, переговори та відповідність. Контракти з постачальниками відіграють вирішальну роль у визначенні умов відносин. Ефективне управління контрактами гарантує, що обидві сторони дотримуються узгоджених умов, зменшує юридичні ризики та забезпечує основу для вирішення суперечок.

Інтеграція цих компонентів в спеціалізоване ПЗ SRM забезпечить комплексну та надійну платформу для ефективного керування відносинами з постачальниками.

2.2 Етапи розробки SRM-системи

Розробка та впровадження спеціалізованого ПЗ для керування взаємовідносинами з постачальниками може принести користь організації, спрощуючи процеси управління постачальниками та покращуючи загальну ефективність. Ключові етапи розробки спеціалізованого ПЗ SRM у вигляді схеми алгоритму наведено на рис. 2.2.



Рисунок 2.2 – Схема алгоритму розробки спеціалізованого ПЗ SRM

Блок 1. Початок.

Блок 2. Планування спеціалізованого ПЗ SRM:

- визначення потреб бізнесу;
- встановлення бюджету та графіку;
- взаємодія з експертами з розробки ПЗ.

Блок 3. Вибір технологій для розробки ПЗ SRM:

- оцінка різних мов програмування;
- вибір системи керування базами даних.

Блок 4. Проєктування інтерфейсу користувача SRM-системи: включення основних функцій.

Блок 5. Розробка ПЗ SRM:

- дотримання найкращих практик розробки ПЗ;
- забезпечення безпеки та конфіденційності даних.

Блок 6. Тестування та розгортання ПЗ:

- проведення ретельного тестування ПЗ;
- підготовка до розгортання ПЗ.

Блок 7. Навчання персоналу та підтримка спеціалізованого ПЗ SRM:

- реалізація програми навчання;
- надання постійної підтримки та оновлень.

Блок 8. Оцінка ефективності впровадження:

- відстеження ключових показників ефективності;
- збір відгуків користувачів і внесення покращень.

Блок 9. Результати задовільні? Якщо Ні, то повернення на Блок 2 для повторних дій щодо планування та реалізації процесу. Якщо так, то перехід на Блок 10 Кінець.

2.2.1 Планування спеціалізованого ПЗ SRM

Правильне планування має вирішальне значення для успіху будь-якого проєкту розробки ПЗ.

Першим кроком у плануванні спеціалізованого ПЗ SRM є визначення конкретних бізнес-потреб. Рекомендується провести ретельний аналіз існуючих процесів управління постачальниками, проблемних точок і бажаних результатів для визначення функцій ПЗ SRM. Цей аналіз стане основою для процесу розробки, гарантуючи, що ПЗ відповідає унікальним потребам організації.

Після визначення бізнес-потреб важливо встановити реалістичний бюджет і графік для проєкту розробки ПЗ SRM з урахуванням різних факторів (складність вимог, кількість користувачів, потреби в інтеграції з існуючими системами та ін.).

Важливо виділити достатні ресурси, як з точки зору бюджету, так і часу, для розробки та впровадження спеціалізованого ПЗ SRM. Форсування процесу або висунення нереалістичних очікувань може призвести до поганих результатів і потенційних невдач.

Також потрібно враховувати потенційну рентабельність інвестицій (Return on Investment, ROI) спеціалізованого ПЗ SRM. Хоча початкові витрати на розробку можуть бути значними, довгострокові вигоди від покращеного управління постачальниками та підвищення операційної ефективності можуть переважити початкові інвестиції.

Ретельне планування та встановлення реалістичного бюджету і графіку забезпечує безперебійний і успішний процес розробки спеціалізованого ПЗ SRM.

Створення спеціалізованого ПЗ SRM вимагає досвіду розробки ПЗ та управління постачальниками. Залучення експертів із розробки ПЗ, які спеціалізуються на інструментах управління взаємовідносинами з постачальниками, може значно підвищити успіх проєкту. Вони можуть надати цінну інформацію та вказівки протягом усього процесу планування та розробки, а також допомогти перетворити потреби бізнесу в технічні вимоги, гарантуючи, що ПЗ відповідає очікуванням і забезпечує бажані функції.

Співпраця з експертами з розробки ПЗ не тільки призведе до створення високоякісного спеціалізованого інструменту для управління взаємовідносинами з постачальниками, але й забезпечить постійну підтримку та обслуговування для забезпечення оптимальної продуктивності ПЗ.

Отже, планування спеціалізованого ПЗ SRM передбачає визначення потреб бізнесу, встановлення реалістичного бюджету та часових рамок, а також залучення експертів із розробки ПЗ.

2.2.2 Вибір технологій для розробки ПЗ SRM

Вибір правильного стеку технологій має вирішальне значення для успіху проекту створення спеціалізованого ПЗ. Стек технологій стосується комбінації мов програмування, фреймворків та інструментів, які використовуються для розробки програмного застосунку.

Вибір правильного стеку технологій передбачає оцінку різних мов програмування та вибір відповідної системи керування базами даних (СКБД).

Огляд різних мов програмування свідчить, що є кілька мов програмування, доступних для створення спеціалізованого ПЗ SRM. Вибір мови програмування залежить від таких факторів, як масштабованість, продуктивність і досвід команди розробників.

Популярні мови програмування для створення спеціалізованого ПЗ SRM та їх характеристика наведені в табл. 2.1.

При виборі мови програмування важливо враховувати такі фактори як наявність кваліфікованих розробників, підтримка спільноти та довгострокова стійкість мови. Аналіз переваг та недоліків кожної мови програмування дозволяє обрати ту, яка найкраще відповідає конкретним вимогам.

Вибір системи керування базами даних (СКБД), яка відповідає за ефективне зберігання, упорядкування та отримання даних, є ще одним критичним рішенням під час створення спеціалізованого ПЗ SRM.

Таблиця 2.1 – Загальна характеристика мов програмування [22]

Назва	Опис
Java	Універсальна та широко поширена мова, відома своєю масштабованістю та надійністю, що пропонує величезну екосистему бібліотек і фреймворків, які можуть прискорити розробку
Python	Гнучка мова, яка підкреслює читабельність коду та простоту використання. Є популярним вибором для швидкої розробки програм через простоту та продуктивність
C#	Потужна мова для створення програм корпоративного рівня на стеку Microsoft, що забезпечує інтеграцію з платформою .NET та підтримку систем на базі Windows.

Огляд деяких популярних варіантів СКБД наведено в табл. 2.2.

Таблиця 2.2 – Загальна характеристика СКБД [22]

Назва	Опис
MySQL	Система керування реляційною базою даних з відкритим кодом, відома своєю продуктивністю та масштабованістю. Широко використовується у вебзастосунках і забезпечує підтримку для обробки структурованих даних.
Oracle	Надійна та багатофункціональна СКБД для програм корпоративного рівня. Забезпечує такі розширені функції, як висока доступність, безпека та масштабованість, що робить її придатною для великомасштабних систем.
MongoDB	База даних NoSQL, яка пропонує гнучкість і масштабованість для обробки великих обсягів даних. Призначена для зберігання неструктурованих даних і забезпечує швидкий і ефективний доступ до інформації.

При виборі СКБД важливо враховувати такі фактори як можливості моделювання даних, мова запитів, підтримка транзакцій і здатність обробляти складні зв'язки між об'єктами даних. Аналіз вимог до ПЗ SRM дозволить обрати СКБД, яка найкраще відповідає потребам щодо зберігання даних, продуктивності та масштабованості.

Оскільки проєкт зазвичай розвивається та постійно з'являються нові технології, можливо знадобитися відповідним чином адаптувати стек технологій. Регулярна переоцінка вибору технологій розробки допомагає переконатися, що ПЗ залишається ефективним, придатним для обслуговування та відповідає бізнес-цілям.

2.2.3 Проєктування інтерфейсу користувача SRM-системи

Інтерфейс користувача (User Interface, UI) спеціалізованого ПЗ для SRM відіграє вирішальну роль у забезпеченні задоволення та адаптації користувачів. Він має бути інтуїтивно зрозумілим, зручним і відповідати вимогам бізнесу. Розробляючи користувацький інтерфейс, важливо віддати перевагу взаємодії з користувачем (User Experience, UX). ПЗ має мати чіткий і логічний макет із зручною навігацією та мінімальним часом навчання для користувачів.

Залучення кінцевих користувачів до процесу проєктування може надати цінну інформацію для створення інтерфейсу користувача, який відповідає їх потребам і очікуванням. Для того, щоб отримати глибше розуміння уподобань користувачів і проблемних моментів потрібно збирати їх відгуки щодо зручності використання та функціональності спеціалізованого ПЗ SRM. Активне врахування відгуків користувачів і вирішення будь-яких проблем чи пропозицій сприятиме подальшому успіху ПЗ та задоволенню користувачів. Включення принципів проєктування, орієнтованого на користувача може допомогти розробити інтерфейс користувача, який задовольняє їх конкретні потреби.

Спеціалізоване ПЗ для керування постачальниками повинно містити основні функції, які відповідають вимогам бізнесу, дозволяють оптимізувати процеси керування постачальниками та підвищити ефективність взаємодії з ними:

- керування профілями постачальників – дозволяє підтримувати повну базу даних ваших постачальників, включаючи їх контактну інформацію, сертифікати та історію ефективності. Ця функція дає змогу легко отримувати доступ до інформації про постачальника й оновлювати її, забезпечуючи точну та актуальну інформацію;

- керування замовленнями на закупівлю – функція, яка дає змогу створювати, відстежувати та ефективно керувати замовленнями на закупівлю, що допомагає автоматизувати процес закупівель, зменшуючи ручні помилки та забезпечуючи своєчасну доставку товарів і послуг;

- моніторинг роботи постачальника – дозволяє оцінити продуктивність постачальників на основі попередньо визначених показників, таких як час доставки, якість і швидкість реагування. Ця функція допомагає визначити найефективніших постачальників і приймати зважені рішення під час вибору або поновлення контрактів;

- управління контрактами – спрощує процес створення, перегляду та керування контрактами з постачальниками. Це централізоване сховище для зберігання контрактних документів, що полегшує доступ і співпрацю. За допомогою керування контрактами можливо відстежувати дати закінчення дії контракту, терміни та умови, забезпечуючи дотримання вимог і мінімізуючи юридичні ризики;

- управління рахунками-фактурами – необхідне для ефективних фінансових операцій. Це дозволяє автоматизувати обробку рахунків-фактур, зменшуючи ручні зусилля та ризик помилок. Управління рахунками дозволяє відстежувати та звіряти платежі, забезпечуючи точні фінансові записи та своєчасні платежі постачальникам (рис. 2.3).



Рисунок 2.3 – Базові функції ПЗ SRM-системи

Включення цих основних функцій у ПЗ SRM-системи дозволить покращити зв'язок і співпрацю з постачальниками, а також підвищити загальну операційну ефективність.

2.2.4 Розробка ПЗ SRM

У процесі розробки вкрай важливо дотримуватися найкращих галузевих практик, щоб забезпечити якість, надійність і зручність обслуговування ПЗ SRM-системи:

- методологія Agile для ітераційної та поступової розробки;
- система контролю версій для керування сховищами вихідного коду та сприяння співпраці;
- перевірка коду та автоматизоване тестування, щоб на ранній стадії виявити та вирішити проблеми.

Дотримання цих найкращих практик призведе до створення надійного програмного рішення для управління взаємовідносинами з постачальниками.

Безпека та конфіденційність даних мають першорядне значення, коли йдеться про керування інформацією про постачальників, тому рекомендується запровадити механізми шифрування, контроль доступу та регулярні перевірки безпеки для захисту конфіденційних даних. Крім того, при розробці спеціалізованого ПЗ системи управління постачальниками слід враховувати дотримання правил захисту (General Data Protection Regulation, GDPR).

2.2.5 Тестування та розгортання ПЗ

Перш ніж розгорнути власне ПЗ для керування постачальниками, дуже важливо провести ретельне тестування, щоб переконатися в його стабільності, функціональності та продуктивності.

Комплексне тестування має охоплювати різні аспекти ПЗ, зокрема:

- модульне тестування для перевірки окремих компонентів;
- тестування інтеграції для забезпечення бездоганної інтеграції з іншими системами;
- тестування продуктивності для оцінки часу відгуку та масштабованості ПЗ;
- тестування безпеки для виявлення потенційних вразливостей;
- тестування прийнятності користувачами для залучення кінцевих користувачів до перевірки зручності та функціональності ПЗ (рис. 2.4).



Рисунок 2.4 – Види тестування ПЗ

Ретельне тестування допоможе виявити та усунути будь-які проблеми чи помилки перед розгортанням спеціалізованого ПЗ SRM у виробництві.

Розгортання спеціалізованого ПЗ для керування відносинами з постачальниками передбачає перенесення ПЗ у робоче середовище, налаштування необхідних параметрів і забезпечення належного підключення до інших систем. Важливо мати чітко визначений план розгортання та

залучити ІТ-команду, щоб забезпечити плавний і безпроблемний процес розгортання [22].

2.2.6 Навчання персоналу та підтримка спеціалізованого ПЗ SRM

Навчання та підтримка є критично важливими аспектами для забезпечення успішного впровадження та використання спеціалізованого ПЗ SRM.

Реалізація комплексного навчання співробітників щодо ефективного використання ПЗ SRM включає навчання їх різноманітним функціям, робочим процесам і найкращим практикам керування постачальниками.

При цьому практичні навчальні заняття, посібники користувача та онлайн-ресурси мають використовуватись, щоб допомогти співробітникам швидко адаптуватися до нового ПЗ.

Оскільки ПЗ розвивається в часі, тому важливо надавати постійну підтримку та оновлення, щоб вирішувати проблеми, запроваджувати нові функції та бути в курсі технологічних тенденцій. Це включає регулярне виправлення помилок, обслуговування та оновлення ПЗ.

Комплексні послуги з підтримки та технічного обслуговування забезпечують максимальну продуктивність і довговічність спеціалізованого ПЗ для управління відносинами з постачальниками.

2.2.7 Оцінка ефективності впровадження

Оцінка ефективності використання спеціалізованого ПЗ SRM має життєво важливе значення для оцінки його впливу на бізнес і визначення областей для вдосконалення. Регулярне відстеження ключових показників ефективності (Key Performance Indicator, KPI), які узгоджені з SRM, дозволяє оцінити успіх ПЗ. Приклади KPI, які слід розглянути, включають:

- оцінки продуктивності постачальника;

- рівень своєчасності доставки;
- економія коштів завдяки покращенню перемовин з постачальниками.

Регулярний аналіз цих ключових показників ефективності допоможе оцінити ефективність прийнятої стратегії для управління постачальниками та приймати рішення на основі даних для постійного вдосконалення.

2.3 Висновки до розділу 2

В цьому розділі відповідно до мети та задач роботи було розроблено методику створення спеціалізованого ПЗ SRM-системи, яке допоможе трансформувати процеси управління постачальниками та досягти успіху в бізнесі.

Створення спеціалізованого ПЗ для керування взаємовідносинами з постачальниками – це складний процес, який вимагає ретельного планування, технічних знань і глибокого розуміння вимог бізнесу. Розроблена методика охоплює всі етапи створення надійного та масштабованого ПЗ SRM, адаптованого до потреб бізнесу.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ SRM-СИСТЕМИ

3.1 Аналіз вимог

Призначення SRM-системи – організація ефективної взаємодії замовників та постачальників послуг (виконавців).

Актуальність розробки обумовлюється тим, що бізнесу доволі часто приходится стикатися з неякісним виконанням робіт, або виконанням не в повному обсязі та не у вказані терміни. Не можна виключати потрапляння на шахраїв, які візьмуть гроші та зникнуть у невідомому напрямку. Також, можуть виникнути спірні питання, вирішити які без сторонньої допомоги важко або взагалі неможливо. Тож потрібен інструмент, який допоможе;

- замовнику знайти виконавця;
- виконавцю провести оцінку робіт, які планується виконати та узгодити кошторис з замовником;
- компанії-посереднику після підписання контракту відстежувати та контролювати хід робіт на всіх стадіях від постановки завдання та оцінки вартості до виконання всього обсягу робіт та прийняття результату замовником;
- зберігати всі обговорення (всі вимоги від замовника та всі узгодження від виконавця) та вносити потрібні зміни в завдання під час виконання.
- забезпечити безпеку платіжних операцій, щоб виконавець зміг отримати винагороду лише після підтвердження виконання робіт замовником;
- захистити виконавця, щоб гарантувати йому оплату за якісно виконану роботу.

Розроблювана система повинна мати інтуїтивно зрозумілий інтерфейс та не вимагати придбання додаткового обладнання або програмного забезпечення.

3.1.1 Функціональні вимоги до системи

Система повинна мати три типи користувачів: замовники, виконавці та адміністратор. Користувач повинен мати змогу зареєструватися в системі використовуючи свою електронну пошту, а потім увійти в систему, використовуючи свою електронну пошту та пароль. Всі дії користувача, які можуть впливати на роботу мають зберігатися в системі.

Замовник повинен мати змогу описати як він бачить кінцевий результат робіт, скільки коштів та часу він готовий виділити для виконання цих робіт.

Виконавець повинен мати змогу вказати свій досвід, види робіт, які він може виконувати та надати приклади завершених робіт; скласти кошторис та передати його на розгляд замовнику. Кошторис повинен бути поділений на фази робіт, які в свою чергу поділені на окремі задачі. Всі фази та задачі повинні мати дату початку та закінчення. Всі задачі повинні містити опис робіт, які планується виконати, вартість матеріалів та самої роботи.

Замовник повинен мати змогу переглянути кошторис та затвердити його або відхилити та направити на доопрацювання. При затвердженні кошторису повинен складатися контракт в електронному вигляді.

Всі зміни та доповнення до контракту повинні складатися в письмовому вигляді та зберігатися в системі. Зміни та доповнення до контракту повинні бути складені виконавцем та після складання кошторису затверджені замовником. Затвердження змін повинно бути виконане за допомогою електронного підпису. Зміни та доповнення після підписання повинні вважатися невід'ємною частиною договору. Замовник та виконавець повинні мати змогу достроково розірвати договір.

Система повинна забезпечити безпеку операцій між замовником та виконавцем. Після підписання договору, замовник повинен перевести кошти за наступну фазу робіт на спеціальний рахунок, де вони повинні зберігатися як гарантія того, що виконавець отримає їх після виконання всіх завдань у поточній фазі робіт.

Система повинна давати змогу переглянути поточний статус проєкту, важливі оновлення та останні дії. Система повинна давати змогу кожному учаснику проєкту відслідковувати кожен етап проєкту.

Система повинна давати змогу спілкування між замовником та виконавцем. Вся переписка повинна зберігатися в системі.

Система повинна надавати інформацію щодо розрахунків. Виконавець повинен мати змогу подати запит на оплату лише після виконання всіх завдань в поточній фазі робіт. Замовник повинен мати змогу дозволити виплатити виконавцю кошти за якісно виконану роботу. Система повинна мати змогу перевести кошти на рахунок виконавця після схвалення замовником.

Система повинна мати можливість зберігати та відображати всі документи, якими замовник та виконавець обмінюються при виконанні договору. Замовник повинен мати можливість складати договори з різними виконавцями одночасно. Виконавець повинен мати можливість складати договори з різними замовниками одночасно. Система повинна надавати замовнику та виконавцю інструменти для перегляду їх договорів на всіх стадіях роботи. Виконавець повинен мати можливість переглянути проєкт наданий замовником та вирішити чи приймати його в роботу чи ні.

Користувачі повинні мати можливість редагувати свої персональні дані. Виконавець повинен мати можливість редагувати своє портфоліо, види робіт, досвід та інші. Користувачі повинні мати можливість змінити пароль на вхід в систему. Система повинна надавати замовнику можливість підбору виконавця за певними критеріями. Система повинна надавати користувачам можливість додавання та зміни банківських реквізитів.

Система повинна давати можливість адміністратору системи отримувати загальну інформацію по всіх проєктах та по кожному проєкту окремо. Адміністратор системи повинен мати доступ до всіх проєктів та користувачів для вирішення спірних питань та оперативного усунення виявлених недоліків.

3.1.2 Нефункціональні вимоги

Користувач повинен мати змогу користуватись застосунком без придбання спеціального обладнання чи встановлення додаткового ПЗ.

Первинне завантаження застосунка не повинно перевищувати 5 с. Повторне завантаження не повинно перевищувати 2 с. Розташування елементів системи повинно підлаштовуватись під роздільну здатність екрану.

Передбачається, що на початку роботи системи в ній буде зареєстровано близько 10 користувачів, через рік кількість планується підвищити до 250 активних користувачів. Передбачається, що через рік одночасно працюючих в системі буде близько 50 користувачів.

Максимально допустимий час очікування відповіді сервера повинен складати 5 с. Середній час очікування відповіді від сервера повинен складати не більше 100 мс.

Пароль на вхід в систему повинен бути довжиною від 8 до 16 символів та включати прописні та строкові букви та цифри. Паролі користувачів повинні зберігатися в базі даних у вигляді хешу. При реєстрації потрібно переконатися, що користувач правильно ввів адресу своєї поштової скриньки шляхом відправлення перевірного листа. Користувач повинен мати право видалити свій обліковий запис з системи.

Максимальний розмір доданого файлу повинен складати до 20 Мб.

Прямий доступ до бази даних повинен бути дозволений лише з певної групи IP-адрес. Користувач повинен взаємодіяти з сервером по протоколу HTTPS. Максимальний термін дії сесії повинен складати 72 години. Застосунок повинен працювати во всіх версіях браузерів, в яких на теперішній час працює мінімум 99 % користувачів. Застосунок повинен працювати на всіх типах підключення до мережі інтернет починаючи з 2G.

Система повинна розмежовувати права доступу до функцій відповідно до ролей користувачів (рис.3.1): замовник, виконавець, адміністратор.

