Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного забезпечення для автоматизації бізнес-процесів студії звукозапису»

Виконав: студент IV курсу
групи 1ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Янголь М. О.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Ткаченко О. М.
(прізвище та ініціали)

Рецензент:. к.т.н., доц. каф ОТ Кадук О. В.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ Романюк О. Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
ФОП Михальнюк О. О.

«21» березня 2025 року

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор Романюк О. Н.

«21» березня 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Янголя Максима Олеговича

1. Тема роботи – Розробка програмного забезпечення для автоматизації бізнес-процесів студії звукозапису.
Керівник роботи: Ткаченко Олександр Миколайович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від «20» березня 2025 р. № 96.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: модуль для створення, редагування клієнтів, проєктів, ліцензій, модуль генерації та відправки ліцензій, модуль фінансового обліку.
4. Зміст текстової частини: аналіз сучасних рішень для автоматизації бізнес-процесів у студіях звукозапису; постановка задачі; проєктування архітектури системи; розробка інтерфейсу користувача; розробка модулів управління клієнтами, ліцензіями, фінансами, проєктами; створення та тестування REST API; реалізація генерації PDF-документів; тестування функціоналу системи; розробка інструкції користувача; вибір інструментів розробки та обґрунтування технологій; фінальні висновки щодо ефективності системи.
5. Перелік ілюстративного матеріалу: інтерфейс StudioCloud, Trello, Sound Studio Management; діаграми діяльності; блок-схеми алгоритмів генерації ліцензій; REST API ендпоінти; знімки екрана інтерфейсу (проєкти, клієнти, ліцензії, фінанси); UML-діаграми; структура PDF-документа ліцензії; структура запитів до бази даних.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ткаченко О. М., к.т.н., доцент кафедри ПЗ	24.03.2025	01.06.2025

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Аналіз стану технологій автоматизації бізнес процесів у студіях звукозапису	25.03. 2025 – 02.04.2025	вик
2.	Розробка моделі системи та методів програмного застосунку	03.04.2025 – 14.04.2025	вик
3.	Розробка програмних модулів програмного застосунку	15.04.2025 – 02.05.2025	вик
4.	Тестування і практичне застосування системи	03.05.25 – 26.05.25	вик
5.	Оформлення матеріалів до захисту БДР	27.05.25 - 30.05.25	вик

Студент (підпис) **Янголь М. О.** (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи (підпис) **Ткаченко О. М.** (прізвище та ініціали)

АНОТАЦІЯ

УДК 681.12

Янголь М.О. Розробка програмного забезпечення для автоматизації бізнес-процесів студії звукозапису.

121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 111 с.

На укр. мові. Бібліогр.:24 назв; рис.: 40; табл. 3.

У бакалаврській кваліфікаційній роботі було розроблено програмні засоби для автоматизації бізнес-процесів студії звукозапису. Веб-додаток включає розробку модулів для управління базою клієнтів, обліку ліцензій, автоматизованої відправки ліцензій, генерацію, фінансову аналітику, а також менеджменту проєктів. Додаток призначений для оптимізації робочих процесів студії, покращення контролю за ліцензіями та спрощення взаємодії з клієнтами.

Програмні модулі, розроблені в проєкті, можуть бути використані звукозаписними студіями, незалежними продюсерами та іншими організаціями, що працюють із музичними та аудіовізуальними проєктами.

Додаток розроблено з використанням мови програмування Java та фреймворку Spring для бекенду, а також TypeScript для фронтенду з фреймворком Next.js. Для збереження даних застосовується база даних MongoDB.

ABSTRACT

Yanhol M.O. Development of software for automation of business processes in a recording studio. Vinnytsia: VNTU, 2025. 111 p.

In Ukrainian language. Bibliographer: 24 titles; fig.: 40; table. 3.

Bachelor's thesis, was developed software tools to automate the business processes of a recording studio. The web application includes the development of modules for customer base management, licence accounting, automated licence dispatch, generation, financial analytics, and project management. The application is designed to optimise the studio's workflows, improve control over licences, and simplify interaction with customers.

The software modules developed in the project can be used by recording studios, independent producers, and other organisations working with music and audiovisual projects.

The application was developed using the Java programming language and the Spring framework for the backend, as well as TypeScript for the frontend with the Next.js framework. The MongoDB database is used to store data.

Keywords: automation, business processes, recording studio, software

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження.

Сучасний розвиток технологій вимагає автоматизації бізнес-процесів для підвищення ефективності роботи студій звукозапису, особливо з урахуванням стрімкої цифрової трансформації у сфері музичної індустрії. Управління клієнтською базою, облік ліцензій, відправка документів та менеджмент проєктів є ключовими аспектами роботи студії, які потребують надійного та функціонального програмного забезпечення [1].

Багато студій звукозапису стикаються з труднощами, пов’язаними з ручним виконанням повторюваних завдань, таких як: обробка заявок клієнтів, облік ліцензії на музичні композиції, менеджмент проєктів та фінансовий облік. Це призводить до зайвих витрат часу, зниження ефективності роботи та підвищення ризику помилок. У цьому контексті розробка спеціалізованої системи автоматизації дозволить значно спростити щоденні операції, зменшити навантаження на персонал і прискорити обробку інформації [2].

Більшість наявних рішень для автоматизації студій є дорогими або недостатньо гнучкими для адаптації під конкретні бізнес-процеси, що ускладнює їхню інтеграцію в індивідуальні робочі середовища. Саме тому актуальною є розробка власної системи, яка забезпечить зручне управління клієнтами, автоматизацію роботи з ліцензіями, ефективний обмін документами та спрощену взаємодію між учасниками проєктів [3].

Зв’язок роботи з науковими програмами, планами, темами.

Дослідження було проведено згідно з планом наукових досліджень на кафедрі програмного забезпечення.

Мета та задачі дослідження. Метою бакалаврської кваліфікаційної роботи є розширення функціональних можливостей системи автоматизації студії звукозапису, яка оптимізує процеси обслуговування клієнтів, ліцензування, документообігу та управління проєктами.

Для досягнення поставленої мети необхідно розв’язати такі задачі:

- розробити зручний та інтуїтивно зрозумілий інтерфейс для ефективного управління замовленнями, проектами, клієнтами, фінансами і ліцензіями;
- розробити модуль фінансової системи, забезпечуючи точність обліку, генерацію звітів;
- розробити модуль для планування та організації роботи, відстеження прогресу і управління проєктами;
- реалізувати модуль для зберігання інформації про проєкти, фінанси, клієнтів, фінансів, ліцензій та взаємодії користувача між ними;
- об'єднати всі модулі в єдину платформу для ефективної автоматизації всіх бізнес-процесів студії.
- провести всебічне тестування для забезпечення стабільності та відповідності вимогам системи.

Об'єкт дослідження - процес автоматизації управління клієнтами, ліцензіями та проєктами у студії звукозапису.

Предмет дослідження - методи та засоби автоматизації управління клієнтами, ліцензіями та проєктами у студії звукозапису

Методи досліджень. В процесі виконання роботи використовувалися методи системного аналізу для виділення основних функціональних вимог до системи, обґрунтування структури та взаємозв’язків компонентів; методи обробки баз даних для створення сховища для збереження даних; методи проектування програмного забезпечення з застосуванням принципів SOLID для розробки REST-API та формування структури клієнт-серверної взаємодії; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Отримав подальшого розвитку метод автоматизації документообігу в студіях звукозапису, який відрізняється від існуючих застосуванням автоматичної генерації PDF-документів і ліцензійних угод на основі динамічних шаблонів, що дозволило прискорити процес створення документів і зменшити кількість ручних операцій порівняно з традиційними методами.

