Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА ПРОГРАМНОГО МОДУЛЮ ДЛЯ ЕФЕКТИВНОГО УПРАВЛІННЯ ОФЕРАМИ ТА ДОГОВОРАМИ ЗАСОБАМИ DJANGO

Виконав: студент 4 курсу, групи КН-22мс
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Палій Н.С.
(прізвище та ініціали)
Керівник: доцент к. ф. – м.н.
Денисюк В.О.
(прізвище та ініціали)
"07" 06 2024 р.
Рецензент: к.т.н., доцент каф САІТ
Козачко О.М.
(прізвище та ініціали)
"07" 06 2024 р.

Допущено до захисту
Зав. кафедри Яровий А.А.
"07" 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 "Інформаційні технології"
Спеціальність – 122 "Комп'ютерні науки"
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
"07" 03 2024 р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Палію Назару Сергійовичу

1. Тема роботи: Розробка програмного модулю для ефективного управління оферами та договорами засобами Django.
Керівник роботи: асистент Денисюк Валерій Олександрович.
Затверджені наказом вищого навчального закладу "11" 03 2024 року № 80

2. Термін подання студентом роботи 27 05 2024

3. Вихідні дані до роботи :
Мова програмування Python;
Використання об'єктно-орієнтованого програмування;
Середовище розробки PyCharm 2023.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; аналіз предметної області управління оферами; розробка програмного модуля управління оферами; програмна реалізація web-ресурсу управління оферами; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
діаграма класів додатку; діаграма варіантів взаємодії користувача; діаграма активності додатку;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Денисюк В.О.		

7. Дата видачі завдання 07.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області управління оферами	06.05.24	09.05.24	
2	Розробка програмного модуля управління оферами	10.05.24	16.05.24	
3	Програмна реалізація web-ресурсу управління оферами	17.05.24	24.05.24	
4	Розробка інструкції користувача	25.05.24	28.05.24	
5	Аналіз отриманих результатів та формування висновків	29.05.24	02.06.24	
6	Оформлення бакалаврської кваліфікаційної роботи	03.06.24	06.06.24	

Студент (підпис) **Палій Н.С.** (прізвище та ініціали)

Керівник роботи (підпис) **Денисюк В** (прізвище та ініціа

АНОТАЦІЯ

Палій Н.С. розробка програмного модулю для ефективного управління оферами та договорами засобами Django. Бакалаврська дипломна робота складається з 89 сторінок формату А4, на яких є 25 рисунків, список використаних джерел містить 31 найменувань.

Метою даної бакалаврської кваліфікаційної роботи є розробка програмного модулю з використанням Django для ефективного управління оферами та договорами, що дозволить забезпечити зручний та надійний інструмент для управління документами і операціями в сфері укладання угод та управління контрактами.

Результатом дипломного дослідження є розроблений програмний модуль на Django для ефективного управління оферами та договорами, що реалізовані мовою програмування Python у програмному середовищі PyCharm 2023.

Ключові слова: Django, Python, управління оферами, управління договорами, ефективність, веб-ресурс.

ABSTRACT

Paliy N.S. Development of a software module for efficient management of offers and contracts using Django. Bachelor's thesis consists of 89 pages of A4 format, including 25 figures, and a list of 31 references.

The aim of this bachelor's qualification work is to develop a software module using Django for efficient management of offers and contracts, which will provide a convenient and reliable tool for document and operation management in the field of agreement negotiation and contract management.

The outcome of the diploma research is a developed software module on Django for efficient management of offers and contracts, implemented using the Python programming language in the PyCharm 2023 software environment.

Keywords: Django, Python, offer management, contract management, efficiency, web resource.

ЗМІСТ

ВСТУП

Актуальність досліджень. У сучасному світі, де бізнес-процеси стають все складнішими, а конкуренція на ринку надзвичайно велика, ефективне управління контрактами стає ключовою складовою успіху для підприємств. Зростання складності бізнес-процесів і постійний конкурентний тиск ставлять підприємства перед завданням швидко реагувати на зміни та вимоги ринку. У цьому контексті використання програмного модулю на базі Django для управління контрактами стає важливим чинником стратегії успіху.

Такий модуль допомагає оптимізувати процес створення, підписання та виконання контрактів. З його допомогою час, витрачений на створення контрактних документів, значно скорочується, а інтегровані засоби шаблонів та автоматизовані процеси допомагають уникнути помилок.

Крім того, модуль забезпечує можливість відстежувати виконання умов контрактів, автоматично генерувати звіти про виконання та аналізувати їх ефективність. Це дозволяє підприємствам уникнути ризиків порушення умов контрактів та забезпечити їхню правову впевненість.

Зростання конкуренції вимагає не лише укладання вигідних угод, але й їх ефективного виконання. Модуль Django надає засоби для відстеження виконання умов контрактів та генерації звітності, що спрощує контроль за виконанням зобов'язань та підтримує високу якість обслуговування.

Отже, управління контрактами за допомогою програмного модулю на базі Django є необхідним елементом стратегії підприємства, що дозволяє успішно функціонувати та забезпечувати конкурентоспроможність на ринку.

Метою дослідження є підвищення ефективності управління оферами та договорами за допомогою розробки програмного модуля на базі фреймворку Django.

Об’єкт дослідження є процес управління оферами та договорами в організаціях, який має потребу в автоматизації та оптимізації.

Предмет дослідження є розробка програмного модулю, який буде забезпечувати функціональність для створення, збереження, оновлення та видалення офер та договорів в системі.

Задачі дослідження:

1. розробка програмного модулю для створення офер та договорів.
2. реалізація функціоналу збереження та оновлення інформації про офери та договори.
3. проведення аналізу потреб користувачів для визначення оптимального інтерфейсу користувача.
4. реалізація адміністративного рівня доступу для керування оферами та договорами.
5. проведення тестування розробленого програмного модулю для перевірки його працездатності та надійності.
6. оптимізація роботи програмного модулю та вдосконалення його функціональності на основі отриманих результатів тестування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ УПРАВЛІННЯ ОФЕРАМИ

1.1 Актуальність теми

Ефективне управління контрактами у сучасному бізнес-середовищі є важливою складовою успішної стратегії для компаній будь-якого розміру і сфери діяльності. Швидке зростання складності бізнес-процесів і постійний конкурентний тиск ставлять підприємства перед завданням ефективно реагувати на зміни умов та потреби клієнтів та партнерів. У цьому контексті використання програмного модулю на базі Django для управління контрактами стає ключовим елементом стратегії успіху.

Програмний модуль на основі Django допомагає компаніям оптимізувати процес створення контрактів, забезпечуючи ефективне управління усіма етапами контрактних відносин, починаючи від створення, підписання і до виконання. З його допомогою витрачений час на створення контрактних документів значно зменшується, а інтегровані засоби шаблонів та автоматизовані процеси допомагають уникнути помилок.

Електронний підпис та інтеграція з електронними документообігами дозволяють проводити підписання контрактів онлайн швидко та безпечно, забезпечуючи зручність для всіх сторін угоди та прискорюючи процес укладання контрактів.

Програмний модуль також надає можливість відстежувати виконання умов контрактів, автоматично генерувати специфікації та звіти про виконання, а також нагадувати про важливі дати та події, що спрощує контроль за виконанням зобов'язань. Додатково, модуль забезпечує можливість проводити аналіз ефективності контрактів, виявляти та аналізувати ризики, а також генерувати звіти для внутрішнього та зовнішнього звітності.

Зростання конкуренції в сучасному бізнес-середовищі підсилює необхідність ефективного управління оферами та договорами. Програмний модуль на базі Django допомагає підприємствам швидко та точно реагувати на

запити клієнтів та укладати договори, забезпечуючи вигідні умови для обох сторін. Крім того, в умовах інтенсивної конкуренції важливо не лише укладати угоди, але і ефективно виконувати їх. Програмний модуль Django дозволяє відстежувати виконання умов контрактів та автоматично генерувати звіти про виконання, що спрощує контроль за виконанням зобов'язань та дозволяє підтримувати високу якість обслуговування та відносини з клієнтами.

Необхідність зменшення ризиків управління контрактами виникає з багатьох факторів, серед яких важливими є постійні зміни умов угод, конкурентний тиск на ринку та необхідність забезпечення правової впевненості. Використання програмного модулю на базі Django допомагає уникнути ризиків порушення умов контрактів, забезпечити правову впевненість та підвищити ефективність діяльності підприємств.

Значна роль у сучасному управлінні контрактами належить аналізу та звітності. Програмний модуль на базі Django надає компаніям можливість проводити детальний аналіз ефективності контрактів, виявляти потенційні ризики та приймати вчасні заходи для їх запобігання. Аналітичні інструменти дозволяють здійснювати глибокий розгляд ефективності різних контрактних стратегій та управлінських рішень, що допомагає підприємствам удосконалювати свою діяльність та досягати стратегічних цілей.

Ефективне управління контрактами набуває великої актуальності в сучасному світі, оскільки суспільство постійно зазнає впливу швидко змінюючихся технологій та ринкових умов. Підприємства повинні бути готові адаптуватися до нових вимог і змінювати умови контрактів відповідно до них. Крім того, конкурентна боротьба на ринку змушує компанії шукати шляхи для оптимізації своїх процесів, включаючи управління контрактами. Тільки ефективне управління контрактами допомагає підприємствам не лише укладати вигідні угоди, а й забезпечувати їх швидке виконання та збереження конкурентних позицій. Зокрема, в умовах цифрової економіки, де безпека даних стає ключовою проблемою, електронне управління контрактами

дозволяє забезпечити захист конфіденційної інформації від кіберзагроз та несанкціонованого доступу. Таким чином, управління контрактами стає необхідним елементом стратегії підприємств, що дозволяє їм успішно функціонувати в сучасних умовах та забезпечувати їх конкурентоспроможність.

Останнім аспектом, на який варто звернути увагу, є забезпечення безпеки та конфіденційності даних. Програмний модуль на базі Django включає в себе заходи забезпечення безпеки, такі як захист від несанкціонованого доступу, шифрування даних та контроль доступу до конфіденційної інформації. Це надає компаніям впевненість у тому, що їхні конфіденційні дані залишаються в безпеці та захищені від потенційних загроз.

1.2 Переваги Django в управлінні контрактами та оплатою

Управління контрактами є ключовим аспектом ефективного бізнесу в сучасному світі. Поява програмних рішень, спрямованих на автоматизацію цього процесу, відкриває нові можливості для підприємств у покращенні своєї діяльності та забезпеченні успішного виконання контрактних зобов'язань. У цьому розділі розглянуто переваги та потенціал програмного модулю на базі Django [1] для управління контрактами.

Переваги програмного модулю на базі Django включають широкий функціонал, гнучкість та надійність. Використання Django дозволяє швидко розробляти та розгортати програмне забезпечення, що є критичним аспектом у сучасних умовах динамічного бізнес-середовища. Крім того, Django забезпечує високий рівень безпеки, що є надзвичайно важливим при управлінні конфіденційною інформацією, яка зазвичай міститься в контрактах.

Додатково, програмний модуль на базі Django може бути інтегрований з іншими системами, такими як електронні документообіги [11] та CRM-системи, подано на рисунку 1.1. Це відкриває нові можливості для

автоматизації та оптимізації процесів управління контрактами, а також сприяє підвищенню ефективності та зниженню ризиків.

Рисунок 1.1 — CRM WorkFlows

У світі електронного бізнесу забезпечення зручної та безпечної можливості оплати є критично важливою для успішної та зручної торгівлі. Використання фреймворку Django відкриває перед розробниками широкі можливості для інтеграції різноманітних платіжних систем, що дозволяє користувачам безпечно та легко здійснювати оплату за товари чи послуги, подано на рисунку 1.2.

Один з переваг використання Django для реалізації оплати полягає в його гнучкості та розширюваності. Завдяки великому спектру плагінів та

бібліотек, розробники можуть інтегрувати різноманітні платіжні системи, такі як Stripe, PayPal, Braintree та інші, з мінімальними зусиллями.

Крім традиційних методів оплати, Django також надає можливість оплати кріптовалютою. З ростом популярності криптовалют, таких як Bitcoin, Ethereum та інші, інтеграція платіжних шлюзів для оплати криптовалютою стає все більш актуальною. Це відкриває нові перспективи для бізнесу, дозволяючи привертати клієнтів, які використовують криптовалюту, і розширюючи глобальний охоплення аудиторії.

Інтеграція онлайн оплати та оплати кріптовалютою з використанням Django дозволяє підприємствам підвищити рівень зручності для клієнтів, розширити ринкові можливості та бути на крок перед конкурентами. Крім того, це створює нові шляхи для розвитку бізнесу та приваблення нових клієнтів, забезпечуючи стійкий ріст та успіх у сучасному електронному бізнес-середовищі.

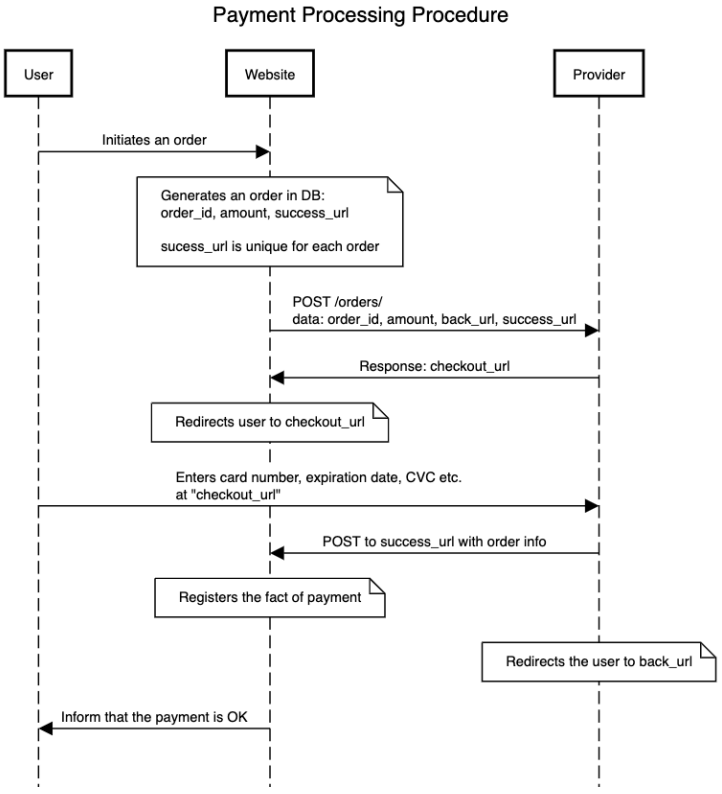

Рисунок 1.2 — Django Payment System

Інтеграція можливості онлайн оплати та оплати криптовалютою з використанням Django має численні переваги для підприємства та створює нові можливості для розвитку бізнесу в цілому:

— забезпечення онлайн оплати дозволяє підприємствам збільшити зручність для клієнтів та знизити бар'єри для здійснення покупок. Це може призвести до збільшення конверсії і, відповідно, до зростання обсягу продажів та прибутку.

— інтеграція оплати криптовалютою відкриває доступ до нових сегментів ринку, таких як люди, які активно користуються цифровими активами. Це дозволяє привернути нових клієнтів і розширити аудиторію своїх товарів або послуг.

— django має вбудовані засоби безпеки, що дозволяють забезпечити безпечну обробку та збереження конфіденційної інформації клієнтів під час транзакцій. Це сприяє підвищенню довіри споживачів та зменшенню ризику втрати даних.

— онлайн оплата та оплата криптовалютою відкривають доступ до продуктів або послуг для клієнтів з усього світу, що сприяє розширенню географії бізнесу та збільшенню його потенційного ринку.

— можливість оплати криптовалютою може привернути увагу технологічно освічених клієнтів, які віддають перевагу цифровим платіжним рішенням. Це дозволяє підприємству залучити нових клієнтів та створити позитивне враження про свою інноваційність та гнучкість.

Інтеграція онлайн платіжних систем та можливостей оплати криптовалютою через фреймворк Django може принести значний прибуток підприємству.

По-перше, підприємство може заробляти на комісійних від кожної транзакції, що здійснюється через платіжну систему. Це може бути як фіксований обсяг комісійного збору, так і відсоток від загальної суми транзакції [4].

Забезпечення зручної та безпечної онлайн оплати може сприяти зростанню обсягу продажів. Клієнти будуть більш схильні здійснювати покупки, оскільки вони матимуть можливість зручно оплачувати товари та послуги через Інтернет.

Крім того, інтеграція можливості оплати криптовалютою відкриває нові ринки для підприємства, подано на рисунку 1.3. Платежі криптовалютою можуть привернути увагу нових клієнтів, що активно використовують ці цифрові активи. Підприємство може також розвивати партнерські відносини з платіжними системами та фінансовими установами, що відкриває додаткові можливості для заробітку та спільних проєктів [9].

Загалом, інтеграція онлайн оплати та оплати криптовалютою з використанням Django створює різноманітні шляхи для підприємства отримати додатковий прибуток та розвиватися на ринку електронної торгівлі.

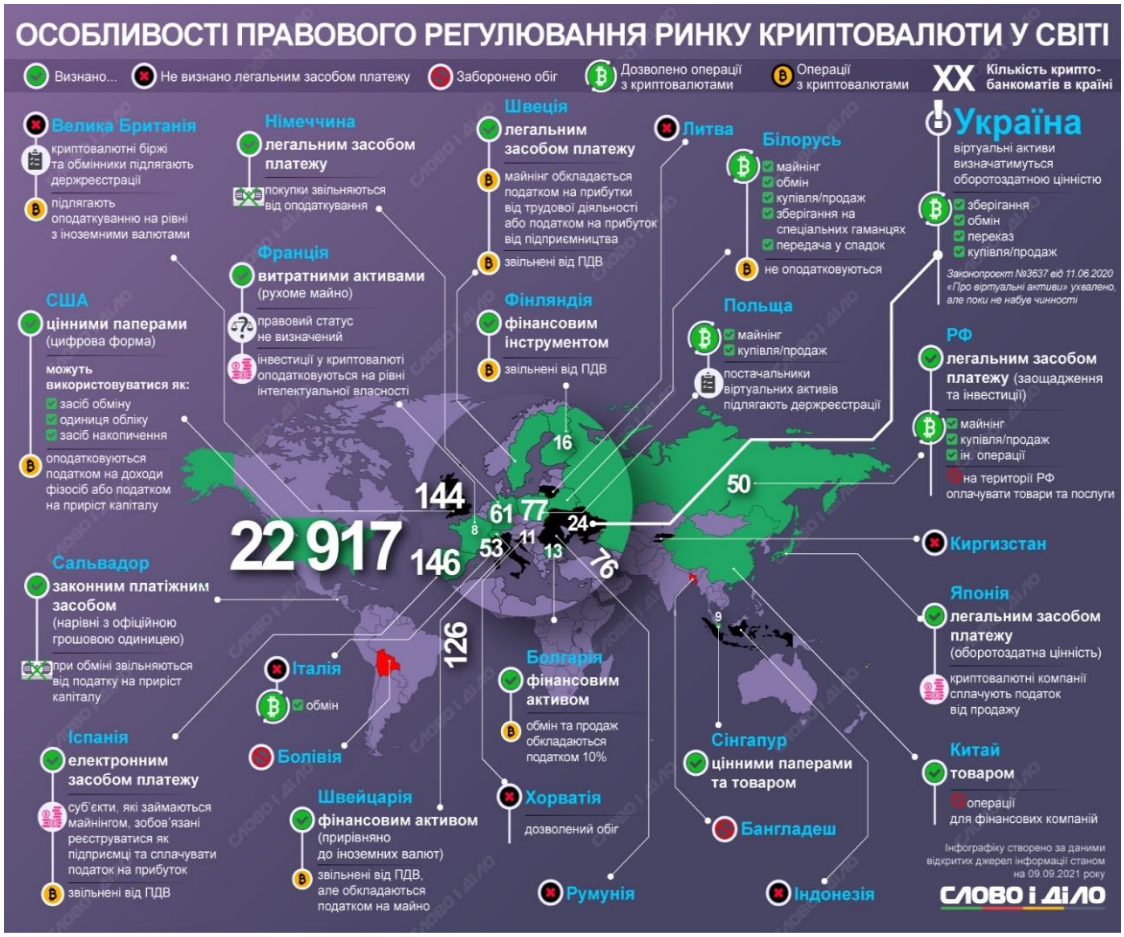

Рисунок 1.3 — Регулювання ринку криптовалюти

1.3 Основні можливості Django для управління контрактами

Функціональні можливості програмного модулю на базі Django для управління контрактами в сучасному бізнес-середовищі мають вирішальне значення для підприємств будь-якого розміру та галузі. Система надає широкий спектр інструментів, спрямованих на автоматизацію та оптимізацію всіх аспектів контрактних відносин, що робить її невід'ємною складовою успішної стратегії.