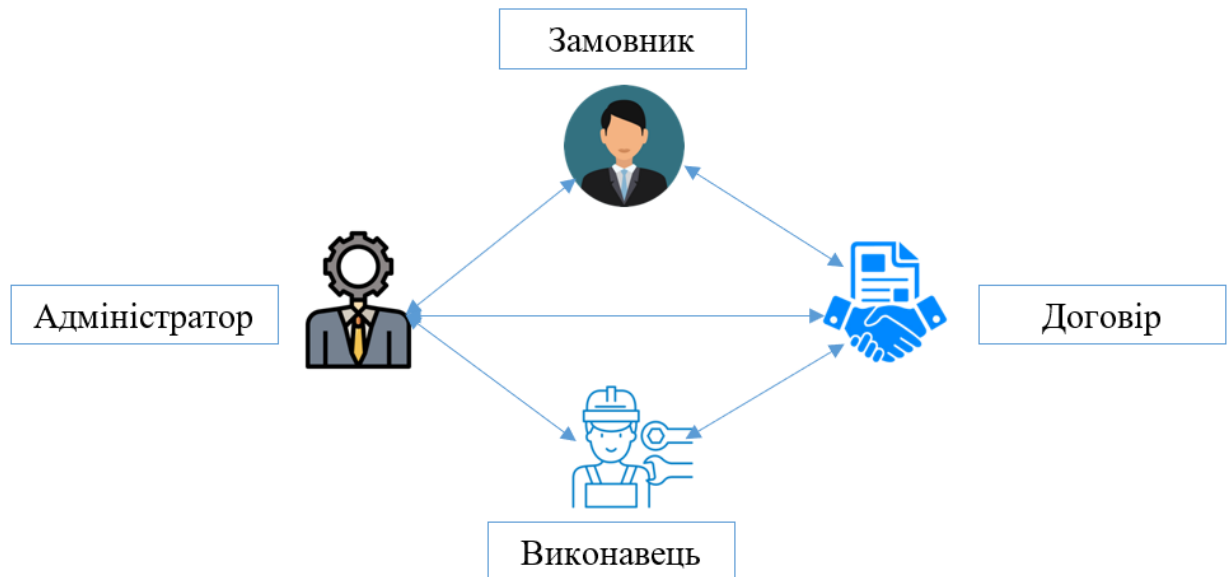


Рисунок 3.1 – Схема взаємодії користувачів системи

Доступ до системи повинен надаватися користувачам лише після їх згоди на збереження та обробку персональних даних, згідно Закону України про захист персональних даних [23].

3.2 Архітектура системи

Архітектура системи складається з фронтенд та бекенд модулів, бази даних (БД), кешу та хмарного сховища (рис.3.2).

Фронтенд модуль призначений для взаємодії з користувачем. В ньому відображається вся інформація застосунку, яку він дозволяє зручно додавати та змінювати.

Бекенд приймає запити від фронтенду, перевіряє чи достатньо у користувача прав на проведення операції і в залежності від результату або дозволяє отримати (змінити) дані або забороняє. Також бекенд призначений для зберігання ключів доступу та керування самим доступом до зовнішніх систем, таких як хмарні сховища, фінансові установи та інші. Секретні ключі не повинні бути розташовані на фронтенді, щоб не давати змогу злоумисникам заволодіти ними.

Модуль кешування призначений для зберігання найбільш затребуваної інформації або тієї, яка рідко змінюється. Завдяки тому, що інформація в ньому зберігається в оперативній пам'яті, процес отримання розташованих там даних значно прискорюється.

БД, на відміну від кешу, призначена для постійного зберігання всієї інформації застосунку.

Для збереження файлів використовуються хмарні сховища. З ними зручно працювати, бо вони виступають незалежним сервісом і користувач отримує доступ на завантаження файлу за унікальним посиланням та не навантажує своїм запитом бекенд чи БД. Також, всі турботи про резервне копіювання та підтримку роботи сервісу перекладаються на плечі хмарного сховища.

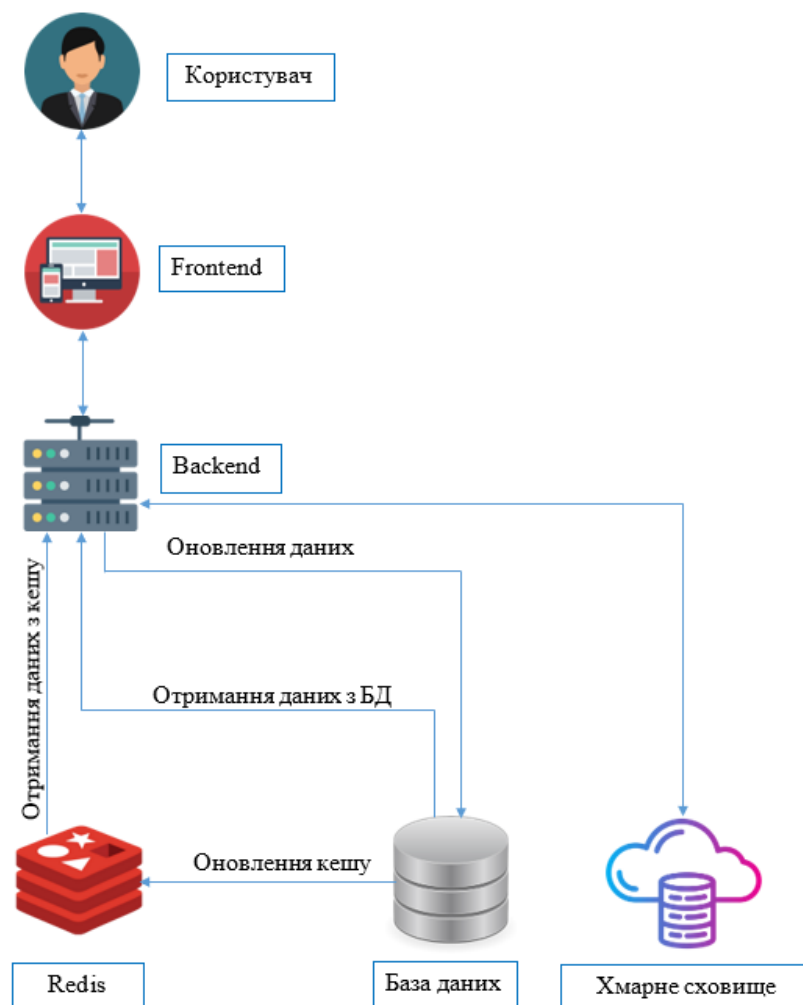


Рисунок 3.2 – Архітектура застосунку

3.3 Аналіз та обґрунтування вибору інструментарію розробки

Аналіз та вибір сучасних технологій потрібно проводити окремо для кожної частини системи, оскільки кожен модуль повинен мати своє функціональне навантаження та виконувати в повному обсязі всі покладені на нього завдання.

3.3.1 Аналіз та обґрунтування вибору технологій розробки для фронтенд частини

Як показав аналіз вимог до системи, користувач повинен мати змогу користуватись застосунком на будь-якій системі та без необхідності встановлення додаткового ПЗ. Таким вимогам задовольняє лише вебзастосунок, адже будь-яка сучасна операційна система підтримує роботу браузерів.

Існує багато мов програмування, які дозволяють створити такий застосунок. Результати аналізу переваг та недоліків найпопулярніших мов програмування PHP, Python, JavaScript, Java наведено в табл. 3.1 [24].

Таблиця 3.1 – Порівняльний аналіз мов програмування вебзастосунків [24]

Мова	Переваги	Недоліки
1	2	3
PHP	Мова спеціалізована для веброзробки. Широко використовується для побудову динамічних вебсайтів та застосунків. Підтримує багатопоточність та великі корпоративні застосунки	Має репутацію не самої надійної мови. Не так багато можливостей за межами веброзробки

Продовження табл. 3.1

1	2	3
Python	Простий та читаний синтаксис. Багато бібліотек для наукових розрахунків та машинного навчання. Широко використовується в науці та штучному інтелекті	Не є нативною мовою для браузерів, треба інтерпретувати за допомогою transpiler. Веброзробка може потребувати додаткових зусиль
JavaScript	Мова розроблена спеціально для браузерів, забезпечує інтерактивність вебсторінок. Є серверна реалізація – Node.js. Багато бібліотек та фреймворків. Оптимізована для фронтенд та вебінтерфейсів	Динамічна типізація може викликати помилки під час виконання
Java	Переносимість коду між різними платформами. Сильна типізація та статична перевірка типів. Підтримує багатопоточність та великі корпоративні застосунки	Потребує компіляції, що може сповільнити розробку. Не є нативною мовою для браузерів

Мова JavaScript перемагає конкурентів завдяки тому, що вона є нативною для веброзробки та забезпечує інтерактивність вебсторінок, що робить її ідеальним вибором для фронтенда вебзастосунку. Згідно даних статистики за минулий рік, близько 98% всіх сайтів використовують JavaScript як мову програмування для клієнтської частини [25].

Задля усунення можливих помилок при динамічній типізації, полегшення читаємості коду, рефакторінгу та повторного використання коду, а також для виявлення помилок ще на етапі розробки потрібно встановити TypeScript.

Аналіз джерел [26]-[28] показав, що використання TypeScript надає наступні можливості:

- статична типізація: TypeScript дозволяє визначати типи даних для змінних, функцій та об'єктів, що допомагає виявити та запобігти безлічі

потенційних помилок на етапі компіляції. Це збільшує надійність та читання коду;

- покращена технологія: використовуючи TypeScript, отримуємо підказки та автодоповнення під час написання коду. Це спрощує процес розробки, зменшує кількість помилок та прискорює розробку;
- читання коду: явне визначення типів робить код зрозумілішим і самодокументуючим;
- рефакторинг: TypeScript спрощує процес рефакторингу, оскільки зміна типів даних або структури коду супроводжується автоматичною перевіркою та попередженнями про можливі проблеми;
- екосистема: TypeScript має велику екосистему інструментів та бібліотек, включаючи підтримку в популярних IDE (наприклад, Visual Studio Code) та плагіни для фреймворків, таких як React;
- дотримання стандартів: TypeScript слідує стандартам ECMAScript і враховує майбутні зміни мови JavaScript, що забезпечує актуальність та сумісність коду;
- зниження помилок часу виконання: завдяки статичній типізації, TypeScript допомагає виявляти помилки до запуску програми, що заощаджує час та ресурси на пошук та виправлення помилок під час виконання;
- команда та спільна розробка: статична типізація допомагає керувати великими кодовими базами та покращує процес спільної розробки, оскільки зменшує невизначеність та конфлікти у коді;
- безпека: TypeScript дозволяє більш надійно обробляти введення користувача та взаємодію з сервером, що допомагає захистити програму від деяких видів атак [26]-[28].

Для прискорення розробки застосунку потрібно використовувати фреймворки та бібліотеки, які допомагають при вирішенні багатьох задач в програмуванні та розробці. Аналіз існуючих бібліотек та фреймворків наведено в табл. 3.2 [29]-[40].

Таблиця 3.2 – Порівняльний аналіз фреймворків та бібліотек [29]-[40]

Критерій	React	jQuery	Angular	Vue.js
Тип фреймворку	Бібліотека для створення UI	Бібліотека для маніпуляцій з DOM	Комплексний фреймворк	Прогресивний фреймворк
Складність	Середня	Низька	Висока	Середня
Оновлення DOM	Віртуальний DOM	Пряма маніпуляція DOM	Пряма маніпуляція DOM	Віртуальний DOM
Компонентний підхід	Так	Ні	Так	Так
Оптимізація продуктивності	Так	Ні	Так	Так
Громадська спільнота	Велике	Велике	Велике	Значна
Використання JSX	Так	Ні	Так	Так
Керування станом	Через setState	Пряма маніпуляція DOM	Складне керування станом	Через data та props
Маршрутизація	Додаткові бібліотеки	Немає	Вбудована маршрутизація	Додаткові бібліотеки
Розширюваність	Широкі можливості для настройки та розширення	Скінчені можливості	Висока розширюваність	Широкі можливості
Офіційна документація	Так	Так	Так	Так
Активна підтримка	Так	Слабка	Так	Так

Згідно статистики використання React є лідером на ринку [41] та може бути обраний в якості інструментарію розробки. Головні переваги React:

- віртуальний DOM: React використовує віртуальний DOM (Virtual Domain, VDOM), що дозволяє ефективно оновлювати лише ті частини вебсторінки, які змінилися, мінімізуючи кількість операцій на реальному DOM, що робить React швидким та ефективним;
- компонентний підхід: React пропонує розробку вебзастосунків з використанням компонентів, які можна багаторазово використовувати та комбінувати, що спрощує організацію коду, його розуміння та підтримку;
- JSX: JSX (JavaScript XML) дозволяє писати компоненти в декларативному стилі, що робить код більш читаним та зрозумілим. Він також забезпечує статичну типізацію та підказки під час розробки;
- велика спільнота: React має величезну спільноту розробників і безліч сторонніх бібліотек і плагінів. Це означає, що можна легко знайти рішення для багатьох завдань і проблем;
- однонаправлений потік даних: React рекомендує однонаправлений потік даних, що сприяє передбачуваності та налагодженню програм, що також спрощує розробку великих та складних програм;
- розробка під мобільні пристрої: з React Native можна створювати мобільні програми для iOS та Android, використовуючи знання та компоненти React, що заощаджує час та зусилля при розробці для кількох платформ;
- інструменти розробника: React має безліч інструментів розробника, таких як React DevTools, які полегшують налагодження та профілювання програм;
- сумісність: React легко інтегрується з іншими бібліотеками та фреймворками, що дозволяє використовувати його у різноманітних екосистемах [42]-[44].

Для прискорення розробки вебзастосунку доцільно обрати UI-бібліотеки, що надають готові набори функцій та компонентів, які можна додавати та змінювати у вебзастосунку (табл. 3.3) [45]-[48].

Таблиця 3.3 – Порівняльний аналіз UI-бібліотек [45]-[48]

Характеристика	Ant Design	Material UI	React Bootstrap	Chakra UI
Тип	UI-компоненти та дизайн	UI-компоненти та дизайн	UI-компоненти та дизайн	UI-компоненти та дизайн
Ліцензія	MIT	MIT	MIT	MIT
Підтримка TypeScript	Так	Так	Так	Так
Стилі	LESS та CSS	JSS, CSS-in-JS	CSS	Emotion та CSS-in-JS
Дизайн	Модерн та креативний	Матеріальний дизайн	Bootstrap	Мінімалістичний та сучасний
Гнучкість компонентів	Висока	Висока	Середня	Висока
Темизація	Підтримує теми	Підтримує теми	Підтримує теми	Підтримує теми
Спільнота та підтримка	Активна спільнота та документація	Активна спільнота та документація	Активна спільнота та документація	Активна спільнота та документація
Іконки	Вбудовані компоненти та пакет «@ant-design/icons»	Вбудовані компоненти та пакет «@mui/icons-material»	Вбудовані компоненти	Вбудовані компоненти
Анімація	Підтримує анімацію	Підтримує анімацію	Частково підтримує	Підтримує анімацію
Документація	Обширна та добре структурована	Обширна та добре структурована	Подрібна документація	Обширна та добре структурована

Після ретельного вивчення документації та відгуків розробників було обрано Ant Design [49]-[51]. Основні переваги:

- модерний та креативний дизайн: Ant Design надає сучасні та креативні дизайн-рішення, які допомагають створювати естетично приємні та інтуїтивно зрозумілі інтерфейси користувача;
- великий набір компонентів: Ant Design включає багатий набір готових компонентів, таких як кнопки, форми, таблиці, графіки, навігаційні елементи і багато інших, що спрощує процес створення інтерфейсу та прискорює розробку;
- підтримка тем: Ant Design має потужну систему темизації, дозволяючи налаштовувати зовнішній вигляд програми відповідно до корпоративного стилю або уподобань проєкту;
- активна спільнота та документація: Ant Design має активну спільноту розробників та велику документацію, що полегшує вирішення проблем та знаходження інформації про компоненти та функціональність бібліотеки;
- підтримка TypeScript: Ant Design добре інтегрується з TypeScript, що робить код більш надійним та полегшує розробку у великих проєктах;
- анімації: Ant Design підтримує анімації, що дозволяє створювати більш інтерактивні і привабливі інтерфейси користувача;
- міжнародна підтримка: Ant Design надає переклади кількома мовами, що робить його придатним для міжнародних проєктів;
- вбудовані іконки: Ant Design надає вбудовані іконки, що робить процес додавання графічних елементів у застосунок зручним;
- популярність: Ant Design широко використовується в спільноті розробників та має безліч проєктів та плагінів, які будуються на його основі [49]-[51].

Яка б не була продумана UI-бібліотека, але все одно виникнуть моменти, коли буде потрібно змінити колір, відступи або щось інше в оформленні компонента. Для вирішення цих питань існує CSS, але це базовий

рівень. Виникають моменти, коли потрібно змінити вигляд компонента в залежності від даних, які приходять з бекенду та існують засоби зробити цей процес простішим та зручнішим для розробника.

Результати аналізу існуючих рішень для React-застосунків наведено в табл. 3.4 [52]-[55].

Таблиця 3.4 – Аналіз рішень для React-застосунків [52]-[55]

Метод стилізації	Переваги
CSS селектори	Простий та широко розповсюджений метод
	Підтримка всіх можливостей CSS включаючи псевдокласи та медіа-запити
	Легко впроваджується в існуючі проекти
Inline стилі	Простота в використанні та розумінні.
	Стилі можуть бути динамічними в залежності від даних.
	Використовується вбудований стиль JSX, що спрощує компоновку компонентів.
CSS Modules	Локальна ізоляція стилів, що запобігає конфліктам класів.
	Модульність – стилі обмежені одним компонентом та можуть бути перевикористані.
	Підтримується автодоповнення та статична перевірка типів
Styled-components	Створення компонентів з вбудованими стилями, що покращує читаємість коду.
	Динамічні стилі на основі стану компоненту.
	Можливість використання JavaScript для стилізації, що дозволяє динамічно змінювати стилі в залежності від даних.
	Автоматична генерація унікальних класів для ізоляції стилів.

Після ретельного аналізу документації, статей, переваг та недоліків було обрано для стилізації Styled-components, оскільки ця бібліотека має ряд переваг [56]-[57]:

- динамічні стилі: Styled-components дозволяє створювати динамічні стилі з урахуванням стану компонента чи даних, переданих до компонента. Це робить стилізацію гнучкішою і дозволяє створювати компоненти, які можуть змінювати свій вигляд залежно від контексту;
- вбудовані стилі: у Styled-components стилі вбудовуються безпосередньо для визначення компонентів. Це робить код більш читаним, тому що стилі та компоненти знаходяться в одному місці, і немає потреби створювати окремі файли для стилів;
- підтримка інтерполяції: можна впроваджувати JavaScript в стилі, використовуючи інтерполяцію, що спрощує динамічну зміну стилів. Це особливо корисно, коли потрібно передавати змінні або обчислені значення в стилі компонента;
- автоматичне створення унікальних класів: Styled-components автоматично генерує унікальні класи кожного компонента. Це забезпечує ізоляцію стилів навіть у разі використання однакових імен компонентів;
- підтримка серверного рендерингу: Styled-components забезпечує хорошу підтримку серверного рендерингу (SSR), що робить її добрим вибором для універсальних застосунків;
- активна спільнота та підтримка: Styled-components має активну спільноту та продовжує розвиватися, що забезпечує підтримку та оновлення бібліотеки [56]-[57].

При розробці будь якого застосунку потрібно постійно підтримувати чистоту коду, щоб будь який файл був написаний в єдиному стилі з єдиними відступами. Це дозволить легко читати код та зробить його більш зрозумілим при перегляді. Але підтримувати такий стан коду вручну складно. Аналіз існуючих інструментів наведено в табл. 3.5 [58]-[60].

Таблиця 3.5 – Аналіз інструментів для форматування коду [58]-[60]

Характеристика	Prettier	Standard	JS Beautifier
Автоматичне форматування	Автоматично форматує код з заданим стилем оформлення	Автоматично форматує код	Надає безліч опцій
Інтеграція з іншими інструментами	Добре інтегрується з іншими інструментами, такими як ESLint	Вимагає додаткової конфігурації з ESLint	Розширення та інтеграції доступні для різних редакторів
Популярність	Популярний та широко використовується в спільноті розробників JavaScript	Популярний, але менше ніж Prettier	Популярний та широко використовується
Не налаштовуваний стиль	Обмежує налаштування форматування. Стил ь визначається самим Prettier та його налаштуваннями	Не дозволяє налаштовувати стиль	Деякі користувацькі параметри можуть бути складними
Незадовільне форматування	Іноді може формувати код не так, як очікувалось, особливо в складних випадках	Не завжди притримується суворих правил	Форматування може бути непередбачуваним

За результатами аналізу було обрано Prettier [58]. Його переваги:

- простота використання: Prettier дуже легко інтегрувати та використовувати. Потрібно лише встановити його, налаштувати кілька параметрів і він автоматично форматує код відповідно до заданих стилів;
- однозначність форматування: Prettier визначає однозначні стандарти форматування, і все, що потрібно зробити, це вказати бажаний стиль;
- безліч мов, що підтримуються: Prettier підтримує безліч мов програмування, включаючи JavaScript, TypeScript, CSS, HTML, JSON, YAML, та інші. Це дозволяє використовувати той самий інструмент для різних мов;

- активна спільнота та підтримка: Prettier має велику активну спільноту розробників, які постійно працюють над покращенням та підтримкою інструменту. Це гарантує оновлення та виправлення помилок;
- інтеграція з іншими інструментами: Prettier може легко інтегруватися з іншими інструментами, такими як ESLint або TypeScript. Це дозволяє об'єднати лінтинг та форматування в один робочий процес;
- широке розповсюдження: Prettier став стандартом у спільноті JavaScript та розробки вебзастосунків. Безліч проєктів та компаній використовують його, що сприяє його популярності;
- активний розвиток: розробка Prettier продовжується і нові функції та покращення регулярно додаються до інструменту [58].

Навіть при ретельному підході до розробки складно гарантувати відсутність орфографічних помилок в назвах функцій та змінних, або тексту, який буде бачити користувач. Тож потрібно забезпечити можливість аналізувати код та знаходити потенційні помилки.

Результати аналізу інструментів, які виконують таку роботу наведено в табл. 3.6 [61]-[64].