2. Отримав подальшого розвитку метод аналізу динаміки росту доходу в системах фінансової аналітики студій звукозапису, який відрізняється від існуючих підходів урахуванням сезонних коливань доходів і витрат із застосуванням корекції на основі статистичних даних про чистий прибуток, що дозволило підвищити точність аналізу фінансових показників.

Практичне значення отриманих результатів. Розроблено алгоритми та програмні модулі, що дозволяють користувачам ефективно управляти клієнтами, проєктами, ліцензіями та фінансами. Реалізовані програмні рішення можуть бути використані як основа для створення подібних систем автоматизації. Окремі компоненти можуть бути адаптовані іншими розробниками для впровадження у власні програмні продукти.

Особистий внесок здобувача.

Усі наукові результати, викладені у бакалаврській кваліфікаційної роботі, отримані автором особисто. Наукові праці опубліковано одноосібно.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні наукові результати, представлені в бакалаврській кваліфікаційній роботі, були висвітлені на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (м. Вінниця, 2025 р.) та на XIII Міжнародній науково-практичній конференції з інформаційних систем і технологій Infocom Advanced Solutions 2025 (м. Київ, 2025 р.).

Публікації. Ключові результати досліджень були оприлюднені в матеріалах LIV Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету та XIII Міжнародної науково-практичної конференції з інформаційних систем і технологій Infocom Advanced Solutions 2025 [4, 5].

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ АВТОМАТИЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ У СТУДІЯХ ЗВУКОЗАПИСУ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану технологій автоматизації бізнес-процесів у студіях звукозапису

Розвиток інформаційних технологій вимагає постійного вдосконалення методів управління бізнес-процесами, зокрема в сфері музичній індустрії. Традиційні методи організації роботи студій звукозапису, які часто базуються на ручному введенні даних та стандартних офісних рішеннях, можуть бути обмеженими через повільну обробку інформації та високий ризик помилок. У цьому контексті, впровадження спеціалізованих програмних модулів для автоматизації управління клієнтською базою, ліцензіями, фінансами та проєктами набуває особливої актуальності.

Одним із ключових факторів розвитку цих технологій є зростання потужності обчислювальних систем та застосування технологій, які дозволяють оперативно обробляти великі обсяги даних. Сучасні платформи забезпечують надійне зберігання, швидке обчислення та інтеграцію різноманітних даних, що є необхідним для комплексного управління бізнес-процесами студії звукозапису.

Програмні рішення, розроблені для автоматизації даних процесів, надають значні переваги у порівнянні з традиційними методами. Головною перевагою є можливість інтеграції різних функціональних модулів в єдину систему. Це сприяє підвищенню ефективності, валідації помилок та автоматизація робочих процесів.

Сучасні програмні модулі можуть також включати аналітичні інструменти, що дозволяють проводити детальний аналіз ефективності роботи студії, відслідковувати дохід і приймати обґрунтовані управлінські рішення на основі отриманих даних. Це забезпечує не лише покращення внутрішніх процесів, але й значні конкурентні переваги на ринку аудіо послуг.

Отже, аналіз сучасних технологій автоматизації бізнес-процесів у студіях звукозапису підтверджує необхідність впровадження інноваційних програмних

засобів. Використання сучасних технологій сприяє створенню високоефективних систем управління, що оптимізують робочі процеси, знижують витрати і підвищують якість надання послуг у сфері звукозапису.

1.2 Порівняльний аналіз аналогів

На ринку існує кілька готових рішень для студій звукозапису, які допомагають автоматизувати бізнес-процеси, організувати клієнтську базу, управління замовленнями, проєктами тощо. Ось кілька популярних варіантів: StudioCloud, Trello, Sound Studio Management.

StudioCloud – це хмарне рішення для автоматизації студій звукозапису. Воно допомагає управляти клієнтами, замовленнями та фінансами, спрощуючи адміністративні завдання [6]. Завдяки хмарним технологіям забезпечує доступ до даних з будь-якого пристрою. Основними перевагами, є хмарний доступ із будь-якого пристрою, управліннями клієнтами, замовленнями тощо. Основними недоліками, є обмеження безкоштовної версії та не має конкретну спеціалізацію на роботу в студії звукозапису. На рисунку 1.1 зображено інтерфейс додатку StudioCloud.

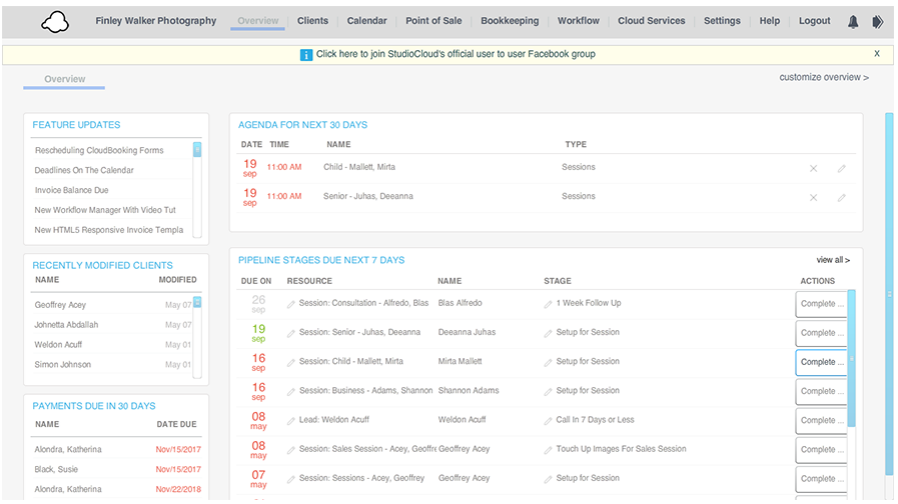


Рисунок 1.1 – Інтерфейс додатку StudioCloud

Trello – це інструмент для управління завданнями, який використовує канбан-дошку [7]. Для студії звукозапису його можна використовувати для організації

сесій, планування записів, управління проектами та комунікації з клієнтами. Переваги Trello включають легкість у використанні, візуалізацію завдань, зручність для командної роботи. Однак безкоштовна версія має обмежений функціонал, і Trello може бути недостатнім для великих проєктів, а також не має вбудованих функцій для фінансового обліку. На рисунку 1.2 зображено інтерфейс додатку Trello.

Рисунок 1.2 – Інтерфейс додатку Trello

Sound Studio Management – це програмне забезпечення, яке допомагає автоматизувати роботу студій звукозапису [8]. Воно дозволяє організовувати управління проектами, клієнтами, замовленнями та фінансами. Для студії звукозапису ця програма є зручним інструментом для планування сесій, відстеження прогресу замовлень і взаємодії з клієнтами. Програма має переваги в тому, що дозволяє зберігати всю інформацію, спрощує облік фінансів і покращує комунікацію між працівниками студії. Але основним недоліком є те, що ця програма доступна лише для операційної системи macOS, що обмежує її використання для користувачів Windows або інших платформ. На рисунку 1.3 зображено інтерфейс Sound Studio Management.