Перш за все, програмний модуль дозволяє легко створювати оферти та контракти, використовуючи готові шаблони або налаштування користувача. Це значно прискорює процес укладання угод та дозволяє індивідуалізувати умови контрактів з урахуванням потреб клієнтів.

Далі, модуль забезпечує можливість здійснення оплати контрактів через інтегровані платіжні системи або криптовалюту, що спрощує процедуру та забезпечує безпеку транзакцій.

Електронний підпис є ще однією важливою функцією модулю, яка дозволяє підписувати контракти онлайн, забезпечуючи швидкість та безпеку цього процесу, подану на рисунку 1.4. Це робить можливим уникнення затримок у укладанні угод та підвищує загальну ефективність бізнесу [11].

Важливим аспектом є також можливість відстежувати стан контрактів, автоматично генерувати звіти та нагадування про важливі дати та події. Це допомагає зберігати контроль над виконанням умов контрактів та попереджувати можливі порушення.

Аналіз ефективності контрактів, який здійснюється модулем на базі Django на основі зібраної статистики, допомагає виявляти потенційні ризики та можливості для оптимізації угод, що є ключовим для стратегічного управління бізнесом [12].

Нарешті, інтеграція з іншими системами, такими як CRM-системи, дозволяє підприємствам забезпечити максимальну ефективність управління контрактами та іншими аспектами бізнесу, уникаючи зайвих труднощів та зберігаючи цілісність даних. Такий підхід сприяє підвищенню продуктивності та конкурентоспроможності підприємства на ринку.

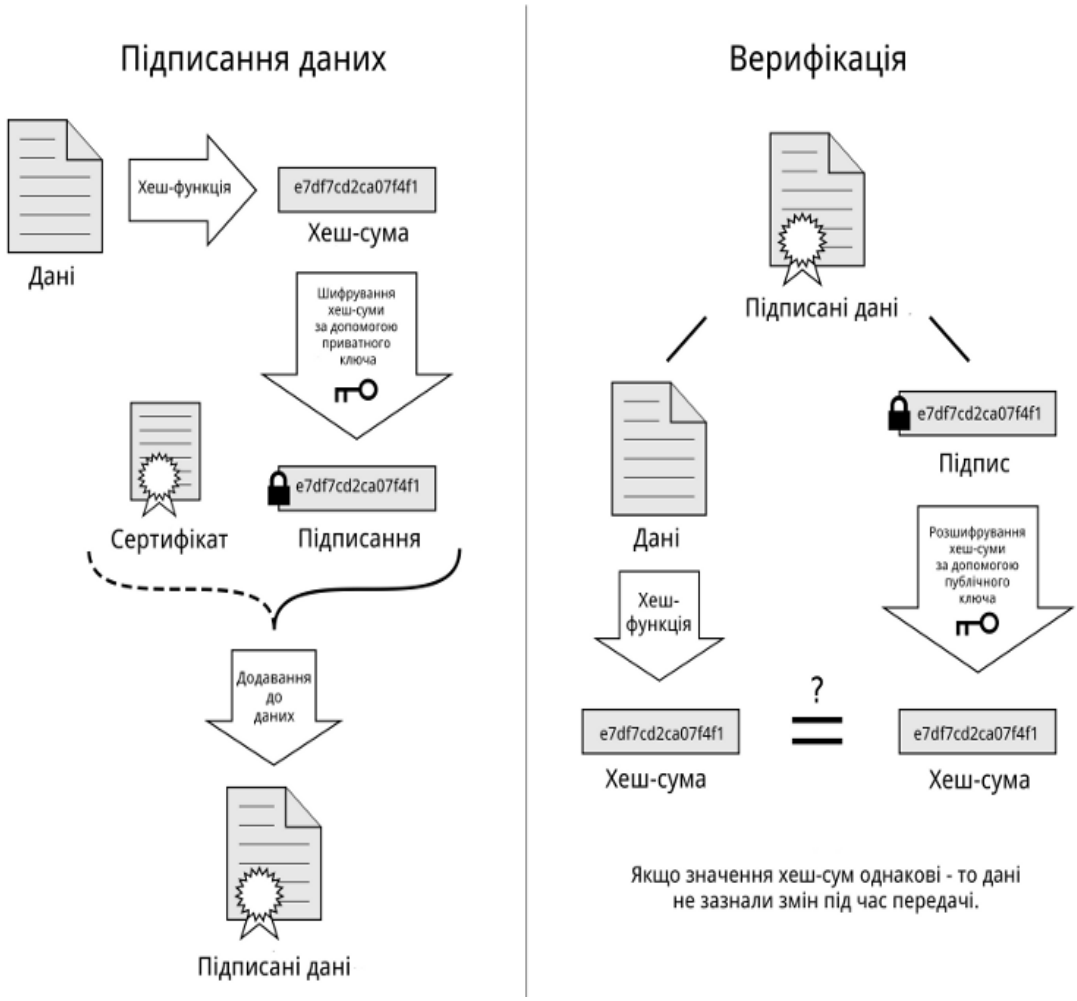

Рисунок 1.4 — Принцип роботи електронного підпису

Інтеграція програмного модулю на базі Django з платіжними системами для забезпечення онлайн оплати контрактів є важливим кроком у підвищенні ефективності та зручності управління фінансовими операціями. Подальше дослідження цього аспекту включає ретельний аналіз переваг,

доступних типів платіжних систем, а також детальне розглядання технічних аспектів інтеграції.

Однією з переваг інтеграції програмного модулю з платіжними системами є зручність для клієнтів[14].

Онлайн оплата контрактів дає їм можливість легко та швидко сплатити рахунки без потреби фізичного відвідування офісу чи використання складних банківських переказів. Це забезпечує клієнтам більшу свободу і комфорт у здійсненні фінансових операцій.

З іншого боку, підприємства отримують можливість отримувати платежі швидше та ефективніше.

Інтеграція з платіжними системами дозволяє автоматизувати процес обробки платежів, що сприяє збільшенню швидкості оплати та зменшенню часу, необхідного для обробки фінансових транзакцій. Це в свою чергу позитивно впливає на фінансову стабільність підприємства та його здатність реагувати на зміни в бізнес-середовищі.

Щодо доступних типів платіжних систем, розгляд може охоплювати широкий спектр варіантів, таких як Stripe, PayPal, Braintree, Square та інші [16], подано на рисунку 1.5.

Кожна з цих систем має свої особливості, переваги та обмеження, і вибір конкретної платіжної системи може залежати від потреб та стратегії підприємства.

Нарешті, технічні аспекти інтеграції включають у себе ряд кроків, починаючи від налаштування з'єднання з платіжними системами і закінчуючи відслідковуванням стану платежів та забезпеченням безпеки транзакцій. Це вимагає ретельного планування і виконання, а також уваги до деталей, щоб забезпечити надійність та безпеку фінансових операцій.

У підсумку, інтеграція програмного модулю Django з платіжними системами відкриває нові можливості для оптимізації процесу оплати контрактів та полегшення взаємодії з клієнтами, що сприяє підвищенню конкурентоспроможності та успішності підприємства.

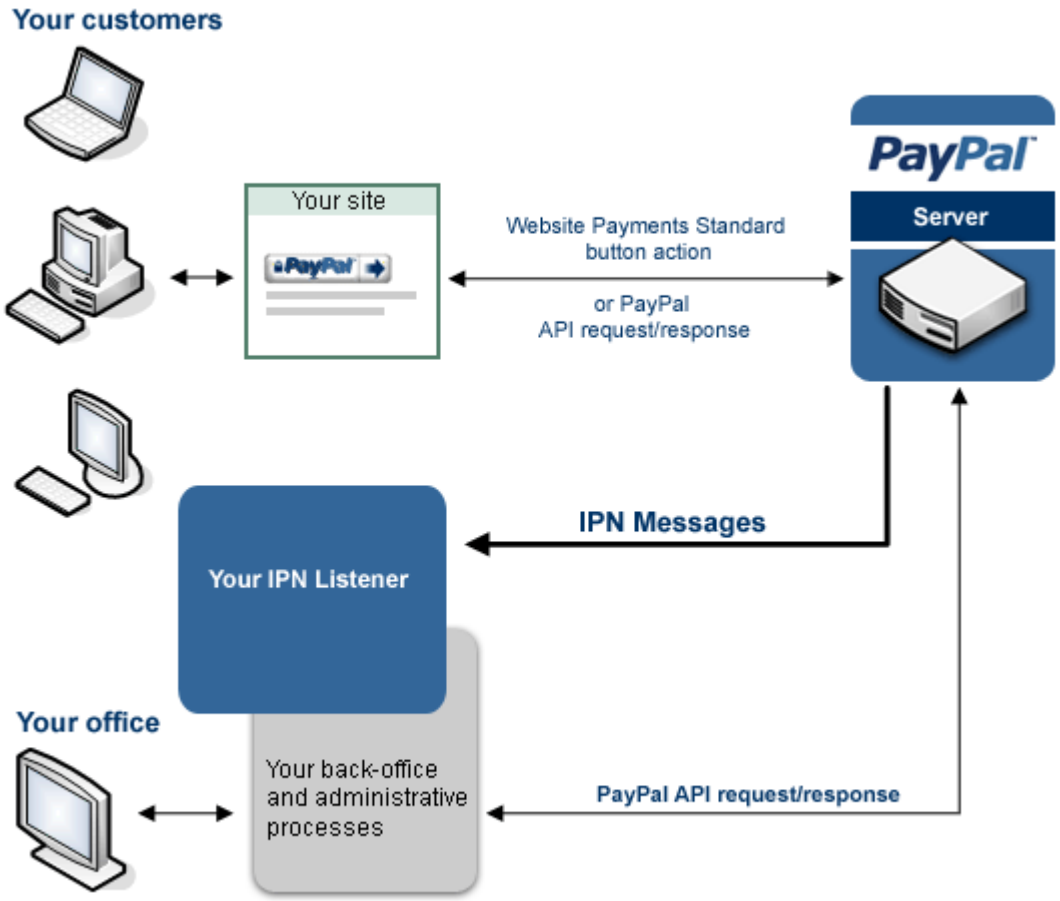

Рисунок 1.5 — Принцип роботи PayPal

Один з ключових функціональних елементів програмного модулю на базі Django - це автоматичне відстеження стану контрактів. Ця можливість надає користувачам зручний і ефективний спосіб контролювати виконання умов угод та важливі дати та події, пов'язані з кожним контрактом.

Модуль автоматично моніторить стан контрактів в реальному часі, сповіщаючи користувачів про будь-які зміни чи події, що стосуються їхніх угод. Наприклад, він може надсилати сповіщення про наближення строків виконання обов'язків, про затримки у виконанні чи про можливі ризики, пов'язані з угодою.

Ця функція допомагає користувачам бути завжди в курсі ситуації з контрактами, забезпечуючи вчасне реагування на будь-які зміни і максимальний контроль над виконанням умов угод.

Генерація звітів і документів:

— ще однією важливою можливістю програмного модулю на базі Django є генерація різноманітних звітів та документів про стан контрактів та їх виконання.

— ця функція дозволяє автоматично створювати різні види звітів і документів, що включають інформацію про статус кожного контракту, деталі його виконання, історію змін, аналітику по ефективності контрактів тощо. Такі звіти можуть бути корисними для внутрішнього аналізу, звітності перед керівництвом, аудиту та комунікації зі зацікавленими сторонами.

— генерація звітів і документів робить процес аналізу та звітності швидшим, простішим і більш ефективним, сприяючи зростанню продуктивності та якості управління контрактами.

1.4 Управління контрактами на сучасному ринку

На сучасному ринку оферів, який характеризується великою конкуренцією, відбуваються постійні зміни та динамічний розвиток. Високий попит на послуги та товари призводить до того, що компанії змушені конкурувати за увагу клієнтів, шукаючи способи вигідної пропозиції. Це відображається в постійному зростанні кількості постачальників, які намагаються привернути увагу споживачів і вийти на лідируючі позиції у своїй галузі.

Для успішного управління контрактами на такому конкурентному ринку необхідно проводити ретельний аналіз та враховувати особливості його функціонування. Постійне моніторинг та аналіз ринкових тенденцій, змін у

попиті та пропозиції, а також руху конкурентів є ключовими елементами стратегії управління контрактами [7].

Крім того, важливо враховувати технологічні інновації, які активно впроваджуються в галузі управління контрактами. Нові технології можуть значно полегшити та оптимізувати процес укладання та виконання контрактів, забезпечуючи компаніям конкурентні переваги. Впровадження цифрових рішень, використання інтелектуальних систем управління, а також автоматизація процесів можуть зробити управління контрактами більш ефективним і продуктивним. Тому важливо постійно оновлювати технологічну базу та бути готовим до інноваційних змін на ринку.

Фреймворк Django визнаний як один із найпотужніших інструментів для розробки веб-додатків, ідеально підходить для управління контрактами. Цей фреймворк забезпечує розробникам зручні інструменти, які дозволяють створювати високоякісні веб-додатки швидко та ефективно. Однією з його основних переваг є гнучкість, яка дозволяє легко адаптувати його до різних потреб сучасного бізнесу.

Основні переваги використання Django полягають у зручності розробки. Він має чистий та легкий синтаксис, що спрощує процес програмування і дозволяє розробникам швидко створювати потужні веб-додатки. Крім того, Django має вбудовані засоби безпеки, що дозволяють захистити додатки від різних видів атак, таких як SQL-ін'єкції, перехоплення сесій та інші загрози [9].

Ще однією перевагою Django є його розширюваність. Фреймворк має велику кількість розширень та сторонніх бібліотек, які допомагають розширити його функціонал та використовувати нові можливості. Крім того, Django легко інтегрується з різними системами керування базами даних, що дозволяє використовувати різні типи СКБД залежно від потреб проєкту [2].

Аналіз ринку оферів у поєднанні з використанням фреймворку Django відкриває можливості для створення ефективних програмних рішень, які відповідають потребам клієнтів і забезпечують конкурентні переваги

компаніям. Це дозволяє розробникам не лише врахувати поточні потреби ринку, а й передбачити майбутні тенденції та вимоги, що є ключовим для успіху в сучасному бізнес-середовищі.

Зважаючи на постійний розвиток технологій та зростання важливості цифровизації у всіх аспектах бізнесу, ідея онлайн оферів стала логічним кроком для багатьох компаній. Традиційний процес укладання контрактів часто пов'язаний з багатьма труднощами, такими як велика кількість паперових документів, необхідність фізичної присутності сторін, складнощі з експрес-доставкою та підписанням. Онлайн офери вирішують ці проблеми, роблячи весь процес більш зручним і швидким [4].

Ідея онлайн оферів з'явилася з потреби уникнути зайвих витрат часу та ресурсів, пов'язаних із традиційним укладанням контрактів. Використання інтернет-платформ для укладання угод дозволяє сторонам швидко обмінюватися пропозиціями, вносити зміни та підписувати документи з будь-якої точки світу, що робить процес більш гнучким та мобільним.

При цьому, переваги онлайн оферів ще більше підсилюються завдяки застосуванню програмного модулю на базі Django. Цей фреймворк дозволяє не лише створювати зручні веб-додатки, але і забезпечує вбудовані засоби безпеки, що гарантують конфіденційність та надійність укладених угод. Крім того, Django легко інтегрується з різними електронними системами підпису, що робить можливим проведення онлайн підписання контрактів безпосередньо на веб-платформі [12].

Такий підхід не лише спрощує процес укладання контрактів, а й забезпечує більшу прозорість та контроль за угодами, оскільки всі етапи проходять через цифрову платформу, де можуть бути збережені всі дані та історія взаємодії між сторонами. Таким чином, використання онлайн оферів у поєднанні з фреймворком Django створює ефективний і безпечний механізм управління контрактами, який відповідає сучасним вимогам бізнесу.

Під час дослідження було виявлено, що сучасний ринок оферів є вельми динамічним та надзвичайно конкурентним середовищем. На цьому

ринку панує великий попит на різноманітні послуги та товари, що спонукає компанії постійно шукати способи вигідних пропозицій для залучення уваги клієнтів. Це зумовлює постійне зростання кількості постачальників і високий рівень конкуренції. Щоб успішно керувати контрактами в такому середовищі, необхідно постійно аналізувати ринкові тенденції та реагувати на зміни у попиті та пропозиції. Такий підхід дозволяє компаніям залишатися в центрі уваги клієнтів та ефективно конкурувати на ринку [13].

Впровадження онлайн оферів у поєднанні з фреймворком Django виявляється неабияк перспективним кроком для компаній. Це дозволяє забезпечити ефективне управління контрактами, використовуючи передові технології та забезпечуючи безпеку та надійність угод. Фреймворк Django забезпечує розробникам зручні інструменти для створення високоякісних веб-додатків швидко та ефективно, що відповідає сучасним вимогам бізнесу.

У своєму загальному висновку варто відзначити, що розробка програмного рішення для управління контрактами з використанням онлайн оферів та фреймворку Django є важливим кроком у напрямку оптимізації бізнес-процесів та підвищення конкурентоспроможності компаній. Це дозволяє не лише забезпечити більшу ефективність управління контрактами, а й створити конкурентні переваги на ринку завдяки зручності, безпеці та інноваційній природі цифрових рішень. Такий підхід дозволяє компаніям залишатися на передній лінії ринку та забезпечувати успішний розвиток у сучасному бізнес-середовищі.

1.5 Висновки

Управління контрактами та ефективна обробка платежів не лише стали ключовими елементами в сучасному бізнесі, але й набули критичної важливості в умовах постійних змін і загострення конкуренції на ринку. Використання програмного модулю на базі Django виявляється вирішальним

для оптимізації цих процесів, надаючи підприємствам необхідний функціонал, гнучкість та надійність.

Аналізуючи мету та завдання дослідження, можна зрозуміти, що Django відкриває унікальні можливості для підприємств у сфері управління контрактами. Його гнучкість дозволяє швидко створювати та розгортати програмне забезпечення, а високий рівень безпеки гарантує захист конфіденційної інформації. Інтеграція з іншими системами, такими як електронні документообіги та CRM-системи, розширює можливості оптимізації та автоматизації процесів управління контрактами.

Розділ, присвячений перевагам програмного модулю на базі Django, виокремлює його значущість для підприємств. Широкий функціонал, гнучкість та надійність Django роблять його привабливим вибором для розробки програмного забезпечення для управління контрактами. Інтеграція з платіжними системами, включаючи можливість оплати криптовалютою, дозволяє підприємствам розширити свої можливості та залучити нових клієнтів, що сприяє зростанню та успіху.

Завершальний висновок підкреслює важливість використання Django для управління контрактами та інтеграції платіжних систем, підкреслюючи його переваги і потенціал для підприємств. Враховуючи потреби сучасного бізнесу у швидкості, безпеці та ефективності, Django виявляється відмінним вибором для оптимізації цих важливих процесів, допомагаючи підприємствам досягати успіху та залишатися конкурентоспроможними.

2 РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ УПРАВЛІННЯ ОФЕРАМИ

2.1 Обґрунтування вибору технологій та інструментів для розробки

При розробці програмного модуля управління контрактами важливо обрати оптимальні технології та інструменти, які забезпечать ефективну та надійну реалізацію функціоналу, а також зручний інтерфейс користувача.

Одним із найпопулярніших та найпотужніших фреймворків для веб-розробки є Django, який використовується для створення високоякісних веб-додатків мовою програмування Python. Django відмінно підходить для розробки програмного модуля управління контрактами з огляду на свою швидкість розробки та гнучкість. За допомогою Django можна швидко розробити веб-додаток з різноманітним функціоналом, від створення, редагування та видалення контрактів до відстеження їхнього статусу та звітності.

Крім Django, для розробки інтерфейсу користувача використовуються стандартні веб-технології, такі як HTML, CSS і JavaScript. Ці технології дозволяють створювати зручний та привабливий інтерфейс для користувачів, забезпечуючи при цьому високий рівень зручності використання [16].

Додатково, можна використовувати фронтенд-фреймворки, такі як React або Angular, для створення більш динамічних та інтерактивних інтерфейсів. Ці фреймворки дозволяють легко керувати взаємодією користувача з додатком та забезпечують високий рівень користувацького досвіду.

Для забезпечення інтеграції з іншими системами може бути використаний REST API, який дозволяє обмінюватися даними з іншими додатками та сервісами. Це особливо корисно, якщо необхідно здійснювати обмін даними з системами електронного документообігу або CRM-системами.

Нарешті, використання системи контролю версій, такої як Git, допоможе забезпечити ефективне керування розробкою програмного модуля та спільну роботу розробників над проєктом. Завдяки системі контролю версій можна легко відстежувати зміни в коді, вносити правки та співпрацювати з іншими учасниками проєкту.

Пайтон та Django обираються для розробки програмного модуля управління контрактами через їхні численні переваги та відповідність потребам проєкту.