Таблиця 3.6 – Аналіз інструментів для перевірки якості коду [61]-[64]

Інструменти	Переваги	Недоліки
1	2	3
JSHint	Простота використання	Обмежена здатність для налаштувань
	Зручний для починаючих розробників	Менш гнучкі можливості для налаштування правил
	Обмежений набір правил за замовченням	Обмежене сповіщення про помилки

Продовження таблиці 3.6

1	2	3
JSLint	Строгі правила оформлення коду	Мала гнучкість налаштувань
	Малий об'єм конфігурації	Не знаходить синтаксичні помилки
StandardJS	Простота та швидкість налаштувань	Не такий гнучкий, як ESLint
	Базується на стандартних правилах	Дає менше змоги на конфігурування
	Прагне до стандартизації стилю коду	Можуть з'явитися розбіжності з кодом
ESLint	Велика гнучкість та здатність до налаштувань	Може знадобитися час для налаштування
	Велика спільнота та екосистема плагінів	Необхідність налаштувань для конкретних проєктів
	Багато правил, які можна налаштувати	Деякі конфігурації можуть бути складними

Аналіз показав, що ESLint є безперечним лідером в своїй галузі. Кожен тиждень розробники всього світу завантажують близько тридцяти шости мільйонів разів цей інструмент [65].

ESLint має декілька переваг в порівнянні з конкурентами [66]-[67]:

- висока гнучкість та можливості для налаштування: ESLint надає широкі можливості для налаштування правил та конфігурації під конкретні потреби проєкту, що дозволяє легко адаптувати ESLint до стилю та вимог команди;
- підтримка ES6 та нових можливостей: ESLint активно оновлюється та підтримує останні версії JavaScript (включаючи ES6 та новіші), що дозволяє використовувати сучасні функції та можливості мови без проблем;

- велика спільнота та екосистема плагінів: ESLint має широке співтовариство користувачів та розробників, що призводить до великої кількості доступних плагінів та розширень. Це забезпечує багату екосистему для виявлення та виправлення різних видів проблем;
- багато правил, що налаштовуються: ESLint надає велику кількість вбудованих правил, а також можливість створення власних правил. Це дозволяє детально контролювати стиль та якість коду;
- активна підтримка та розробка: ESLint активно підтримується та розробляється спільнотою. Нові версії та оновлення випускаються регулярно, що гарантує актуальність та надійність інструменту;
- підтримка багатьох інтегрованих середовищ розробки (Integrated Development Environment, IDE): ESLint інтегрується з популярними IDE, такими як Visual Studio Code, Sublime Text, та іншими. Це полегшує використання інструменту у повсякденній розробці;
- багатомовна підтримка: ESLint підтримує не лише JavaScript, а й різні діалекти, такі як JSX (для React) та TypeScript [66]-[67].

Оскільки весь код програми буде розділений на частини і збережений в окремі файли, то для збирання цих шматочків коду потрібно підібрати збирач модулів. Цей збирач повинен також надавати змогу зменшити об'єм коду для пришвидшення його завантаження на пристрій користувача. Результати аналізу існуючих збирачів коду наведено в табл. 3.7 [68]-[73].

Таблиця 3.7 – Аналіз переваг та недоліків збирачів коду [68]-[73]

Інструмент збирання	Переваги	Недоліки
1	2	3
Webpack	Потужні можливості конфігурації	Складне налаштування
	Широка спільнота та плагіни	Довгий час навчання
	Code splitting та динамічне завантаження	

Продовження таблиці 3.7

1	2	3
Parcel	Простота в використанні	Обмежене налаштування
	Автоматичне виявлення залежностей	Може бути повільним в деяких випадках
Rollup	Оптимізований для бібліотек та модулів	Менш підходить для великих застосунків
	Підтримка Tree-shaking	Обмежене конфігурування
Browserify	Використовує модулі Node.js в браузері	Може бути застарілим для деяких проєктів
	Добре підходить до бібліотек	Обмежене конфігурування
Gulp	Автоматизація різних задач	Не є повноцінним інструментом збирання
	Багато плагінів для задач	Потребує більше налаштувань

Найпопулярнішим збирачем модулів є Webpack, що тижня його завантажують близько двадцяти сими мільйонів разів [74]. Проаналізувавши статті [75]-[76] можна зробити висновок, що Webpack обирають через наступні особливості:

- потужні можливості конфігурації: що дозволяє створювати складні та високооптимізовані робочі процеси з урахуванням потреб проєкту;
- Code splitting: Webpack підтримує поділ коду на бандли, що дозволяє оптимізувати завантаження ресурсів вебпрограми та зменшити початкове завантаження сторінки;
- підтримка плагінів та завантажувачів: Webpack має багату екосистему плагінів та завантажувачів, які полегшують інтеграцію з різними бібліотеками, фреймворками та інструментами;
- велика спільнота та документація: забезпечує підтримку та наявність безлічі ресурсів для навчання та вирішення проблем;
- довгострокова підтримка: Webpack стабільний та активно розвивається, що робить його надійним інструментом для проєктів [75]-[76].

3.3.2 Аналіз та обґрунтування вибору бази даних

Існує величезна кількість різновидів БД, які відрізняються за різними критеріями (рис. 3.3) [77]-[80].

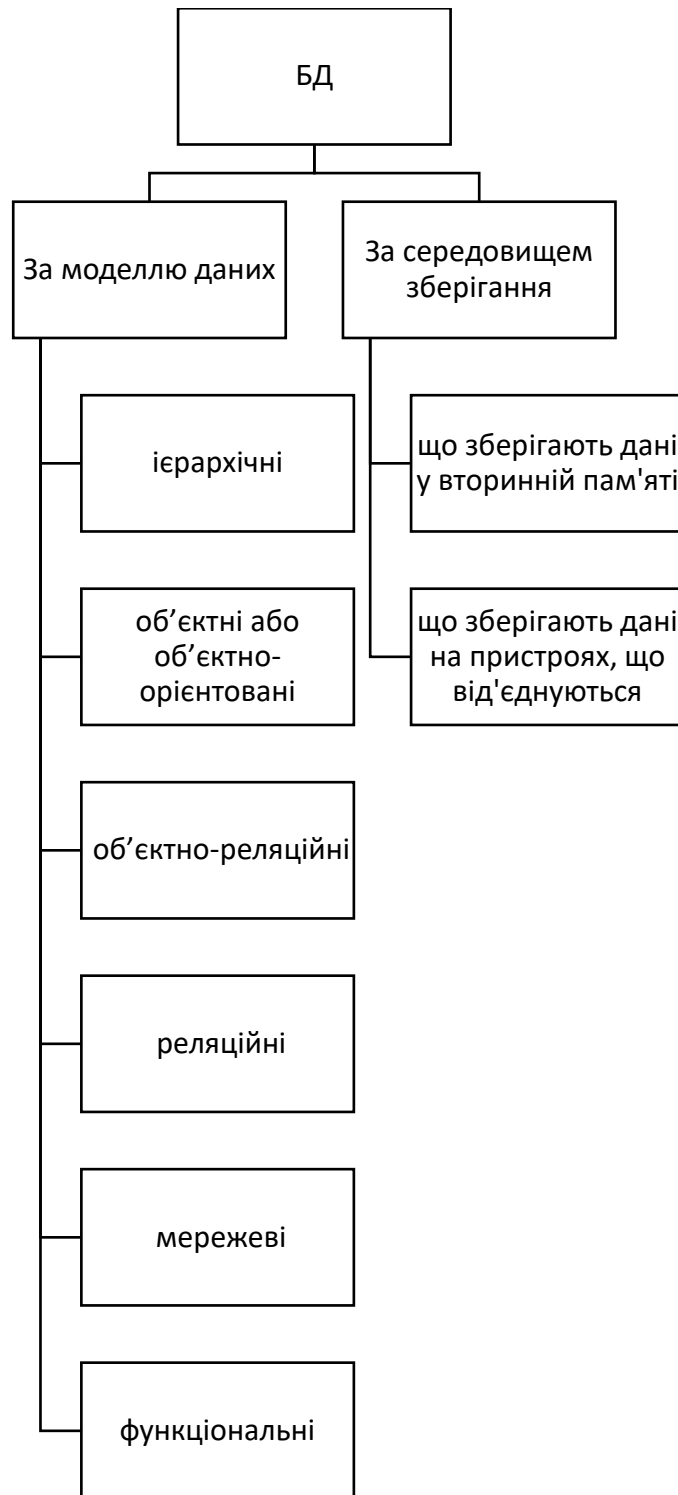


Рисунок 3.3 – Класифікація БД [77]-[80]

При цьому у всіх класах так чи інакше використовуються всі середовища зберігання, наприклад, для резидентних баз даних СКБД записує в постійну пам'ять журнали передзапису, а для традиційних баз використовується кеш в оперативній пам'яті. Результати аналізу найпопулярніших СКБД зведено в табл. 3.8 [77]-[80].

Таблиця 3.8 – Аналіз існуючих СКБД [77]-[80]

СКБД	Переваги	Недоліки
1	2	3
Oracle	Висока надійність та стійкість	Високі витрати на ліцензії та обслуговування
	Багаті можливості для управління даними	Складне налаштування та керування
	Підтримка високої продуктивності та масштабу	Закрита схема даних
	Широкий набір функціональних можливостей	Потрібне навчання та досвід для адміністрування
MySQL	Безкоштовна та з відкритим вихідним кодом	Обмежені можливості для великих підприємств
	Легко впроваджується та керується	Обмежена підтримка для складних запитів
	Хороша продуктивність	Обмежені можливості масштабування
	Широке співтовариство та екосистема	Недостатня підтримка транзакцій та узгодження

Продовження таблиці 3.8

1	2	3
Microsoft SQL Server	Інтеграція з екосистемою Microsoft	Ліцензійна модель може бути дорогою
	Висока продуктивність	Обмежено підтримку платформ, відмінних від Windows
	Широкі можливості для розробників	Недостатня спільнота з відкритим вихідним кодом
	Зручне управління та моніторинг	Недостатня масштабованість
PostgreSQL	З відкритим вихідним кодом	Не така широка підтримка графічних інтерфейсів
	Висока продуктивність	Не така поширена як MySQL
	Розширювана та гнучка схема даних	Потрібно більше зусиль для налаштування
	Підтримка безлічі розширень та доповнень	

Після аналізу статей [81]-[83] було обрано БД PostgreSQL. Це потужна та популярна СКБД з безліччю переваг:

- відкритий вихідний код: PostgreSQL поширюється під ліцензією PostgreSQL, яка дозволяє використовувати його безкоштовно та вільно розповсюджувати, змінювати та вносити до нього зміни. Це означає, що PostgreSQL доступний всім без додаткових витрат на ліцензії;
- гнучка схема даних: PostgreSQL надає широкі можливості для створення гнучких та складних схем даних. Можна визначати типи даних користувача, створювати збережені процедури та функції на різних мовах, а також працювати з JSON та іншими неструктурованими даними;

- підтримка багатьох розширень: система підтримує безліч розширень, які дозволяють розширити функціональність PostgreSQL. Ці розширення дозволяють додавати нові типи даних, індекси, функції та інші можливості без зміни вихідного коду СКБД;
- висока продуктивність: PostgreSQL забезпечує високу продуктивність при правильному налаштуванні. Він має механізми оптимізації запитів, підтримує паралельне виконання запитів, а також індекси та подання для прискорення доступу до даних;
- розширюваність та масштабованість: PostgreSQL легко масштабується і може бути налаштований для роботи з великими обсягами даних. Він підтримує реплікацію, шардування та кластеризацію для забезпечення відмовостійкості та високої доступності;
- транзакції та узгодження: PostgreSQL забезпечує надійне керування транзакціями та забезпечує узгодженість даних. Він підтримує ACID (Atomicity, Consistency, Isolation, Durability) – набір характеристик, що гарантує цілісність даних;
- активна спільнота: існує активна спільнота розробників та користувачів PostgreSQL, яка підтримує та розвиває систему. Це забезпечує регулярні оновлення та нові версії, а також великі ресурси, включаючи документацію та форуми підтримки;
- безліч клієнтських інтерфейсів та інструментів: існує багато клієнтських застосунків та інструментів для керування PostgreSQL, що робить його зручним для розробників та адміністраторів;
- підтримка різних операційних систем: PostgreSQL підтримує різні операційні системи, включаючи Linux, Windows, MacOS та інші [81]-[83].

Зазвичай замовник та виконавець будуть знаходитись в системі одночасно. Якщо один з них щось напише іншому, то треба буде зайти в систему та відкрити це повідомлення, файл або задачу. Отже вони обидва будуть отримувати однакову інформацію з БД. Для прискорення роботи системи, цю інформацію треба зберігати в якомусь місті, з якого її отримання

буде займати мінімальний проміжок часу та не завантажувати БД однаковими запитами. Є також інформація, яка рідко змінюється, але доступ до неї мають всі користувачі. Наприклад, база поштових індексів, або список видів робіт.

Результати аналізу систем, які дозволяють пришвидшити отримання даних, наведено в табл. 3.9 [84]-[87].

Таблиця 3.9 – Огляд систем керування даними [84]-[87]

Критерій/ система	Memcached	Apache Kafka	Redis	Amazon ElastiCache
1	2	3	4	5
Тип даних	Прості ключ-значення	Потокові події	Ключ-значення	Прості ключ-значення
Зберігання даних	В оперативній пам'яті	Довгострокове зберігання	В оперативній пам'яті	В оперативній пам'яті (в ElastiCache)
Швидкість доступу	Дуже висока	Дуже висока	Дуже висока	Дуже висока
Масштабованість	Обмежена	Висока	Обмежена	Легко масштабується в AWS
Черги повідомлень	Не підтримує	Основне призначення	Не підтримує	Не основне призначення
Підтримка даних в пам'яті	Так (або може бути конфігуровано)	Так	Так	Так
Екосистема та інтеграція	Обмежена	Багата екосистема	Багата екосистема	Інтеграція з AWS-сервісами
Складність налаштувань	Проста	Складна	Проста	Спрощена в AWS
Розподіл	Обмежений	Високий	Обмежений	Керується в AWS
Відмовостійкість	Обмежена	Висока	Обмежена	Керується в AWS

Продовження таблиці 3.9

1	2	3	4	5
Переваги	Дуже висока продуктивність в оперативній пам'яті	Підтримка поточкових даних та подій	Дуже висока продуктивність	Керований сервіс з простим налаштуванням
Недоліки	Не підходить для довгострокового зберігання даних	Складне налаштування та керування	Обмежена масштабуємість	Залежить від екосистеми Amazon та не підходить для гібридних середовищ

Після ретельного вивчення документації та статей [88]-[90] було обрано Redis, що надає наступні основні переваги:

- висока продуктивність: Redis відомий своєю неймовірно високою продуктивністю завдяки зберігання даних в оперативній пам'яті та використанню ефективних алгоритмів. Це робить його ідеальним для сценаріїв, де швидкий доступ до даних критичний, таких як кешування;
- простий інтерфейс: Redis надає простий API, що складається з команд ключ-значення, що робить його легким у використанні та інтеграції. Це дозволяє швидко розпочати роботу з Redis та використовувати його для різних завдань;
- різноманітність структур даних: Redis підтримує різні структури даних, такі як рядки, списки, хеші, множини та багато інших. Це дозволяє вибирати найкращу структуру даних для конкретного завдання;
- підтримка транзакцій: Redis підтримує транзакції, що дозволяє виконувати кілька операцій як атомарну операцію. Це корисно для забезпечення цілісності даних;
- реплікація та відмовостійкість: Redis підтримує механізми реплікації та високої доступності, що дозволяє створювати відмовостійкі

конфігурації та забезпечувати надійність даних;

- підтримка плагінів та розширень: існує безліч сторонніх модулів та розширень, які дозволяють розширити функціональність Redis для конкретних завдань, таких як повнотекстовий пошук, графові бази даних та багато іншого;

- активна спільнота та екосистема: Redis має активну спільноту розробників, безліч бібліотек та клієнтів для різних мов програмування, що полегшує інтеграцію та обслуговування;

- Опенсорс: Redis є проєктом з відкритим вихідним кодом, що означає, що він безкоштовний і може бути налаштований та використаний як заманеться;

- можливості Pub/Sub: Redis підтримує публікацію та підписку на повідомлення, що робить його придатним для створення систем обміну повідомленнями та реалізації патерну "спостерігач" [88]-[90].

3.3.3 Аналіз та обґрунтування вибору хмарного сховища для файлів

Файли, як і прості дані, можна зберігати безпосередньо в БД, а можна зберігати в хмарному сховищі. Зберігання файлів у БД є доцільним при конкурентному доступі на зміну файлу. Враховуючи, що файли будуть статичні та не будуть змінюватись з часом, краще зберігати їх окремо, що надає наступні переваги:

- звернення до файлів швидше, ніж отримання запису з БД;
- кешування файлів виконується автоматично засобами операційної системи та серверу, а кешування засобами СКБД викликає втрату продуктивності через витіснення простих запитів з пам'яті;
- кількість файлів може бути дуже великою без втрати потужності серверу.

Результати аналізу популярних хмарних сховищ наведено в табл. 3.10.

Таблиця 3.10 – Порівняльний аналіз хмарних сховищ [91]-[97]

Хмарне сховище	Безкоштовне сховище	Максимальний розмір файлу	Особливості
Google Drive	До 15 ГБ	5 ТБ	Інтеграція з Gmail, Google Workspace
Dropbox	До 2 ГБ	50 ГБ	Просте спільне використання, Dropbox Paper
Microsoft OneDrive	5 ГБ (безкоштовно)	100 ГБ	Інтеграція з Office 365, SharePoint
Apple iCloud	5 ГБ (безкоштовно)	До 2 ТБ	Синхронізація з пристроями Apple, iCloud Drive
Amazon S3	Перші 12 місяців безкоштовно (обмежений розмір)	Величезні розміри	Використовується для зберігання даних та застосунків
Box	10 ГБ	5 ТБ	Представляє фокус на безпеці та керуванні
Dropbox Business	В залежності від плану	До 100 ТБ	Для підприємств, розширені функції управління

Розглянувши обговорення переваг та недоліків в роботах [98]-[101], було обрано сховище Google, що надає наступні переваги:

- безкоштовне та просте використання: Google надає безкоштовне хмарне сховище для користувачів з обліковим записом Google. Google Drive легко інтегрується з іншими сервісами та програмами Google, що робить його зручним для зберігання файлів;

- безпека та контроль доступу: можливо налаштовувати права доступу до файлів, визначати, хто має право переглядати, редагувати чи коментувати файли. Google також надає інструменти управління безпекою даних;

- великий обсяг сховища: Google Drive надає 15 ГБ безкоштовного сховища, і можливо розширити його, вибравши платні плани, якщо потреби у сховищі збільшаться;
- інтеграція зі сторонніми програмами: Google Drive підтримує інтеграцію з багатьма сторонніми програмами та сервісами, що розширює його функціональність;
- інструменти для розробників: Google надає API для розробників, що дозволяє створювати користувацькі програми, що взаємодіють з Google Drive [98]-[101].

3.3.4 Аналіз та обґрунтування вибору технологій розробки для бекенд частини

Бекенд кожного застосунка – це його логіка, яка схована від користувача. Саме тут оброблюються всі запити від користувача. Бекенд забезпечую контроль доступу до БД та інших сервісів, до яких користувач, з міркувань безпеки, не повинен мати прямого доступу. Він перевіряє та оброблює паролі, створює токени, розмежовує рівні доступу та виконує заплановані дії. Як і для фронтенд частини, так і для бекенд існує багато мов програмування.

В таблиці 3.11 наведено результати порівняльного аналізу популярних мов програмування для бекенду [102]-[120].

Згідно статистики сайту [121] Node.js є найпопулярнішим між конкурентів. Після аналізу статей [122]-[124], було обрано Node.js, що надає наступні переваги:

- асинхронне та подієво-орієнтоване програмування: Node.js підтримує асинхронні операції, що дозволяє обробляти безліч запитів клієнтів без блокування потоку виконання. Це підвищує продуктивність та чутливість вебзастосунків;

Таблиця 3.11 – Порівняльний аналіз мов програмування бекенду [102]-[120]

Мова	Опис	Основні фреймворки	Переваги	Недоліки
JavaScript	Мова сценаріїв на стороні клієнта та сервера.	Node.js, Express.js	Універсальність, велика спільнота, асинхронне програмування підходить для повного стеку розробки.	Відсутність багатозадачності при однопоточній моделі Node.js
Python	Універсальна мова із широким застосуванням.	Django, Flask, FastAPI	Простий синтаксис, велика спільнота, безліч бібліотек, добре підходить для розробки вебсервісів та API.	Не настільки швидкий, як деякі інші мови, GIL може викликати проблеми із паралельним виконанням завдань.
Ruby	Динамічна та елегантна мова.	Ruby on Rails	Простий синтаксис, сильна спільнота, швидка розробка вебзастосунку.	Продуктивність може стати проблемою великих проєктів.
Java	Потужна, об'єктно-орієнтована мова.	Spring, Java EE	Портативність, багатопоточність, висока продуктивність, надійність.	Багатослівний і вимагає багато коду для досягнення цілей.
PHP	Мова для веброзробки з великою історією.	Laravel, Symfony	Простота вивчення, доступні хостинги, популярність для вебсайтів та CMS.	Репутація як застарілої мови та небезпечної.
Go (Golang)	Молода та ефективна мова.	Gin, Echo	Висока продуктивність, компіляція виконуваний код, багатопоточність.	Обмежена кількість бібліотек та фреймворків у порівнянні з іншими.
C#	Мова розробки Microsoft.	ASP.NET Core	Відмінна інтеграція з Windows, багатопотоковість, хороша продуктивність.	Обмеження з крос-платформності та залежність від .NET Framework.

- те саме мовне оточення на клієнті та сервері: використання JavaScript як на клієнтській, так і на серверній стороні полегшує розробку та обслуговування програм, оскільки немає необхідності в перекладі коду з однієї мови на іншу;
- велика спільнота та безліч бібліотек: Node.js має активну спільноту розробників та велику екосистему бібліотек та модулів. Це полегшує розробку, оскільки є безліч готових рішень для різних завдань;
- висока продуктивність: Node.js використовує двигун V8 від Google Chrome, який забезпечує високу продуктивність JavaScript-коду. Це особливо корисно для обробки множини одночасних запитів;
- модульність і пакетний менеджер NPM: Node.js включає NPM (Node Package Manager), який дозволяє встановлювати і керувати сторонніми бібліотеками і модулями. Це робить розробку більш ефективною та зручною;
- швидкість розробки: JavaScript має простий та інтуїтивно зрозумілий синтаксис, що прискорює процес розробки. Крім того, середовище Node.js дозволяє легко створювати прототипи та протестувати нові ідеї;
- підтримка WebSocket та реального часу: Node.js добре підходить для створення програм, які вимагають обміну даними в режимі реального часу, таких як чати, онлайн-ігри та потокова передача даних;
- широке використання вебфреймворків: Існує безліч популярних вебфреймворків, таких як Express.js, Koa.js та Nest.js, які спрощують створення вебзастосунків з використанням Node.js;
- швидкість розгортання та масштабування: Node.js дозволяє швидко розгорнути програми та легко масштабувати їх на горизонтальному рівні, що корисно для обробки зростання навантаження;
- активний розвиток та підтримка: Node.js постійно розвивається та оновлюється, і має активну спільноту розробників, що гарантує її актуальність та підтримку у довгостроковій перспективі [122]-[124].