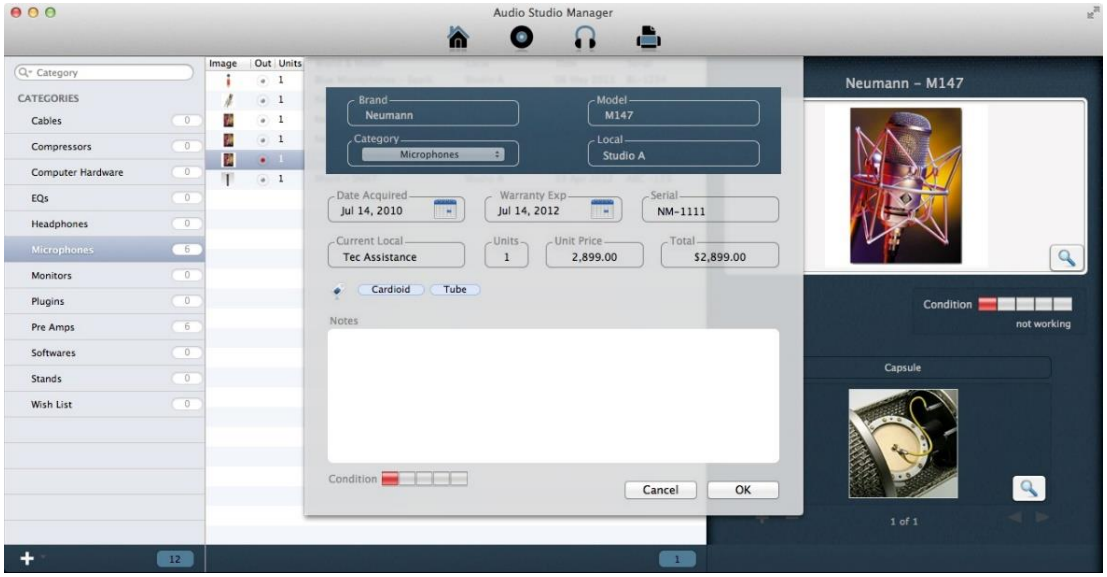


Рисунок 1.3 – Sound Studio Management

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	Sound Studio Management	Trello	StudioCloud	SoundWave
Управління клієнтами	+	-	+	+
Фінансова аналітика	+	-	+	+
Управління проєктами	+	+	+	+
Генерація ліцензій	-	-	-	+
Звітність	+	-	-	+
Загальна оцінка	80%	40%	60%	100%

Згідно з порівнянням різних програмних рішень для автоматизації бізнес-процесів студії звукозапису, розробка власного програмного модуля є найбільш доцільною для забезпечення необхідних функцій і максимізації ефективності.

Отже, розробка індивідуального рішення не тільки дозволяє покрити всі недоліки існуючих програм, але й відкриває можливості для додавання нових функцій, які можуть підвищити ефективність роботи студії. Враховуючи високий рівень контролю та адаптивності, власний програмний засіб дозволить значно підвищити продуктивність і забезпечити більш зручну роботу з усіма аспектами бізнес-процесів студії звукозапису.

1.3 Аналіз методів розв’язання задачі

Розробка програмних модулів для автоматизації бізнес-процесів студії звукозапису вимагає комплексного підходу до вирішення завдань, пов'язаних із управлінням замовленнями, клієнтами, проектами, фінансами та іншими аспектами діяльності студії. Існують різні методи автоматизації, кожен з яких має свої переваги та недоліки.

Один з основних аспектів – це організація ефективного процесу управління замовленнями та клієнтами та проектами. Для цього важливо створити зручну систему, що дозволяє менеджерам студії швидко обробляти замовлення, контролювати виконання проектів і моніторити фінансові потоки. Використання зручних інтерфейсів для відстеження всіх етапів роботи з клієнтами та виконання проектів є основним елементом цієї системи.

Іншим важливим аспектом є стандартизація процесів. Це включає в себе стандарти для управління проектами, взаємодії з клієнтами, укладання контрактів, фінансові операції тощо. Встановлення чітких стандартів дозволяє забезпечити точність і відповідність усіх етапів роботи з клієнтами та студією. Крім того, стандартизація процесів дозволяє значно зменшити ймовірність помилок і забезпечити кращу організацію роботи в студії. Не менш важливим є інтеграція програмних модулів з іншими системами, Це дозволяє створити єдину платформу

для всіх операцій, забезпечуючи ефективне управління і контролювання всіх аспектів роботи студії.

Однак жоден підхід не є ідеальним без допрацювань та адаптацій до специфічних потреб студії звукозапису. Кожен етап розробки вимагає ретельного аналізу і налаштування для досягнення оптимального результату [9]. Важливою складовою є постійне вдосконалення системи.

1.4 Постановка задач бакалаврської кваліфікаційної роботи

Оцінка переваг і недоліків існуючих програмних рішень для автоматизації бізнес-процесів студії звукозапису дозволила визначити низку завдань, які слід вирішити для ефективної розробки програмного забезпечення:

- розробити зручний та інтуїтивно зрозумілий інтерфейс для ефективного управління замовленнями, проектами, клієнтами, фінансами і ліцензіями;
- розробити модуль фінансової системи, забезпечуючи точність обліку, генерацію звітів;
- розробити модуль для планування та організації роботи, відстеження прогресу і управління проєктами;
- реалізувати модуль для зберігання інформації про проєкти, фінанси, клієнтів, фінансів, ліцензій та взаємодії користувача між ними;
- об'єднати всі модулі в єдину платформу для ефективної автоматизації всіх бізнес-процесів студії.
- провести всебічне тестування для забезпечення стабільності та відповідності вимогам системи.

1.5 Висновки

Розвиток інформаційних технологій в сфері автоматизації бізнес-процесів студій звукозапису є важливим етапом для підвищення ефективності їх роботи, зменшення ймовірності помилок та покращення якості надання послуг. Існуючі програмні рішення мають свої переваги та недоліки, і хоч деякі з них надають базову автоматизацію, все ж існує потреба в розробці власних спеціалізованих

інструментів, які б задовольняли всі специфічні вимоги студій звукозапису.

Аналіз наявних програмних засобів показав, що розробка індивідуального програмного рішення є оптимальним варіантом для студії звукозапису, оскільки це дозволяє врахувати всі вимоги, забезпечити високий рівень інтеграції та адаптації до конкретних бізнес-процесів. Власні модулі для управління клієнтами, фінансами, проектами, ліцензіями та іншими аспектами діяльності студії дозволяють автоматизувати процеси, зменшити час обробки даних і підвищити ефективність.

Завдяки комплексному підходу до розробки та інтеграції різних функціональних модулів, можна створити систему, яка забезпечить оптимальне управління всіма бізнес-процесами студії звукозапису, а також забезпечить високий рівень контролю та зручність у використанні для співробітників студії. Наявність централізованої бази даних дозволяє забезпечити швидкий доступ до необхідної інформації, покращити комунікацію між підрозділами та забезпечити надійне збереження даних.

Подальше вдосконалення системи дозволить підтримувати її стабільну роботу та відповідність змінним вимогам ринку. Постійна адаптація до нових технологій, а також можливість розширення функціоналу за потреби, сприятимуть підвищенню конкурентоспроможності студії на ринку.