Пайтон є однією з найпопулярніших мов програмування у світі, завдяки своїй простоті, зрозумілій синтаксису та великій кількості бібліотек і фреймворків. Він має широке застосування в різних галузях, включаючи веб-розробку, аналітику даних, штучний інтелект та інше. Вибір Пайтона для розробки програмного модуля управління контрактами забезпечує швидку реалізацію проєкту та зручне відладження коду [17].

Django, у свою чергу, є потужним фреймворком для розробки веб-додатків мовою програмування Python. Він надає широкий функціонал та вбудовані інструменти для швидкого створення веб-додатків, включаючи аутентифікацію, авторизацію, роботу з базами даних, адміністративний інтерфейс та інше. Основні переваги Django включають:

— Швидкість розробки: Django пропонує зручні інструменти та готові компоненти, що дозволяють швидко розробляти функціонал веб-додатку без необхідності писати код з нуля.

— Гнучкість: Django надає можливість налаштування та розширення, що дозволяє адаптувати фреймворк під конкретні потреби проєкту.

— Безпека: Django має вбудовані засоби захисту від таких загроз, як SQL-ін'єкції, XSS атаки та інші типи вразливостей.

— Адміністративний інтерфейс: Django має вбудований інструмент для адміністрування веб-додатків, що дозволяє легко керувати даними та користувачами без необхідності написання додаткового коду.

Спільнота та документація: Django має активну спільноту користувачів та добре документований код, що полегшує роботу з фреймворком та вирішення можливих проблем [18].

— Бекенд на Django є одним з найпоширеніших варіантів для створення потужних веб-додатків та API. Django надає ряд інструментів і можливостей для швидкої розробки та розгортання бекенд-частини веб-додатків. Ось деякі ключові аспекти бекенду на Django та Django REST Framework (DRF):

— Моделі та ORM: Django надає зручний спосіб визначення моделей даних для бази даних за допомогою ORM (Object-Relational Mapping). Це дозволяє розробникам працювати з базою даних на рівні об'єктів Python, що спрощує взаємодію з даними та зменшує кількість коду [14].

— Відображення (Views): Django дозволяє визначати відображення, які обробляють HTTP-запити та повертають HTTP-відповіді. Відображення можуть бути функціями або класами, і вони можуть працювати з моделями даних, обробляти запити користувачів та повертати відповіді.

— URL-адреси: Django дозволяє визначати шаблони URL-адрес для маршрутизації запитів до відповідних відображень. Це дозволяє структурувати ваш веб-додаток та робити його більш легко зрозумілим.

— Шаблони (Templates): Django надає систему шаблонів для рендерингу HTML-сторінок на основі даних з бекенду. Це дозволяє розділити логіку бекенду та фронтенду, полегшуючи розробку та обслуговування веб-додатків.

— Аутентифікація та авторизація: Django має вбудовану систему аутентифікації та авторизації, яка дозволяє керувати доступом користувачів до різних частин веб-додатку.

Django REST Framework (DRF): DRF - це потужний інструмент для створення веб-сервісів та API на базі Django. Він надає ряд додаткових

функцій, таких як автоматична документація API, серіалізатори для перетворення даних у JSON або інші формати, перевірка даних та багато іншого.

Розширення та налаштування: Django дозволяє легко розширювати та налаштовувати ваш бекенд відповідно до ваших потреб. Існують багато розширень та сторонніх бібліотек, які допоможуть вам розширити функціонал веб-додатку.

При розробці веб-додатків на Django, важливою частиною є його взаємодія з фронтендом. Для цього часто використовується сучасний JavaScript фреймворк, такий як React.js, який дозволяє розробникам створювати динамічні та інтерактивні інтерфейси, подано на рисунку 2.1.

React.js - це потужний JavaScript фреймворк для створення користувацьких інтерфейсів, який надає ряд переваг:

— Компонентна архітектура: React сприяє побудові додатків за допомогою невеликих, незалежних компонентів, які можна легко перевикористовувати та комбінувати. Це полегшує розробку та підтримку коду, особливо великих проєктів.

— Віртуальний DOM: React використовує віртуальний DOM для оптимізації процесу оновлення інтерфейсу. Це дозволяє зменшити кількість операцій маніпулювання реальним DOM, що пришвидшує роботу додатків та зменшує навантаження на браузер.

— Односторінкові додатки (SPA): React ідеально підходить для створення SPA, де весь контент завантажується разом з першою загрузкою сторінки, а подальша навігація відбувається без перезавантаження сторінки. Це забезпечує швидку та плавну роботу додатку [19].

— Реактивність: React дозволяє створювати реактивні інтерфейси, які миттєво відображають зміни в даних без необхідності оновлення сторінки. Це дозволяє створювати більш інтерактивні та користувацько-орієнтовані додатки.

Взаємодія між бекендом на Django та фронтендом на React може відбуватися за допомогою HTTP запитів, зазвичай через REST або GraphQL API. Бекенд на Django надає API, яке дозволяє фронтенду отримувати та відправляти дані серверу. Наприклад, фронтенд може виконувати запити GET, POST, PUT або DELETE для отримання, створення, оновлення та видалення ресурсів на сервері, подано на рисунку 2.1.

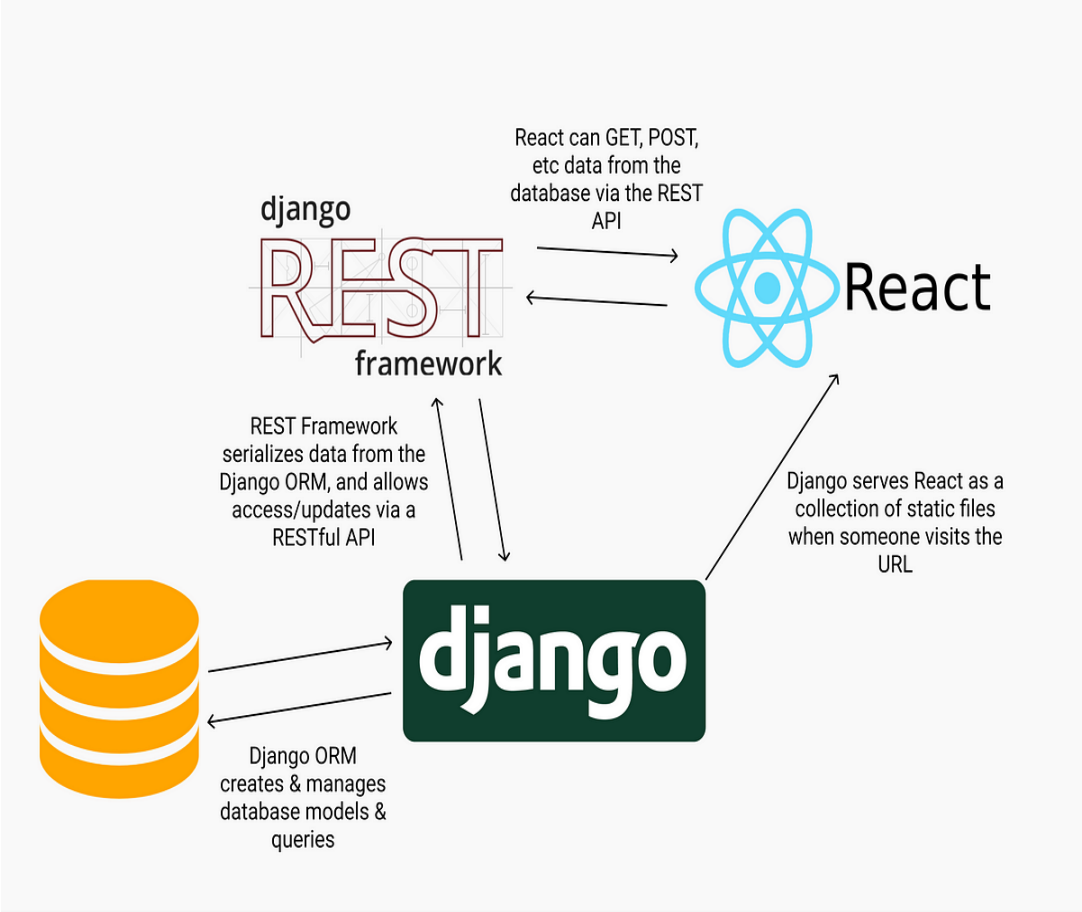

Рисунок 2.1 — Взаємодія Django з React

При цьому, Django REST Framework (DRF) дозволяє швидко створювати REST API на базі Django, надаючи зручний інтерфейс для визначення серіалізаторів, відображень та маршрутів для API.

PyCharm та WebStorm - це інтегровані середовища розробки (IDE), які надають розширені можливості для створення веб-додатків з використанням фреймворків Django та React [20].

PyCharm, подано на рисунку 2.2:

— Підтримка Django: PyCharm надає повноцінну підтримку Django, подано на рисунку 2.2, що дозволяє розробникам створювати бекенд-додатки швидше та ефективніше. Він має інтегровані інструменти для автоматичного створення та налаштування проєктів Django, підтримки шаблонів, моделей та контролерів, а також вбудований сервер для локального розгортання та налагодження додатків [21].

— Рефакторинг та аналіз коду: PyCharm має широкий набір інструментів для рефакторингу коду, що дозволяє покращувати структуру та читабельність програмного коду. Крім того, він надає можливості аналізу коду для виявлення потенційних проблем та помилок, що допомагає забезпечити високу якість програмного забезпечення[18].

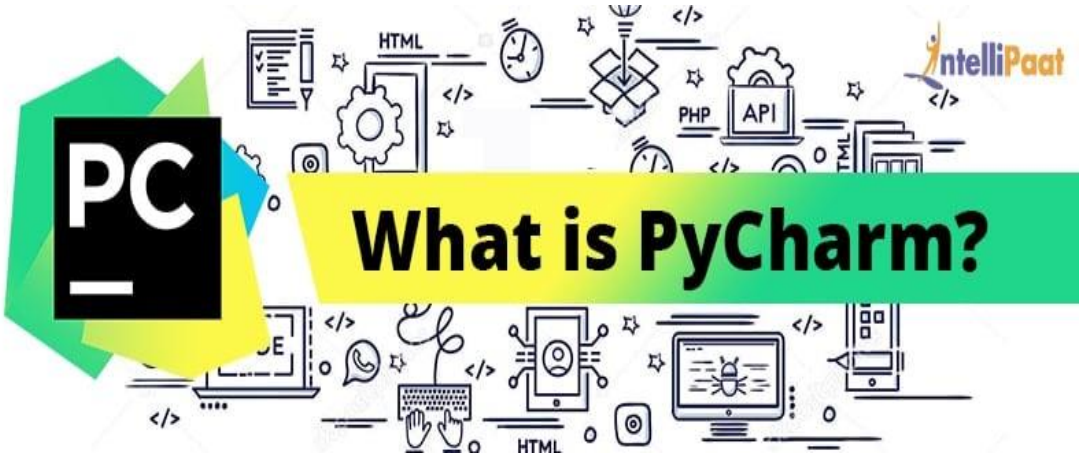

Рисунок 2.2 — PyCharm

WebStorm, подано на рисунку 2.3:

— Підтримка React: WebStorm володіє вбудованою підтримкою для розробки веб-інтерфейсів на React, що робить процес розробки фронтенду більш зручним та продуктивним. Він надає автодоповнення коду,

підсвічування синтаксису JSX, вбудовані інструменти для роботи з компонентами та багато іншого, подано на рисунку 2.3.

— Інструменти для розробки веб-додатків: WebStorm має ряд корисних інструментів, які допомагають розробникам створювати високоякісні веб-додатки. Серед них - вбудований веб-сервер для локального тестування, інтеграція з системами контролю версій (наприклад, Git), інструменти для автоматичного перевірки та тестування коду, а також засоби для налагодження веб-додатків.

Рисунок 2.3 — WebStorm

З використанням PyCharm та WebStorm, розробники можуть максимально продуктивно працювати з фреймворками Django та React. Ці IDE надають розширені можливості для розробки як бекенду, так і фронтенду веб-додатків, забезпечуючи зручність, швидкість та надійність в процесі розробки.

Вибір PyCharm та WebStorm для роботи з фреймворками Django та React у цій роботі обґрунтовується їхніми перевагами та властивостями, які роблять їх ідеальними інструментами для розробки веб-додатків:

Інтегрованість з фреймворками: Обидві IDE надають повноцінну підтримку для фреймворків Django та React, що спрощує розробку бекенду та фронтенду веб-додатків[17].

— Розширені можливості рефакторингу та аналізу коду: Інструменти PyCharm та WebStorm допомагають покращувати структуру та читабельність програмного коду, а також виявляти потенційні проблеми та помилки, що підвищує якість розробленого програмного забезпечення.

— Зручність використання: Інтерфейси PyCharm та WebStorm інтуїтивно зрозумілі та дружні до користувача, що дозволяє розробникам швидко орієнтуватися та працювати з великими проєктами [18].

— Широкий спектр інструментів та плагінів: Обидві IDE мають велику кількість розширень та плагінів, які дозволяють розширити їх функціональність та адаптувати під потреби конкретного проєкту.

— Інтеграція з системами контролю версій: PyCharm та WebStorm підтримують інтеграцію з популярними системами контролю версій, такими як Git, що дозволяє зручно керувати версіями програмного коду та спільно працювати над проєктом у команді.

— Підтримка різних мов програмування: PyCharm підтримує мову програмування Python, а WebStorm - JavaScript, TypeScript, HTML, CSS та інші, що дозволяє розробляти як бекенд, так і фронтенд веб-додатків у єдиному середовищі [20].

Нюанси використання можуть включати в себе необхідність налаштування окремих інструментів для певних задач, а також можливі проблеми з продуктивністю при роботі з великими проєктами. Однак, з правильним налаштуванням та вивченням функціоналу ці нюанси можна

легко подолати, забезпечуючи комфортну та продуктивну роботу з фреймворками Django та React.

2.2 Розробка діаграми класів

Розробка діаграми класів в програмній інженерії можна порівняти з будівельним кресленням або картою міста перед подорожжю.

Ця діаграма виступає як план, який вказує, як всі частини програми пов'язані між собою.

На початку процесу створення програмного забезпечення, коли розробники мають багато ідей і концепцій, діаграма класів слугує важливим інструментом для організації цих ідей у логічну структуру.

Подібно до того, як архітектор малює плани будинку перед будівництвом, розробники використовують діаграму класів, щоб визначити, які класи і як вони будуть взаємодіяти між собою.

Для кращого зрозуміння, уявімо, що ми розробляємо програму для інтернет-магазину.

На діаграмі класів ми можемо мати класи для продуктів, користувачів, замовлень та платежів.

Кожен з цих класів матиме взаємозв'язки з іншими, наприклад, клас "замовлення" буде мати посилання на клас "користувач", який розміщує замовлення, і на клас "продукт", який входить до складу замовлення.

Діаграма класів допомагає розробникам зрозуміти структуру програми і виявити можливі проблеми архітектури ще до того, як почнеться активний процес програмування.

Наприклад, вона допомагає виявити, які класи можуть бути занадто сильно залежними один від одного або які класи можуть бути перевантаженими функціональністю [19].

Основні причини важливості розробки діаграми класів включають:

— Візуальне представлення структури системи: Діаграма класів надає графічне представлення класів програми та їх взаємозв'язків, що дозволяє розробникам швидше та зрозуміліше аналізувати архітектурні аспекти проєкту.

— Збільшення зрозуміння коду: Розглядання діаграми класів допомагає розробникам краще зрозуміти взаємозв'язки між класами, їх функціональність та взаємодію.

— Виявлення слабких місць та покращення архітектури: Шляхом аналізу діаграми класів можна виявити слабкі місця в архітектурі програми та вчасно вжити заходів для їх виправлення, що сприяє покращенню якості та продуктивності проєкту.

— Підтримка комунікації в команді розробників: Діаграма класів служить ефективним інструментом для спілкування між членами команди розробників, оскільки вона дозволяє лаконічно та зрозуміло висловлювати архітектурні концепції та ідеї.

— Покращення підтримки та розвитку програми: Розробка діаграми класів створює базу для подальшого розвитку та підтримки програми, спрощуючи процес розуміння та модифікації існуючого коду.

У нашому випадку, ми можемо побудувати діаграму класів, що відображатиме основні класи та їх взаємозв'язки в системі для управління контрактами та пропозиціями.

Діаграма може включати класи такі як Contract, Offer, UserFSM, Organization, Document, Payment і так далі.

Кожен з цих класів буде мати свої атрибути та методи, а також взаємозв'язки з іншими класами.

Наприклад, клас Contract може мати зв'язки з класами Offer, UserFSM та Document, оскільки кожен контракт пов'язаний з конкретною пропозицією, користувачем та документом.

Діаграма класів також може відображати різні статуси контрактів та пропозицій, такі як "pending", "accepted", "rejected" і так далі [21].

Крім того, наша діаграма може включати взаємозв'язки між користувачами та організаціями, оскільки користувачі можуть бути пов'язані з певними організаціями, а також між пропозиціями та контрактами, оскільки пропозиції можуть бути прийняті та перетворені на контракти.

Коли ми говоримо про розробку діаграми класів, важливо врахувати, що це не просто статичне зображення структури програми.

Це інструмент, який допомагає розробникам отримати глибше розуміння архітектури програми, виявити потенційні проблеми та взаємозв'язки між класами.

Наприклад, розглянемо клас Offer у нашій системі. Цей клас представляє пропозиції, які користувачі можуть створювати для укладення контрактів.

У діаграмі класів ми можемо побачити, як Offer пов'язаний з класом UserFSM через поле user, яке вказує на користувача, що створив пропозицію. Також ми можемо побачити зв'язок між Offer та Contract, оскільки пропозиція може бути прийнята та перетворена на контракт [25].

Інший приклад - це взаємозв'язок між класами Organization та UserFSM. У системі організації можуть мати співробітників, які представлені класом UserFSM.

Тому в діаграмі класів ми можемо побачити зв'язок багато-до-багатьох між цими класами через поле users, що вказує на користувачів, що належать до певної організації, а також зворотній зв'язок через поле organization в класі UserFSM, що вказує на організацію, якій належить користувач, подано на рисунку 2.4.

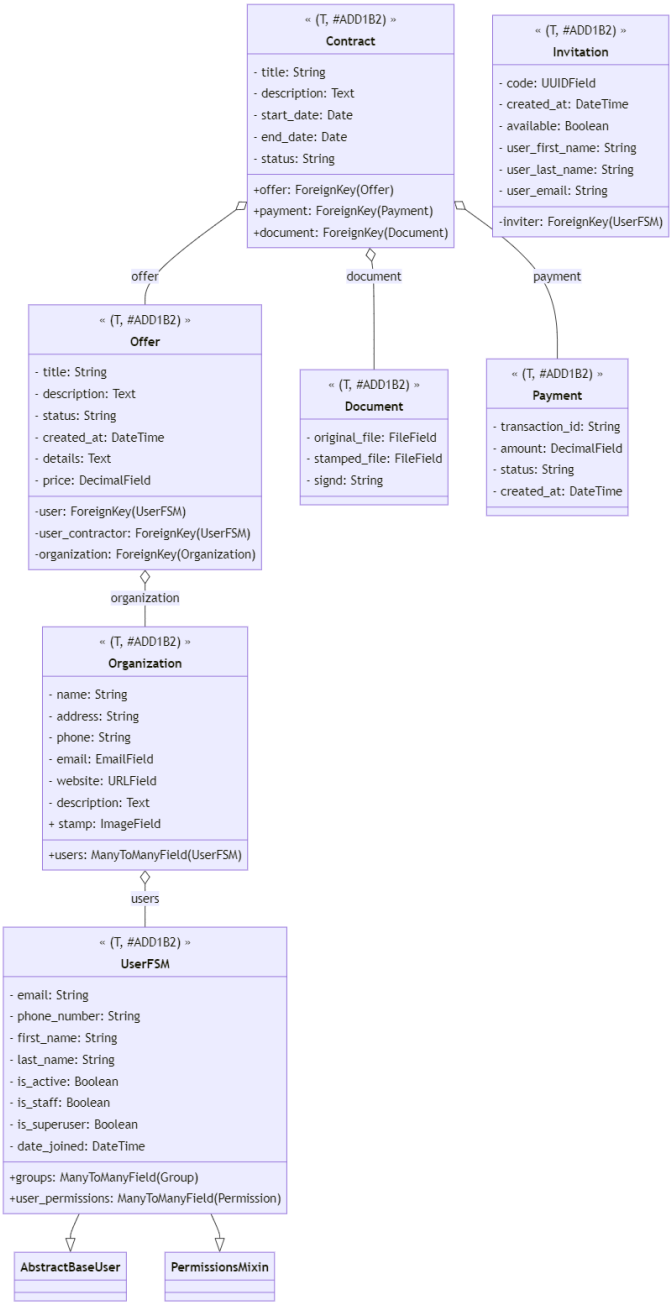

Рисунок 2.4 — Діаграма класів додатку

Клас UserFSM:

Цей клас відповідає за представлення користувачів у системі. Основні поля класу описують основну інформацію про користувача, таку як електронна адреса, номер телефону, ім'я, прізвище, статус активності тощо. Крім того, є поля, що відповідають за права користувача (наприклад,

is_superuser), дату реєстрації (date_joined) та зв'язки з іншими моделями, такими як групи користувачів та дозволи.