Для отримання даних з бекенду треба обрати протокол або мову запитів. Це дуже важливе питання, адже воно впливає на всю взаємодію фронт частини з бекендом. Від вибору залежить чи отримаємо ми всі необхідні дані за допомогою одного запиту, чи нам потрібно буде поступово, крок за кроком отримувати інформацію. Відповідно, скільки часу користувач буде чекати на ті дані, які йому необхідні. Та з якими засобами зв'язку нам самим, як розробникам буде зручно працювати. У кожного засобу зв'язку є свої переваги та недоліки.

Результати аналізу технологій зв'язку фронтенду з бекендом та базою даних наведено в табл. 3.12 [125]-[134].

Таблиця 3.12 – Аналіз особливостей технологій зв'язку [125]-[134]

Мова/ Протокол	Переваги	Недоліки
REST	Простота використання та розуміння. Добре підходить для CRUD-операцій. Стабільність та надійність	Обмеженість у можливостях запитів..Відсутність стандартизації у структурі даних. Надмірність даних при частих запитах
GraphQL	Гнучкість запитів (запитуєте лише потрібне). Однорідні кінцеві точки. Зменшення обсягу даних, що передаються	Складність у реалізації на сервері та клієнті. Потенційна загроза виконання складних запитів
gRPC	Ефективна та швидка взаємодія. Застосування Protocol Buffers для узгодженості. Вбудована підтримка схем та серіалізації даних	Складність у читанні та налагодженні даних у мережевому трафіку. Складність впровадження у деяких мовах програмування
SQL	Потужні запити та операції з даними. Добре підходить для аналітики та звітності. Широка підтримка реляційних баз даних	Не завжди підходить для нереляційних баз даних. Складність у розробці безпечних SQL-запитів. Обмеження в гнучкості та масштабованості

За підсумка аналізу статей [135]-[137] було обрано GraphQL, що надає наступні переваги:

- гнучкість запитів: GraphQL дозволяє клієнтам запитувати тільки ті дані, які їм дійсно потрібні, і нічого зайвого. Це зменшує надмірність даних та збільшує продуктивність, тому що не потрібно пересилати та обробляти зайві дані;
- єдина кінцева точка: на відміну від API REST, де кожен ендпоінт може надавати різні дані, GraphQL використовує єдину кінцеву точку для всіх запитів. Це спрощує роботу з API та покращує передбачуваність;
- схема даних: GraphQL заснований на строгій схемі даних, яка описує, які типи даних доступні і як вони пов'язані. Це робить процес розробки API більш структурованим та зрозумілішим;
- рефлексивне API: GraphQL надає можливість інтроспекції, що означає, що клієнти можуть витягувати метадані про схему даних і типи, що полегшує створення інструментів і документації;
- батчінг запитів: GraphQL дозволяє об'єднувати кілька запитів в один, що зменшує кількість мережевих запитів і може знизити затримку при завантаженні даних;
- підтримка реального часу: GraphQL може бути використаний для створення застосунків реального часу, оскільки підтримує підписки на події та оновлення даних у режимі реального часу;
- екосистема інструментів: існує безліч інструментів та бібліотек для роботи з GraphQL, включаючи клієнтські бібліотеки, серверні фреймворки та інструменти для тестування та налагодження;
- масштабованість: GraphQL дозволяє створювати складніші запити, що особливо корисно у застосунках з великим обсягом даних. Ви можете отримати лише необхідну інформацію, не перевантажуючи сервер непотрібними запитами;
- клієнтська незалежність: клієнти мають велику автономію у визначенні, які дані їм потрібні, що спрощує розробку мобільних та

вебзастосунків, а також покращує досвід користувача;

– екосистема спільноти: GraphQL має активну спільноту розробників та велику кількість ресурсів для навчання та підтримки [135]-[137].

Ці переваги зробили GraphQL популярним серед розробників та сприяють його використанню у різноманітних типах проєктів.

3.4 Висновки до розділу 3

В цьому розділі відповідно до мети та задач роботи було виконано аналіз вимог до розроблюваного програмного застосунка, запропоновано його архітектуру та розглянуто технології розробки. В результаті обґрунтованого вибору було сформовано такий стек технологій:

- мова програмування фронтенд модуля – JavaScript (React);
- мова програмування бекенд модуля – JavaScript (Nodejs);
- UI-бібліотека – Ant Design;
- стилізація – Styled-Component;
- форматування коду – Prettier;
- інструмент для перевірки якості коду та поверхневого виправлення помилок – ESLint;
- інструмент збирання коду – Webpack;
- база даних – PostgreSQL;
- побудова запитів до БД – GraphQL;
- керування кешем – Redis;
- зберігання файлів – Google Drive.

Це основні технології. Можливо, в процесі розробки знадобляться додаткові інструменти, але вони вже будуть вирішувати окремі питання на значно нижчому рівні та не будуть сильно впливати на всю розробку.

4 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ SRM-СИСТЕМИ

4.1 Структура та алгоритми функціонування системи

Структура системи об'єднує дві частини – фронтенд та бекенд.

Структурна схема фронтенд частини, що відображає блоки програми та їх взаємозв'язок, представлена на рис.4.1.

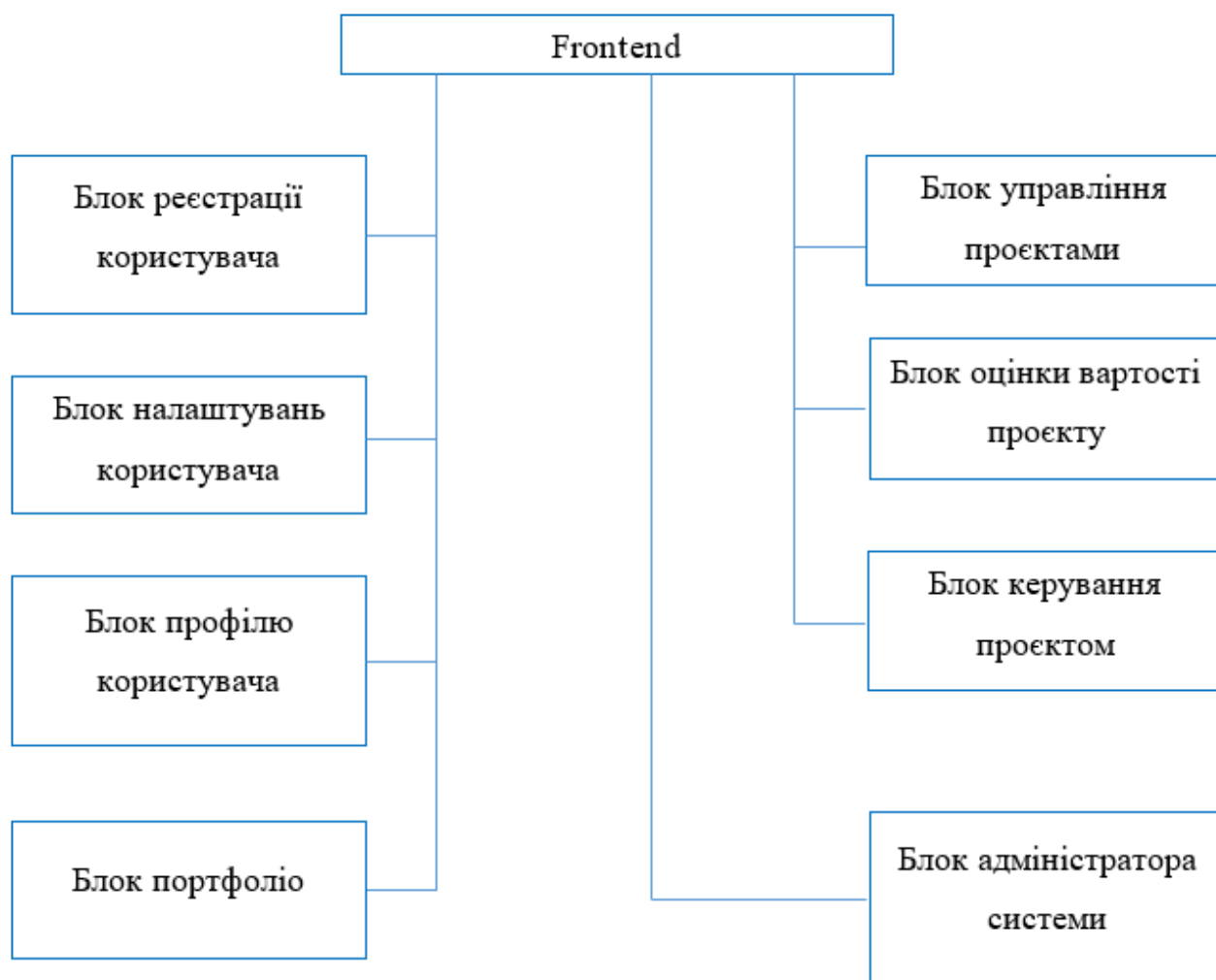


Рисунок 4.1 – Структурна схема фронтенд частини

Фронтенд частина складається з наступних блоків:

- Блок реєстрації користувача. Призначений для реєстрації замовників та виконавців. В залежності від обраної ролі йде свій шлях реєстрації;
- Блок налаштувань користувача. Цей блок призначений для налаштувань роботи в застосунку. Можна змінити пароль, налаштувати двофакторну аутентифікацію, вказати банківські реквізити та інше;
- Блок профілю користувача. Тут можна змінити свій аватар, написати про себе. Замовник може вказати, що він більш за все цінує в виконавці, та іншу інформацію, яка може спростити пошук виконавця. Виконавець може вказати інформацію, яка може описати його як спеціаліста в своїй справі. Вказати адреси офісів та допуски які він отримав на виконання певних видів робіт;
- Блок портфолію. Виконавець може показати свої завершені роботи. Це допоможе замовнику визначитися з вибором виконавця;
- Блок управління проектами. В цьому блоці можна буде переглянути всі проекти на всіх стадіях роботи. Замовник зможе додавати нових виконавців для своїх проектів, щоб вони склали кожен свій кошторис та було між чим обирати;
- Блок оцінки вартості проекту. Цей блок призначений для того, щоб виконавець зміг скласти кошторис та відправити його на розгляд замовнику, а замовник зміг його переглянути. Після чого або погодитись або відправити з зауваженнями на доопрацювання;
- Блок керування проектом. Це головна частина проекту, де можна керувати контрактом на всіх його стадіях;
- Блок адміністратора системи. Цей блок дозволяє керувати всім застосунком. Тут адміністратор має доступ до всіх проектів та користувачів. Також адміністратор може зайти в систему під будь-яким користувачем та «дивитись його очима», діяти як це робить сам користувач.

Схема взаємодії блоків представлена на рис.4.2.

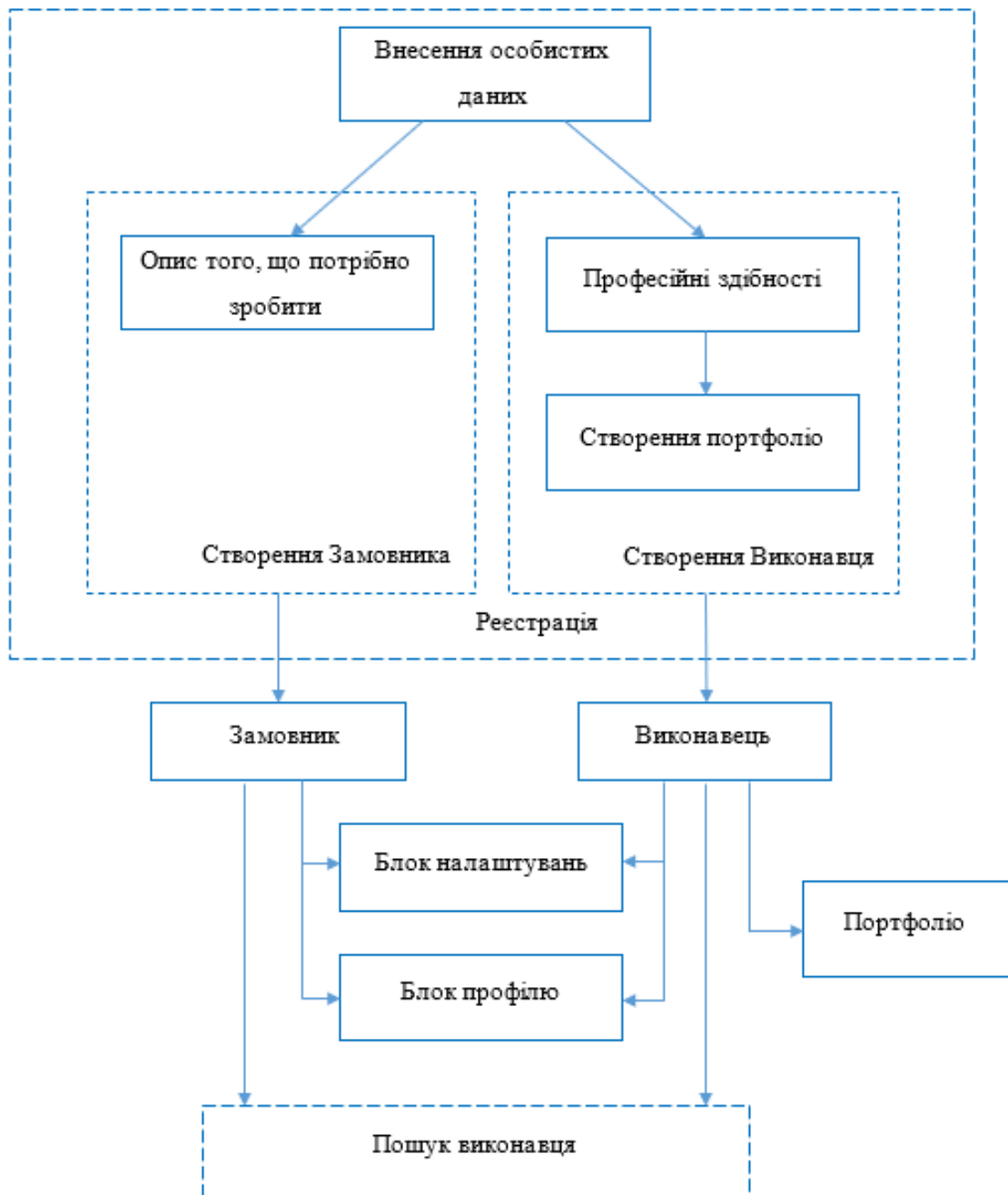


Рисунок 4.2 – Схема взаємодії блоків системи

Внесення особистих даних. З цього блоку буде починатися робота системи. Вхідними параметрами буде введення імені, прізвища, логіну (електронної пошти користувача) та паролю. Після вводу підтвердження паролю, користувач буде мати змогу перейти до наступної сторінки. Введені данні будуть передані на сторону бекенду, де пароль повинен буде збережений у вигляді хешу. Хеш – це набір випадкових символів, що генерується на основі текстового пароля за допомогою спеціального

алгоритму хешування. Хешування пароля дозволяє захистити пароль від читання зловмисниками, тому що звернути хеш назад у вихідний пароль неможливо. Після чого користувач може вибрати свою роль, тобто хто він є, Замовник або Виконавець. Якщо обирає роль Виконавця, то переходить на сторінку, де потрібно вказати свій головний та додаткові офіси. Адреса складається з області, назви населеного пункту, вулиці, дома та індексу. Ця інформація потрібна для того, щоб застосунок мав змогу пропонувати Замовнику лише тих Виконавців, які знаходяться поряд с місцем проведення майбутніх робіт.

Опис того, що потрібно зробити. Якщо користувач обрав роль Замовника, він потрапляє в цей блок. Тут він має можливість вказати адресу, де саме планує зробити ремонт. На наступних сторінках має вказати таку інформацію:

- що саме він хоче зробити;
- в якому стилі;
- за який час;
- площу робіт;
- яку суму згоден витратити;
- додати фото або інші файли, які дадуть змогу більш детально зрозуміти, що він хоче отримати в результаті.

Професійні здібності. Тут виконавець вносить наступну інформацію:

- якими професіями він володіє;
- які типи робіт він може виконувати;
- на які види робіт він має допуски;
- в якому стилі оформлення він має досвід роботи.

Після цього Виконавець переходить до блоку Створення портфолію. В цьому блоці він має змогу заповнити анкету на одну або декілька виконаних робіт, щоб показати Замовнику на що він здатен. Інформація, яку надає Виконавець:

- назва проєкту;

- опис проекту;
- область та населений пункт, де проводився ремонт;
- стиль, в якому виконували роботи;
- вартість;
- час, витрачений на роботу;
- дата проведення роботи;
- фотографії;

Після заповнення всіх початкових даних на виході ми маємо повністю сформованого або Замовника або Виконавця, в залежності від вибору користувача.

Вони обидва отримують доступ до Блоку Налаштувань. В цьому блоці можна:

- змінити адресу електронної пошти;
- змінити пароль;
- налаштувати двофакторну автентифікацію;
- вказати банківські реквізити;

Наступні пункти повинні бути доступні лише користувачу з роллю Виконавець:

- вказати мінімальну суму, яку розраховує витратити Замовник. Якщо сума буде менша, застосунок не буде рекомендувати цього виконавця замовнику;

- відстань в кілометрах від офісу Виконавця до місця проведення робіт. Також буде впливати на пропонування цього виконавця замовнику.

Блок профілю. Дозволяє додати та змінювати наступну інформацію: ім'я; прізвище; аватар. Додатково Виконавець повинен мати змогу змінити або додати адреси своїх офісів та назву своєї компанії. Також в цьому блоці Виконавець повинен мати змогу редагувати всі параметри, які він вказував у блоці Професійні здібності. Додатково на сторінці повинна бути кнопка переходу до редагування портфоліо (Блок портфоліо описаний вище).

Схему алгоритму пошуку виконавця наведено на рис. 4.3.

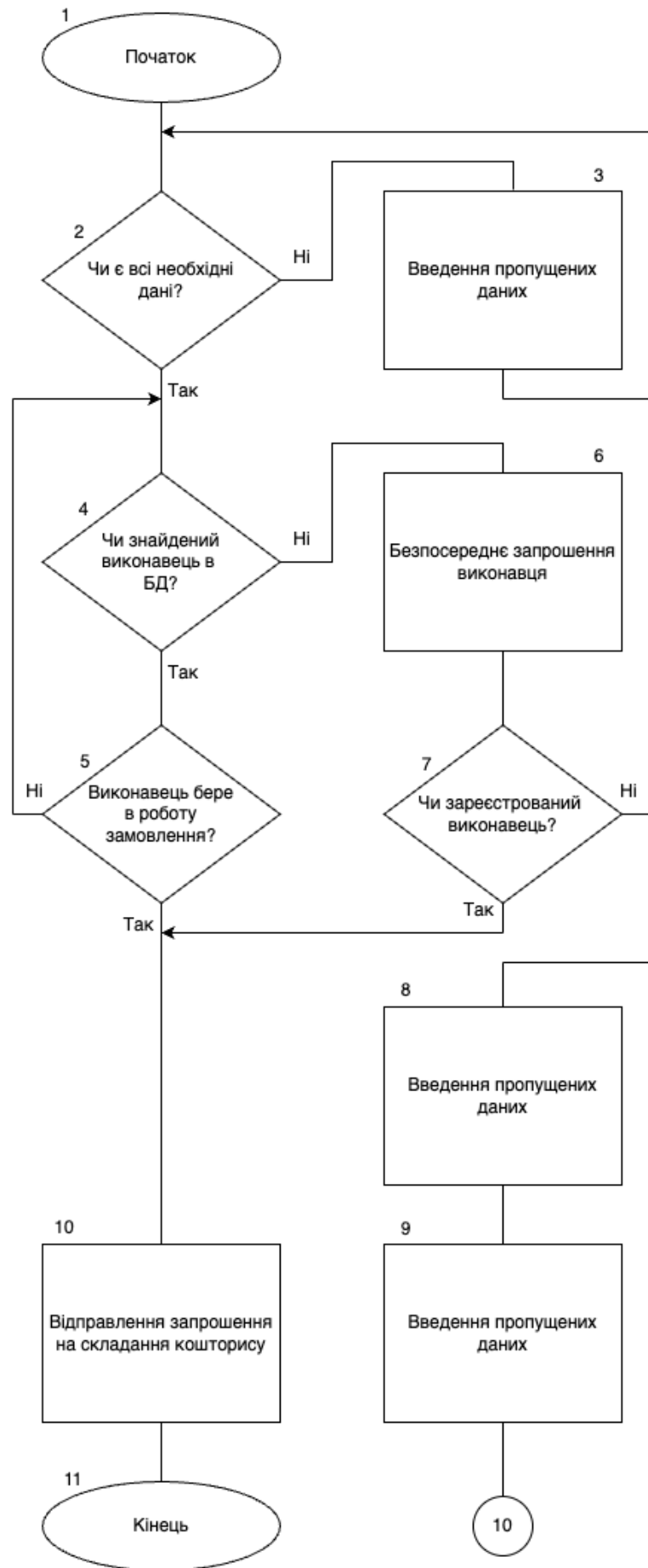


Рисунок 4.3 – Схема алгоритму пошуку виконавця

Блок 1. Початок.

Блок 2. Перевірка чи є всі вхідні дані. Вхідними даними є повністю заповнені дані про перелік робіт які потрібно виконати, сума та час які Замовник готовий витратити на роботи та адреса об'єкта. При введенні даних про роботу, яку необхідно виконати Замовник може зупинитися на будь-якому етапі з різних причин, тож треба перед початком пошуку Виконавця бути впевненим, що всі необхідні дані внесені в систему. Якщо якихось даних не вистачає – переадресуємо Замовника на сторінку вводу даних.