Таким чином, розробка програмного забезпечення для автоматизації бізнес-процесів студії звукозапису є важливим кроком у розвитку індустрії, сприяючи не лише підвищенню ефективності, але й наданню високоякісних послуг клієнтам. Вона забезпечує студії стратегічну перевагу, допомагаючи швидше реагувати на запити клієнтів, знижувати операційні витрати, підвищувати рівень обслуговування та оптимізувати використання ресурсів. Інвестиції у створення і розвиток власного ПЗ є довгостроковим вкладенням у стабільність і зростання бізнесу.

2 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних даних системи

Для реалізації застосунку для автоматизації бізнес-процесів студії звукозапису будуть використані мови програмування Java з фреймворком Spring Boot, TypeScript з фреймворком Next.js, а також база даних MongoDB. Java – це об’єктно-орієнтована мова програмування, яка використовується для створення надійних і масштабованих серверних застосунків через віртуальну машину Java (JVM). Spring Boot – це фреймворк на основі Java, який спрощує розробку веб-додатків, надаючи готові інструменти для роботи з API, базами даних і конфігурацією серверів [10]. TypeScript – це надбудова над JavaScript із статичною типізацією, яка ідеально підходить для створення сучасних інтерактивних інтерфейсів у поєднанні з Next.js, фреймворком для швидкої розробки веб-додатків [11]. MongoDB – це документо-орієнтована NoSQL база даних, яка забезпечує гнучке зберігання даних і швидкий доступ до них, що ідеально підходить для управління інформацією про проєкти, клієнтів і фінанси [12].

Розробка програмного забезпечення для автоматизації бізнес-процесів студії звукозапису передбачає кілька етапів. Спочатку буде створено модуль із зручним та інтуїтивно зрозумілим інтерфейсом на базі TypeScript і Next.js, який відповідатиме за ефективне управління замовленнями, проектами, клієнтами, фінансами та ліцензіями. Цей модуль забезпечить просту навігацію та швидкий доступ до функціоналу системи. Далі буде розроблено модуль фінансової системи на Spring Boot, який інтегруватиметься з MongoDB для точного обліку, автоматичної генерації звітів і управління фінансовими даними студії.

Наступним кроком стане створення модуля для планування та організації роботи, який дозволить відстежувати прогрес проєктів, керувати завданнями та координувати діяльність команди. Також, буде розроблено модуль для зберігання та управління інформацією про проєкти, фінанси, клієнтів і ліцензії, забезпечуючи швидкий доступ до даних і їх взаємодію в системі.

Усі модулі будуть об’єднані в єдину платформу, яка забезпечить комплексну автоматизацію бізнес-процесів студії звукозапису. Для розробки будуть використані інструменти, такі як IntelliJ IDEA для роботи з Java і Spring Boot, а також інтегроване середовище для TypeScript і Next.js. База даних MongoDB слугуватиме для ефективного зберігання та обробки інформації. Після завершення розробки буде проведено всебічне тестування системи, щоб переконатися в її стабільності, безпеці та відповідності вимогам.

Отже, розробка програмного забезпечення для автоматизації бізнес-процесів студії звукозапису на базі Java з Spring Boot, TypeScript з Next.js і MongoDB є складним завданням, яке вимагає створення кількох взаємопов’язаних модулів. Успіх проєкту залежить від якісного проектування, інтеграції всіх компонентів і ретельного тестування. Завдяки використанню IntelliJ IDEA та сучасних технологій, таке програмне забезпечення зможе оптимізувати бізнес-процеси студії, підвищити ефективність роботи та задовольнити потреби користувачів [13].

2.2 Розробка моделі системи

Для створення програмного забезпечення, призначеного для автоматизації бізнес-процесів студії звукозапису, необхідно ретельно продумати його архітектуру, функціонал і взаємозв’язки між різними компонентами. На початковому етапі розробки було проведено детальний аналіз потреб системи, що допомогло визначити основні цілі, які має досягти програмний продукт.

Серед головних вимог: створення зручного інтерфейсу для управління проєктами, клієнтами та ліцензіями; фінансова аналітика із можливістю створення звітів; організація та контроль виконання проєктів; забезпечення надійного зберігання даних, генерація ліцензій та інтеграція з поштовим сервісом; об’єднання всіх функцій у єдину систему, а також гарантування її стабільності через ретельне тестування. Ці критерії лягли в основу проектування структури застосунку.

Особливу увагу приділено інтеграції всіх функціональних блоків в єдину цілісну систему. Важливо було досягти не просто сукупності окремих модулів, а забезпечити їх повноцінну взаємодію, що дає змогу об’єднати всі бізнес-процеси

студії у злагоджений цифровий механізм. Крім того, однією з ключових вимог було гарантування стабільної та безпечної роботи програмного забезпечення за рахунок ретельного тестування та впровадження сучасних підходів до контролю якості.

Створене програмне забезпечення базується на принципі модульності, який є одним із найефективніших підходів у сучасній розробці. Це забезпечує гнучкість системи, її легку адаптацію до нових вимог, а також можливість розширення функціоналу без суттєвого впливу на основну архітектуру. Кожен модуль системи виконує чітко визначену роль, що полегшує як підтримку, так і подальший розвиток проєкту [14].

На рисунку 2.6 зображено діаграму діяльності, яка ілюструє загальну логіку роботи користувача із системою та послідовність обробки основних бізнес-процесів.