Діаграма класів:

На діаграмі класів, клас UserFSM представлений прямокутником з ім'ям "UserFSM", який має список полів, які описують основну інформацію про користувача. У нашому випадку, поля позначені за допомогою приватного модифікатора доступу (-) та описані у текстовому вигляді. Крім того, на діаграмі показані зв'язки класу UserFSM з іншими класами, такими як AbstractBaseUser та PermissionsMixin, які відображають успадкування цього класу від базових класів користувачів Django.

Робота класу:

Клас UserFSM відіграє важливу роль у системі, оскільки він є основним представленням користувачів. Він забезпечує зберігання та управління основною інформацією про користувачів, такою як їх особисті дані та статус активності. Крім того, за допомогою зв'язків з іншими класами, такими як групи користувачів та дозволи, клас UserFSM забезпечує реалізацію системи аутентифікації та авторизації користувачів. Таким чином, цей клас відіграє важливу роль у забезпеченні безпеки та правильної роботи системи.

Модель запрошення (Invitation):

Модель Invitation відображає інформацію про запрошення користувачів до системи. Кожен об'єкт цієї моделі містить дані про конкретне запрошення, включаючи ідентифікатор запрошення, дату створення, код запрошення, інформацію про того, хто створив запрошення, а також дані про запрошеного користувача (якщо він є) [24].

Зв'язок з користувачем (UserFSM):

У моделі Invitation є зв'язок з моделлю користувачів (UserFSM) через поле inviter, яке є зовнішнім ключем ForeignKey. Це поле посилається на конкретного користувача (inviter), який створив запрошення. Таким чином, кожне запрошення пов'язане з користувачем, який його створив [22].

Роль моделі Invitation:

Модель Invitation використовується для створення запрошень користувачам до системи. Це може бути корисно в ситуаціях, коли адміністратор системи хоче надати доступ до системи новим користувачам за допомогою запрошень. Це також може бути корисно для відстеження, хто і коли надіслав запрошення, а також для відстеження статусу запрошення (наприклад, чи активне воно чи вже використане).

Зв'язок з користувачем:

Зв'язок з моделлю користувачів (UserFSM) дозволяє встановити, хто створив запрошення. Це важливо для відстеження власника запрошення та можливості взаємодії з ним (наприклад, відправлення повідомлень про статус запрошення тощо). Такий зв'язок допомагає забезпечити безпеку та відстеження дій користувачів у системі [23].

Клас Offer:

— Клас Offer використовується для створення та управління пропозиціями, які надаються в системі.

— Поля класу, такі як заголовок, опис, ціна, статус, дата створення, деталі та організація, дозволяють зберігати інформацію про кожну пропозицію.

— Поле user вказує на користувача, який створив пропозицію, тоді як поле user_contractor вказує на користувача, який приймає її.

— Пропозиція може бути пов'язана з певною організацією через поле organization.

— Клас Offer взаємодіє з іншими класами, зокрема з класами Organization та UserFSM.

Клас Contract:

— Клас Contract використовується для представлення та управління контрактами між сторонами.

— Поля класу, такі як заголовок, опис, дати початку та завершення, статус контракту, дозволяють зберігати інформацію про кожний контракт.

— Контракт може мати прив'язки до пропозиції (Offer), платежу (Payment) та документу (Document), що відповідають за цей контракт.

— Клас Contract дозволяє створювати, оновлювати та видаляти контракти в системі.

— Взаємодія з іншими класами, такими як Offer, Payment та Document, дозволяє відстежувати статус контрактів та їх виконання.

Клас Organization:

— Клас Organization використовується для представлення та управління організаціями або компаніями в системі.

— Поля класу, такі як назва, адреса, телефон, електронна пошта, веб-сайт та опис, дозволяють зберігати інформацію про кожну організацію.

— Організація може мати зв'язки з користувачами (UserFSM), які пов'язані з цією організацією через поле users.

— Клас Organization дозволяє створювати, оновлювати та видаляти інформацію про організації в системі.

— Зображення печатки організації також зберігається у класі Organization.

Клас Document:

— Клас Document використовується для зберігання та управління документами в системі.

— Поля класу, такі як оригінальний файл, підписаний файл та інформація про підпис, дозволяють зберігати інформацію про кожен документ.

— Клас Document дозволяє завантажувати, зберігати та відображати документи в системі.

2.3 Розробка інших UML-діаграм

Розробка інших UML-діаграм, таких як діаграми варіантів використання, діаграми послідовності та діаграми активності, є важливим

етапом у процесі створення програмного забезпечення. Ці діаграми доповнюють діаграму класів, розширюючи наше уявлення про програму та дозволяючи нам краще зрозуміти її логіку та функціональність.

Діаграми варіантів використання:

— Діаграми варіантів використання допомагають визначити, як користувачі будуть взаємодіяти з системою та які дії вони можуть виконувати.

— Вони вказують на ключові функціональні вимоги до програми, а також дозволяють зрозуміти потреби користувачів з точки зору їхніх взаємодій з системою.

— Діаграми варіантів використання допомагають уникнути недорозумінь між розробниками та замовниками щодо очікувань від програми.

Діаграми послідовності:

— Діаграми послідовності дозволяють відслідковувати послідовність подій та взаємодію об'єктів у системі, подано на рисунку 2.4.

— Вони показують, як об'єкти взаємодіють один з одним у реальному часі, показуючи послідовність повідомлень та дій, що відбуваються між ними.

— Діаграми послідовності допомагають виявити потенційні проблеми у логіці програми та оптимізувати процеси взаємодії між об'єктами [21].

Діаграми активності:

— Діаграми активності використовуються для моделювання послідовності дій або процесів у системі.

— Вони показують, як об'єкти виконують дії у системі та які стани вони приймають під час цих дій.

— Діаграми активності часто використовуються для моделювання бізнес-процесів, алгоритмів або сценаріїв взаємодії користувачів з системою.

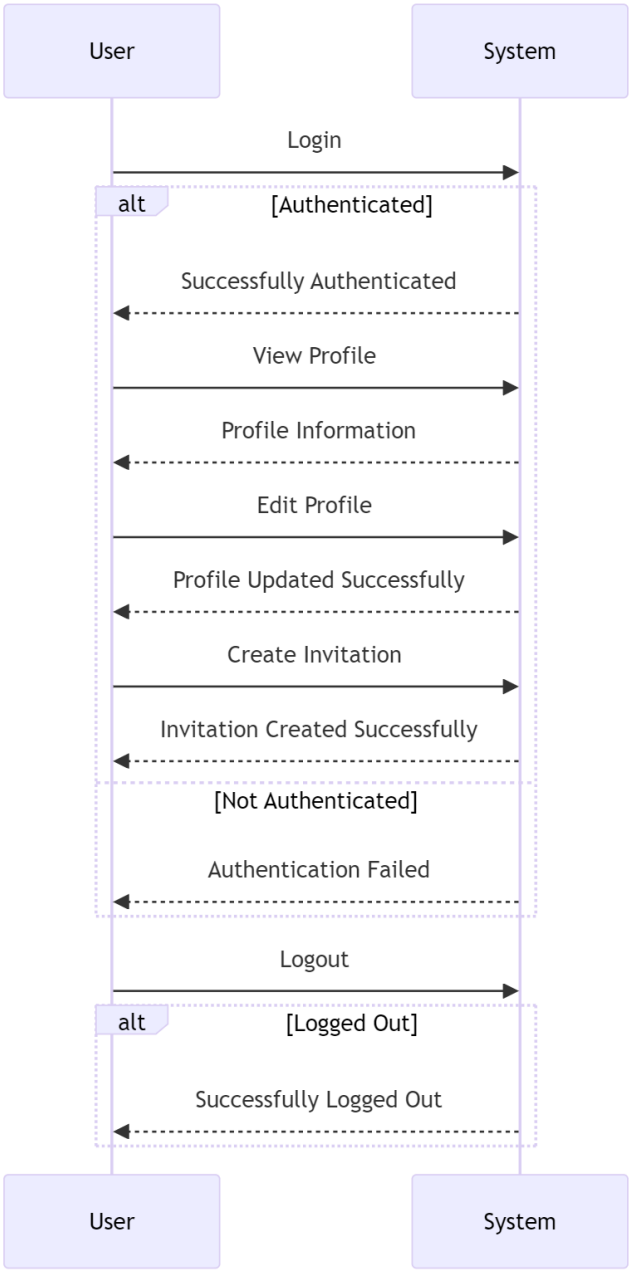

Рисунок 2.4 — Діаграма варіантів взаємодії користувача з системою

Діаграма послідовності використання демонструє взаємодію користувача з системою під час автентифікації, перегляду та редагування профілю, створення запрошень та виходу з системи:

— Користувач намагається увійти в систему, виконавши дію "Login".

— Система перевіряє дані автентифікації.

— Якщо користувач успішно автентифікується, система повідомляє його про успішну автентифікацію та надає можливість переглянути профіль.

— Якщо автентифікація не вдалася, система повідомляє користувача про невдачу.

— Користувач переглядає інформацію профілю.

— Користувач редагує свій профіль.

— Після успішного редагування профілю система повідомляє користувача про успішне оновлення профілю.

— Користувач створює запрошення.

— Система підтверджує успішне створення запрошення.

— Користувач виходить з системи, виконавши дію "Logout".

— Система підтверджує успішний вихід користувача з системи.

Важливість діаграми активності в процесі розробки програмного забезпечення полягає в тому, що вона дозволяє візуалізувати послідовність дій, які виконуються системою або користувачем для досягнення певних цілей. Ключові моменти, які роблять діаграму активності важливою:

Візуалізація послідовності дій: Діаграма активності дозволяє чітко уявити послідовність подій і дій, що відбуваються у системі або процесі.

Аналіз і оптимізація процесів: Вона допомагає розробникам та аналітикам виявляти можливі проблеми та оптимізувати процеси, спрощуючи їх або забезпечуючи більш ефективну роботу.

Виявлення ризиків та можливостей: Діаграма активності дозволяє виявити можливі ризики та можливості, пов'язані з процесом, що допомагає розробникам приймати обґрунтовані рішення [22].

Комунікація і спілкування: Вона служить ефективним інструментом комунікації між членами команди розробників, а також з зацікавленими сторонами, які можуть бути не знайомі з технічними деталями проєкту.

Документація та навчання: Діаграма активності може використовуватися як частина документації проєкту або для навчання нових

членів команди, щоб вони краще зрозуміли процеси та функціональність системи, подано на рисунку 2.5.

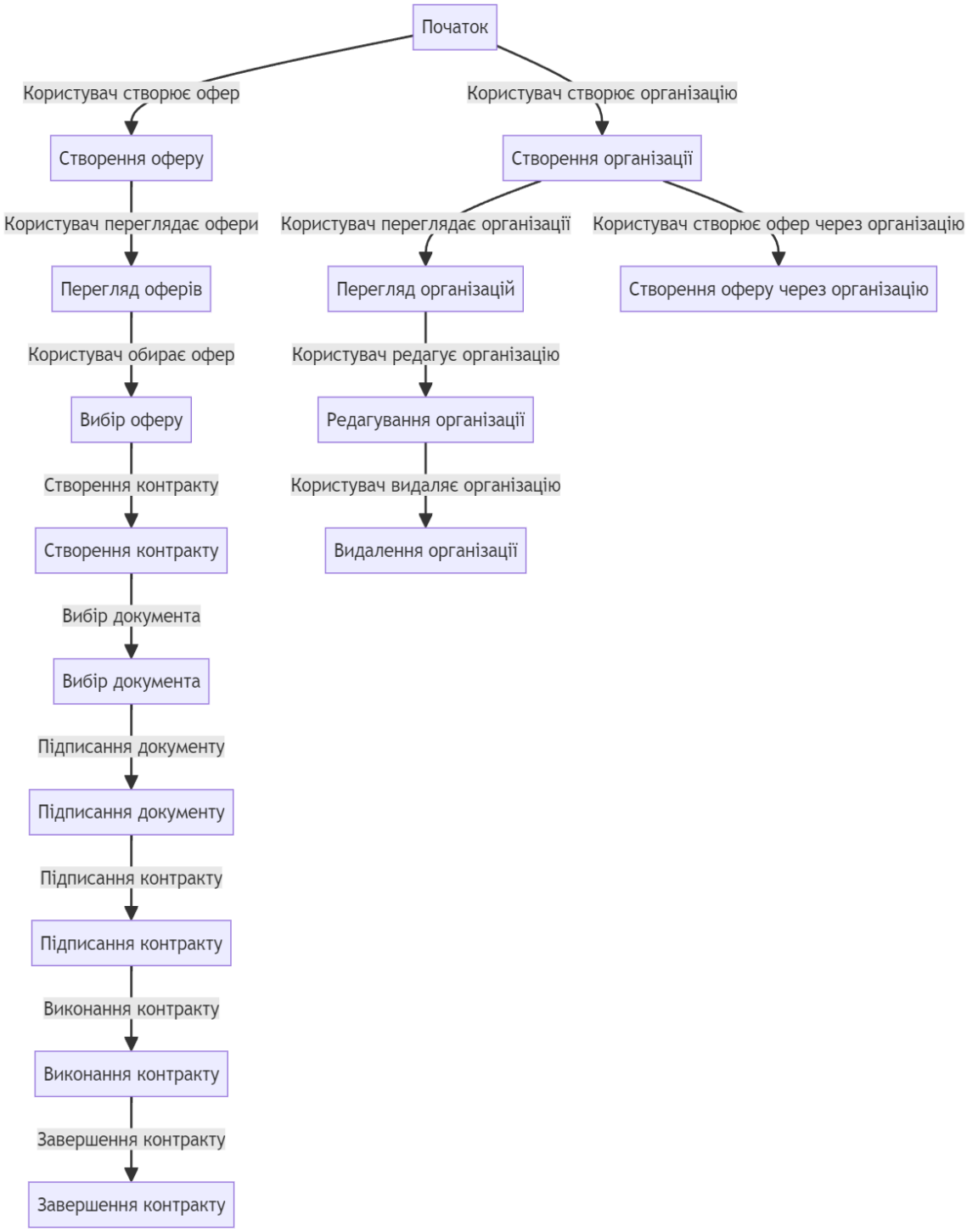

Рисунок 2.5 — Діаграма активності додатку

Діаграма активності, яку ми створили, відображає послідовність дій користувачів та системи у процесі створення та виконання контракту.

Цей процес є ключовим для бізнесу, оскільки контракти визначають умови співпраці між сторонами і можуть впливати на виконання різних видів робіт, обмін товарів або послуг, юридичні відносини та багато іншого [24].

Основна мета цієї діаграми полягає в тому, щоб візуалізувати та узгодити робочі процеси, забезпечити зрозуміння того, як система функціонує у реальному часі.

Вона також допомагає ідентифікувати можливі проблеми або недоліки у процесі та розробляти стратегії для їх вирішення.

Створення цієї діаграми допомагає команді проєкту:

— Зрозуміти логіку роботи системи: Це дозволяє кожному учаснику проєкту отримати чітке уявлення про послідовність подій та взаємодію між об'єктами системи.

— Виявити можливі проблеми та ризики: Аналіз діаграми допомагає виявити слабкі місця у процесі та підготуватися до них.

— Покращити ефективність: Розуміння процесу дозволяє виявити можливі місця для оптимізації та покращення продуктивності.

— Забезпечити згоду всіх учасників: Діаграма активності служить інструментом для спілкування між усіма учасниками проєкту та забезпечує згоду щодо логіки роботи системи.

На рисунку 2.6 подано діаграму активності, яка ілюструє всі ці аспекти. Завдяки цьому інструменту, команда проєкту може ефективно планувати, виконувати та контролювати всі етапи створення та виконання контрактів, що в кінцевому результаті сприяє підвищенню ефективності бізнес-процесів.

Діаграма також забезпечує чітке розуміння всіх учасників процесу, включаючи їхні ролі та обов'язки, що сприяє злагодженій роботі команди. Кожен етап процесу, від ініціації до завершення, детально представлений, що дозволяє легко відстежувати прогрес та вчасно виявляти будь-які проблеми.

Крім того, діаграма допомагає забезпечити відповідність усіх дій нормативним вимогам та внутрішнім стандартам компанії. Це, у свою чергу, підвищує якість та надійність виконуваних контрактів, що є критично важливим для успіху бізнесу.

Рисунок 2.6 — Взаємодія юзера з контрактами та оферами

Діаграма представляє послідовність дій у взаємодії користувача з додатком через його фронтенд і бекенд. Основні кроки включають відкриття додатку користувачем, запит даних користувача від фронтенду до бекенду, створення нового контракту, перевірку можливості створення контракту на бекенді, збереження даних контракту на бекенді, підтвердження створення контракту на фронтенді, оновлення списку контрактів на фронтенді, створення нового оферу, перевірку можливості створення оферу на бекенді, подано на рисунку 2.7, збереження даних оферу на бекенді та підтвердження створення оферу на фронтенді [25].

— Відкриття додатку: Користувач відкриває додаток.

— Запит на дані користувача: Фронтенд надсилає запит до бекенду для отримання даних користувача.

— Відправка даних користувача: Бекенд відправляє дані користувача на фронтенд.

— Створення нового контракту: Користувач створює новий контракт через фронтенд.

— Перевірка можливості створення контракту: Бекенд перевіряє можливість створення контракту.

— Відповідь про можливість створення: Бекенд повертає відповідь про можливість створення контракту на фронтенд.

— Збереження даних контракту: Фронтенд зберігає дані контракту на бекенді.

— Підтвердження створення контракту: Бекенд підтверджує створення контракту на фронтенді.

— Оновлення списку контрактів: Фронтенд оновлює список контрактів.

— Створення нового оферу: Користувач створює новий офер через фронтенд.

— Перевірка можливості створення оферу: Бекенд перевіряє можливість створення оферу.

— Відповідь про можливість створення: Бекенд повертає відповідь про можливість створення оферу на фронтенд.

— Збереження даних оферу: Фронтенд зберігає дані оферу на бекенді.

— Підтвердження створення оферу: Бекенд підтверджує створення оферу на фронтенді

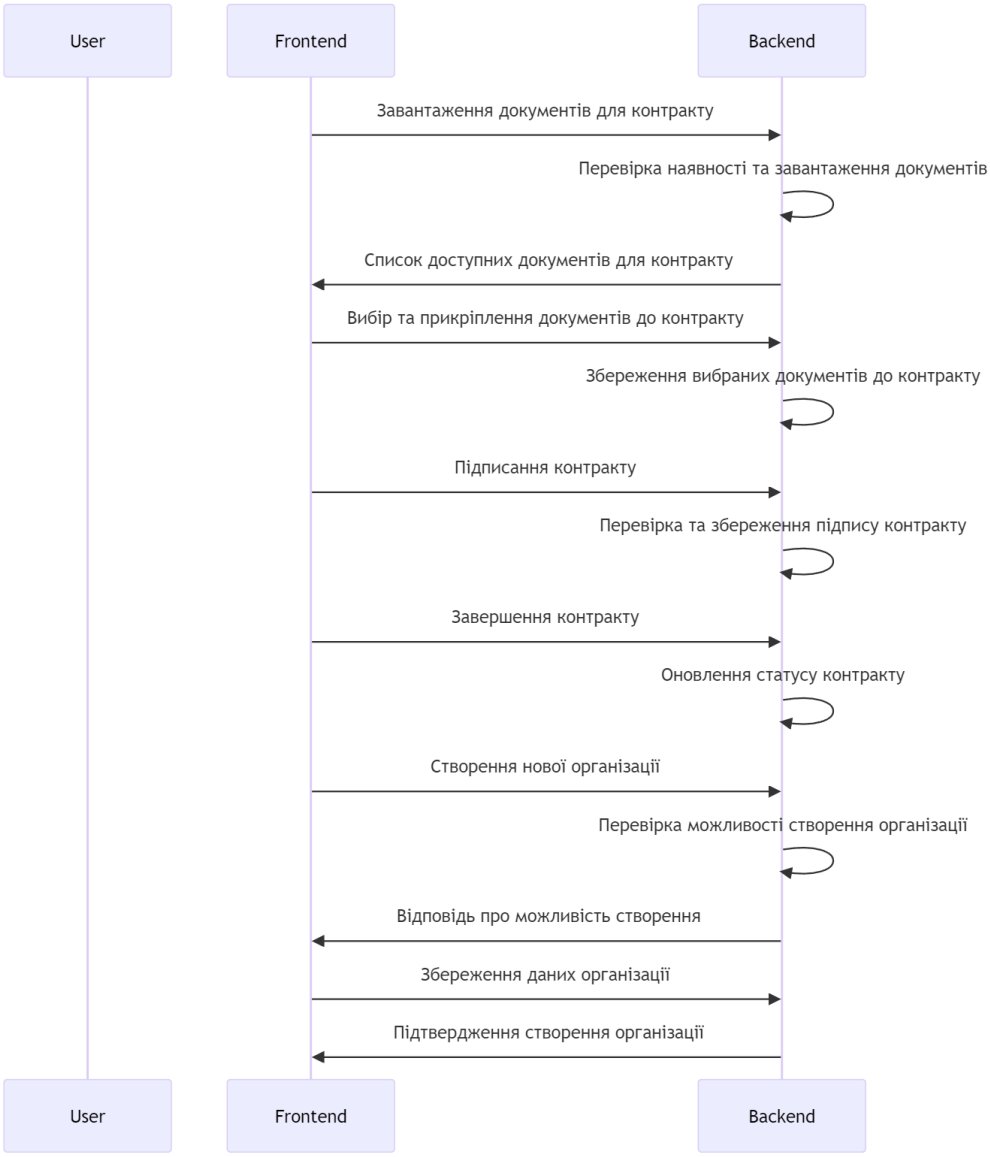

Рисунок 2.7 — Взаємодія юзера з документами та організаціями

Ця діаграма представляє послідовність дій між фронтендом і бекендом під час взаємодії користувача з додатком.