Блок 3. Цей блок є частиною реєстрації. При перевірці інформації про замовлення робіт Замовник має бути переадресований на першу не заповнену сторінку даних.

Блок 4. Якщо всі початкові дані є, замовник повинен потрапити на сторінку пошуку Виконавця. При переході до цієї сторінки, застосунок повинен запропонувати Замовнику тих виконавців, які на думку системи найбільш підходять до його вимог. Пошук повинен проходити з врахуванням вимог Виконавця, які він може вказати в блоці Налаштувань, та з врахуванням потреб Замовника, вказаних при створенні проєкту, а також за професійними здібностями Виконавця, які він вказував при реєстрації або в блоці Профілю.

Виконавці повинні бути представлені у вигляді карточок, в яких буде вказана наступна інформація:

- ім'я та прізвище виконавця;
- на чому він спеціалізується;
- які типи робіт виконує;
- кнопка переходу до перегляду портфоліо;
- кнопка додавання цього Виконавця до списку відібраних виконавців;
- кнопка відхилення цього виконавця.

Замовник може обрати виконавців з запропонованих застосунком або ввести електронну адресу того, з ким хоче співпрацювати – Блок 6. Після того

як Замовник обрав Виконавців їм буде відправлене повідомлення, з описом замовлення.

Блок 6. В цьому блоці Замовник вводить електронну адресу та ім'я того Виконавця, якого він знає для відправлення запрошення на співпрацю. На цій сторінці повинен бути текст запрошення, який буде включати опис майбутнього проєкту. Замовник повинен мати змогу змінити текст повідомлення.

Блок 7. Перевірка, чи зареєстрований такий користувач. Якщо така електронна пошта є в системі, то запрошення йде по стандартному шляху так, якби цього Виконавця обрав Замовник за рекомендацією системи.

Блок 8. Якщо такого користувача в системі не знайдено, тоді йому на пошту буде відправлене ще одне повідомлення з посиланням на реєстрацію в застосунку.

Блок 9. Якщо запрошений Виконавець перейде за посиланням, він потрапить до стандартного блоку реєстрації Виконавця.

Блок 10. Відправлення стандартного запрошення на складання кошторису.

Блок 11. Кінець. Перехід до складання кошторису та створення контракту.

Схему алгоритму складання кошторису та створення контракту наведено на рис.4.4.

Блок 1. Початок.

Блоку 2. Оцінка проєкту.

Після отримання запрошення на складання кошторису, Виконавець повинен мати змогу перейти до блоку, де вказана вся інформація, яку надав Замовник при створенні свого замовлення (Опис того, що потрібно зробити) та прийняти рішення щодо своєї участі в цій роботі. Якщо він погоджується прийняти участь, то переходить до створення кошторису.

Блок 3. Відмова від проєкту.

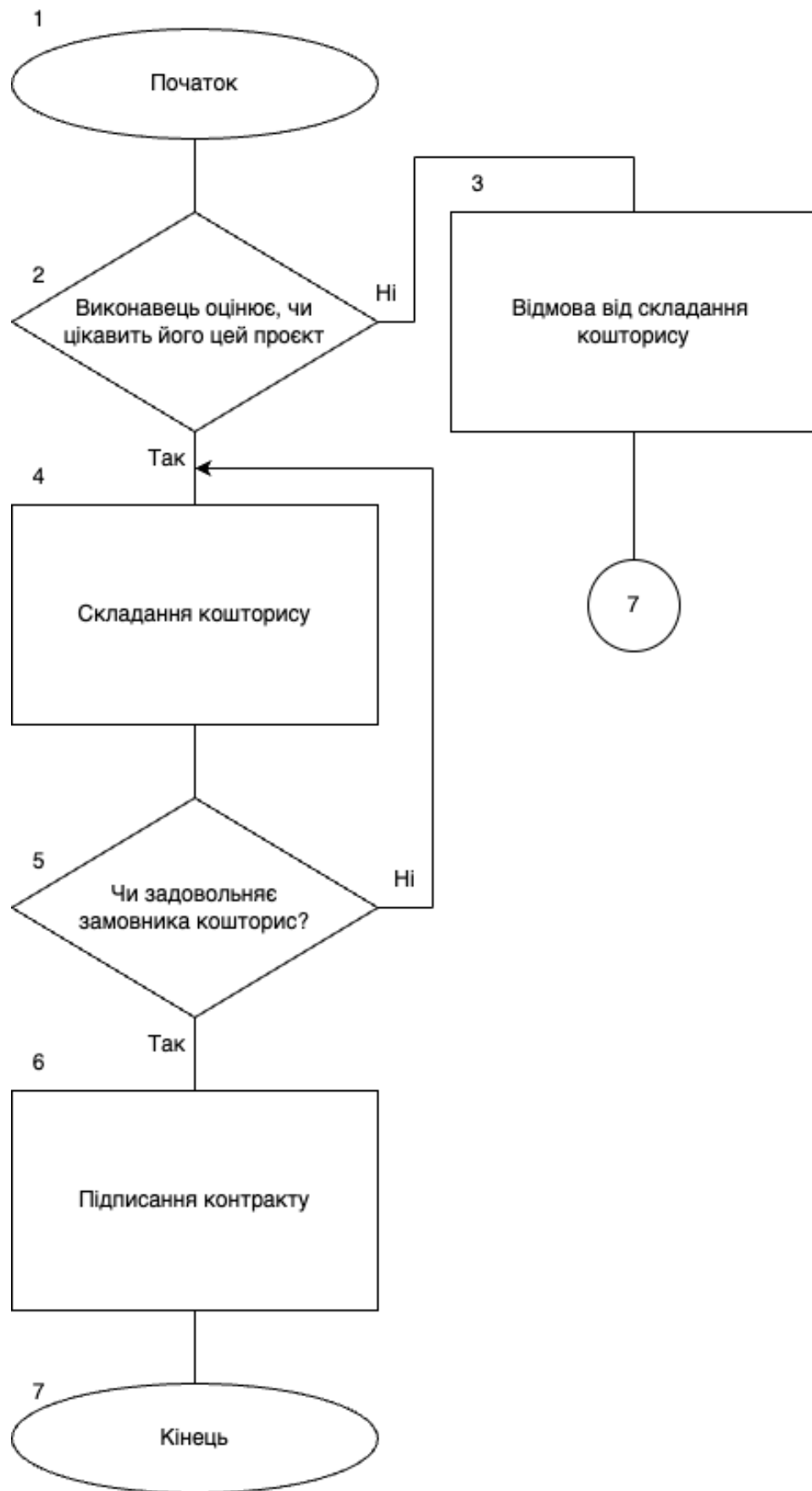


Рисунок 4.4 – Схема алгоритму складання кошторису та створення контракту

Блок 4. Складання кошторису. Сторінка складання кошторису повинна бути у вигляді таблиці. Виконавець повинен мати змогу додавати фази робіт, а до кожної фази мати змогу додати мінімум одну задачу.

Таблиця повинна мати наступні поля:

- назва фази або задачі;
- опис цієї фази або задачі;
- дата початку;
- дата закінчення;
- вартість матеріалів;
- вартість роботи;
- вартість інших витрат;
- вартість розмітки;
- загальні витрати.

Вартість по фазі повинна розраховуватись автоматично, як сума вартостей всіх її задач. Виконавець повинен мати змогу додати до кошторису фотографії та інші файли, які він вважає за потрібне для ознайомлення замовником. Після заповнення всіх колонок, Виконавець відправляє кошторис на затвердження Замовнику.

Блок 5. Замовник ознайомлюється з кошторисом. Замовник не повинен бачити вартість матеріалів, роботи та інше, він повинен бачити назву фази чи задачі, їх опис, дати початку та закінчення та остаточну вартість фаз та задач. Замовник повинен мати змогу продивитись всі додані до кошторису документи. Після ознайомлення, якщо у Замовника є зауваження, він повинен мати змогу відправити кошторис на доопрацювання додавши свої коментарі. В такому випадку, кошторис потрапляю знову до Виконавця, де він має змогу переглянути всі зауваження та змінити кошторис. Після цього повторно відправити його на затвердження замовнику. Якщо замовника все задовольняє він натискає відповідну кнопку та переходить до підписання договору.

Блок 6. Підписання контракту. При переході до цього блоку замовник за допомогою свого паролю підписує договір. При цьому він повинен переказати на рахунок Застосунку суму в розмірі вартості першої фази робіт.

Блок 7. Кінець.

Блок керування проектом можна розділити на модулі, навігація між якими буде реалізована за допомогою бокового меню сторінки керування проектом. Структурна схема Блоку керування проектом, що відображає модулі та їх взаємозв'язок, представлена на рис.4.5.



Рисунок 4.5 – Структурна схема Блоку керування проектом

Модуль інформації по проекту. Цей модуль повинен містити інформацію про поточний статус проекту – відображати на який процент вже виконаний проєкт, показувати останні події, які відбулися в проєкті та інформацію на яку потрібно відреагувати користувачу.

Модуль керування завданнями. Це місце де кожен учасник проекту повинен мати можливість відслідковувати кожен етап проекту і створювати в

режимі реального часу контрольний список завдань. Сторінка завдань повинна містити усі завдання з договору. Лише користувач Виконавець повинен мати можливість оновлювати данні завдань контракту та його статус. Візуально сторінка повинна бути розділена на три стовпчики: необхідно зробити, в роботі та виконані.

Завдання повинні бути представлені у вигляді карток. Картки повинні містити опис робіт, які планується виконати в даному завданні та період виконання цього завдання. При створенні контракту всі завдання повинні знаходитися в колонці «Необхідно виконати». Коли виконавець приступає до виконання завдання, він перетаскує відповідну картку в колонку «В роботі». Після виконання, переміщує цю картку в колонку «Виконано». Таким чином можна візуально спостерігати хід роботи.

Модуль внутрішнього чату. Це сторінка, де всі сторони проєкту можуть спілкуватися, обговорювати різні питання та обмінюватись документами. Обидві сторони контракту повинні бути впевнені, що добре розуміють один одного, що нема непорозумінь та пропущених бесід.

Модуль розрахунків. Повинен відображати кошторис по етапах робіт. В цьому блоці Виконавець повинен мати змогу створити запит на оплату, який може бути затверджений лише після завершення всіх завдань на цьому етапі. Замовник або підтвердить його за допомогою електронного підпису, або відхилить з поясненням причини.

Модуль контракту. На цій сторінці повинен бути детальний опис всього проєкту з відображенням всіх фаз та завдань. Тут повинен відображатися кошторис, але без можливості його редагування. В цьому розділі повинна бути можливість достроково завершити контракт. Якщо Замовник відмовляється від продовження роботи, він повинен мати змогу або погодитись на переказ всіх коштів за поточну фазу роботи Виконавцю, або погодити лише часткову оплату з описом обставин. В цьому випадку Виконавець або погоджується на отримання часткової оплати або приводить свої аргументи.

Модуль змін в проєкті. Яким би ретельним не було планування, але не все можливо передбачити заздалегідь. Також, в певний момент, замовник, ймовірно, захоче ввести деякі зміни. Коли план змінюється його треба узгодити, задокументувати та затвердити з обох сторін, щоб уникнути плутанини. Ця сторінка повинна містити саме ці зміни в письмовому вигляді. Зміни повинні відображатися у вигляді таблиці, яка буде мати наступні поля:

- поточний номер цієї зміни;
- назву завдання, до якого складається зміна;
- статус цієї зміни: відкрита, схвалена, в очікуванні, відкладена, відхилена та закрита;
- коментар до зміни;
- дата створення;
- сума, на яку буде змінена вартість;
- дата узгодження.

Ініціювати зміни повинен Виконавець. Якщо в процесі цих змін також змінюється вартість робіт, це потрібно узгодити з Замовником.

Модуль доданих документів. Всі документи які коли-небудь були додані за допомогою чату або іншим засобом, повинні бути показані в цьому блоці. Відображати ці документи потрібно у вигляді карток. Картка повинна містити зображення першої сторінки документу, назву файлу, ім'я та прізвище того, хто додав цей документ та дату його додавання.

Сторінка повинна мати інструменти сортування документів за ім'ям файлу, ім'ям користувача та датою створення файлу.

Також користувачі повинні мати можливість завантажити сюди нові файли.

Модуль керування співробітниками. Самому вести всі справи стосовно договору може бути важко, а вважаючи, що таких договорів може бути декілька може просто не вистачити часу. Для полегшення ведення справи, можна додати в проєкт співробітників. Цей модуль призначений для

додавання та курування співробітниками. Виконавець та Замовник повинні мати можливість додавати співробітників лише для ведення власних справ.

Головні користувачі повинні мати змогу змінювати права доступу співробітників до проєкту. Повинно існувати три рівня доступу:

- читання задач та чату. Цей користувач повинен мати доступ лише на перегляд модулю інформації по проєкту, задач, чату, та списку документів;
- редагування задач та чату. Може дивитись всі модулі, писати в чат, створювати зміни до проєкту від імені Виконавця, але не може затверджувати зміни від Замовника, та не може додавати інших співробітників;
- повний доступ. Цей співробітник може виконувати будь-які дії від імені головних користувачів, окрім додавання нових співробітників.

Цей модуль повинен бути представлений у вигляді таблиці. Таблиця повинна містити ім'я та прізвище співробітника, чий інтереси від представляє (замовника чи виконавця), надані права доступу та кнопку видалення цього співробітника.

4.2 Опис особливостей програмної реалізації

Вся робота системи будується на взаємодії окремих частин коду – функцій класів, запитів на отримання інформації та мутацій (зміни та додавання даних у базу даних).

4.2.1 Реалізація бекенд частини

Проаналізуємо структуру файлів проєкту на стороні бекенду. Нам потрібно розробити функції, які відповідають за авторизацію, проведення банківських операцій, для роботи з БД, відправленням пошти, для роботи з хмарним сховищем, тощо. Загальну структуру папок на бекенді наведено на рис. 4.6.

```
graph TD; src[✓ src] --> auth[> auth]; src --> billing[> billing]; src --> config[> config]; src --> db[✓ db]; src --> gql[✓ gql]; src --> notifications[✓ notifications]; src --> test[> test]; src --> utils[> utils]; db --> dataLoaders[> dataLoaders]; db --> dataUtils[> dataUtils]; db --> match[> match]; db --> migrations[> migrations]; db --> models[> models]; db --> storage[> storage]; db --> types[> types]; gql --> checks[> checks]; gql --> resolvers[✓ resolvers]; gql --> subscriptions[> Subscriptions]; resolvers --> Mutations[> Mutations]; resolvers --> Queries[> Queries]; resolvers --> Types[> Types]; notifications --> emails[> emails]; notifications --> senders[> senders]; notifications --> subscriptions[> subscriptions];
```

- ✓ src
 - > auth
 - > billing
 - > config
 - ✓ db
 - > dataLoaders
 - > dataUtils
 - > match
 - > migrations
 - > models
 - > storage
 - > types
 - ✓ gql
 - > checks
 - ✓ resolvers
 - > Mutations
 - > Queries
 - > Subscriptions
 - > Types
 - ✓ notifications
 - > emails
 - > senders
 - > subscriptions
 - > test
 - > utils

Рисунок 4.6 – Структура папок на бекенді

Більш докладно структура папок описана в табл. 4.1.

Таблиця 4.1 – Опис папок

Папка	Опис
1	2
auth	Тут потрібно зберігати функції, які відповідають за реєстрацію користувача, його автентифікацію, створення токена, його перевірку та оновлення паролю.
billing	Функції для проведення банківських операцій
config	Налаштування конфігурації. Налаштування для локальної розробки, тестування на девелоп-сервері та для продакшена. Тут повинні зберігатися паролі та токени для взаємодії з зовнішніми середовищами.
db	Все, що стосується роботи з базою даних
dataLoaders	Завантаження даних з бази даних
	«dataUtils» – функції додаткової обробки даних. Наприклад, об'єднання адреси (область, населений пункт, вулиця, номер дому, квартира) в один рядок.
	«match» – функції пов'язані з пошуку виконавця.
	«migrations» – міграції, тобто зміни бази даних. В процесі роботи при будь-якому плануванні все одно доведеться якось змінювати або доповнювати структуру таблиць бази даних. Будуть додаватися нові таблиці, змінюватись збереження існуючих даних. Кожна така зміна повинна зберігатися в окремому файлі. При запуску бекенду система повинна перевіряти, чи виконані всі міграції і якщо якась ще не виконана – виконати її.
	«models» – в цій папці будуть зберігатися дані для опису роботи з кожною моделлю даних окремо. На кожную модель нам потрібні функції для виконання CRUD-операцій, тобто створення, читання, оновлення та видалення запису.
	«storage» – функції для взаємодії з хмарним сховищем.
	«types» – опис типів даних

Продовження таблиці 4.1

1	2
gql	Ця папка повинна містити функції для роботи з GraphQL
	«checks» – тут будуть розташовані засоби перевірки доступу до даних
	«resolvers» – ця папка буде містити всі запити до бази даних та типи даних: – «mutations» – в папці мутацій будуть зберігатися всі запити на зміну та оновлення даних; – «queries» – в цій папці повинні зберігатися всі запити на отримання даних; – «subscriptions» – тут будуть зберігатися функції для обробки підписок на події. Ці і реакція на нове повідомлення в чаті, на створення заявки на зміну в договорі та інші. Щоб обидві сторони договору мали можливість отримувати актуальну інформацію без перезавантаження сторінки; – «types» – типи даних.
	Ця папка призначена для роботи по відправленню електронних листівок.
notifications	«emails» – ця папка повинна містити шаблони електронних листівок
	«sersders» – тут будуть зберігатися функції для отримання даних з бази даних, їх обробки та відправлення листівок.
	«subscriptions» – ця папка буде містити функції для відправлення електронних листівок, якщо потрібно реагувати на нові події.
test	Папка для розташування тестів на всі функції проєкту
utils	В різних частинах проєкту може виникнути потреба виконувати одну й ту саму операцію. Щоб не дублювати код, який потім буде складно підтримувати, доцільно винести їх в окремі функції та викликати там де потрібно. Для зберігання таких функцій і призначена ця папка.

Таблиця 4.2 містить список розроблених класів та їх опис.

Таблиця 4.2 – Опис розроблених класів

Назва	Опис
ChangeOrdersLoader	Повинен описувати роботу зі змінами до контракту
ChatsLoader	Повинен описувати роботу з чатами
EmployeesLoader	Повинен описувати роботу зі співробітниками
CompaniesLoader	Повинен описувати роботу з компаніями виконавців
ContractActivitiesLoader	Повинен описувати роботу з логуванням подій в контракті
ContractsLoader	Повинен описувати роботу з контрактами
EsignsLoader	Повинен описувати роботу з електронними підписами
EstimatesLoader	Повинен описувати роботу з кошторисами
FilesLoader	Повинен описувати роботу з файлами
LicensesLoader	Повинен описувати роботу з допусками на виконання видів робіт, які є у виконавця
MessagesLoader	Повинен описувати роботу з повідомленнями в чаті
PaymentHistoriesLoader	Повинен описувати роботу з банківськими платежами
PhasesLoader	Повинен описувати роботу з фазами робіт
PortfoliosLoader	Повинен описувати роботу з портфоліо
ProjectsLoader	Повинен описувати роботу проектами (до підписання договору це тільки проєкт)
RolesLoader	Повинен описувати роботу ролями користувачів
TasksLoader	Повинен описувати роботу з задачами в фазах
UsersLoader	Повинен описувати роботу з користувачами

Таблиця 4.3 містить список розроблених функцій та їх опис.

Таблиця 4.3 – Опис розроблених функцій

Назва	Опис
generateAuthToken	Генерація токена
getCurrentUser	Отримання даних про користувача при розшифруванні токена
isAdmin	Перевірка, чи є цей користувач Адміністратором системи
validateContractAccess	Перевірка, чи є у цього користувача доступ до контракту
validEsign	Перевірка електронного підпису
formatDate	Перетворення дати так, щоб день або місяць містили дві цифри завжди (додає 0, якщо дата менша за 10)
formatMoney	Перетворення коштів так, щоб було видно гривні та копійки

4.2.2 Реалізація фронтенд частини

Для реалізації фронтенд частини потрібно розробити компоненти, з яких будуть створюватись сторінки, вирішити де зберігати запити на отримання та зміну даних, а також функції, які будуть використовуватись в різних частинах проєкту. Структура папок на фронтенді наведена на рис. 4.7.



Рисунок 4.7 – Структура папок на фронтенді

Папка компонентів «components» містить всі компоненти системи. Немає сенсу на кожній окремій сторінці створювати свою унікальну модальну форму або поле вводу будь-якої інформації. Краще зробити один компонент, у якого буде декілька властивостей і використовувати його де потрібно. Це не тільки збереже час, а й гарантуватиме, що цей компонент буде однаково оформлений у всьому застосунку. В табл. 4.4 наведено опис компонентів застосунку.

Таблиця 4.4 – Компоненти застосунку

Назва	Опис компоненту
1	2
AddEmployee	Компонент повинен мати форму для додавання нового співробітника
ApproveDeclineForm	Форма яка повинна містити текст, чекбокс для підтвердження умов, або відхилення з можливістю обґрунтувати своє рішення. Буде використовуватись при підписанні контракту та при ухвалі чи відхиленні змін до проєкту
ChatMembers	Компонент для додавання та видалення співробітника з чату
ConfirmModal	Модальне вікно в яке можна додавати інший компонент з кнопками підтвердження чи відхилення. Буде використовуватись там, де потрібне додаткове підтвердження дії.
CreateChat	Компонент для створення нового чату
CurrencyInput	Компонент для вводу суми грошей
DebounceInput	Компонент, який передає введені данні лише після того, як користувач закінчить вводити весь текст. Спрацьовує лише коли користувач не вводить данні впродовж 500 мс. Буде використовуватись там, де йде пошук будь-яких даних на бекенді. Це знизить навантаження на бекенд та базу даних.