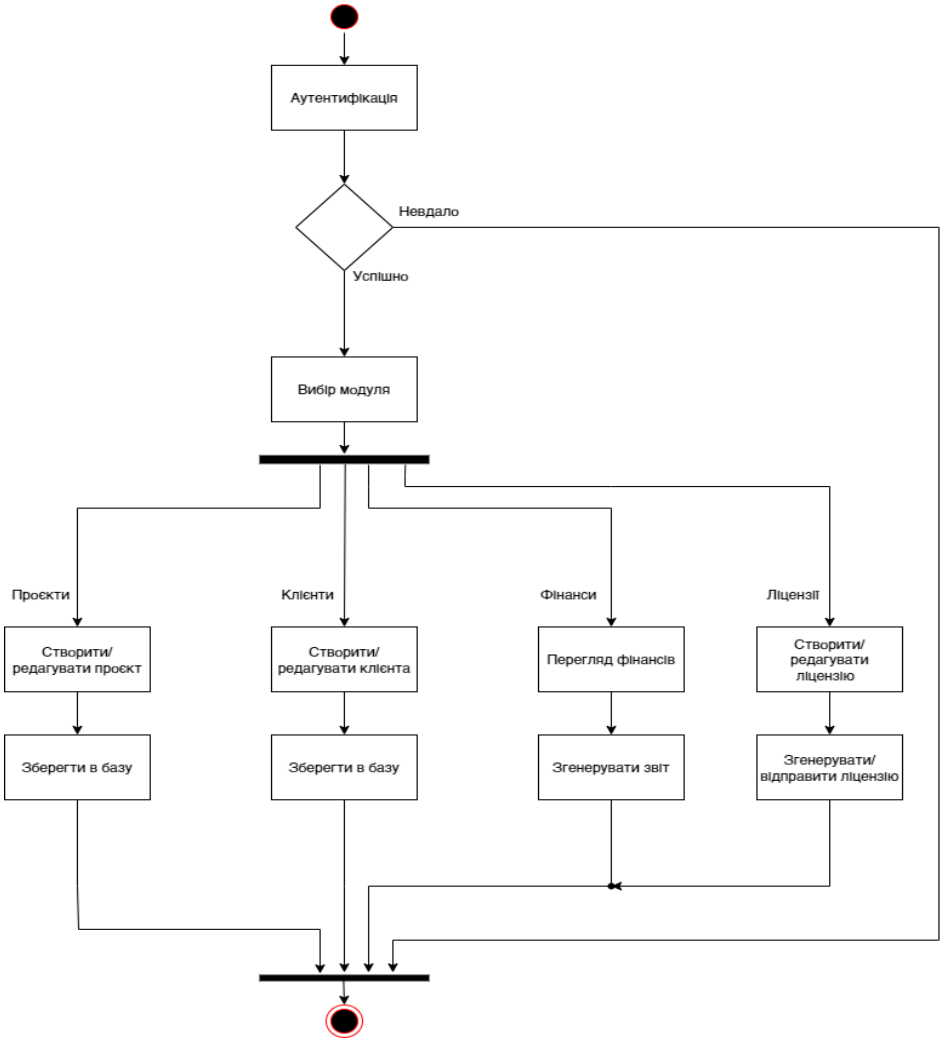


Рисунок 2.6 – Діаграма діяльності

Кожен модуль відповідає за певну задачу: інтерфейс користувача, фінансовий облік, планування проєктів, управління даними тощо. Для реалізації проєкту обрано мову програмування Java разом із фреймворком Spring Boot для серверної логіки, TypeScript із фреймворком Next.js для фронтенду, а також базу даних MongoDB для зберігання інформації. Spring Boot спрощує розробку серверної частини, надаючи інструменти для створення REST API, управління залежностями та інтеграції з MongoDB. Next.js забезпечує швидке створення інтерактивних інтерфейсів із підтримкою серверного рендерингу, що покращує швидкість роботи та зручність для користувача. MongoDB, як NoSQL база даних, дозволяє гнучко зберігати дані про проєкти, клієнтів, фінанси та ліцензії, забезпечуючи швидкий доступ до інформації.

Для створення звітів, зокрема тих, що стосуються фінансових операцій і ліцензій, у системі було використано бібліотеку iTextPDF. Ця бібліотека дає змогу генерувати PDF-документи програмним шляхом, що є важливим для автоматизації звітності студії звукозапису. iTextPDF підтримує форматування тексту, вставку таблиць, графіків і налаштування дизайну документа, що робить звіти зручними для аналізу та друку [15].

Отже, створення системи передбачало ґрунтовний підхід до вивчення вимог, розробки архітектури та забезпечення зручного інтерфейсу. Застосування Java з Spring Boot, TypeScript з Next.js, MongoDB і бібліотеки iTextPDF дозволило розробити ефективне та гнучке рішення для автоматизації бізнес-процесів студії звукозапису, яке відповідає всім заявленим потребам і забезпечує комфортну роботу для користувачів.

2.3 Розробка методу автоматизації документообігу

До впровадження вдосконаленого методу автоматизації документообігу в студіях звукозапису процес створення та обробки ліцензійних угод був переважно ручним. Зазвичай студії використовували статичні шаблони документів, які заповнювалися вручну на основі даних клієнтів і проєктів. Після цього готові

документи надсилалися клієнтам через електронну пошту або месенджери, що вимагало значних затрат часу.

Основні проблеми цього підходу включали:

- високу ймовірність людських помилок під час заповнення шаблонів, наприклад, неправильне введення даних про клієнта, вартості ліцензії чи умов угоди;
- трудомісткість процесу, який міг займати багато часу на створення однієї ліцензії, що ставало критичним при великій кількості клієнтів;
- відсутність централізованого зберігання документів, що ускладнювало їх пошук і відстеження історії змін;
- відсутність інтеграції з іншими модулями, що призводило до дублювання даних і зниження ефективності роботи.

Для вирішення зазначених проблем було розроблено вдосконалений метод автоматизації документообігу. Цей метод дозволяє створювати, редагувати, зберігати та відправляти ліцензійні угоди у форматі PDF, використовуючи динамічні шаблони, які автоматично заповнюються даними з бази даних. Автоматизований процес значно скорочує час, необхідний для створення документів, зменшує ймовірність помилок і забезпечує централізоване зберігання всіх документів у базі даних. Інтеграція з поштовим сервісом дозволяє автоматично надсилати документи клієнтам, що підвищує ефективність взаємодії та забезпечує прозорість документообігу.

Метод автоматизації документообігу було реалізовано через кілька ключових етапів, які забезпечують його ефективність і надійність:

- проведення аналізу вимог документів;
- розробка модуля генерації документів;
- інтеграції з поштовим сервісом;
- валідація та безпека даних.

У результаті розробки вдосконаленого методу автоматизації документообігу створено ефективний модуль, який значно оптимізував процес створення, зберігання та надсилання ліцензійних угод у студіях звукозапису. Використання

бібліотеки iTextPDF забезпечило гнучке створення структурованих PDF-документів, а інтеграція з JavaMailSender дозволила автоматизувати відправку документів клієнтам.

2.4 Розробка методу аналізу динаміки росту доходу

До впровадження запропонованого методу аналізу динаміки росту доходу в студіях звукозапису фінансовий аналіз зазвичай проводився вручну або з використанням базових інструментів, таких як електронні таблиці (наприклад, Microsoft Excel або Google Sheets). Аналіз доходів часто обмежувався підрахунком загальної суми доходів за місяць без урахування сезонних коливань, які суттєво впливають на попит на послуги студії, наприклад, зростання замовлень у період концертних сезонів (весна та осінь) або спад у зимові місяці. Такий підхід мав низку недоліків:

- ручна обробка даних призводила до помилок у розрахунках, таких як неправильне підсумовування або ігнорування сезонних факторів;
- підготовка фінансових звітів займала значний час (від 30 хвилин до кількох годин залежно від обсягу даних), що знижувало оперативність прийняття управлінських рішень;
- дані про доходи та витрати часто зберігалися в різних джерелах, що ускладнювало їх консолідацію та аналіз.

Для подолання зазначених обмежень було розроблено вдосконалений метод аналізу динаміки росту доходу. Цей метод забезпечує автоматизований збір, обробку та аналіз фінансових даних, враховуючи сезонні коливання, що дозволяє студіям звукозапису отримувати точні та оперативні фінансові звіти. Метод реалізовано як частину модуля фінансової аналітики, який інтегрується з іншими компонентами системи.

Етапи реалізації методу аналізу динаміки росту доходу:

- збір даних через REST API;
- побудова сезонного індексу;
- корекція доходів і витрат із урахуванням сезонності.

Для побудови сезонного індексу було розроблено алгоритм, який визначає, наскільки дохід або витрати в певному місяці відрізняються від середньорічного значення.