Основні кроки, описані в цій діаграмі, включають завантаження, вибір та прикріплення документів до контракту, підписання контракту, завершення контракту, а також створення нової організації.

Важливість цієї послідовності дій:

— Оптимізація процесу: Ця послідовність дій допомагає оптимізувати процес взаємодії між фронтендом і бекендом, дозволяючи користувачам легко працювати з додатком.

— Покращення користувацького досвіду: Швидка та ефективна робота додатку сприяє покращенню загального користувацького досвіду, що є ключовим для задоволення потреб користувачів.

— Забезпечення правильності даних: Чітко визначені кроки допомагають у забезпеченні правильності та цілісності даних, що важливо для успішної роботи додатку.

— Покращення продуктивності: Якщо взаємодія між фронтендом і бекендом відбувається без перешкод, це сприяє покращенню продуктивності розробки та впровадженню нових функцій.

— Забезпечення безпеки: Чітко визначені кроки дій допомагають забезпечити безпеку даних та запобігти можливим порушенням безпеки.

2.4 Висновки

Аналізуючи вимоги до програмного продукту та обираючи оптимальні технології для його реалізації, було визначено кілька ключових аспектів. Спочатку виявлено значущість використання фреймворку Django для швидкої та ефективної розробки веб-додатків. Django відомий своєю гнучкістю та надійністю, що робить його вигідним вибором для програмного модуля управління оферами. Також враховується роль стандартних веб-технологій,

таких як HTML, CSS і JavaScript, у створенні зручного та привабливого інтерфейсу користувача.

Крім того, використання фронтенд-фреймворків, таких як React або Angular, дозволяє створювати динамічні та інтерактивні інтерфейси, що підвищує користувацький досвід.

Інтеграція з іншими системами здійснюється за допомогою REST API, що сприяє обміну даними та забезпечує гнучкість взаємодії з іншими додатками та сервісами.

Використання системи контролю версій, такої як Git, відіграє важливу роль у керуванні розробкою програмного модуля.

Git дозволяє відстежувати зміни в коді та співпрацювати з іншими учасниками проєкту.

У контексті візуалізації структури програмного продукту, використання UML-діаграм, таких як діаграми класів та інші, допомагає краще розібратися з функціональністю та взаємодією компонентів, що підвищує розуміння архітектури програми та поліпшує комунікацію між розробниками.

Таким чином, аналіз та вибір відповідних технологій та інструментів формують основу для успішної реалізації програмного модуля управління оферами, забезпечуючи ефективність, надійність та зручність використання для кінцевого користувача.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-РЕСУРСУ УПРАВЛІННЯ ОФЕРАМИ

3.1 Обґрунтування вибору мови програмування

Кожна мова програмування особлива і кожна краще підходить для своєї мети. Для досягнення успішного результату важливо вибрати мову, яка найкраще відповідає вимогам проєкту і забезпечує ефективну та швидку розробку.

Для бекенд розробки, що стосується серверної частини веб-додатків, розглянемо три ключові мови програмування: Python, Java та Node.js.

Python є однією з найпопулярніших мов програмування для бекенду. Він відомий своєю простотою та лаконічністю, що робить його дуже популярним серед розробників. Особливо важливо відзначити його співпрацю з фреймворком Django, який є потужним інструментом для швидкої та ефективної розробки веб-додатків. Django забезпечує широкий спектр функцій, включаючи аутентифікацію користувачів, роботу з базами даних та обробку URL-адрес.

Java також є популярним вибором для бекенд розробки, особливо для великих та масштабованих проєктів. Вона відома своєю надійністю та високою швидкістю виконання, що робить її ідеальним вибором для великих корпоративних систем. Java має широкий екосистему бібліотек та фреймворків, які полегшують розробку складних програмних рішень.

Node.js, заснований на мові JavaScript, став дуже популярним у останні роки. Його основна перевага полягає в тому, що він дозволяє використовувати одну мову як для фронтенду, так і для бекенду. Це спрощує розробку та підтримку коду, оскільки розробники можуть використовувати ті ж самі знання та інструменти для обох частин додатка. Node.js також славиться своєю високою швидкістю виконання, асинхронним програмуванням та великою кількістю доступних модулів.

Кожна з цих мов має свої унікальні переваги та використовується в залежності від конкретних потреб та вимог проєкту.

Основні аспекти розглянуті в таблиці 3.1

Таблиця 3.1 — Основні аспекти бекенд мов програмування

Аспект	Python	Java	Node.js
Фреймворки	Django, Flask, FastAPI	Spring Boot, Spark	Express.js, Koa.js, NestJS
Простота	Простий у вивченні та використанні	Багато стандартних бібліотек і шаблонів	Простий у використанні, асинхронне програмування
Продуктивність	Зазвичай повільніший за Java, але досить ефективний	Висока продуктивність завдяки JVM	Висока продуктивність завдяки асинхронності
Надійність	Надійний і стабільний	Відомий своєю надійністю і стабільністю	Схильний до помилок, але може бути надійним із правильним налаштуванням
Екосистема	Велика кількість бібліотек та інструментів	Обширна екосистема, багато бібліотек та фреймворків	Швидко зростаюча екосистема з багатьма модулями та пакетами
Поширення	Широко використовується в веб-розробці, науці, машинному навчанні	Часто використовується в корпоративному програмуванні	Популярний вибір для додатків у реальному часі та мікросервісів
Підтримка	Активне співтовариство та багато ресурсів	Великі компанії та активне співтовариство	Велике співтовариство розробників

На основі порівняння мов програмування для бекенду, Python, зокрема у поєднанні з фреймворком Django, виділяється як перспективний вибір для багатьох проєктів.

React:

— React - це бібліотека JavaScript для побудови інтерфейсів користувача (UI), розроблена компанією Facebook. Вона дозволяє створювати компоненти UI, які можна легко перевикористовувати та оновлювати. Основні особливості React включають компонентний підхід та використання віртуального DOM.

— Компонентний підхід: React базується на ідеї розділення інтерфейсу на невеликі незалежні компоненти. Ці компоненти можуть бути легко керовані та маніпульовані, що спрощує розробку та підтримку коду [26].

— Віртуальний DOM: React використовує віртуальний DOM, який є абстракцією реального DOM-дерева сторінки. Він дозволяє React оптимізувати процес оновлення веб-сторінок, зменшуючи кількість зайвих операцій та покращуючи продуктивність додатків.

Angular:

— Angular - це фреймворк для створення односторінкових веб-додатків (SPA) та динамічних веб-сторінок, розроблений компанією Google. Він надає розробникам інструменти для розробки як фронтенду, так і бекенду.

— Модульність: Angular пропонує потужні інструменти для побудови додатків за допомогою модулів. Модулі дозволяють розробникам розділити функціональність додатка на невеликі, незалежні компоненти, що спрощує підтримку та масштабування коду [28].

— Двостороннє зв'язування даних: Angular надає механізм для автоматичного оновлення даних у вигляді моделі та їх відображення у вигляді інтерфейсу користувача. Це дозволяє зберігати дані та їх відображення взаємозалежними, що спрощує розробку та підтримку додатків.

jQuery:

— jQuery - це бібліотека JavaScript, яка спрощує взаємодію з HTML, CSS та обробку подій. Вона надає розробникам потужні інструменти для маніпулювання DOM-елементами та взаємодії з користувачем [27].

— Селектори: jQuery надає розширений функціонал для вибору елементів на веб-сторінці за допомогою CSS-подібних селекторів. Це дозволяє зручно маніпулювати DOM-елементами та змінювати їх властивості.

Основні аспекти, розглянемо в таблиці 3.2

Таблиця 3.2 – Позитивні та негативні аспекти фрейморків

Характеристика	React	Angular	jQuery
Тип	Бібліотека JavaScript	Фреймворк JavaScript	Бібліотека JavaScript
Спрощення роботи з DOM	+ Використовує віртуальний DOM для оптимізації оновлень сторінок.	+ Має свій механізм двостороннього зв'язування даних та власний робочий цикл змін.	+ Забезпечує легкий доступ до DOM-елементів та просту маніпуляцію ними.
Компонентний підхід	+ Так	+ Так	- Ні
Модульність	- Залежить від інших бібліотек для деяких функцій.	+ Вбудований механізм модулів для підтримки модульної архітектури.	- Ні
Величина	+ Легкий	- Великий	~ Малий-середній
Вступний поріг навчання	+ Низький	- Високий	+ Низький
Підтримка	+ Спільнота React широка та активна.	+ Підтримується Google та спільнотою Angular.	- Широко використовується, але не активно оновлюється.

Після аналізу позитивних та негативних аспектів React можна зробити висновок на його користь:

— Легкий та ефективний: React використовує віртуальний DOM для оптимізації роботи з реальним DOM, що забезпечує високу продуктивність та ефективність відображення змін на сторінці.

— Компонентний підхід: React працює з компонентами, що сприяє створенню чіткої та легко змінної структури додатку. Це дозволяє підтримувати високу рівень модульності та перевикористання коду.

— Низький вступний поріг навчання: React має простий та логічний синтаксис, що робить його доступним для початківців. Крім того, велика активна спільнота та широкий обсяг ресурсів дозволяють легко знайти підтримку та відповіді на питання.

— Підтримка: React має широку та активну спільноту, що регулярно оновлюється та розширюється. Це забезпечує стабільність, безпеку та можливість отримання важливих оновлень та покращень.

Тому, з огляду на ці фактори, React виявляється відмінним вибором для фронтенд розробки, забезпечуючи високу ефективність, гнучкість та легкість використання для розробників будь-якого рівня [26].

Для низького рівня програмування, де ефективність та швидкість виконання програм є критичними, часто використовуються мови програмування, які дозволяють прямо взаємодіяти з апаратним забезпеченням. Серед них основними є C та C++.

Мова програмування C відома своєю ефективністю та можливістю прямого доступу до пам'яті комп'ютера. Вона зазвичай використовується для розробки операційних систем, системного програмування та драйверів пристроїв. C надає велику контроль над ресурсами комп'ютера та дозволяє оптимізувати роботу програми під конкретні потреби.

C++, як розширення мови C, надає додаткові можливості об'єктно-орієнтованого програмування, такі як класи та спадкування. Вона також

використовується для розробки операційних систем, драйверів пристроїв та вбудованих систем. C++ дозволяє реалізувати великі та складні проєкти, забезпечуючи високий рівень контролю та оптимізації швидкості виконання.

Ці мови є популярними у сферах, де важлива мінімізація накладних витрат та максимальна ефективність програм. Вони дозволяють розробникам працювати на найнижчому рівні абстракції, безпосередньо взаємодіючи з апаратним забезпеченням та оптимізуючи програми для максимальної швидкості та ефективності.

Хоча мови програмування C та C++ є популярними у сферах, де важлива мінімізація накладних витрат та максимальна ефективність програм, вони не є ідеальними вибором для розробки веб-додатків. Це пов'язано з тим, що веб-додатки зазвичай потребують швидкого розвитку, гнучкості та можливості взаємодії з користувачем на високому рівні абстракції.

З цих причин розгляд детальніших аспектів мов C та C++ у контексті розробки веб-додатків не є необхідним, оскільки існують інші мови та технології, які надають більш швидкий та ефективний спосіб створення веб-додатків з урахуванням потреб сучасного Інтернету.

3.2 Створення web-ресурсу управління оферами

Вибір технологій для реалізації веб-додатку для управління оферами був обгрунтований ретельним аналізом вимог проєкту та характеристиками доступних інструментів. На підставі цього аналізу було прийнято рішення використовувати такі технології:

Python та Django: Python був обраний як мова програмування завдяки своїй простоті, читабельності та ефективності. Django, як високорівневий фреймворк для Python, надавав ряд вбудованих функцій і можливостей, що значно спростили розробку веб-додатку та забезпечили його надійність та безпеку.

Django REST Framework (DRF): DRF був обраний для розробки API для взаємодії з клієнтською частиною додатку. Він надавав зручні засоби для створення RESTful API на основі Django, що спрощувало роботу з обміном даними між сервером та клієнтом.

React: React був обраний для створення клієнтської частини веб-додатку через свою ефективну та зручну модель компонентів, що дозволяла реалізувати динамічний та інтерактивний інтерфейс користувача.

WebStorm та PyCharm: Інтегровані середовища розробки WebStorm (для фронтенду) та PyCharm (для бекенду) були обрані з урахуванням їхньої потужності, зручного інтерфейсу та вбудованих інструментів, що полегшували розробку, налагодження та тестування коду.

Ці вибори були зроблені з урахуванням потреб проєкту та мети створення веб-додатку для управління оферами, спрямованої на ефективність, надійність та зручність користування.

Почнемо зі створення проєкту на GitHub. Створення репозиторію на GitHub важливо з кількох причин. По-перше, це забезпечує збереження версій коду в хмарному сховищі, що робить процес розробки більш гнучким і безпечним. Ви завжди можете повернутися до попередніх версій коду у разі необхідності. По-друге, GitHub надає зручний інтерфейс для співпраці з іншими членами команди та для прийняття внесків від спільноти. Це дозволяє ефективно керувати процесом розробки та спільно працювати над проєктом з іншими розробниками.

Після створення репозиторію на GitHub та налаштування основи Django проєкту, наступним кроком є встановлення необхідних залежностей та налаштування середовища розробки. У розділі залежностей проєкту, зазначимо модулі, які потрібно встановити для правильної роботи додатку.

```
django~=3.2.0
djangorestframework~=3.12.0
django-filter~=2.4.0
django-constance~=2.7.0
```

django-post-office~=3.1.0

django-guid~=1.0.0

django-rest-framework-authtoken~=2.0.0

Ці залежності вказують на необхідні пакети Django та його розширень, таких як Django REST Framework, які допоможуть вам зробити ваш веб-додаток більш потужним та функціональним. Після додавання залежностей до проєкту потрібно встановити їх за допомогою pip командою:

pip install -r requirements.txt

Після встановлення необхідних залежностей потрібно перейти до налаштування Django додатку. Це включає в себе створення та налаштування моделей для збереження даних про офери, створення та налаштування REST API за допомогою Django REST Framework для забезпечення взаємодії з цими даними, а також налаштування маршрутизації та інших налаштувань Django проєкту, розглянуто на рисунку 3.1.

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "constance",
    "constance.backends.database",
    "post_office",
    "django_guid",
    "djoser",
    "rest_framework",
    "rest_framework.authtoken",
    "django_filters",
    "userfsm",
    "organization",
    "contract",
    "document",
    "corsheaders"
]
```

Рисунок 3.1 — Налаштовані моделі Django

Після створення проєкту на Django важливо налаштувати його параметри, щоб забезпечити правильну роботу та взаємодію з іншими компонентами додатку, подано на рисунку 3.2.

Нижче наведені деякі ключові параметри налаштування Django проєкту:

— MEDIA_ROOT та MEDIA_URL: Ці параметри вказують Django, де зберігати та як доступатися до медіа-файлів, таких як зображення, відео тощо.

— STATIC_ROOT та STATIC_URL: Аналогічно до медіа-файлів, ці параметри вказують місце та URL для зберігання та доступу до статичних файлів, таких як CSS та JavaScript.

— CORS_ALLOWED_ORIGINS: Цей параметр вказує, які джерела можуть здійснювати запити на ваш сервер через CORS (Cross-Origin Resource Sharing).

— AUTH_USER_MODEL та AUTHENTICATION_BACKENDS: Ці параметри вказують Django, яку модель користувача використовувати для аутентифікації користувачів та які методи аутентифікації використовувати.

— DJOSER: Це налаштування визначає параметри для бібліотеки Django Rest Framework для управління користувачами та аутентифікацією.

— REST_FRAMEWORK: Ці параметри визначають параметри автентифікації, фільтрації, пагінації та форматування даних для REST API.

— SIMPLE JWT : Для Django забезпечує простий та зручний механізм роботи з JWT-токенами для автентифікації веб-додатків.

Налаштування цих параметрів забезпечує стабільну та безпечну роботу вашого проєкту на Django, а також полегшує інтеграцію з іншими системами та компонентами. Вони дозволяють створити масштабовану та гнучку архітектуру, яка може легко адаптуватися до змінних вимог бізнесу та технологічних оновлень.

```
DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"
FRONTEND_DIR = os.path.join(MANAGE_ROOT, "apps/esm/contrib/frontend")
WEBPACK_LOADER = {
    "DEFAULT": {
        "BUNDLE_DIR_NAME": "frontend/bundles/",
        "STATS_FILE": os.path.join(FRONTEND_DIR, "webpack/stats/base.json"),
    },
}

AUTH_USER_MODEL = "userfsm.UserFSM"
AUTHENTICATION_BACKENDS = [
    "userfsm.backends.EmailBackend",
    "userfsm.backends.PhoneBackend",
]

DJOSER = {
    "PASSWORD_RESET_CONFIRM_URL": "#/password/reset/confirm/{uid}/{token}",
    "USERNAME_RESET_CONFIRM_URL": "#/username/reset/confirm/{uid}/{token}",
    "ACTIVATION_URL": "auth/activate/{uid}/{token}",
    "SEND_ACTIVATION_EMAIL": False,
}


REST_FRAMEWORK = {
    "DEFAULT_AUTHENTICATION_CLASSES": (
        "rest_framework_simplejwt.authentication.JWTAuthentication",
        "rest_framework.authentication.TokenAuthentication",
    ),
    "DEFAULT_FILTER_BACKENDS": ("django_filters.rest_framework.DjangoFilterBackend",),
    "DEFAULT_PAGINATION_CLASS": "rest_framework.pagination.PageNumberPagination",
    "DATETIME_FORMAT": "%m/%d/%Y",  #  %H:%M:%S"
    "PAGE_SIZE": 10,
}
```

Рисунок 3.2 — Налаштування параметрів Django

Після створення Django проєкту та налаштування його параметрів, наступним етапом є розробка API точок за допомогою Django Rest Framework (DRF), подано на рисунку 3.3. API точки відповідають на запити від клієнтів і надають необхідні дані. Для цього використовуються різні компоненти DRF:

— Маршрутизатори (Routers): Маршрутизатори допомагають визначити URL-шляхи до різних API-точок. Вони автоматично генерують URL-шляхи для створених вами видів (views) та їх пов'язаних методів. Це спрощує налаштування URL-шляхів та забезпечує структурованість вашого API.

— Серіалізатори (Serializers): Серіалізатори відповідають за перетворення моделей Django у формати, зрозумілі для веб-клієнтів. Вони дозволяють перетворити дані у формат JSON або інші формати для передачі через мережу. Серіалізатори також відповідають за валідацію даних, які надходять через API.

— Види (Views): Види визначають логіку обробки запитів та відправки відповідей. Вони приймають запити, обробляють їх та повертають відповіді. Види можуть використовувати серіалізатори для обробки даних, а також доступ до моделей для отримання або збереження інформації в базі даних, подано на рисунку 3.4.