Продовження таблиці 4.4

1	2
DnD	Компонент, який допоможе перетаскувати об'єкти з одного стовпчика в інший. Буде використовуватись для зміни статусу задач.
EditChat	Компонент редагування чату. Повинен дозволяти змінювати назву чату, керувати його користувачами, тощо.
EditChatMembers	Компонент додавання та видалення членів чату.
ErrorHandler	Компонент, який буде бачити користувач у разі виникнення критичної помилки.
InviteEmployeeModal	Компонент запрошення співробітника. Повинен містити поля для вводу ім'я, прізвища, електронної адреси та рівню доступу.
Icon	Компонент для відображення аватарів в чаті та маленьких зображень доданих документів. Повинен мати можливість відкрити зображення на весь екран, або відкрити файл.
Image	Компонент для демонстрації картинки на весь екран, Повинен показувати назву файлу. Буде використовуватись в тих компонентах де є можливість переглянути файли.
Header	Компонент, який буде показувати поточного користувача та надавати можливість переходити до списку проєктів.
NewMessage	Компонент для створення нового повідомлення. Повинен мати поле для вводу тексту, а також можливість додавання файлу.
ManageRouter	Головний компонент, який буде керувати роутами. Відображення сторінки буде залежати від адреси де знаходиться користувач. Отже від прив'язки адреси до компонента чи сторінки залежить що побачить користувач. Це зручно, адже користувач може додати будь-яку сторінку до обраних та одразу відкривати потрібну інформацію.
ManageRouterLayout	Компонент бокової панелі, де можна переходити до сторінок керування проєктом (блоки керування проєктом).

Продовження таблиці 4.4

1	2
PercentageInput	Компонент для вводу інформації у процентах. Буде використовуватись при складанні кошторису та створенні змін до контракту.
PhoneNumberInput	Компонент для вводу телефонного номеру. Буде використовуватись при реєстрації користувачів та при внесенні змін до профілю користувача.
PreviewImageList	Компонент, який дозволить показувати список зображень або іконок документів. Повинен мати можливість фільтрувати зображення за назвою чи датою додавання. Буде використовуватись на сторінці з документами та при перегляді портфоліо.
PreviewSlider	Компонент слайдер зображень. Буде використовуватись при перегляді та виборі виконавця.
ShowMoreText	Компонент, який дозволить обрізати текст до певного розміру з можливістю відкрити його повністю. Буде використовуватись при перегляді великих текстів.
Spinner	Компонент, який буде бачити користувач при зверненні до бекенду. Щоб буде зрозуміло, що застосунок оброблює запит.
TextAreaField	Компонент для вводу тексту більше за одно строку. Буде використовуватись при складанні листа при запрошенні виконавця, введенні опису завдання та фази при складанні кошторису та в інших місцях.

Папка для зберігання всіх запитів до бекенду «graphql». Тут будуть зберігатися запити на отримання, додавання та зміну інформації. В табл. 4.5 наведено опис запитів у системі.

Таблиця 4.5 – Опис запитів у системі

Назва	Опис
1	2
adminGetUsers	Запит адміністратора системи на отримання всіх користувачів
adminGetContracts	Запит адміністратора системи на отримання всіх контрактів
adminUpdateRole	Запит адміністратора системи на зміну ролі користувача
adminDeleteUser	Запит адміністратора системи на видалення користувача
register	Запит на реєстрацію користувача
getCurrentUser	Запит на отримання даних поточного користувача.
login	Запит на вхід в систему
resetPassword	Запит на зміну паролю
updateUserBasicInfo	Запит на оновлення таких даних користувача як ім'я, прізвище, електронна адреса та норма телефону
updateUserImages	Запит на оновлення аватара користувача
getChangeOrders	Запит на отримання змін до поточного контракту
createChangeOrder	Запит на створення зміни до поточного контракту
editChangeOrder	Запит на редагування зміни до поточного контракту
deleteChangeOrder	Запит на видалення зміни
approveChangeOrder	Запит на схвалення змін до договору
declineChangeOrder	Відхилення змін до контракту
getChatListByUser	Запит на отримання списку чатів
getChatMembers	Запит на отримання користувачів чату
addChatMember	Запит на додавання користувача до чату
removeChatMember	Запит на видалення користувача з чату
createGroupChat	Запит на створення чату
getContractUsers	Запит на отримання списку користувачів контракту
getManageContracts	Запит на отримання всіх підписаних контрактів поточного користувача
getContract	Запит на отримання деталей поточного контракту
createEstimate	Запит на створення кошторису
updateEstimate	Запис на внесення змін до кошторису

Продовження таблиці 4.5

1	2
declineEstimate	Запит на відхилення кошторису
acceptEstimate	Запит на ухвалення кошторису
getEmployees	Запит на отримання всіх співробітників у поточному контракті
createEmployee	Запит на створення співробітника
updateEmployee	Запит на оновлення даних співробітника
deleteEmployee	Запит на видалення співробітника
getContractSummary	Запит на отримання загальної інформації по контракту
getContractActivities	Запит на отримання списку подій в контракті
getUserRequiredActions	Запит на отримання списку дій, на які поточний користувач повинен звернути увагу
getContractFiles	Запит на отримання файлів поточного контракту
createFile	Запит на додавання файлу
deleteFile	Запит на видалення файлу
getPhases	Запит на отримання списку фаз в контракті
addPhase	Запит на додавання фази в кошторис
updatePhase	Запит на оновлення фази в кошторису
deletePhase	Запит на видалення фази
getTasks	Запит на отримання всіх завдань проєкту
addTask	Запит на додавання завдання в фазу
updateTask	Запит на оновлення завдання
deleteTask	Запит на видалення завдання
getPortfolio	Запит на отримання портфоліо
createPortfolio	Запит на створення портфоліо
updatePortfolio	Запит на оновлення портфоліо
deletePortfolio	Запит на видалення портфоліо
createProject	Запит на створення замовником проєкту
updateProject	Запит на оновлення проєкту
deleteProject	Запит на видалення проєкту
getProjects	Запит на отримання всіх проєктів поточного користувача

Окрім запитів, потрібно постійно підтримувати інформацію у актуальному стані. Робити кожні декілька хвилин всі запити на отримання інформації – дуже не продуктивна справа. Отже потрібно щоб був механізм, який би сповістив відкриту сторінку з даними, що інший користувач щось змінив. Для вирішення таких питань потрібно зробити підписку на певні події (табл. 4.6).

Таблиця 4.6 – Опис підписок на події

Назва	Опис
taskboardUpdated	Підписка на зміни в задачах. Потрібно, щоб оновлювався список задач при додаванні нової задачі або будь-якій зміні існуючої задачі
contractActivitiesCreated	Підписка на будь-які зміни в проєкті. Потрібно, щоб користувач зміг бачити що в контракті йдуть зміни. Наприклад, доданий коментар до задачі, або файл. Отже повинен оновлюватись список останніх подій одразу при їх виникненні.
changeOrderUpdated	Підписка на оновлення списку змін у контракті або якоїсь однієї зміни.
contractFilesUpdated	Підписка на оновлення списку файлів у контракті.
employeesUpdated	Підписка на додавання або видалення співробітника.
chatListUpdated	Підписка на зміни у списку чатів.
chatUpdated	Підписка на нові повідомлення в чаті.
requiredActionsUpdated	Підписка на оновлення списку подій, на які потрібна реакція користувача.
projectsUpdated	Підписка на оновлення стану проєктів.

Папка “pages” містить всі сторінки проєкту. У таблиці 4.7 наведено опис сторінок застосунку.

Таблиця 4.7 – Опис сторінок застосунку

Назва	Опис
SignUp	Сторінка реєстрації користувача
Login	Сторінка логіна
TermOfUse	Сторінка умов користування застосунком
Users	Сторінка адміністратора, де можна переглянути всіх зареєстрованих користувачів
Contracts	Сторінка адміністратора, де можна переглянути всі контракти застосунка
ReportsTable	Сторінка адміністратора, де можна переглянути звіти по застосунку
CreateProjectPage	Сторінка для заповнення інформації про перелік робіт, які потрібно виконати
ExecutorSignUp	Сторінка, де виконавець заповнює дані про свою фірму, види робіт, які він вміє виконувати.
ProjectsPage	Сторінка зі всіма проєктами користувача.
CustomerEstimate	Сторінка, де замовник може переглянути складений кошторис
ExecutorEstimate	Сторінка, де виконавець складає кошторис
MatchPage	Сторінка, де йде пошук вибір або запрошення виконавця
Dashboard	Сторінка огляду контракту.
Documents	Сторінка з документами
Chats	Сторінка чатів
ChangeOrders	Сторінка змін у контракті
Employers	Сторінка співробітників
Payments	Сторінка історії платежів
Taskboard	Сторінка задач
SettingsPage	Сторінка налаштувань користуча
Profile	Сторінка профайлу користувача
Portfolio	Сторінка портфоліо

В папці «utils» так як і на бекенді будуть зберігатися функції, які будуть використовуватись у всьому фронтенді. Список функцій фронтенду наведено в табл. 4.8.

Таблиця 4.8 – Опис функцій фронтенду

Назва	Опис
isAdmin	Перевірка поточного користувача на права адміністратора
withProtected	Перевірка поточної адреси. Якщо користувач ще не увійшов в систему, він може потрапити лише на публічні роути, такі як логін чи реєстрація. Якщо сторінка містить інформацію про якийсь проєкт або контракт, така сторінка називається захищена та повинна бути доступна лише після авторизації користувача. Якщо користувач не пройшов автентифікацію, потрібно зробити його переадресацію на сторінку логіну.
transformToKopeck	Функція для переведення вартості в копійки. Всі числа, які відображають вартість повинні зберігатися в базі даних як цілі числа без дрібної частини.
transformKopecksToHrivnia	Оборотний процес перетворення копійок у гривні
getDaysDiff	Отримання різниці між датами. Буде використовуватись при перерахунку тривалості фази, якщо потрібно внести зміни до контракту.
handleError	Функція обробки помилки при запитах. Повинна виводити повідомлення, з описом помилки.
isImage	Перевірка типу файлу, чи є він зображенням. Буде використовуватись для відкриття зображення на весь екран
isDocument	Перевірка, чи є цей файл pdf-документом, для відкриття його засобами браузера.
isCustomer	Перевірка користувача, чи є він замовником.
isExecutor	Перевірка, чи є користувач виконавцем.
upload	Функція для завантаження файлу.
getUserAvatar	Функція для отримання посилання на аватар користувача.
getUserFullName	Отримання ім'я та прізвища поточного користувача для відображення на екрані.

4.3 Опис інтерфейсу

Інтерфейс системи має велике значення для будь якої програми. Зручний та приємний інтерфейс одразу розташовує користувача до себе та приносить задоволення при роботі. Також він повинен бути інтуїтивно зрозумілим, щоб не було потреби постійно шукати необхідне в керівництві користувача.

При створенні інтерфейсу програми були забезпечені наступні властивості:

- природність інтерфейсу;
- узгодженість інтерфейсу;
- дружній інтерфейс;
- принцип «зворотного зв'язку»;
- простота інтерфейсу;
- загальнодоступність;
- естетична привабливість [138].

Природність інтерфейсу. Полягає в тому, що він не змушує користувача суттєво змінювати звичні йому способи вирішення завдань. У застосунку, що розробляється, ця властивість буде використовуватися наступним чином: повідомлення та результати виконання не повинні вимагати будь-яких додаткових пояснень, повинні бути збережені позначення та термінологія даної предметної області, повинні використовуватись знайомі поняття та образи, що будуть забезпечувати інтуїтивне розуміння інтерфейсу.

Узгодженість інтерфейсу. Дозволить користувачеві використовувати раніше набуті знання для з'ясування нових аспектів роботи. У застосунку це можна буде простежити в наступному: будуть використовуватись одне і те саме найменування для виконання тих чи інших функцій, які здаються користувачеві аналогічними. Буде використовуватись стандартний інтерфейс роботи в середовищі Windows і т. д.

Дружній інтерфейс. Не для кого не секрет, що вивчення будь-якого програмного продукту відбувається методом спроб і помилок, тому для цього в програмі необхідно передбачити запобігання таким ситуаціям, які найімовірніше закінчилися б помилками.

Принцип «зворотного зв'язку». Цей принцип має на увазі, що кожна дія користувача повинна супроводжуватись візуальним підтвердженням того, що програма сприйняла команду. Тут потрібно також врахувати, що типовий користувач здатний витерпіти лише кілька секунд очікування реакції у відповідь від свого електронного «співрозмовника».

Простота інтерфейсу. Перед користувачем повинна відображатися мінімально необхідна інформація. Необхідно також уникати багатослівних командних імен та повідомлень та розміщувати елементи управління на екрані з урахуванням їх смислового значення та логічного взаємозв'язку.

Загальнодоступність. Під цим розуміється сукупність характеристик інтерфейсу, що забезпечують його просте освоєння та ефективне застосування незалежно від ступеня готовності користувача.

Естетична привабливість. полягає в тому, щоб на екрані формувалося таке середовище, яке б допомагало користувачеві зрозуміти інформацію, що надається, зосереджуватися на ній і, по можливості, зменшувати навантаження на людину (неприпустимо використання яскравих фарб в заголовках меню, різких переміщень, лякаючих зображень і т. д.) [138].

Інтерфейс програми надає декілька сторінок.

Сторінка розділу керування контрактом розділена на три зони:

- верхня – ліворуч знаходяться кнопки переходу до списку проєктів та кнопка для переходу до керування контрактом. Праворуч, кнопка допомоги, яка відкриває інструкцію з використання поточної сторінки, іконка з кнопкою переходу до чату, яка показує що є нові повідомлення в чаті та аватар поточного користувача з його ім'ям. При натисканні на ім'я з'являється випадаючий список з переходом до налаштувань системи, налаштувань профілю користувача та виходом з системи;

- ліворуч є бокова панель для навігації по всім сторінкам контракту;
- остання частина екрану призначена для відображення робочої інформації по кожній сторінці.

Сторінка з завданнями (рис. 4.8) надає можливість користувачам бачити прогрес виконання завдань. Спочатку всі завдання знаходяться в колонці «Треба зробити». Коли Виконавець приймає в роботу якесь завдання, він переміщує його в колонку «В процесі», після виконання в «Виконано». Біля кожного підпису колонки показана кількість завдань в цьому статусі. У правому верхньому куті є кнопка для додавання нового завдання.

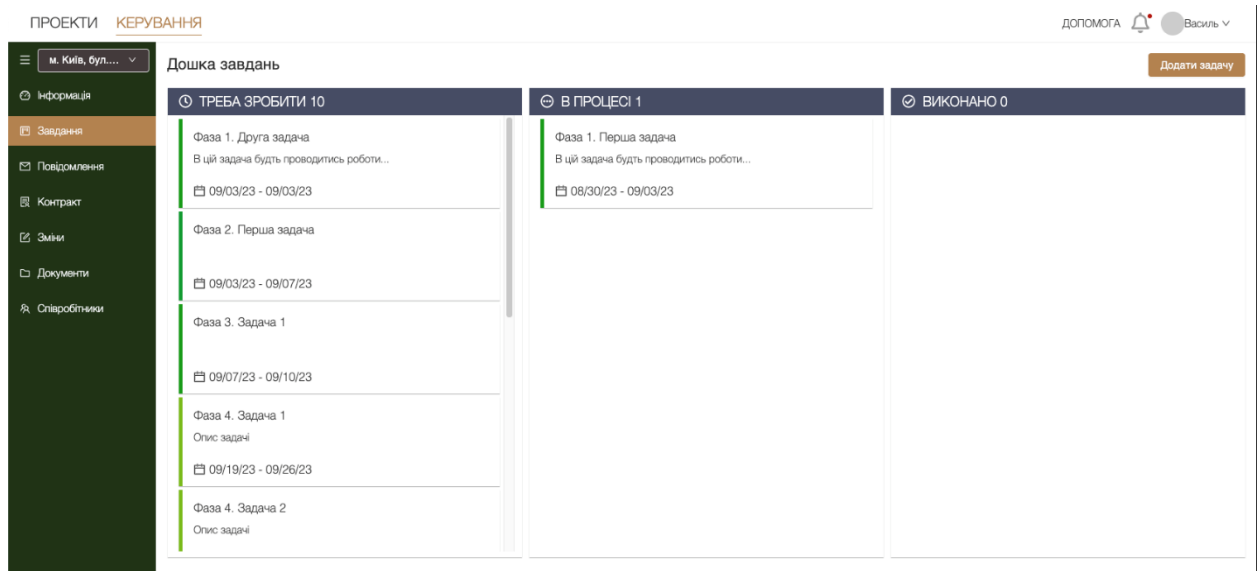


Рисунок 4.8 – Сторінка з завданнями

На сторінці чату (рис. 4.9) користувачі можуть вести переписку. На панелі ліворуч знаходиться список чатів. Чати розділено на дві категорії – групові, де може спілкуватися більше двох людей та персональні, де переписка відбувається суто між двома користувачами. За допомогою кнопки «+» біля груп чатів можна додавати нові чати у відповідні категорії. Зверху знаходиться назва чату, далі йде переписка. Для зручності, переписка умовно поділяється на дні. Дата представлена у вигляді дня тижня, місяця, дня та року. Праворуч вказаний час відправлення повідомлення. Знизу є кнопка додавання файлу та

поле для текстового повідомлення. Саме повідомлення представлене аватаром, ім'ям та прізвищем користувача.

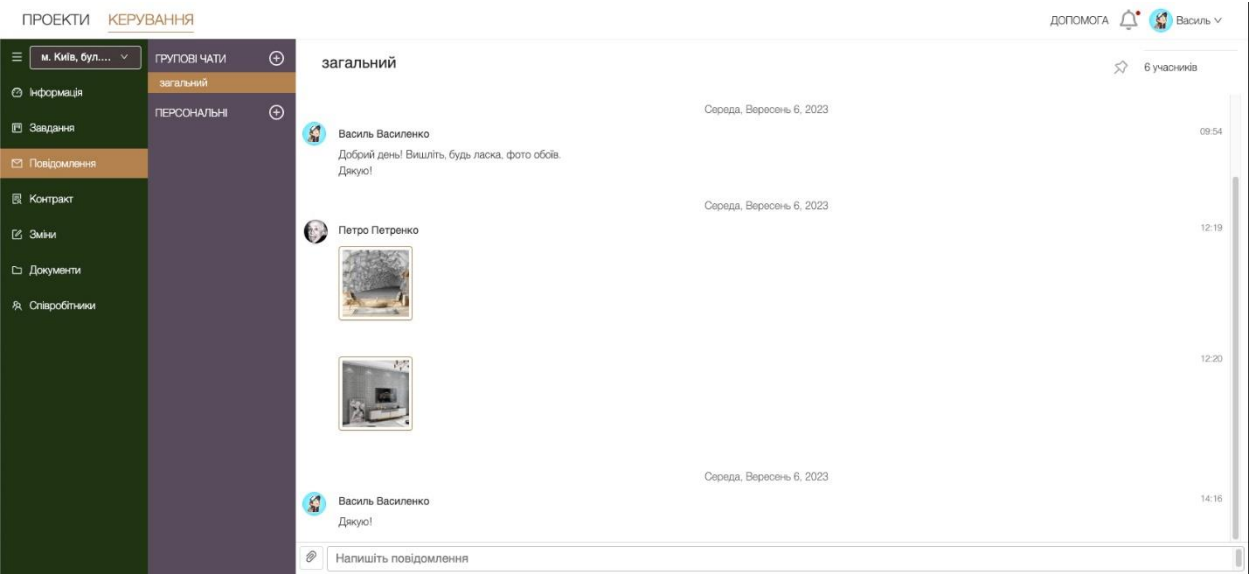


Рисунок 4.9 – Сторінка чату

Сторінка контракту (рис. 4.10) представлена у вигляді адреси (зверху) та таблиці, де можна переглянути назви фаз робіт та їх опис. Кожна фаза включає свої завдання та опис до них.

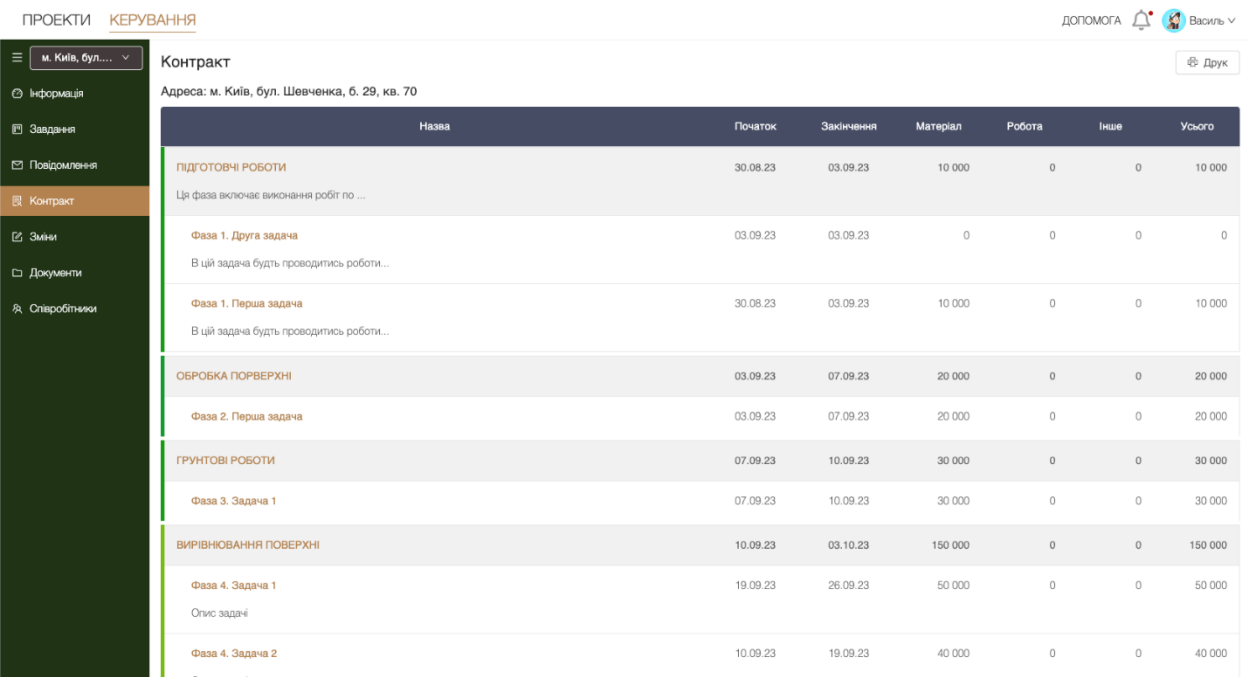


Рисунок 4.10 – Сторінка контракту

Також є можливість переглянути дату початку та плановану дату завершення, вартість матеріалів, роботи, інших чинників, які впливають на кінцеву вартість та загальну вартість по задачі та фазах. Знизу наведено загальну вартість по контракту.