Алгоритм включає:

- збір історичних даних про доходи за попередні роки (якщо такі є в базі) або за поточний рік;
- розрахунок середньомісячного доходу за формулою:

$$\text{Середній дохід} = \frac{\text{Загальний річний дохід}}{\text{Кількість місяців}};$$

- визначення сезонного індексу для кожного місяця:

$$\text{Сезонний індекс}_m = \frac{\text{Дохід за місяць}}{\text{Середній дохід}},$$

де m – номер місяця;

- нормалізація індексу для уникнення аномалій, наприклад, при нульовому доході в окремих місяцях.

На основі сезонного індексу коригуються доходи для отримання більш точних фінансових прогнозів. Наприклад, якщо сезонний індекс для травня становить 1,5 (на 50% вище середнього), дохід за травень множиться на коефіцієнт корекції, щоб врахувати підвищений попит. Для місяців із низьким попитом наприклад, лютий із індексом 0,8 дохід нормалізується для уникнення завищених прогнозів.

Запропонований метод аналізу динаміки росту доходу, реалізований у системі, значно покращив процес фінансового аналізу в студіях звукозапису. Завдяки автоматизованому збору даних через REST API та урахуванню сезонних коливань.

2.5 Розробка алгоритмів роботи системи

Для створення програмного забезпечення, призначеного для автоматизації бізнес-процесів студії звукозапису було розроблено загальний алгоритм функціонування системи, схему якого представлено на рисунку 2.7.

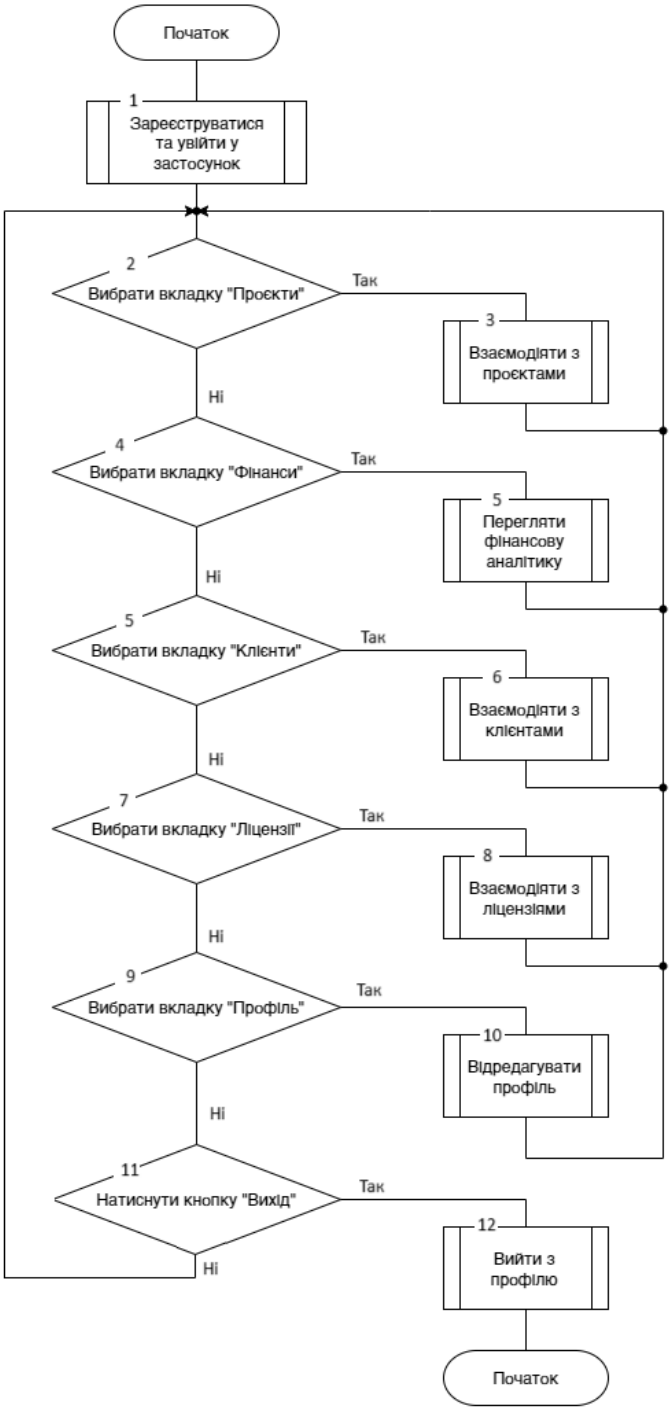


Рисунок 2.7 – Схема загального алгоритму роботи застосунку

На представленій схемі, зображено алгоритм взаємодії користувача із програмним забезпеченням після реєстрації та входу в застосунок. Після авторизації користувач потрапляє до головного інтерфейсу, де має змогу обирати різні функціональні вкладки, що відповідають за основні модулі системи. Зокрема, передбачена можливість вибору вкладки «Проєкти», яка дозволяє взаємодіяти з поточними проєктами, керувати ними або переглядати їхній стан. У випадку

вибору вкладки «Фінанси», користувач отримує доступ до перегляду фінансової аналітики та відповідної інформації. Вкладка «Клієнти» надає змогу керувати взаємодією з клієнтами системи, а вкладка «Ліцензії» дозволяє працювати з ліцензійними даними. Крім цього, користувач може перейти до розділу «Профіль», де можливо відредагувати особисту інформацію або налаштування облікового запису. На будь-якому етапі роботи із застосунком передбачена можливість натиснути кнопку «Вихід», що призводить до завершення сесії та повернення до початкового етапу. Блок-схема демонструє циклічність процесу — після виконання дії користувач завжди має змогу повернутися до вибору вкладок і продовжити роботу в системі. Такий підхід забезпечує гнучкість, зручність навігації та логічну послідовність взаємодії з інтерфейсом.

2.6 Висновки

У другому розділі було здійснено аналіз вхідних і вихідних даних програмного забезпечення для автоматизації бізнес-процесів студії звукозапису. Розроблено методи для автоматизації документообігу та фінансової аналітики, що дозволяють ефективно працювати з ліцензіями та доходами студії. Створено алгоритми, які визначають послідовність дій користувача в системі та забезпечують логіку її функціонування. Усі ці елементи стали основою для подальшої реалізації програмних модулів і перевірки їхньої працездатності в реальних умовах. Для моделювання роботи програми було використано UML діаграми [16].

3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ДЛЯ АВТОМАТИЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ СТУДІЇ ЗВУКОЗАПИСУ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Для розробки програмного забезпечення, призначеного для автоматизації бізнес-процесів студії звукозапису, необхідно ретельно підібрати технології, які забезпечать ефективну роботу всіх модулів системи. Java з фреймворком Spring Boot є оптимальним вибором для створення серверної частини, оскільки Spring Boot забезпечує швидку розробку REST API, зручне управління залежностями та інтеграцію з базами даних, такими як MongoDB [17]. TypeScript у поєднанні з фреймворком Next.js ідеально підходить для розробки клієнтської частини, дозволяючи створювати швидкі та інтерактивні інтерфейси з підтримкою серверного рендерингу. MongoDB, як NoSQL база даних, забезпечує гнучке зберігання даних про проєкти, клієнтів, фінанси та ліцензії, що є важливим для масштабованості системи.