```python
from django.db import models
from django.contrib.auth import get_user_model
from organization.models import Organization
from document.models import Document
from django.utils.translation import gettext_lazy as _

User = get_user_model()


nazar
class Offer(models.Model):
    STATUS_CHOICES = (
        ('pending', 'Pending'),
        ('accepted', 'Accepted'),
        ('rejected', 'Rejected'),
    )
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, blank=True)
    user_contractor = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, related_name='user_contractor',
                                        blank=True)
    title = models.CharField(max_length=255, blank=True, null=True)
    description = models.TextField(blank=True, null=True)
    price = models.DecimalField(max_digits=10, decimal_places=2, blank=True, null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    organization = models.ForeignKey(Organization, on_delete=models.CASCADE, blank=True, null=True)
    details = models.TextField(blank=True, null=True)
    status = models.CharField(
        max_length=20,
        choices=STATUS_CHOICES,
        default="pending",
    )

    nazar
    def __str__(self):
        return self.title


nazar
class Payment(models.Model):
    transaction_id = models.CharField(max_length=255)
    amount = models.DecimalField(max_digits=10, decimal_places=2)
    status = models.CharField(max_length=50)
    created_at = models.DateTimeField(auto_now_add=True)
```

Рисунок 3.3 — Налаштування моделі оферу

```python
class ContractViewSet(ModelViewSet):
    queryset = Contract.objects.all()
    serializer_class = ContractSerializers
    pagination_class = None
    permission_classes = [IsAuthenticated]

    nazar
    @action(detail=True, methods=['post'])
    def sign(self, request, pk=None):
        contract = self.get_object()
        if contract.offer.organization:
            if contract.status != 'signed':
                sign_pdf = generate_pdf(contract, sign=True)
                contract.document.stamped_file.save(f'{contract.title}_signed.pdf', sign_pdf)
                contract.status = 'signed'
                contract.save()
                return Response({'status': 'signed'}, status=status.HTTP_200_OK)
        elif contract.offer.user_contractor:
            signer = Signer()
            contract.document.signd = signer.sign("Signed")
            sign_pdf = generate_pdf(contract, sign=True, signer=contract.document.signd)
            contract.document.stamped_file.save(f'{contract.title}_signed.pdf', sign_pdf)
            contract.status = 'signed'
            contract.save()
            return Response({'status': 'signed'}, status=status.HTTP_200_OK)

    nazar
    @action(detail=True, methods=['get'])
    def download(self, request, pk=None):
        contract = self.get_object()
        if contract.status == 'signed':
            file_path = contract.document.stamped_file.path
        elif contract.status == 'pending':
            file_path = contract.document.original_file.path
        else:
            return Response({'status': 'Contract not signed'}, status=status.HTTP_400_BAD_REQUEST)

        response = FileResponse(open(file_path, 'rb'), content_type='application/pdf')
        response['Content-Disposition'] = f'attachment; filename="{contract.title}.pdf"'
        return response
```

Рисунок 3.4 — Налаштування Api точок для контрактів

В цьому прикладі створюються Api точки для моделі Contract:

from rest_framework.viewsets import ModelViewSet: Імпортує ModelViewSet, який дозволяє використовувати всі стандартні методи для створення, оновлення, видалення та перегляду об'єктів моделі.

from ..models import Contract, Offer: Імпортує моделі Contract та Offer з файлу моделей.

from rest_framework.permissions import IsAuthenticated: Імпортує клас дозволів IsAuthenticated, який вимагає, щоб користувач був аутентифікований для доступу до цих API точок.

from .serializers import ContractSerializers, OfferSerializers: Імпортує серіалізатори для моделей Contract та Offer.

from ..utils import generate_pdf: Імпортує функцію generate_pdf з власного модулю утиліт.

from rest_framework.decorators import action: Імпортує декоратор action, який дозволяє додавати додаткові методи до ModelViewSet.

from rest_framework.response import Response: Імпортує клас Response для створення відповідей API.

from rest_framework import status: Імпортує клас status, який містить коди статусів HTTP для використання в відповідях API.

from django.http import FileResponse: Імпортує клас FileResponse для створення відповіді з файлом.

from django.core.signing import Signer: Імпортує клас Signer для підписування документів.

from django.db.models import Q: Імпортує Q-об'єкт для складних запитів до бази даних.

Клас ContractViewSet успадкований від ModelViewSet і містить методи для створення, оновлення, видалення та перегляду об'єктів моделі Contract. Крім стандартних методів, цей клас містить два додаткові дії (sign та download), які використовуються для підписання договорів та завантаження файлів. Методи sign та download використовують декоратор @action, щоб позначити їх як додаткові дії.

```
from rest_framework import routers
from .viewsets import ContractViewSet, OfferViewSet
react_router = routers.DefaultRouter()


react_router.register("contract", ContractViewSet, basename="contract")
react_router.register("offer", OfferViewSet, basename="offer")
```

Рисунок 3.4 — Визначення Url шляхів для представлень

Налаштування Url маршрутів для APi точок:

from rest_framework import routers: Імпортує клас DefaultRouter з модуля routers бібліотеки Django REST Framework.

from .viewsets import ContractViewSet, OfferViewSet: Імпортує класи ContractViewSet та OfferViewSet, які визначають API точки для моделей Contract та Offer.

react_router = routers.DefaultRouter(): Створює екземпляр маршрутизатора DefaultRouter.

react_router.register("contract", ContractViewSet, basename="contract"): Реєструє API точку для моделі Contract. Перший параметр "contract" - це шлях до API точки у веб-адресі, другий параметр ContractViewSet - це клас, який визначає поведінку для цієї API точки, а параметр basename="contract" вказує базове ім'я, яке буде використовуватися для генерації іменованих маршрутів, подано на рисунку 3.5.

```
from django.contrib import admin
from django.urls import path,include
from rest_framework import routers
from contract.drf_api.routers import react_router as contract_router
from organization.drf_api.routers import react_router as organization_router
from userfsm.drf_api.routers import react_router as user_router


nazar
class DefaultRouter(routers.DefaultRouter):
    nazar
    def extend(self, reg_routers):
        self.registry.extend(reg_routers.registry)


router = DefaultRouter()
router.extend(contract_router)
router.extend(organization_router)
router.extend(user_router)

urlpatterns = [
    path("admin/", admin.site.urls),
    path("api/v1/", include(router.urls)),
    path("api/v1/auth/", include("userfsm.urls")),
]
```

Рисунок 3.5 — Узагальнення маршрутів Django додатку

Цей фрагмент коду використовується для збирання всіх зареєстрованих маршрутів API в одному місці.

class DefaultRouter(routers.DefaultRouter): Це підклас DefaultRouter з бібліотеки Django REST Framework.

Він розширює функціонал базового DefaultRouter.

def extend(self, reg_routers): Це метод, який дозволяє розширити маршрутизатор DefaultRouter, додавши до нього маршрути з інших маршрутизаторів.

self.registry.extend(reg_routers.registry): Цей рядок коду додає маршрути з іншого маршрутизатора до поточного маршрутизатора.

router = DefaultRouter(): Створюємо екземпляр маршрутизатора.

router.extend(contract_router), router.extend(organization_router), router.extend(user_router): Додаємо маршрути з інших маршрутизаторів (contract_router, organization_router, user_router) до поточного маршрутизатора.

urlpatterns: Це список URL-шляхів Django.

В ньому визначаються шляхи для адміністративного інтерфейсу, API точок та шляхів аутентифікації.

path("api/v1/", include(router.urls)): Додає URL-шлях "api/v1/", який включає всі маршрути, зібрані маршрутизатором router.

Цей процес забезпечує централізоване збирання всіх API маршрутів в одному місці за допомогою маршрутизатора DefaultRouter, що полегшує управління URL-шляхами великого проєкту Django.

Це підвищує читабельність та підтримку коду, оскільки всі маршрути зібрані в одному місці.

Крім того, це дозволяє легко додавати нові маршрути або змінювати існуючі без необхідності вносити зміни в декілька файлів.

Такий підхід також спрощує тестування та відлагодження, оскільки всі маршрути доступні через єдину точку входу.

3.3 Тестування web-ресурсу управління оферами

Тестування додатку є важливим аспектом розробки, який гарантує його надійність, безпеку та коректність функціонування. Воно допомагає виявити та виправити помилки до того, як вони зможуть вплинути на кінцевих користувачів. Регулярне та ретельне тестування дозволяє забезпечити стабільну роботу системи, відповідність її вимогам та очікуванням, а також підтримувати високу якість коду в довгостроковій перспективі [31], подано на рисунку 3.6.

Для тестування web-ресурсу управління оферами використовуються різні підходи та інструменти, які забезпечують комплексну перевірку всіх аспектів додатку. Основні типи тестування, які застосовуються, включають:

— Юніт-тестування - перевірка окремих функцій та методів на коректність їх виконання.

— Інтеграційне тестування - перевірка взаємодії між різними компонентами системи.

— Функціональне тестування - перевірка функціональності додатку згідно з вимогами.

— Тестування безпеки - перевірка системи на вразливості та захист від потенційних атак.

— Тестування продуктивності - оцінка швидкодії та ефективності додатку під різними навантаженнями [29].

Інструменти для тестування

Для тестування Django проєкту використовуються такі інструменти та бібліотеки:

— Django Test Framework: Вбудований у Django модуль для написання та запуску тестів.

— Pytest: Потужний інструмент для написання та виконання тестів, який підтримує як юніт-, так і інтеграційне тестування.

— Django REST Framework: Забезпечує додаткові засоби для тестування API.

— Selenium: Інструмент для автоматизованого тестування веб-інтерфейсів, який дозволяє здійснювати функціональне тестування з точки зору користувача.

Юніт-тестування

Юніт-тестування є одним з ключових етапів перевірки коду. Для написання юніт-тестів у Django використовується вбудований тестовий фреймворк [31]. Наприклад, тестування моделі Offer може виглядати наступним чином:

```
from django.test import TestCase
from .models import Contract, Offer

new *
class OfferModelTest(TestCase):
    new *
    @classmethod
    def setUpTestData(cls):
        Offer.objects.create(title='Test Offer', price=100)

    new *
    def test_name_content(self):
        offer = Offer.objects.get(id=1)
        expected_object_name = f'{offer.title}'
        self.assertEquals(expected_object_name, 'Test Offer')

    new *
    def test_price_content(self):
        offer = Offer.objects.get(id=1)
        expected_object_name = offer.price
        self.assertEquals(expected_object_name, 100)
```

Рисунок 3.6 — Юніт тестування моделі Offer

Інтеграційне тестування

Інтеграційне тестування дозволяє перевірити, як різні компоненти додатку взаємодіють між собою, подано на рисунку 3.7. Наприклад, можна перевірити роботу API:

```python
from django.urls import reverse
from django.contrib.auth import get_user_model
from .models import Organization

User = get_user_model()

class OrganizationCreateTestCase(APITestCase):
    def setUp(self):
        self.user = User.objects.create_user(first_name='testuser', password='testpass',email="kolya2002@gmail.com",phone_number="1234567890")
        self.client.login(first_name='email', password='testpass')

    def test_create_organization(self):
        url = reverse('organization-list')
        data = {
            'name': 'Test Organization',
            'email': 'test@organization.com',
        }
        response = self.client.post(url, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        self.assertEqual(Organization.objects.count(), 1)
        self.assertEqual(Organization.objects.get().name, 'Test Organization')
```

Рисунок 3.7 — Інтеграційне тестування для моделі Organization

Функціональне тестування

Функціональне тестування перевіряє, чи відповідає додаток визначеним функціональним вимогам, подано на рисунку 3.8. Наприклад, можна автоматизувати перевірку входу користувача за допомогою Django Test Client:

```python
from django.test import TestCase
from django.urls import reverse
from .models import UserFSM

class UserLoginTest(TestCase):
    def setUp(self):
        self.user = UserFSM.objects.create_user(first_name='testuser', password='testpass',email="kolya2002@gmail.com",phone_number="1234567890")