Сторінка співробітників (рис. 4.11) зверху праворуч містить кнопку для запрошення співробітника. Нижче є таблиця, в якій можна переглянути всіх співробітників на контракті. В таблиці є наступні поля:

- ім'я, прізвище;
- представник – чиї інтереси він представляє (замовника чи виконавця);
- його права в цьому контракті;
- кнопки керування цим співробітником (в поточній версії – це кнопка видалення).

Ім'я	Представник	Доступ	Дії
Петро Петренко	Замовника	Замовник або Виконавець	
Василь Василенко	Виконавця	Замовник або Виконавець	
Test Owner Owner	Замовника	Читання задач та чату	Видалити
Андрій Андрієнко	Замовника	Читання задач та чату	Видалити
Денис Денисенко	Виконавця	Читання задач та чату	Видалити
Сергій Сергієнко	Виконавця	Повний доступ	Видалити

Рисунок 4.11 – Сторінка з інформацією про співробітників

Запуск застосунку для локальної розробки передбачає наступні дії:

- перейти в терміналі до директорії з бекенд частиною;
- встановити модулі залежності за допомогою команди: «yarn

install»;

- запустити бекенд за допомогою команди: «yarn dev»;
- перейти в терміналі до директорії з фронтенд частиною;
- встановити модулі залежності за допомогою команди: «yarn

install»;

- запустити фронтенд за допомогою команди: «yarn dev».

4.4 Тестування програми

Тестування ПЗ – це необхідний процес під час розробки, під час якого виявляються проблеми у роботі програми. Для запобігання помилок, потрібно написати тести на всі функції, запити та мутації, щоб бути впевненим, що внесені зміни в одному місці не привели до помилки в іншому.

- для тестування роботи функціоналу застосунка було розроблено та використано юніт-тести. Процес написання юніт-тестів включав декілька кроків [139]:

- визначення тестованих «юнітів». В якості «юнітів» виступали всі функції, запити та мутації бекенд застосунку;

- розробка тестових випадків – при створенні випадків особлива увага приділялася захисту від несанкціонованого доступу до даних. Всі запити на отримання чи зміну даних повинні відбуватися лише після перевірки токена, який прийшов разом з запитом;

- написання тестового коду за допомогою фреймворку Jest (результати наведено в Додатку А.1).

- запуск тестів – повинен проводитись після будь-яких змін в кодовій базі, навіть якщо на перший погляд ці зміни не на що не впливають.

- аналіз результатів і виправлення помилок. Якщо функція перестала правильно обробляти дані, треба це виправити та знову перевірити шляхом тестування, якщо змінилась логіка роботи – оновити тест.

Фрагмент тексту програми наведено в Додатку А.2.

4.5 Висновки до розділу 4

В цьому розділі відповідно до мети та завдань роботи було виконано конструювання спеціалізованого ПЗ SRM-системи. Розроблено структурні схеми та алгоритми функціонування для фронтенд та бекенд частин вебзастосунку. Представлено структуру папок, описано класи, функції. Описано інтерфейс системи та особливості тестування.

Розроблений вебзастосунок дозволяє виконувати такі функції:

- замовляти виконання ремонтних робіт;
- автоматизувати пошук виконавців;
- складати кошторис роботи;
- вести внутрішню переписку протягом всього часу;
- складати контракт на певний перелік робіт;
- вносити корективи під час роботи;
- узгоджувати або відхиляти зміни в контракті;
- гарантувати захищення коштів;
- вести історію замовлень та виконаних робіт.

ВИСНОВКИ

Як показали проведені дослідження, за останні кілька років управління взаємовідносинами з постачальниками зазнало значних змін через зміну технологій та зростання глобальних масштабів економіки. Сучасні управління взаємовідносинами з постачальниками (SRM) дозволяють ефективно керувати бізнес-процесами, але зазвичай не забезпечують урахування всіх особливостей бізнес-діяльності кожної окремої компанії.

Розроблена спеціалізована SRM-система призначена для організації взаємодії замовника виконання ремонтних робіт у власній будівлі або квартирі та виконавця цих робіт. Її впровадження є доцільним, оскільки дозволить надати гарантії вчасного та якісного виконання робіт замовнику та оплати виконаних робіт постачальнику послуг. Завдяки фіксації у електронному вигляді всіх вимог, змін, розрахунків вартості та оплаті робіт через посередника вдасться уникнути багатьох непорозумінь.

Вебзастосунок реалізовано на мові програмування JavaScript з використанням бібліотеки React та бекенду, який також розроблено на мові програмування JavaScript на платформі Node.js. База даних PostgreSQL.

Наукова новизна роботи полягає в тому, роботи полягає в тому, що розроблено методику створення спеціалізованої SRM-системи, що базується на методології Agile та охоплює всі етапи створення надійного та масштабованого ПЗ, адаптованого до потреб бізнесу.

Практична цінність роботи полягає в тому, що впровадження розробленого спеціалізованого ПЗ SRM-системи допоможе трансформувати процеси взаємодії постачальників та замовників послуг та досягти успіху в бізнесі.

В подальшій роботі можливо розширити функціонал застосунку та удосконалити роботу з системою за рахунок додавання кнопки для редагування email та кнопки повторної відправки листа з запрошенням зареєструватися для нових користувачів системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. SRM-система [Електрон. ресурс]. – Режим доступу: <https://wezom.com.ua/ua/blog/srm-system>.
2. A complete guide to supplier relationship management (SRM) [Electronic resource]. – Access mode: <https://www.jaggaer.com/blog/complete-guide-srm/>.
3. Everything you need to know about SRM and CRM [Electronic resource] / J. Serrao. – Access mode: <https://www.gainfront.com/blog/srm-crm-everything-you-need-to-know/>.
4. Differentiated CRM – the modern strategy to increase company performance growth / [H. Lošťáková, M. Bednaříková, L. Branská et al.]. – Czech Republic, 2009. – 272 p.
5. Gros, I. Supply systems: supply chain management / I. Gros, S. Grosova // Czech Republic, 2012. – 187 p.
6. Lambert, D. M. Supplier relationship management as a macro business process, / D. M. Lambert, M. A. Schwieterman // Supply chain management: an international journal, United Kingdom. – 2012. – Vol. 17, № 3. – P. 337-352.
7. Ling, L. Supply Chain Management / L. Ling // United Kingdom, 2007. – 346 p.
8. Vlckova, V. Ways of cooperation within supply chain in metallurgy from the point of view of company processing steel coils / V. Vlckova // Metallurgy and materials: 24th International conference, 3 -5 June 2015, Brno, Czech Republic: proceedings. – 2015. – P. 2118-2124.
9. Cao, M. Supply chain collaboration: Impact on collaborative advantage and firm performance, / M. Cao, Q. Zhang // Journal of operations management, netherlands. – 2011. – Vol. 29, № 3. – P 163-180.
10. Gros, I. The secret of the modern purchase / I. Gros, S. Grosova. – Czech Republic. – 2006. – 184 p.

11. Sodomka, P. Information systems in business practice / P. Sodomka, H. Klčova. – Czech Republic. – 2010. – 501 p.
12. Ballou, R. H. New managerial challenges from supply chain opportunities / R. H. Ballou, S. M. Gilbert, A. Mukherjee // Industrial marketing management. – Netherlands. – 2000. – Vol. 29, № 1. – P. 7-18.
13. How to improve supplier relationship management (SRM System)? [Electronic resource]. – Access mode: <https://www.procurementflow.com/blog/supplier-relationship-management-system-srm>.
14. Supplier relationship management as a tool for strengthening partnerships in supply chain / [L. Branská, M. Paták, Z. Pecinova et al.] // Social sciences & arts: 3rd International multidisciplinary scientific conferences, 22 - 31 August, 2016, Albena, Bulgaria: proceedings. – 2016. – P. 1-8.
15. Supplier relationship management software: how you can use it to improve supplier relationships [Electronic resource]. – Access mode: <https://www.prokuria.com/post/supplier-relationship-management-software>.
16. 6 Steps to implement your own supplier relationship management (SRM) program [Electronic resource] / L. K. Pedersen. – Access mode: <https://www.leanlinking.com/blog/implement-supplier-relationship-management-program>.
17. What is supplier relationship management (SRM)? Tips for a strategic process arielle kushner [Electronic resource]. – Access mode: <https://www.approve.com/blog/supplier-relationship-management/>.
18. Leverage supplier relationships. Boost efficiency and gain competitive advantage with an effective SRM strategy [Electronic resource]. – Access mode: <https://www.gep.com/supplier-relationship-management-guide>.
19. 5 SRM software examples [Electronic resource]. – Access mode: <https://www.kodiakhub.com/blog/srm-software-examples>.

20. 7 best SRM software providers: a comparison between vendors [Electronic resource]. – Access mode: <https://www.kodiakhub.com/blog/7-best-srm-software-providers-a-comparison-between-vendors>.

21. CRM for suppliers? You need an SRM platform [Electronic resource]. – Access mode: <https://www.kodiakhub.com/blog/crm-for-suppliers-you-need-an-srm-platform#:~:text=A%20business%20is%20covering%20many,whom%20you%20are%20a%20customer>.

22. Supplier relationship management software: a complete guide [Electronic resource]. – Access mode: <https://wesoftyou.com/outsourcing/supplier-relationship-management-software-guide/>.

23. Закон України №2297–VI. Про захист персональних даних [Електрон. ресурс]. – Режим доступу: <https://www.president.gov.ua/documents/2297vi-11567>.

24. Most used programming languages among developers worldwide as of 2023 [Electronic resource]. – Access mode: <https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/>.

25. Top JavaScript statistics you must know to kickstart your business in 2024 [Electronic resource] / N. Raval. – Access mode: <https://radixweb.com/blog/top-javascript-usage-statistics>.

26. TypeScript vs JavaScript: the difference you should know [Electronic resource] / N. Raval. – Access mode: <https://radixweb.com/blog/typescript-vs-javascript>.

27. What is TypeScript? Benefits of TypeScript and disadvantages [Electronic resource] / P. Zaitsev. – Access mode: <https://itjet.io/blog/what-is-typescript>.

28. TypeScript vs. JavaScript: your go-to guide [Electronic resource] / D. Monesi. – Access mode: <https://www.toptal.com/typescript/typescript-vs-javascript-guide>.

29. React [Electronic resource]. – Access mode: <https://reactjs.org>.
30. React Getting Started / [Electronic resource]. – Access mode: <https://legacy.reactjs.org/docs/getting-started.html>.
31. React [Electronic resource]. – Access mode: <https://github.com/facebook/react>.
32. jQuery [Electronic resource]. – Access mode: <https://jquery.com>.
33. jQuery API [Electronic resource]. – Access mode: <https://api.jquery.com>.
34. jQuery — new wave JavaScript [Electronic resource]. – Access mode: <https://github.com/jquery/jquery/>.
35. Angular – deliver web apps with confidence / [Electronic resource]. – Access mode: <https://angular.io>.
36. Introduction to the Angular docs [Electronic resource]. – Access mode: <https://angular.io/docs>.
37. Angular – the modern web developer's platform [Electronic resource]. – Access mode: <https://github.com/angular/angular>.
38. The progressive JavaScript framework [Electronic resource]. – Access mode: <https://vuejs.org>.
39. Introduction [Electronic resource]. – Access mode: <https://vuejs.org/v2/guide>.
40. Vue.js [Electronic resource]. – Access mode: <https://github.com/vuejs/vue>.
41. Web frameworks [Electronic resource]. – Access mode: <https://insights.stackoverflow.com/survey/2021#most-popular-technologies-webframe-prof>.
42. 8 benefits of ReactJS: is it worth using in your project? [Electronic resource] / O. Kopachovets. – Access mode: <https://procoders.tech/blog/advantages-of-using-reactjs/>.

43. Top 10 benefits of React.js for application development [Electronic resource]. – Access mode: <https://ncube.com/top-10-benefits-of-react-js-for-application-development/>.
44. 15 advantages of ReactJS for application development [Electronic resource] / M. Gothankar. – Access mode: <https://www.yourteaminindia.com/blog/advantages-of-reactjs>.
45. Ant design 5.0 [Electronic resource]. – Access mode: <https://ant.design/>.
46. Ready to use material design components [Electronic resource]. – Access mode: <https://material-ui.com/>.
47. React bootstrap [Electronic resource]. – Access mode: <https://react-bootstrap.netlify.app/>.
48. Create accessible React apps with speed [Electronic resource]. – Access mode: <https://chakra-ui.com/>.
49. With ant design, create React components like a pro [Electronic resource]/ A. Dhyani. – Access mode: <https://opensenselabs.com/blog/articles/ant-design-create-react-components-pro>.
50. Ant design 101 – introduction to a design system for enterprises [Electronic resource]. – Access mode: <https://www.uxpin.com/studio/blog/ant-design-introduction/>.
51. Full review. Ant design VS material UI [Electronic resource] / Scott. – Access mode: <https://pixso.net/reviews/ant-design-vs-material-ui/>.
52. CSS selectors [Electronic resource]. – Access mode: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_selectors.
53. Inline CSS guide – how to style an HTML tag directly [Electronic resource] / A. Haddad. – Access mode: <https://www.freecodecamp.org/news/inline-css-guide-how-to-style-an-html-tag-directly/>.
54. css-modules [Electronic resource]. – Access mode: <https://github.com/css-modules/css-modules>.

55. CSS for the component age [Electronic resource]. – Access mode: <https://styled-components.com/>.
56. How to use styled components in your React apps [Electronic resource] / K. Makode. – Access mode: <https://www.freecodecamp.org/news/styled-components-in-react/>.
57. CSS modules vs styled-components [Electronic resource]. – Access mode: <https://stackshare.io/stackups/css-modules-vs-styled-components>.
58. Prettier [Electronic resource]. – Access mode: <https://github.com/prettier/prettier>.
59. JavaScript Standard Style [Electronic resource]. – Access mode: <https://github.com/standard/standard>.
60. JS Beautifier [Electronic resource]. – Access mode: <https://github.com/beautify-web/js-beautify>.
61. JSHint [Electronic resource]. – Access mode: <https://jshint.com/>.
62. JSLint [Electronic resource]. – Access mode: <https://www.jshint.com/>.
63. JavaScript standard style [Electronic resource]. – Access mode: <https://standardjs.com/>.
64. Find and fix problems in your JavaScript code [Electronic resource]. – Access mode: <https://eslint.org/>.
65. ESLint [Electronic resource]. – Access mode: <https://www.npmjs.com/package/eslint>.
66. ESLint vs. JSHint: a comparison and how to choose [Electronic resource]. – Access mode: <https://www.testim.io/blog/eslint-vs-jshint/>.
67. How to improve your JavaScript code quality with ESLint [Electronic resource] / A. Dhiman. – Access mode: <https://blog.cloudanalogy.com/how-to-improve-your-javascript-code-quality-with-eslint/>.
68. Webpack [Electronic resource]. – Access mode: <https://webpack.js.org/>.
69. Parcel [Electronic resource]. – Access mode: <https://parceljs.org/>.
70. rollup.js [Electronic resource]. – Access mode: <https://rollupjs.org/>.

71. Browserify [Electronic resource]. – Access mode: <http://browserify.org/>.
72. A toolkit to automate & enhance your workflow / [Electronic resource]. – Access mode: <https://gulpjs.com/>.
73. Webpack or Browserify & Gulp: which is better? [Electronic resource] / E. Grosse. – Access mode: <https://www.toptal.com/front-end/webpack-browserify-gulp-which-is-better>.
74. Webpack [Electronic resource]. – Access mode: <https://www.npmjs.com/package/webpack>.
75. Webpack configuration [Electronic resource]. – Access mode: <https://open-amt-cloud-toolkit.github.io/docs/2.3/Reference/UIToolkit/webpackConfig/>.
76. Understanding Webpack's code splitting feature [Electronic resource] / N. Sebastian. – Access mode: <https://blog.bitsrc.io/understanding-webpacks-code-splitting-feature-3077768e4066>.
77. Oracle [Electronic resource]. – Access mode: <https://www.oracle.com/>.
78. MySQL [Electronic resource]. – Access mode: <https://www.mysql.com/>.
79. Microsoft SQL Server [Electronic resource]. – Access mode: <https://www.microsoft.com/en-us/sql-server/>.
80. PostgreSQL: the world's most advanced open source relational database [Electronic resource]. – Access mode: <https://www.postgresql.org/>.
81. PostgreSQL vs Oracle: 6 critical differences [Electronic resource] / A. Phaujdar. – Access mode: <https://hevodata.com/learn/postgresql-vs-oracle/>.
82. Difference between MySQL and Oracle [Electronic resource]. – Access mode: <https://www.javatpoint.com/mysql-vs-oracle>.
83. PostgreSQL vs SQL server: 16 critical differences [Electronic resource] / S. Ravoof. – Access mode: <https://kinsta.com/blog/postgresql-vs-sql-server/>.
84. Memcached [Electronic resource]. – Access mode: <https://memcached.org/>.

85. Apache Kafka [Electronic resource]. – Access mode: <https://kafka.apache.org/>.
86. Real-time speed and simplicity [Electronic resource]. – Access mode: <https://redis.com/>.
87. Amazon ElastiCache [Electronic resource]. – Access mode: https://aws.amazon.com/elasticache/?nc1=h_ls.
88. Memcached vs Redis [Electronic resource]. – Access mode: <https://www.baeldung.com/memcached-vs-redis>.
89. Amazon ElastiCache vs Kafka [Electronic resource]. – Access mode: <https://stackshare.io/stackups/amazon-elasticache-vs-kafka>.
90. Apache Kafka vs. Redis: are they truly comparable? [Electronic resource] / J. Carder. – Access mode: <https://www.openlogic.com/blog/apache-kafka-vs-redis>.
91. Google Drive [Electronic resource]. – Access mode: <https://drive.google.com>.
92. Securely collaborate on your content anywhere, anytime / [Electronic resource]. – Access mode: <https://www.dropbox.com>.
93. Microsoft OneDrive [Electronic resource]. – Access mode: <https://onedrive.live.com>.
94. Apple iCloud [Electronic resource]. – Access mode: <https://www.icloud.com>.
95. Amazon S3 [Electronic resource]. – Access mode: <https://aws.amazon.com/s3>.
96. Simplify how you work [Electronic resource]. – Access mode: <https://www.box.com>.
97. Everything your business needs to work efficiently, all in one place [Electronic resource]. – Access mode: <https://www.dropbox.com/business>.
98. Dropbox vs. Google Drive: Which is right for you? / [Electronic resource] / M. Rebelo. – Access mode: <https://zapier.com/blog/dropbox-vs-google-drive/>.

99. OneDrive vs iCloud: PC or Mac cloud storage? [Electronic resource] / N. Turner. – Access mode: <https://blog.internxt.com/onedrive-vs-icloud/>.
100. Amazon S3 vs Box [Electronic resource]. – Access mode: <https://www.softwareadvice.com/cloud-storage/amazon-s3-profile/vs/box-enterprise/>.
101. OneDrive vs. Google Drive: which cloud storage option is best? [Electronic resource] / A. Pell. – Access mode: <https://zapier.com/blog/onedrive-vs-google-drive/>.
102. JavaScript [Electronic resource]. – Access mode: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
103. About documentation [Electronic resource]. – Access mode: <https://nodejs.org/en/docs/>.
104. Python 3.12.0 documentation [Electronic resource]. – Access mode: <https://docs.python.org/3/>.
105. Django documentation [Electronic resource]. – Access mode: <https://docs.djangoproject.com/en/4.2/>.
106. Flask [Electronic resource]. – Access mode: <https://flask.palletsprojects.com/en/2.1.x/>.
107. FastAPI [Electronic resource]. – Access mode: <https://flask.palletsprojects.com/en/2.1.x/>.
108. Ruby [Electronic resource]. – Access mode: <https://ruby-doc.org/>.
109. Ruby on Rails Guides (v7.1.2) [Electronic resource]. – Access mode: <https://guides.rubyonrails.org/>.
110. Java documentation [Electronic resource]. – Access mode: <https://docs.oracle.com/en/java/>.
111. Spring framework documentation [Electronic resource]. – Access mode: <https://docs.spring.io/spring-framework/reference/>.
112. Java EE platform specification [Electronic resource]. – Access mode: <https://javaee.github.io/javaee-spec/>.

113. PHP manual [Electronic resource]. – Access mode: <https://www.php.net/manual/en/>.

114. Laravel [Electronic resource]. – Access mode: <https://www.php.net/manual/en/>.

115. Symfony documentation [Electronic resource]. – Access mode: <https://symfony.com/doc/current/index.html>.

116. Go [Electronic resource]. – Access mode: <https://go.dev/doc/>.

117. Gin web framework [Electronic resource]. – Access mode: <https://pkg.go.dev/github.com/gin-gonic/gin#section-readme>.

118. Echo [Electronic resource]. – Access mode: <https://echo.labstack.com/>.

119. C# documentation [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/dotnet/csharp/>.

120. ASP.NET documentation [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0>.

121. Most used web frameworks among developers worldwide, as of 2023 [Electronic resource]. – Access mode: <https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/>.

122. Top 5 NodeJS pros and cons: what they mean for your business [Electronic resource] / D. Koteshev. – Access mode: <https://anywhere.epam.com/business/node-js-pros-and-cons>.

123. 15 great advantages of Node JS in software development [Electronic resource] / Z. Aftab. – Access mode: <https://enou.co/blog/advantages-of-node-js/>.

124. Advantages of Node js [Electronic resource] / I. Sharma. – Access mode: <https://tatvasoft.com/outsourcing/2021/07/advantages-of-node-js.html>.

125. What is REST? [Electronic resource]. – Access mode: <https://restfulapi.net/>.

126. Hypertext Transfer Protocol – HTTP 1.1 [Electronic resource]. – Access mode: <https://www.w3.org/Protocols/rfc2616/rfc2616.html>.