Spring Boot – це потужний фреймворк на основі Java, який значно спрощує розробку серверних застосунків. Він надає готові інструменти для створення REST API, управління залежностями через Spring Dependency Injection, а також інтеграцію з базами даних, такими як MongoDB [18]. Spring Boot дозволяє швидко налаштувати проєкт завдяки вбудованій автоматичній конфігурації, що зменшує кількість шаблонного коду. У контексті автоматизації бізнес-процесів студії звукозапису Spring Boot забезпечує створення стабільної серверної частини, яка обробляє запити від клієнтського інтерфейсу, виконує бізнес-логіку тощо.

MongoDB – це сучасна NoSQL база даних, яка використовує документо-орієнтовану модель для зберігання даних. MongoDB зберігає дані у форматі JSON-подібних документів, що дозволяє легко працювати зі складними структурами даних, такими як інформація про проєкти, клієнтів, фінанси та ліцензії. Основною перевагою MongoDB є її здатність швидко обробляти великі обсяги даних і підтримувати горизонтальне масштабування, що забезпечує високу продуктивність

навіть при зростанні обсягу інформації. Крім того, MongoDB підтримує інтеграцію з Java через Spring Data MongoDB, що спрощує доступ до даних і їх обробку в серверній частині системи.

IntelliJ IDEA – це потужне інтегроване середовище розробки, яке підтримує розробку на Java, надаючи інструменти для автодоповнення коду, налагодження та інтеграції з Spring Boot і Next.js. Це середовище значно прискорює процес створення програмного забезпечення, дозволяючи розробникам зосередитися на логіці програми.

Cursor IDE – це інноваційне середовище розробки, яке використовує штучний інтелект для прискорення процесу написання коду. Воно підтримує мови програмування, такі як Java і TypeScript, і пропонує інтелектуальні функції, такі як автодоповнення коду на основі контексту, генерація коду за описом і автоматичне виправлення помилок [19].

Бібліотека iTextPDF використовується для генерації PDF-звітів, таких як фінансові звіти чи документи ліцензій. iTextPDF надає широкий набір функцій для створення структурованих документів, включаючи додавання таблиць, форматування тексту та налаштування стилів, що робить її ідеальним інструментом для автоматизації звітності в студії звукозапису.

React, який використовується в рамках Next.js, є бібліотекою для створення компонентно-орієнтованих інтерфейсів користувача. Вона дозволяє розробляти модульні та повторно використовувані компоненти, такі як таблиці ліцензій, форми для додавання проєктів чи фінансових звітів, забезпечуючи високу швидкість роботи інтерфейсу та зручність для користувача.

Поєднання MongoDB, Spring Boot, та Next.js створює потужний набір інструментів для розробки програмного забезпечення для автоматизації бізнес-процесів студії звукозапису. MongoDB забезпечує гнучке та швидке зберігання даних, Spring Boot спрощує створення серверної логіки, IntelliJ IDEA і Cursor IDE прискорюють процес розробки завдяки інтелектуальним функціям і зручному інтерфейсу. Разом ці технології дозволяють створити ефективну, масштабовану та зручну систему, яка відповідає потребам користувачів.

3.2 Розробка інтерфейсу програмного додатку

При розробці інтерфейсу програмного забезпечення особливу увагу було приділено його зрозумілості та зручності для користувача [20]. Основним кольором інтерфейсу обрано темний фон із відтінками чорного, що асоціюється з професійним середовищем студії звукозапису, а також сприяє зменшенню напруги очей під час тривалої роботи. Допоміжними кольорами обрано білий для тексту та фіолетовий для кнопок і акцентів, що створює контраст і полегшує сприйняття інформації.

Усі елементи інтерфейсу розташовані логічно та інтуїтивно зрозуміло, що дозволяє швидко орієнтуватися в системі навіть новим користувачам. Використання фірмового стилю сприяє формуванню впізнаваного бренду та підвищує загальне враження від роботи із застосунком.

При відкритті програмного застосунку користувача зустрічає сторінка аутентифікації, яка забезпечує захищений доступ до персоналізованих даних та функціоналу системи. На рисунку 3.1 зображено сторінку аутентифікації.

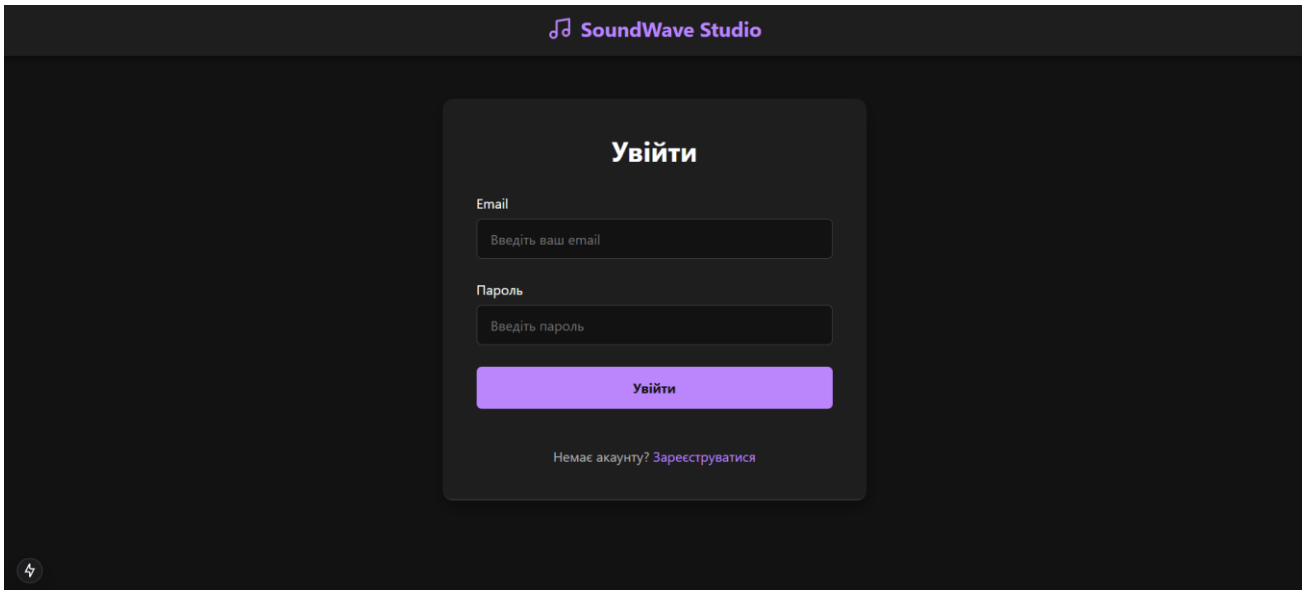


Рисунок 3.1 – Сторінка аутентифікації

Навігація у застосунку відіграє важливу роль, оскільки вона забезпечує швидкий і зручний доступ до всіх ключових функцій системи, дозволяючи користувачам зручного його використовувати. Містить у собі основні підрозділи

застосунку та поле з користувачем та кнопкою «Вихід». Також навігація присутня майже на всіх сторінках. На рисунку 3.2 зображено навігацію.

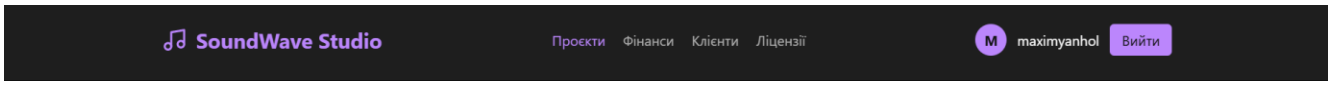


Рисунок 3.2 – Навігація

На рисунку 3.3 зображена сторінка «Проєкти». Користувач має можливість редагувати проєкти, створювати їх та редагувати.