    def test_user_login(self):
        response = self.client.post(reverse('login'), {'email': 'kolya2002@gmail.com', 'password': 'testpass'})
        self.assertEqual(response.status_code, 200)
```

Рисунок 3.8 — Функціональне тестування моделі UserFSM

Однак, автоматизовані тести не можуть повністю замінити ручне тестування. Тому важливо також проводити ручне тестування, щоб перевірити додаток з точки зору кінцевого користувача. Це включає:

— Перевірку користувацького інтерфейсу (UI): Переконайтесь, що всі елементи відображаються коректно та працюють згідно з очікуваннями.

— Тестування користувацьких сценаріїв: Пройдіть через різні сценарії використання додатку, щоб переконатися в його функціональності та зручності.

— Перевірку сумісності: Тестування додатку на різних браузерах та пристроях, щоб забезпечити коректну роботу для всіх користувачів.

— Тестування безпеки: Перевірка на вразливості, наприклад, SQL-ін'єкції, XSS (міжсайтовий скриптинг) та інші потенційні загрози.

Цей процес дозволяє виявити проблеми, які можуть не бути виявлені автоматизованими тестами, і забезпечити високу якість кінцевого продукту [30].

Починати тестування програми слід з логіну в систему. Це важливо, оскільки процес аутентифікації є першим кроком для доступу до функціоналу додатку. Якщо цей процес не працює належним чином, користувачі не зможуть скористатися додатком, що робить інші тести менш актуальними. Логін також перевіряє, чи правильно налаштовані взаємодія між фронтендом і бекендом, а також чи правильно зберігаються і обробляються дані користувачів.

Авторизація за допомогою JWT токенів

JSON Web Tokens (JWT) є популярним методом для забезпечення аутентифікації та авторизації в веб-додатках. Процес авторизації за допомогою JWT складається з кількох ключових етапів:

— Вхід користувача: Користувач вводить свої облікові дані (ім'я користувача та пароль).

— Створення токену: Якщо дані користувача вірні, сервер генерує JWT токен, який містить інформацію про користувача і підписується секретним ключем.

— Збереження токену: Токен повертається користувачу і зазвичай зберігається в браузері (наприклад, в localStorage або в cookie).

— Доступ до ресурсів: При кожному наступному запиті до сервера користувач відправляє свій JWT токен в заголовках запиту.

— Верифікація токену: Сервер перевіряє підпис токену і, якщо він валідний, дозволяє доступ до запитуваних ресурсів.

Тестування авторизації зображено на рисунку 3.9.

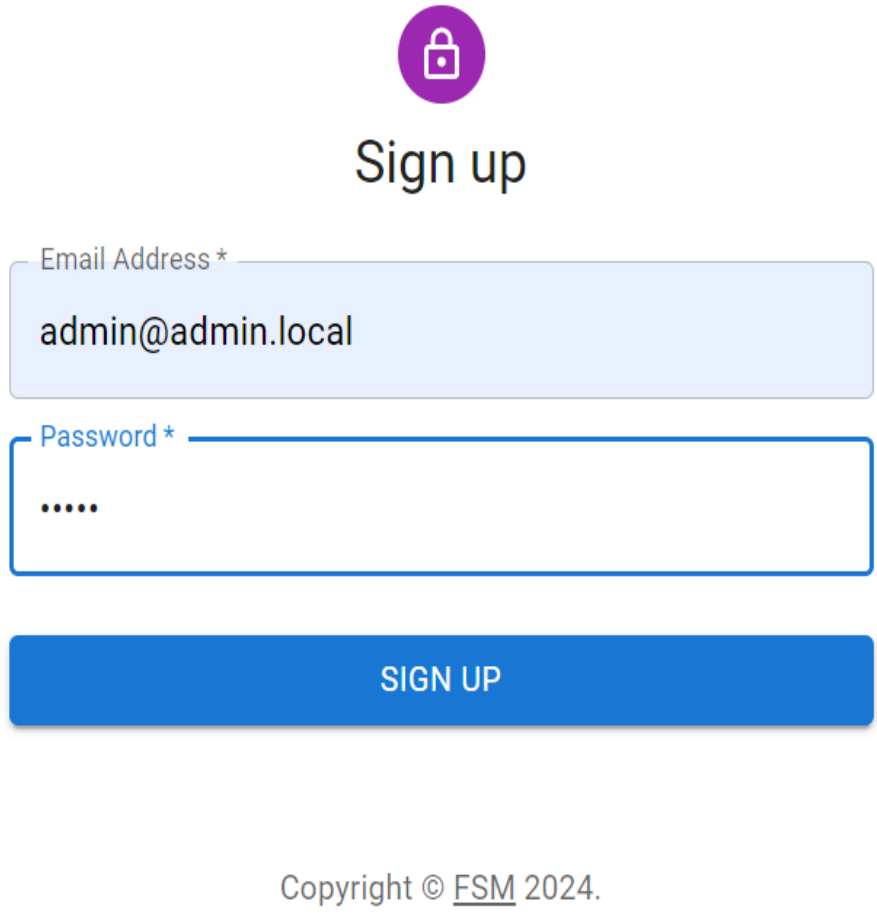

Рисунок 3.9 — Форма логіну користувача

Авторизація за допомогою JSON Web Tokens (JWT) у комбінації з Django на бекенді та React на фронтенді є популярним підходом до забезпечення безпеки та керування доступом у сучасних веб-додатках. Цей метод забезпечує надійний механізм для ідентифікації користувачів та захисту ресурсів.

Як працює авторизація з використанням JWT:

— Вхід користувача (логін):

— Користувач вводить свої облікові дані (ім'я користувача/електронну пошту та пароль) у форму входу на фронтенді.

— Дані надсилаються на бекенд-сервер Django за допомогою POST-запиту.

Генерація токену на сервері:

— Сервер отримує облікові дані та перевіряє їх на коректність.

— Якщо облікові дані правильні, сервер генерує два JWT-токени: access token та refresh token.

— Access token містить інформацію про користувача та має короткий термін дії (зазвичай кілька хвилин).

— Refresh token використовується для оновлення access token і має довший термін дії (зазвичай кілька днів).

Передача токенів на фронтенд:

— Сервер повертає обидва токени (або лише access token) у відповіді на запит.

— Фронтенд (React) зберігає access token та refresh token у безпечному місці (наприклад, у localStorage або sessionStorage).

— Доступ до захищених ресурсів:

— При кожному запиті на захищений ресурс фронтенд надсилає access token у заголовку Authorization: Bearer <access_token>.

— Сервер перевіряє валідність токену. Якщо він дійсний, сервер надає доступ до запитуваного ресурсу.

Оновлення токену:

— Коли термін дії access token закінчується, фронтенд може надіслати запит на сервер з refresh token для отримання нового access token.

— Сервер перевіряє refresh token та, якщо він дійсний, генерує новий access token.

Вихід користувача (лог-аут):

— Коли користувач виходить із системи, access token та refresh token видаляються з клієнтської сторони.

Після того як ми налаштували та протестували функціонал авторизації в нашому додатку, наступним кроком є перевірка логіки підписання і створення контракту. Це важливий аспект, оскільки він безпосередньо впливає на ключові функціональні можливості нашого веб-ресурсу управління оферами.

Тестування логіки підписання контракту

Створення контракту:

— Перевірити, чи можливо створити контракт через відповідний API-ендпоінт.

— Перевірити правильність збереження даних контракту в базі даних.

— Переконатися, що створений контракт має початковий статус (наприклад, "pending").

Підписання контракту, зображено на рисунку 3.10:

— Перевірити можливість підписання контракту через API-ендпоінт.

— Переконатися, що після підписання статус контракту змінюється на "signed".

— Перевірити збереження підписаного файлу контракту в правильному місці та його доступність для завантаження.

— Переконатися, що для різних типів користувачів (наприклад, організація або користувач-конрактор) підписання працює коректно.

Завантаження контракту, зображено на рисунку 3.11:

— Перевірити можливість завантаження підписаного контракту через відповідний API-ендпоінт.

— Переконатися, що завантажений файл відповідає підписаному контракту.

Тестування цих аспектів забезпечить впевненість у тому, що логіка створення та підписання контракту працює належним чином, і користувачі можуть безпечно та ефективно взаємодіяти з системою.

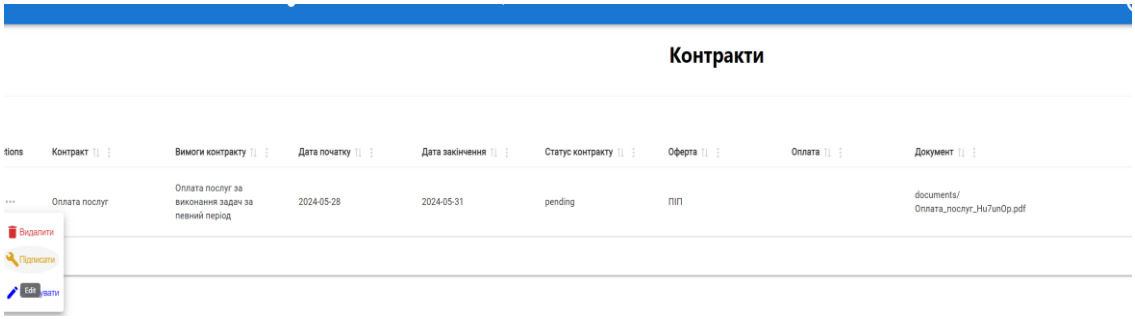

Рисунок 3.10 — Підписання контракту організацією

Отримання контракту:

— Користувач надсилає запит на підписання контракту через відповідний API-ендпоінт.

— Система отримує об'єкт контракту на основі ідентифікатора (ID), переданого в запиті.

Перевірка статусу та умов підписання:

— Якщо контракт належить організації, система перевіряє, чи його статус не є "signed".

— Якщо статус контракту відрізняється від "signed", він готується до підписання.

— Якщо контракт належить користувачу-конрактору, система використовує об'єкт Signer для створення підпису.

Генерація та збереження підписаного файлу:

— Використовується функція generate_pdf для створення PDF-документа з підписом.

— Підписаний файл зберігається в відповідному полі контракту (stamped_file).

—

Оновлення статусу контракту:

— Статус контракту змінюється на "signed".

— Збереження змін у базі даних.

Відповідь клієнту:

— Система відправляє відповідь клієнту із статусом "signed".

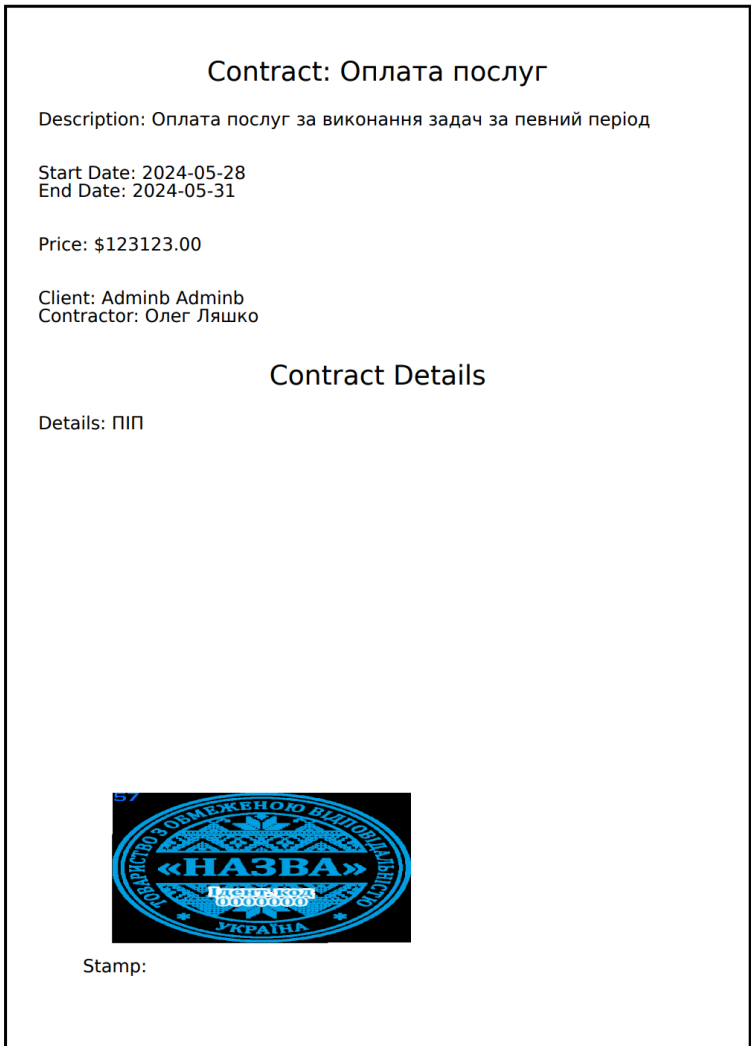

Рисунок 3.11 — Завантажений контракт

Отримання контракту:

— Користувач надсилає запит на завантаження контракту через відповідний API-ендпоінт.

— Система отримує об'єкт контракту на основі ідентифікатора (ID), переданого в запиті.

Перевірка статусу контракту:

— Якщо статус контракту є "signed", система отримує шлях до підписаного файлу.

— Якщо статус контракту є "pending", система отримує шлях до оригінального файлу.

Створення відповіді:

— Система створює об'єкт FileResponse з відкритим файлом контракту для читання в режимі "rb".

— У відповідь додається заголовок "Content-Disposition" з назвою файлу для завантаження.

Відправка файлу клієнту:

— Система відправляє файл клієнту у форматі PDF.

3.4 Висновки

Успішне створення web-ресурсу для управління оферами вимагало ретельного аналізу та вибору відповідних технологій, реалізації програмного забезпечення та всебічного тестування. У цьому розділі детально розглянуто процес вибору мови програмування та фреймворків, а також кроки, необхідні для розробки та тестування проєкту.

Проаналізувавши сучасні технології, ми вибрали Python з Django для бекенд-розробки та React для фронтенд-розробки. Django забезпечує надійний та безпечний бекенд, тоді як React дозволяє створювати динамічні інтерфейси користувача. Налаштування проєкту включало конфігурацію медіа- та

статичних файлів, налаштування CORS та інтеграцію Simple JWT для безпечної авторизації.

Розробка DRF API точок, маршрутизаторів, серіалізаторів та видів дозволила створити ефективну систему обробки запитів, включаючи створення, підписання та завантаження контрактів. Особлива увага була приділена тестуванню основних функцій додатку, що гарантувало високу якість та безпеку його роботи.

Завершуючи, можна підкреслити, що використання Django та React для створення web-ресурсу управління оферами забезпечило надійність, гнучкість та зручність, необхідні для оптимізації процесів управління оферами в сучасному бізнесі.

ВИСНОВКИ

В ході виконання бакалаврської кваліфікаційної роботи виконано поставлені задачі:

— розроблено програмний модуль для створення оферів та договорів.

— реалізовано функціонал збереження та оновлення інформації про офери та договори.

— проведено аналіз потреб користувачів для визначення оптимального інтерфейсу користувача.

— реалізовано адміністративний рівень доступу для керування оферами та договорами.

— проведено тестування розробленого програмного модулю для перевірки його працездатності та надійності.

— оптимізовано роботу програмного модулю та вдосконалено його функціональність на основі отриманих результатів тестування.

Для успішного виконання проєкту було важливо перш ніж все провести глибоке дослідження предметної області, щоб зрозуміти потреби та вимоги користувачів у банківській сфері. Це включало в себе аналіз існуючих програмних рішень, виявлення їхніх переваг та недоліків, а також визначення потреб у новому програмному забезпеченні.

У рамках проєкту було проведено проєктування системи, де була визначена загальна архітектура програмного забезпечення, створено діаграми класів та взаємодії компонентів, описані основні підходи до розробки.

Оскільки виконання проєкту передбачало розробку програмного коду, було важливо вибрати відповідні інструменти та технології. Мовою програмування була обрана Python через її гнучкість та широкий спектр бібліотек для обробки даних та розробки веб-застосунків. Для роботи з базами

даних використовувалися SQL бази даних, що забезпечувало ефективне зберігання та опрацювання інформації.

Протягом реалізації проєкту було важливо також звернути увагу на тестування програмного забезпечення. Було проведено модульне, інтеграційне та функціональне тестування, щоб переконатися у коректності роботи розроблених модулів.

В результаті виконання проєкту було розроблено та впроваджено програмне забезпечення, яке відповідає вимогам та потребам користувачів у банківській сфері. Розроблені модулі допомагають оптимізувати фінансові процеси та поліпшити ефективність управління ресурсами.

Усі заплановані завдання були успішно виконані відповідно до вимог проєкту, а пояснювальна записка була оформлена з урахуванням методичних вказівок до виконання бакалаврської кваліфікаційної роботи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Django for beginners [Електронний ресурс]. — Режим доступу до ресурсу: https://djangobook.com
2. Python Doc [Електронний ресурс]. — Режим доступу до ресурсу: https://www.python.org
3. Django Doc [Електронний ресурс]. — Режим доступу до ресурсу: https://docs.djangoproject.com/en/4.2/
4. Stackoverflow [Електронний ресурс]. — Режим доступу до ресурсу: https://stackoverflow.com/questions/11923317/creating-django-forms
5. Scikit-learn [Електронний ресурс]. — Режим доступу до ресурсу: https://scikit-learn.org/stable/
6. Kaggle [Електронний ресурс]. — Режим доступу до ресурсу: https://www.kaggle.com/datasets/kaushil268/disease-prediction-using-machine-learning/?select=Training.csv
7. Django forum [Електронний ресурс]. — Режим доступу до ресурсу: https://readthedocs.org/projects/django/downloads/
8. Medium [Електронний ресурс]. — Режим доступу до ресурсу: https://towardsdatascience.com/document-your-machine-learning-project-in-a-smart-way-35c68aa5fc0e
9. Django CRM [Електронний ресурс]. — Режим доступу до ресурсу: https://django-crm.readthedocs.io/en/latest/
10. Django Rest [Електронний ресурс]. — Режим доступу до ресурсу: https://www.django-rest-framework.org
11. CRM BLOG [Електронний ресурс]. — Режим доступу до ресурсу: https://gowombat.team/blog/posts/7-reasons-to-use-a-customer-relationship-management-crm-system
12. CRM PULSE [Електронний ресурс]. — Режим доступу до ресурсу: https://sendpulse.com/support/glossary/crm

13. CRM BUILD [Електронний ресурс]. — Режим доступу до ресурсу: https://bambooagile.eu/insights/how-to-build-a-crm-system-from-scratch
14. Payment System [Електронний ресурс]. — Режим доступу до ресурсу: https://reintech.io/blog/integrating-payment-system-django
15. Making free payments in a Django projects [Електронний ресурс]. — Режим доступу до ресурсу: https://stackoverflow.com/questions/77853470/making-free-payments-in-a-django-projects
16. Payment Processing [Електронний ресурс]. — Режим доступу до ресурсу: https://djangopackages.org/grids/g/payment-processing/
17. React [Електронний ресурс]. — Режим доступу до ресурсу: https://react.dev
18. React Doc [Електронний ресурс]. — Режим доступу до ресурсу: https://legacy.reactjs.org/docs/create-a-new-react-app.html
19. React App Doc [Електронний ресурс]. — Режим доступу до ресурсу: https://create-react-app.dev
20. React + Django Doc [Електронний ресурс]. — Режим доступу до ресурсу: https://medium.com/@devsumitg/how-to-connect-reactjs-django-framework-c5ba268cb8be
21. DigitalOcean Doc [Електронний ресурс]. — Режим доступу до ресурсу: https://www.digitalocean.com/community/tutorials/how-to-build-a-modern-web-application-to-manage-customer-information-with-django-and-react-on-ubuntu-18-04
22. React Blog [Електронний ресурс]. — Режим доступу до ресурсу: https://blog.logrocket.com/using-react-django-create-app-tutorial/
23. User Django [Електронний ресурс]. — Режим доступу до ресурсу: https://docs.djangoproject.com/en/5.0/topics/auth/customizing/
24. Custom Django User [Електронний ресурс]. — Режим доступу до ресурсу: https://testdriven.io/blog/django-custom-user-model/

25. Build User Model [Електронний ресурс]. — Режим доступу до ресурсу: https://medium.com/@johnthuo/building-your-own-django-custom-user-model-292bade451bd
26. React Material Table [Електронний ресурс]. — Режим доступу до ресурсу: https://www.material-react-table.com
27. jQuery [Електронний ресурс]. — Режим доступу до ресурсу: https://jquery.com
28. Angular [Електронний ресурс]. — Режим доступу до ресурсу: https://angular.dev
29. Django testing [Електронний ресурс]. — Режим доступу до ресурсу: https://docs.djangoproject.com/en/5.0/topics/testing/
30. Django tutorial testing [Електронний ресурс]. — Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Testing
31. DRF testing [Електронний ресурс]. — Режим доступу до ресурсу: https://www.django-rest-framework.org/api-guide/testing/

ДОДАТКИ

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програмного модуля для ефективного управління оферами та договорами засобами Django

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 98,66% Схожість 1,34%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Палій Н.С.

Керівник роботи _____ Денисюк В.О.

ДОДАТОК Б. ІНСТРУКЦІЯ КОРИСТУВАЧА

Для перегляду основних функцій додатку, можна скористуватись DRF оглядачем кінцевих точок (рисунок В.1)

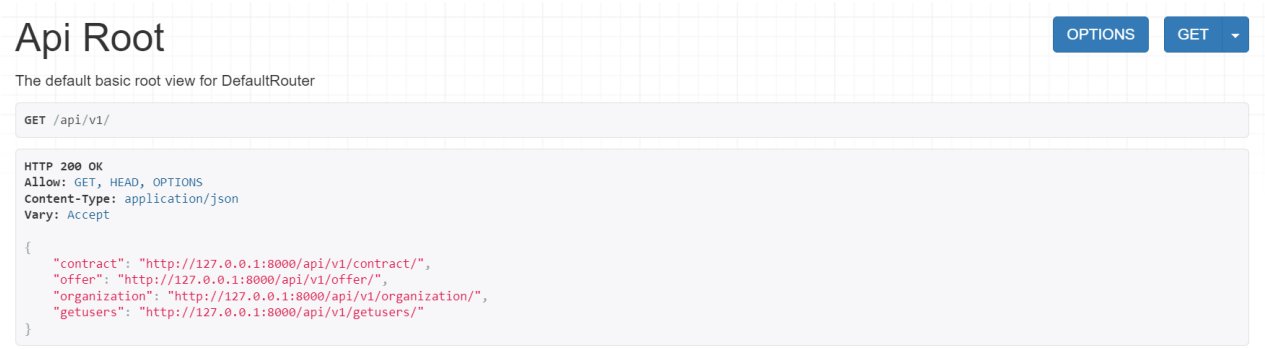

Рисунок В.1 — Оглядач DRF кінцевих точок

Для авторизації користувача, потрібно використовувати окремі API поінти, моделі UserFSM (рисунок В.2)

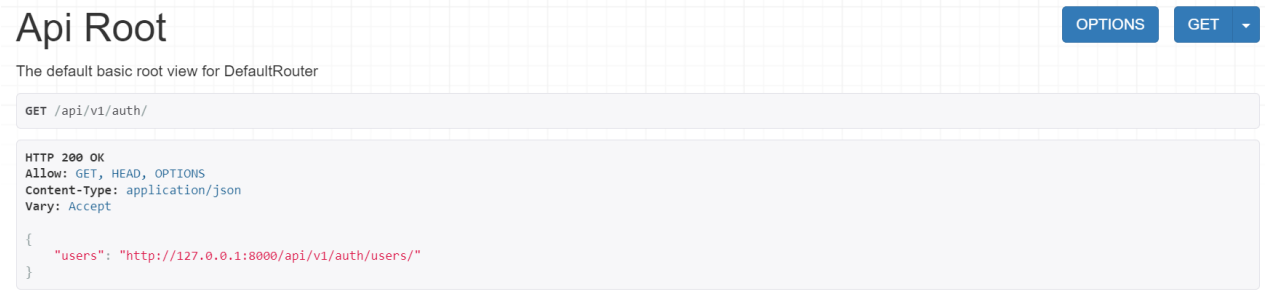

Рисунок В.2 — API поінти моделі UserFSM

Після чого нам запропонує пройти авторизацію за налаштованою нами формою (рисунок В.3). Є можливість авторизації двома методами, через номер телефону користувача та через електронну пошту, будь-який метод, буде зв’язаний з кінцевим шляхом нашого запиту, який і налаштовує авторизацію, в відповідь ми отримаємо токен доступу до нашого додатку

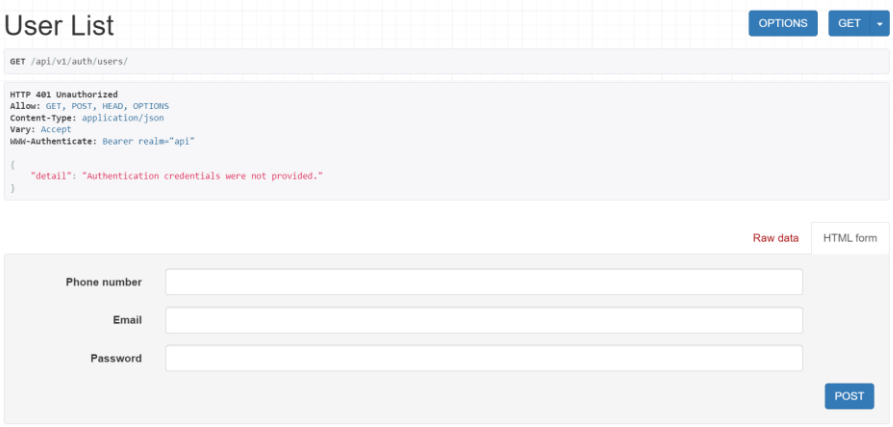

Рисунок В.3 — Точка авторизації користувача

Після успішної авторизації, користувач отримає доступ до основної частини додатку (рисунок В.4).

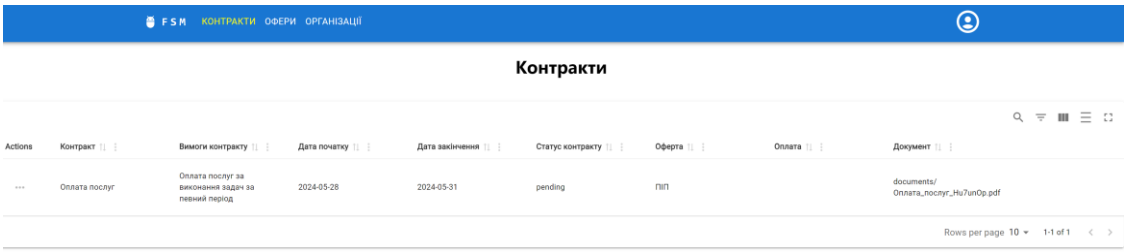

Рисунок В.4 — Головна сторінка додатку

За допомогою налаштованих Actions буде змога змінювати, підписувати та скачувати документ контракту (рисунок В.5)

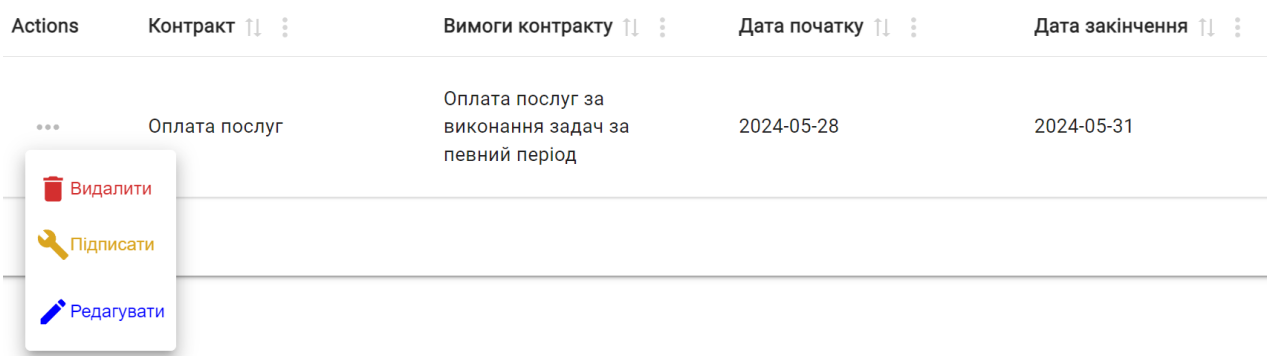

Рисунок В.5 — Основні Actions додатку

ДОДАТОК В. ФРАГМЕНТ ЛІСТИНГУ ПРОГРАМНОГО КОДУ

```
from io import BytesIO
from django.core.files.base import ContentFile
from reportlab.lib.pagesizes import letter
from reportlab.lib.units import inch
from reportlab.lib.utils import ImageReader
from reportlab.pdfgen import canvas
from reportlab.lib.styles import ParagraphStyle
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle
from reportlab.lib import colors
from reportlab.pdfbase.ttfonts import TTFont
from reportlab.pdfbase import pdfmetrics
from reportlab.lib.fonts import addMapping

from reportlab.lib.pagesizes import letter
from reportlab.lib.units import inch

pdfmetrics.registerFont(TTFont('DejaVuSans', 'DejaVuSans.ttf'))
addMapping('DejaVuSans', 0, 0, 'DejaVuSans')
addMapping('DejaVuSans', 0, 1, 'DejaVuSans-Bold')


def generate_pdf(contract, sign=False,signer=None):
    buffer = BytesIO()
    doc = SimpleDocTemplate(buffer, pagesize=letter, rightMargin=72, leftMargin=72, topMargin=72, bottomMargin=72)

    title_style = ParagraphStyle(name='Title', fontName='DejaVuSans-Bold', fontSize=18, spaceAfter=12, alignment=1,
                                 textColor=colors.black)
    normal_style = ParagraphStyle(name='Normal', fontName='DejaVuSans', fontSize=12, spaceAfter=12,
                                  textColor=colors.black)
    justify_style = ParagraphStyle(name='Justify', fontName='DejaVuSans', fontSize=12, spaceAfter=12,
                                   textColor=colors.black)

    elements = []

    title = Paragraph(f"<b>Contract: {contract.title}</b>", title_style)
    elements.append(title)
    elements.append(Spacer(1, 12))

    description = Paragraph(f"<b>Description:</b> {contract.description}", justify_style)
    elements.append(description)
    elements.append(Spacer(1, 12))

    dates = Paragraph(f"<b>Start Date:</b> {contract.start_date}<br/><b>End Date:</b> {contract.end_date}",
```