127. GraphQL [Electronic resource]. – Access mode: <https://graphql.org/>.

128. Apollo docs [Electronic resource]. – Access mode: <https://www.apollographql.com/docs/>.

129. Documentation [Electronic resource]. – Access mode: <https://grpc.io/docs/>.

130. Protocol Buffers [Electronic resource]. – Access mode: <https://protobuf.dev/>.

131. MySQL documentation [Electronic resource]. – Access mode: <https://dev.mysql.com/doc/>.

132. Documentation [Electronic resource]. – Access mode: <https://www.postgresql.org/docs/>.

133. Microsoft SQL documentation [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/sql/?view=sql-server-ver16>.

134. Database documentation [Electronic resource]. – Access mode: <https://docs.oracle.com/en/database/>.

135. GraphQL is the better REST [Electronic resource]. – Access mode: <https://www.howtographql.com/basics/1-graphql-is-the-better-rest/>.

136. REST vs. GraphQL vs. gRPC – which API to choose? [Electronic resource]. – Access mode: <https://www.baeldung.com/rest-vs-graphql-vs-grpc>.

137. GraphQL vs. REST vs. SQL vs. gRPC vs. OData vs. MongoDB / [Electronic resource]. – Access mode: <https://ritza.co/articles/graphql-vs-rest-vs-sql-vs-grpc-vs-odata-vs-mongodb/>.

138. Rules and tips for good UI design in web application [Electronic resource] / A. Kryzhanovska, M. Sharapova. – Access mode: <https://gearheart.io/articles/role-web-application-user-interface-design/>.

139. Що таке Unit тести і як їх писати / [Електрон. ресурс]. – Режим доступу: <https://foxminded.ua/yunit-testy/>.

ДОДАТОК А
Текст програми

A.1 Приклади Unit-тестів

A.1.1 Тест 1

```
import * as assert from 'assert';
import { createHash } from 'crypto';
import { encrypt, decrypt } from '../utils/aes256';

describe('utils/aes256', () => {
  it('should allow to encrypt and decrypt string', () => {
    const secret = 'password';
    const data = 'hihi';

    const encrypted = encrypt(data, secret);
    assert.notEqual(encrypted, data);

    const decrypted = decrypt(encrypted, secret);
    const decryptedWithBuffer = decrypt(
      encrypted,
      createHash('sha256')
        .update(secret, 'utf8')
        .digest()
    );
    assert.equal(decrypted, data);
    assert.equal(decryptedWithBuffer, data);
  });
});
```

A1.2 Тест 2

```
import * as assert from 'assert';
import { execQuery } from '../utils/index';

const LOGIN_MUTATION = `mutation ($email: String!,
$password: String!) {
  login(email: $email, password: $password) {
    token
    user {
      id
    }
  }
}`;

describe('graphql/resolvers/Mutation/login', () => {
  it('should allow to login with existing user', async () => {
    {
```

```

        const { data, errors } = await execQuery(LOGIN_MUTATION,
{ email: 'default@test.com', password: 'password' }, null);

        assert.ok(!errors, 'there should be no errors');
        assert.ok(data!.login.token, 'token must be present');
        assert.equal(data!.login.user.id, '0281c17b-8460-4221-
843f-546cd0268b46');
    });

    it('should return error if password is not correct', async
() => {
        const { data, errors } = await execQuery(
            LOGIN_MUTATION,
            { email: 'default@test.com', password: 'random-string'
},
            null
        );

        assert.ok(errors, 'it should return errors');
        assert.ok(!data, 'there should be no data');
    });
});

```

A1.3 Тест 3

```

import * as assert from 'assert';
import { execQuery, getCurrentUser } from '../.../index';

const UPDATE_USER_BASIC_INFO_MUTATION = `mutation
($firstName: String, $lastName: String, $phone: String, $about:
String) {
  updateUserBasicInfo(firstName: $firstName, lastName:
$lastName, phone: $phone, about: $about) {
    id
    email
    firstName
    lastName
    phone
    about
  }
}`;

describe('graphql/resolvers/Mutation/updateUserBasicInfo', ()
=> {
    it('should update info', async () => {
        const email = 'updated@test.com';
        const currentUser = await getCurrentUser(email);
        const updatedInfo = {
            firstName: 'NewName',
            lastName: 'NewLastName',

```

```

        phone: '23232323232'
    };

    const { errors, data } = await
    execQuery(UPDATE_USER_BASIC_INFO_MUTATION, updatedInfo,
    currentUser);

    assert.ok(!errors, 'there should be no errors');
    assert.ok(!data!.updateUserBasicInfo.about, 'there
    should be no value');
    assert.equal(data!.updateUserBasicInfo.email, email);
    assert.equal(data!.updateUserBasicInfo.firstName,
    updatedInfo.firstName);
    assert.equal(data!.updateUserBasicInfo.lastName,
    updatedInfo.lastName);
    assert.equal(data!.updateUserBasicInfo.phone,
    updatedInfo.phone);
    });
});

```

A1.4 Тест 4

```

import * as assert from 'assert';
import { execQuery, getCurrentUser } from '../../index';

const GET_PROJECT_QUERY = `query ($projectId: ID!) {
  getProject(projectId: $projectId) {
    id
  }
}`;

describe('graphql/resolvers/Query/getProject', () => {
  it('should allow to get my project', async () => {
    const currentUser = await getCurrentUser('for-
    get@test.com', 'c9caf642-6cbf-4371-a97e-ca756339eaaa');
    const projectId = 'afaff0ea-54d2-4abc-8947-
    9f005a465270';
    const { errors, data } = await
    execQuery(GET_PROJECT_QUERY, { projectId }, currentUser);

    assert.ok(!errors, 'there should be no errors');
    assert.equal(data!.getProject.id, projectId);
  });

  it('should not allow to get someone's else project', async
  () => {
    const currentUser = await getCurrentUser('for-
    create@test.com', 'b79207c2-9db1-47f8-9f8d-5e06b170f413');
    const projectId = 'afaff0ea-54d2-4abc-8947-
    9f005a465270';

```

```

        const { errors, data } = await
execQuery(GET_PROJECT_QUERY, { projectId }, currentUser);

        assert.ok(errors, 'there should be an error');
        assert.ok(!data, 'there should be no data');
    });
});

```

A.2 Фрагмент текста програми

```

import { Phase, PHASE_TABLE } from
'../../../../../db/types/phase';
import { UserRole } from
'../../../../../db/types/role';
import { EmployeePermission } from
'../../../../../db/types/collaborator';
import { PhaseModel } from
'../../../../../db/models/PhaseModel';
import { defMutation, GraphQLError } from
'../../../../../index';
import { PhaseCostInput } from
'../../PhaseCostInput';
import { validateContractAccess, WithContractAccess
} from ' ../../../../checks/validateContractAccess';

type TArgs = { phaseId: string; cost: PhaseCostInput
};

type TReturn = Phase;

defMutation<TReturn, TArgs>(
  `updatePhaseCost(
    phaseId: ID!,
    cost: PhaseCostInput!
  ): Phase! @authenticated @contractPaid(path:
"phaseId")`,
  (_root, { phaseId, cost }, ctx) => {
    return ctx.db.getClient<Phase>(async client =>
{
      const hasContractAccess =
ctx.sql.contractAccess(ctx.sql.raw('"contractId"'),
ctx.currentUser!.roleId, {
        minPermission: EmployeePermission.Full,
        role: UserRole.Executor
      });

```

```

        const {
          rows: [phase]
        } =
        await
client.query<WithContractAccess<Phase>>(
  ctx.sql`
    SELECT *,
      ${hasContractAccess}
    AS
    "contractAccess"
    FROM ${PHASE_TABLE}
    WHERE "id" = ${phaseId}
  `
);
if (!phase)
  throw
GraphQLError.notFound('phase');
validateContractAccess(phase);

const updatedPhase =
  await
PhaseModel.update.exec(
  client,
  {
    id: phaseId,
    materialCost: cost.material,
    laborCost: cost.labor,
    otherCost: cost.other
  },
  ctx
);
if (!updatedPhase)
  throw
  new
GraphQLError('phase not updated');

return updatedPhase;
});
}
);

```

ДОДАТОК Б
Слайди презентації

**ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
ВЗАЄМОДІЇ ЗАМОВНИКІВ ТА ПОСТАЧАЛЬНИКІВ ПОСЛУГ**

Виконав
студент групи КНТ-222м

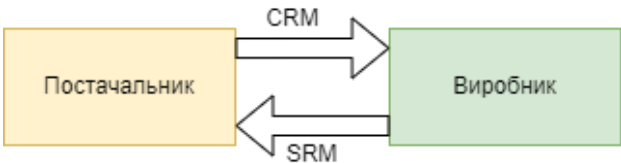
Ігор ФЕДЬКОВ

Керівник
к.т.н., доцент

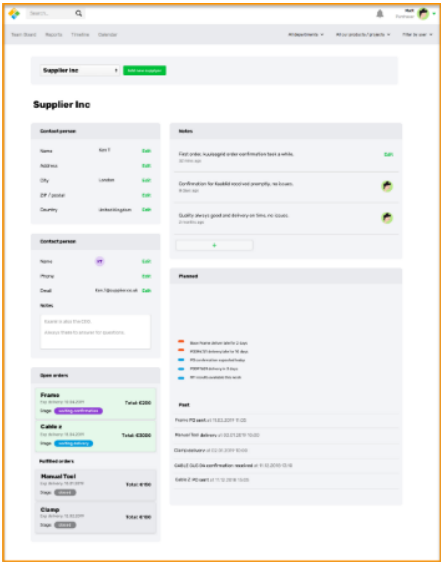
Анжеліка ПАРХОМЕНКО

Рисунок Б.1 – Слайд № 1

АКТУАЛЬНІСТЬ РОБОТИ



Зв'язок між SRM і CRM в системі постачання



Приклад екранної форми SRM-системи

Рисунок Б.2 – Слайд № 2

МЕТА ТА ЗАВДАННЯ ДИПЛОМНОЇ РОБОТИ

Мета роботи – дослідження та програмна реалізація спеціалізованого програмного забезпечення автоматизації взаємодії замовників та постачальників послуг для підвищення економічної ефективності бізнес-практик.

Завдання роботи:

- розробити методика програмної реалізації спеціалізованої SRM-системи;
- виконати проектування ПЗ SRM-системи;
- виконати розробку та тестування спеціалізованого ПЗ SRM-системи.

3

Рисунок Б.3 – Слайд № 3

МЕТОДИКА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ SRM-СИСТЕМИ



Структура SRM-системи



Базові функції ПЗ SRM-системи



Схема алгоритму розробки спеціалізованого ПЗ SRM

4

Рисунок Б.4 – Слайд № 4

ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ SRM-СИСТЕМИ

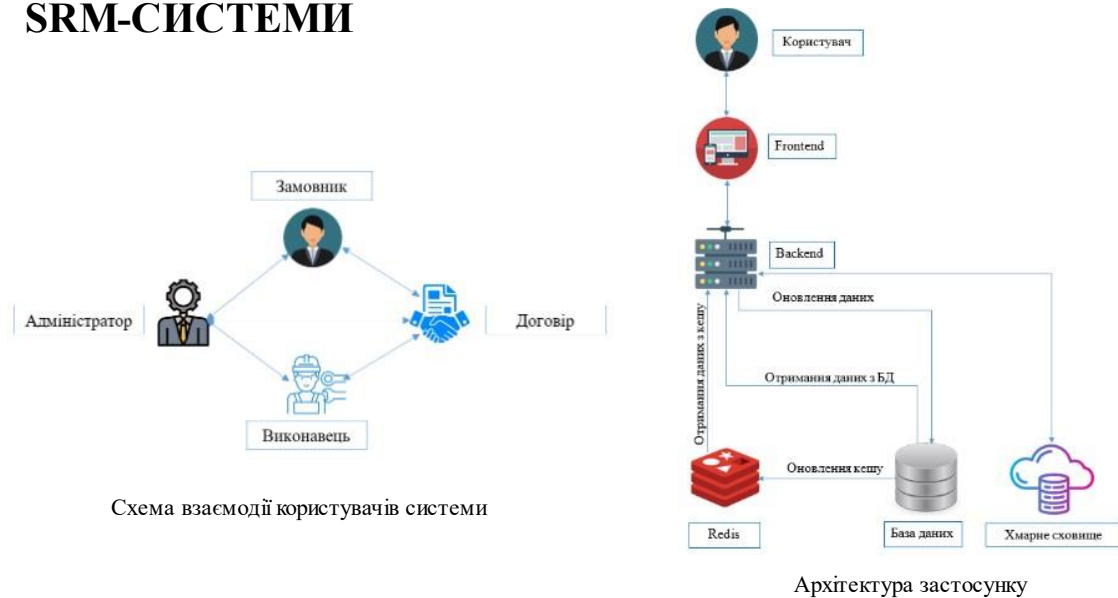


Схема взаємодії користувачів системи

Архітектура застосунку

5

Рисунок Б.5 – Слайд № 5

ВИБІР ІНСТРУМЕНТАРІЮ РОЗРОБКИ

Мова	Переваги	Недоліки
PHP	Мова спеціалізована для використання в веб-застосунках. Підтримує корпоративні застосунки.	Складність, Повільність, Слабка безпека.
Python	Простий та зрозумілий синтаксис. Широко використовується в науковій сфері.	Висока витратність пам'яті.
JavaScript	Мова розробки веб-застосунків. Широко використовується в веб-застосунках.	Висока витратність пам'яті.
Java	Платформонезалежна. Широко використовується в корпоративних застосунках.	Висока витратність пам'яті.

Критерій	React	Angular	Vue.js
Тип фреймворку	Бібліотека для створення UI	Бібліотека для створення UI	Бібліотека для створення UI
Складність	Середня	Висока	Низька
Оновлення DOM	Віртуальний DOM	Віртуальний DOM	Віртуальний DOM
Компонентний підхід	Так	Так	Так
Оптимізація продуктивності	Так	Так	Так
Громадська спільнота	Велика	Велика	Велика
Використання JSX	Так	Так	Так
Керування станом	Через setState	Через setState	Через setState
Маршрутизація	Додаткові бібліотеки	Додаткові бібліотеки	Додаткові бібліотеки
Розширюваність	Широко можливі налаштування та розширення	Широко можливі налаштування та розширення	Широко можливі налаштування та розширення
Офіційна документація	Так	Так	Так
Активна підтримка	Так	Так	Так

Метод стилізації	Переваги
Ant Design	Простий та швидко розроблюваний метод. Підтримка всіх можливостей CSS включаючи складні медіа-запити.
Material UI	Легко впроваджується існуючі проекти. Простота в використанні та розумінні. Стили можуть бути динамічними в залежності від даних.
React Bootstrap	Використовується вбудований стиль JSX, що спрощує компоновку компонентів.
Chakra UI	Локальна ізоляція стилів, що запобігає конфліктам класів. Модульність – стилі обмежені одним компонентом та можуть бути перевикористані. Підтримується автодоповнення статична перевірка типів. Створення компонентів з вбудованими стилями, що покращує читабельність коду. Динамічні стилі на основі стану компоненту. Можливість використання JavaScript для стилізації, що дозволяє динамічно змінювати стилі в залежності від даних. Автоматична генерація унікальних класів для ізоляції стилів.

Порівняльний аналіз мов програмування

Порівняльний аналіз фреймворків та бібліотек

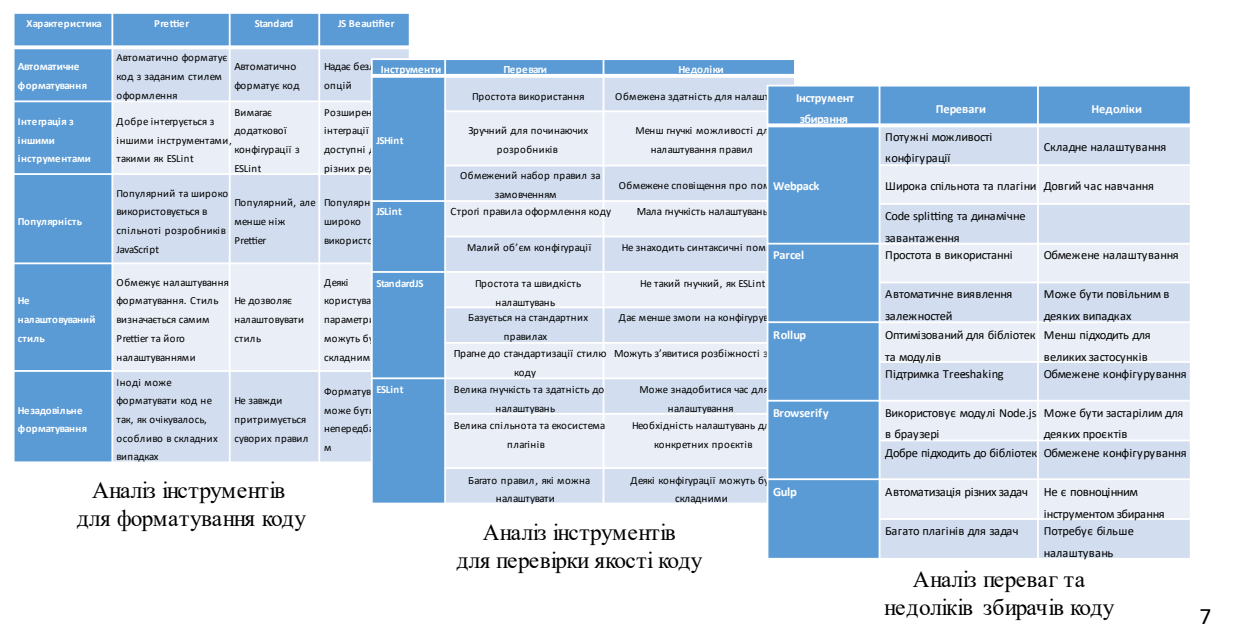
Порівняльний аналіз UI-бібліотек

Аналіз рішень для React-застосунків

6

Рисунок Б.6 – Слайд № 6

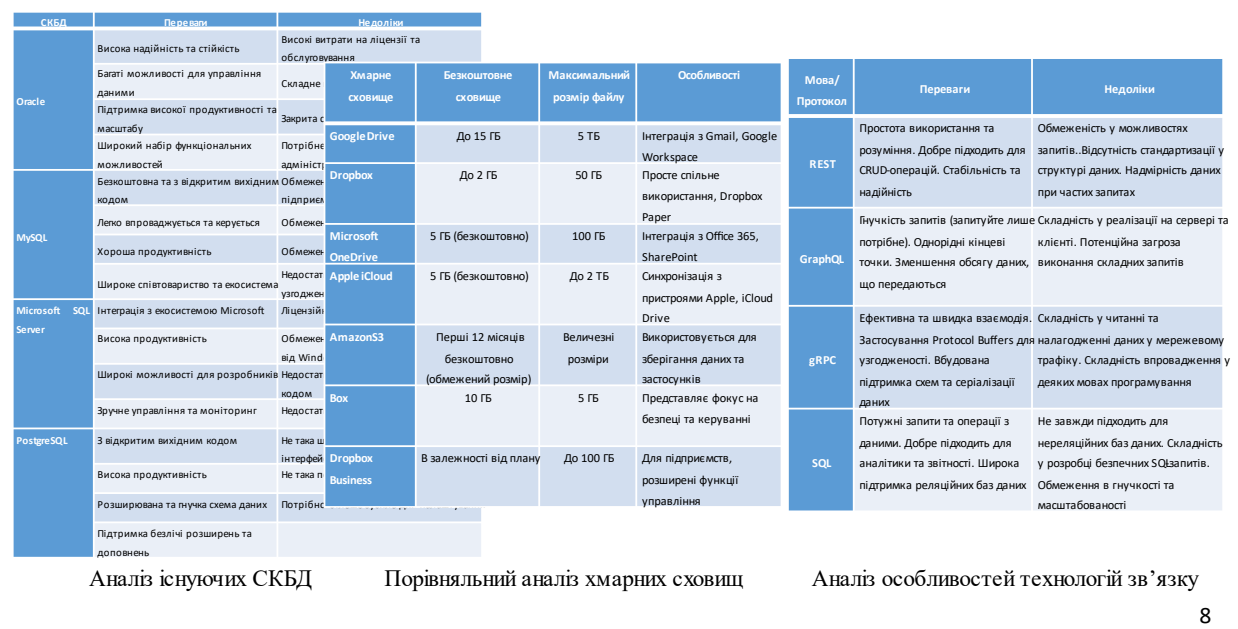
ВИБІР ІНСТРУМЕНТАРІЮ РОЗРОБКИ



7

Рисунок Б.7 – Слайд № 7

ВИБІР ІНСТРУМЕНТАРІЮ РОЗРОБКИ



8

Рисунок Б.8 – Слайд № 8

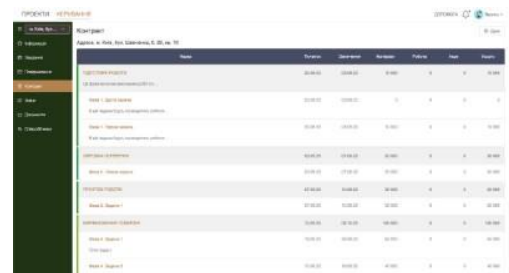
ІНТЕРФЕЙС СИСТЕМИ



Сторінка з завданнями



Сторінка чату



Сторінка контракту



Сторінка з інформацією про співробітників

11

Рисунок Б.11 – Слайд № 11

ВИСНОВКИ

- ✓ Розроблена спеціалізована SRM-система призначена для організації взаємодії замовника виконання ремонтних робіт у власній будівлі або квартирі та виконавця цих робіт.
- ✓ Вебзастосунок реалізовано на мові програмування JavaScript з використанням бібліотеки React та бекенду, який також розроблено на мові програмування JavaScript на платформі Node.js. База даних PostgreSQL.
- ✓ Наукова новизна роботи полягає в тому, що розроблено методику створення спеціалізованої SRM-системи, що базується на методології Agile та охоплює всі етапи створення надійного та масштабованого ПЗ, адаптованого до потреб бізнесу.
- ✓ Практична цінність роботи полягає в тому, що впровадження розробленого спеціалізованого ПЗ SRM-системи допоможе трансформувати процеси взаємодії постачальників та замовників послуг та досягти успіху в бізнесі.
- ✓ В подальшій роботі можливо розширити функціонал застосунку та удосконалити роботу з системою за рахунок додавання кнопки для редагування email та кнопки повторної відправки листа з запрошенням зареєструватися для нових користувачів системи.

12

Рисунок Б.12 – Слайд № 12