```
                normal_style)
    elements.append(dates)
    elements.append(Spacer(1, 12))

    if contract.offer and contract.offer.price:
        price = Paragraph(f"<b>Price:</b> ${contract.offer.price}", normal_style)
        elements.append(price)
        elements.append(Spacer(1, 12))

    if contract.offer and contract.offer.user and contract.offer.user_contractor:
        users_info = Paragraph(
            f"<b>Client:</b>                                {contract.offer.user.first_name}
{contract.offer.user.last_name}<br/><b>Contractor:</b>
{contract.offer.user_contractor.first_name} {contract.offer.user_contractor.last_name}",
            normal_style)
        elements.append(users_info)
        elements.append(Spacer(1, 12))
    elif contract.offer and contract.offer.user and contract.offer.organization:
        org_info = Paragraph(
            f"<b>Client:</b>                                {contract.offer.user.first_name}
{contract.offer.user.last_name}<br/><b>Organization:</b> {contract.offer.organization.name}",
            normal_style)
        elements.append(org_info)
        elements.append(Spacer(1, 12))


    contract_tile = Paragraph(f"<b>Contract Details</b>", title_style)
    elements.append(contract_tile)
    elements.append(Spacer(1, 12))


    if contract.offer.details:
        contract_details = Paragraph(f"<b>Details:</b> {contract.offer.details}", justify_style)
        elements.append(contract_details)
        elements.append(Spacer(1, 12))

    doc.build(elements, onFirstPage=lambda canvas, doc: draw_frame(canvas, doc, contract,
sign=sign,signer=signer))

    pdf = buffer.getvalue()
    buffer.close()
    return ContentFile(pdf)


def draw_frame(canvas, doc, contract, sign=False,signer=None):
    canvas.saveState()


    frame_width = 500
    frame_height = 700
    canvas.setStrokeColor(colors.black)
    canvas.setLineWidth(2)
```

```
    canvas.rect((letter[0] - frame_width) / 2, (letter[1] - frame_height) / 2, frame_width,
frame_height, stroke=1,
                fill=0)


    stamp_x = (letter[0] - frame_width) / 2 + inch
    stamp_y = (letter[1] - frame_height) / 2 + inch
    canvas.setFont("DejaVuSans", 12)
    if sign:
        canvas.drawString(stamp_x - 20, stamp_y - 20, "Stamp:")
    else:
        canvas.drawString(stamp_x, stamp_y - 20, "Stamp Here:")
    canvas.setStrokeColor(colors.black)
    canvas.setLineWidth(1)
    canvas.rect(stamp_x, stamp_y, 2 * inch, 1 * inch, stroke=1, fill=0)
    if contract.offer.organization and contract.offer.organization.stamp and sign:
        stamp_image_path = contract.offer.organization.stamp.path
        stamp_image = ImageReader(stamp_image_path)
        canvas.drawImage(stamp_image, stamp_x, stamp_y, width=200, height=100)
    elif contract.offer.user_contractor and sign and signer:
        canvas.drawString(stamp_x - 20, stamp_y - 35, signer)

    canvas.restoreState()
from rest_framework.viewsets import ModelViewSet
from ..models import Contract, Offer
from rest_framework.permissions import IsAuthenticated
from .serializers import ContractSerializers, OfferSerializers
from ..utils import generate_pdf
from rest_framework.decorators import action
from rest_framework.response import Response
from rest_framework import status
from django.http import FileResponse
from django.core.signing import Signer
from django.db.models import Q

class ContractViewSet(ModelViewSet):
    queryset = Contract.objects.all()
    serializer_class = ContractSerializers
    pagination_class = None
    permission_classes = [IsAuthenticated]

    @action(detail=True, methods=['post'])
    def sign(self, request, pk=None):
        contract = self.get_object()
        if contract.offer.organization:
            if contract.status != 'signed':
                sign_pdf = generate_pdf(contract, sign=True)
                contract.document.stamped_file.save(f'{contract.title}_signed.pdf', sign_pdf)
                contract.status = 'signed'
                contract.save()
                return Response({'status': 'signed'}, status=status.HTTP_200_OK)
        elif contract.offer.user_contractor:
```

```
            signer = Signer()
            contract.document.signd = signer.sign("Signed")
            sign_pdf = generate_pdf(contract, sign=True, signer=contract.document.signd)
            contract.document.stamped_file.save(f'{contract.title}_signed.pdf', sign_pdf)
            contract.status = 'signed'
            contract.save()
            return Response({'status': 'signed'}, status=status.HTTP_200_OK)

    @action(detail=True, methods=['get'])
    def download(self, request, pk=None):
        contract = self.get_object()
        if contract.status == 'signed':
            file_path = contract.document.stamped_file.path
        elif contract.status == 'pending':
            file_path = contract.document.original_file.path
        else:
            return Response({'status': 'Contract not signed'},
status=status.HTTP_400_BAD_REQUEST)

        response = FileResponse(open(file_path, 'rb'), content_type='application/pdf')
        response['Content-Disposition'] = f'attachment; filename="{contract.title}.pdf"'
        return response




class OfferViewSet(ModelViewSet):
    queryset = Offer.objects.all()
    serializer_class = OfferSerializers
    pagination_class = None
    permission_classes = [IsAuthenticated]

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)
```

ДОДАТОК Г. ГРАФІЧНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО МОДУЛЮ ДЛЯ ЕФЕКТИВНОГО УПРАВЛІННЯ ОФЕРАМИ ТА ДОГОВОРАМИ ЗАСОБАМИ DJANGO

Виконав: студент 4 курсу, групи КН-22мс
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Палій Н.С.
(прізвище та ініціали)
Керівник: к.т.н., асистент каф. КН
Денисюк О.В.
(прізвище та ініціали)
«07» 06 2024 р.

Вінниця ВНТУ – 2024 рік

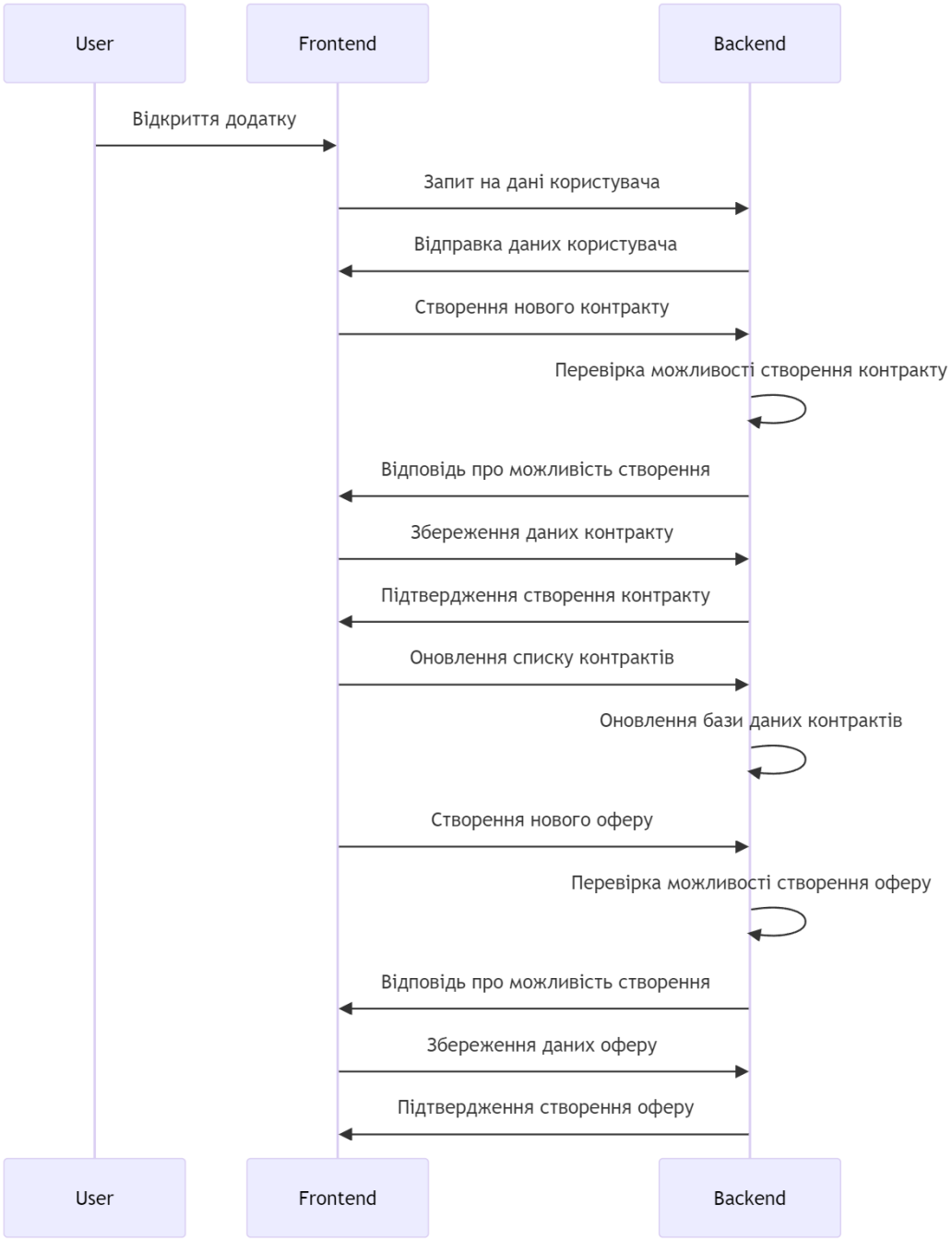

Рисунок Д.1 — Взаємодія юзера з контрактами та оферами

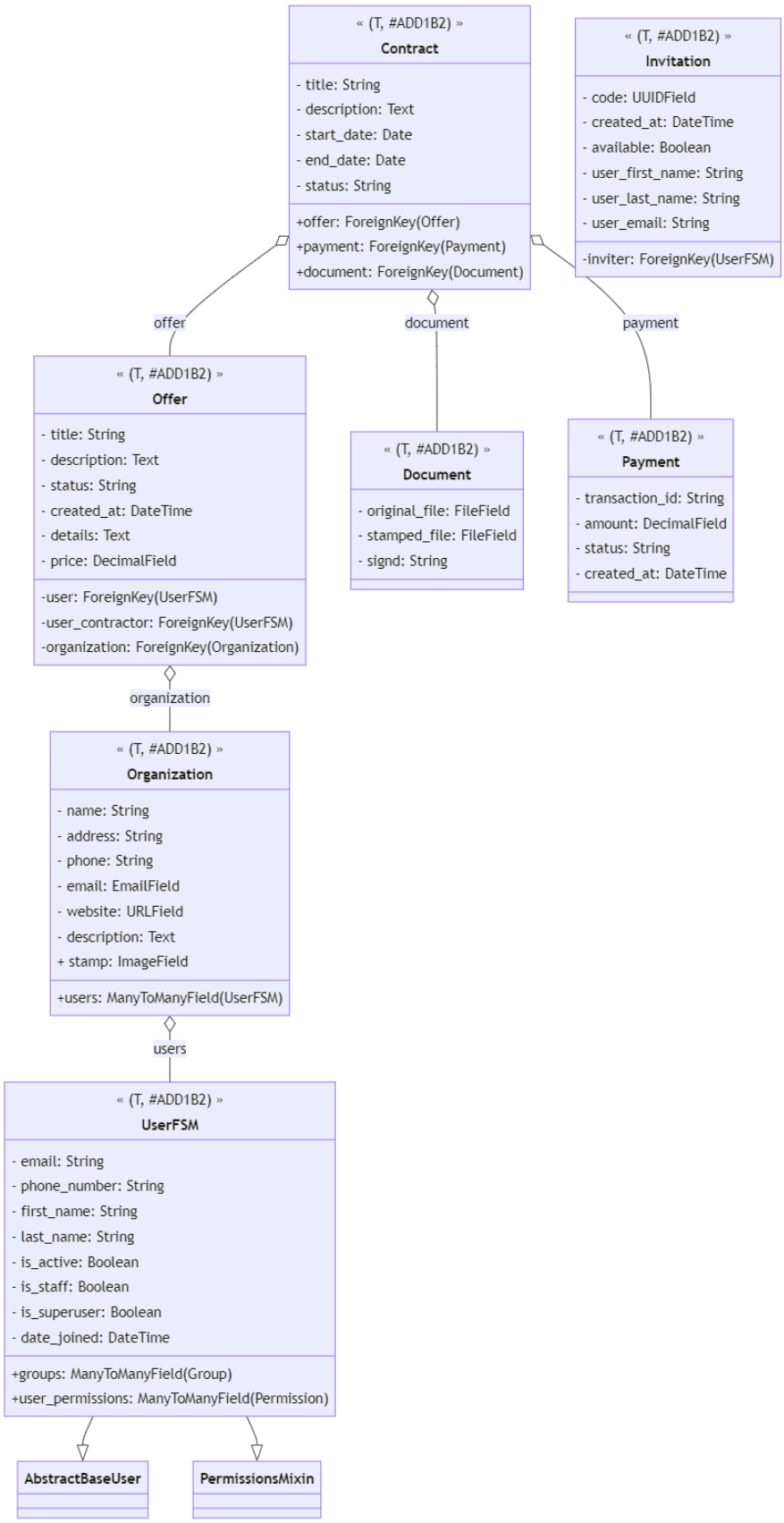

Рисунок Д.2 — Діаграма класів

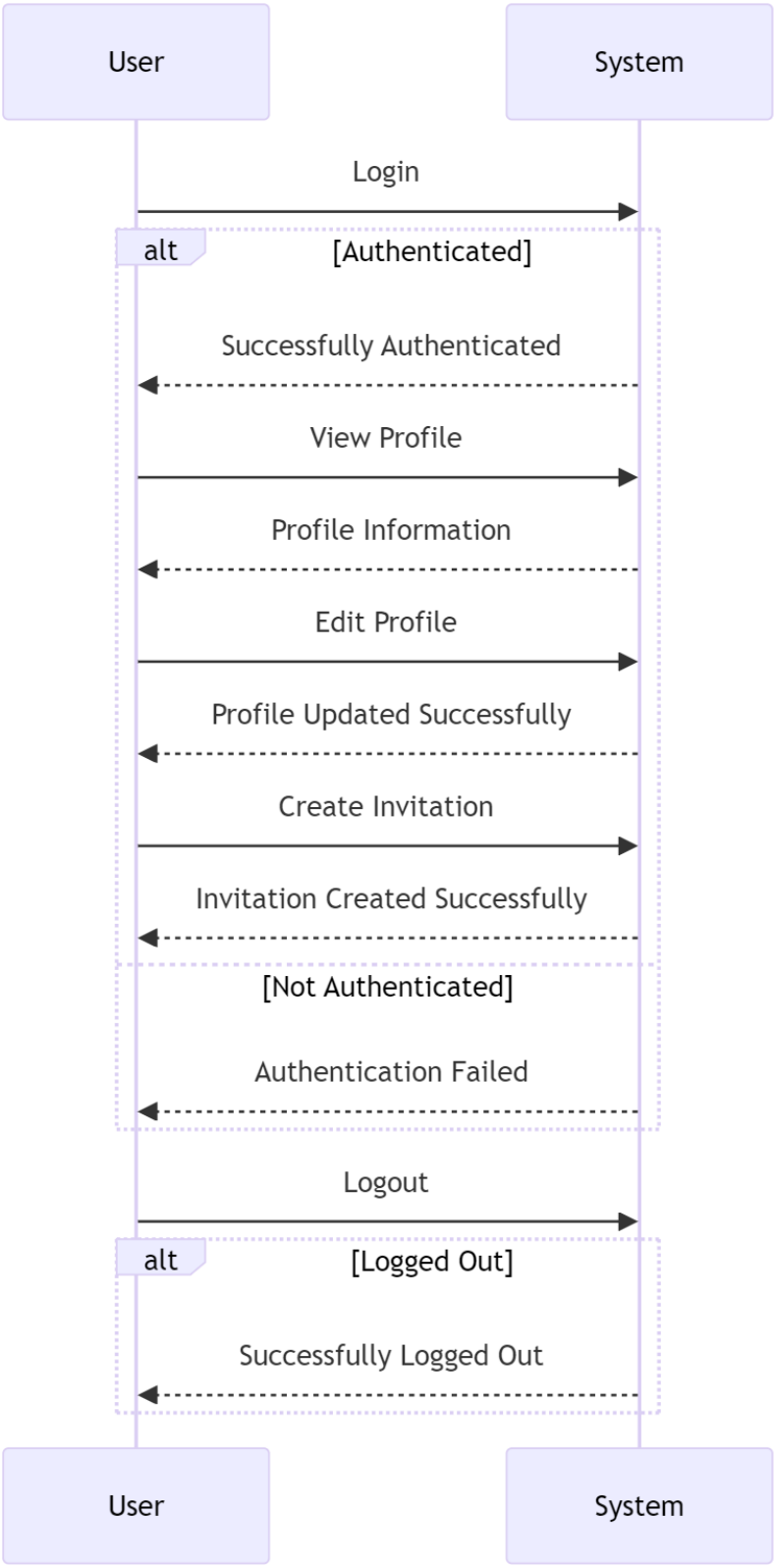

Рисунок Д.3 — Діаграма варіантів взаємодії користувача

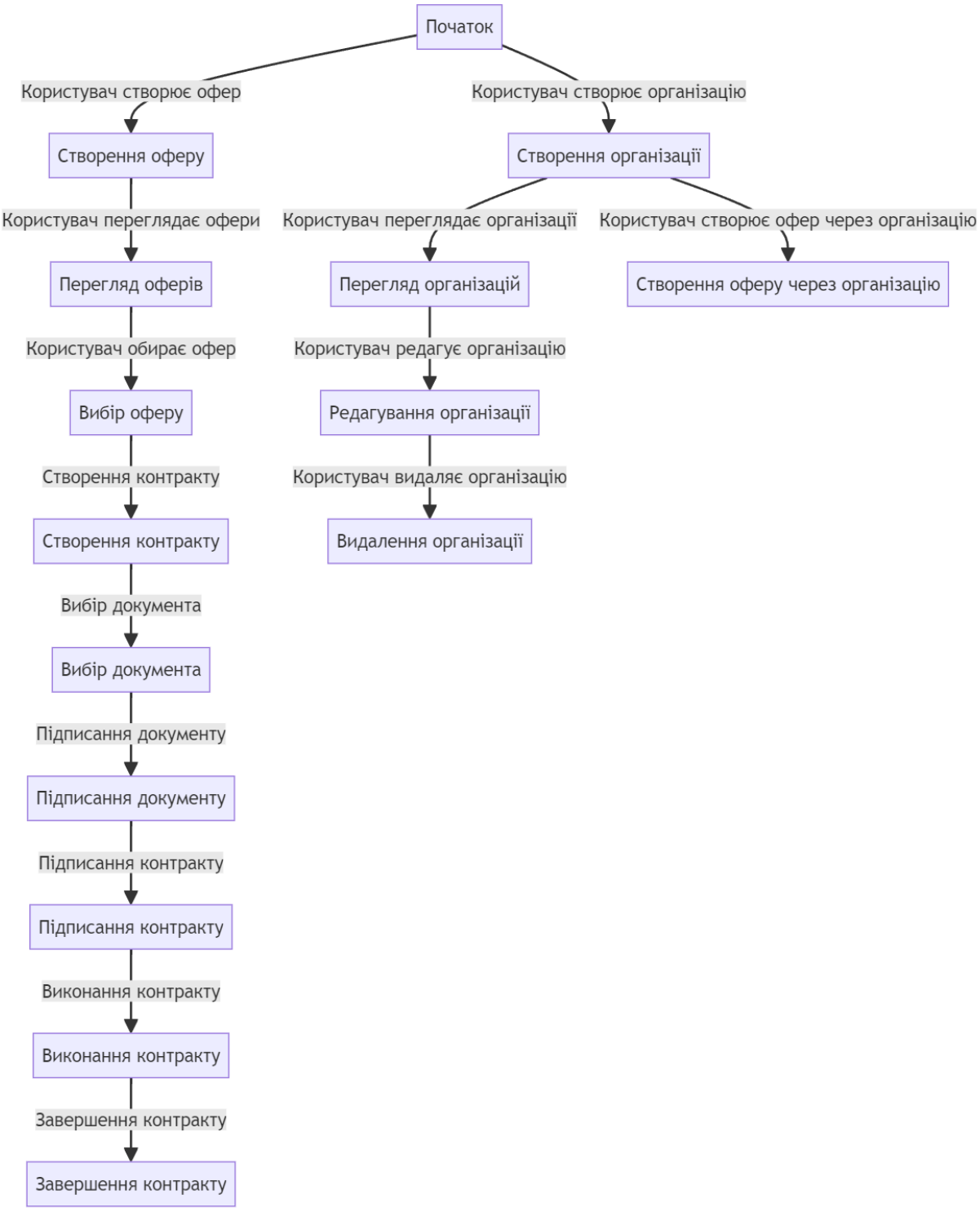

Рисунок Д.4 — Діаграма активності додатку