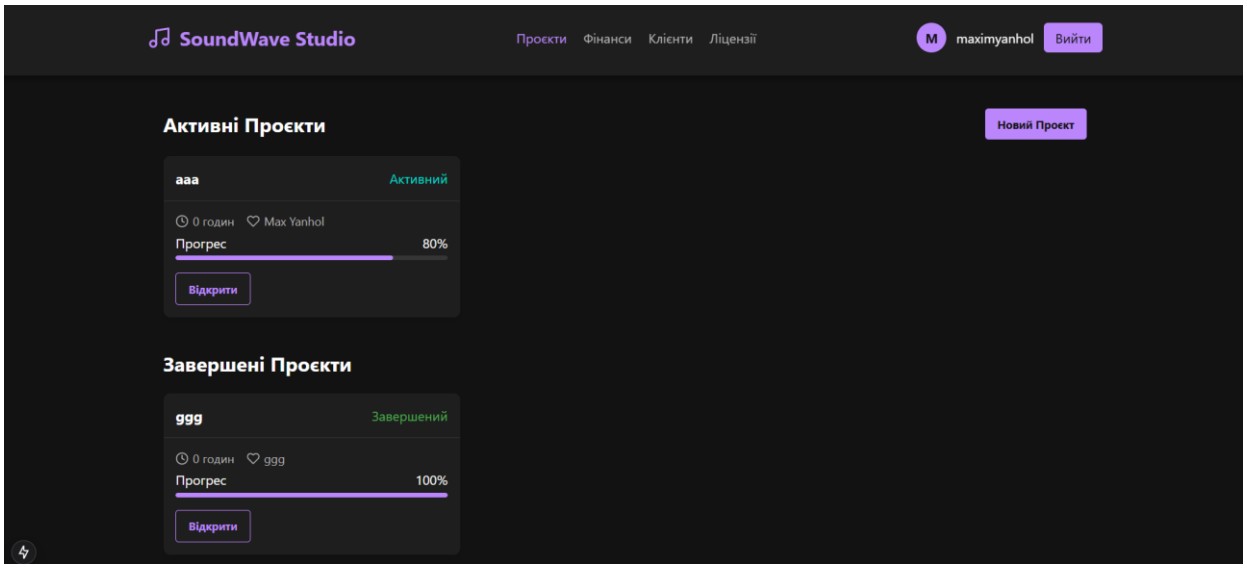


Рисунок 3.3 – Сторінка «Проєкти»

Також, на рисунку 3.4 зображено сторінку «Клієнти», в якому можна створювати, додавати та редагувати клієнтів.

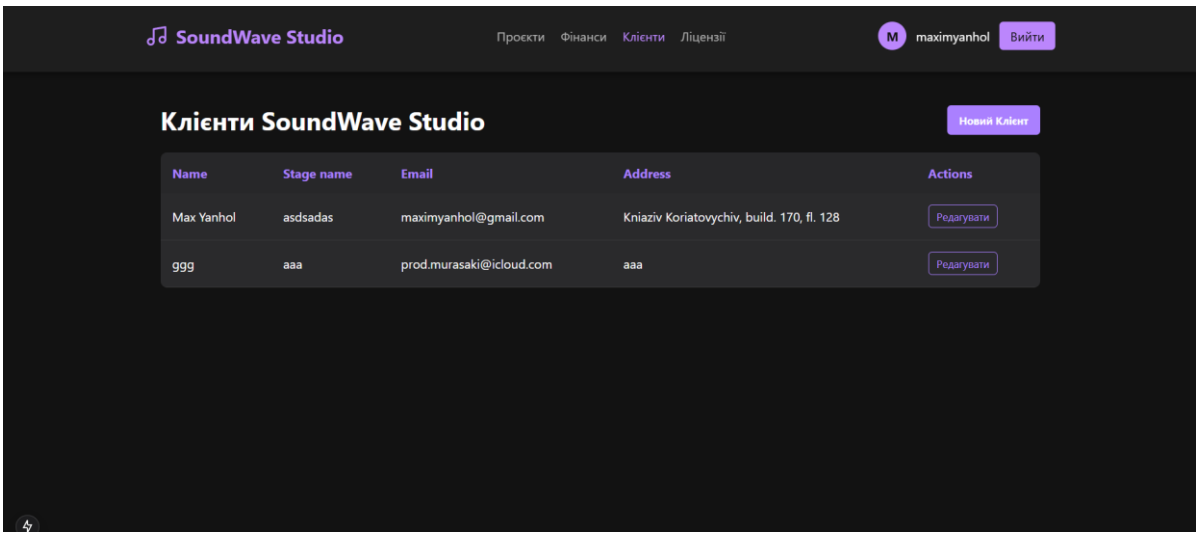


Рисунок 3.4 – Сторінка «Клієнти»

На рисунку 3.5 зображено сторінку «Ліцензії», за допомогою якої, можна створювати ліцензії, генерувати локально на свою систему та відправляти їх клієнтам на пошту.

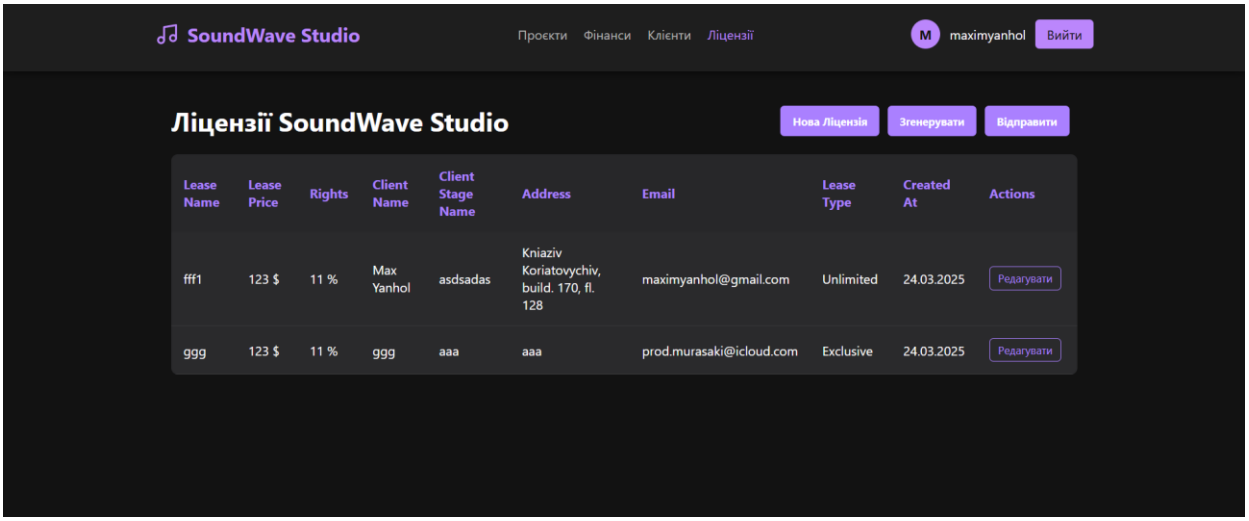


Рисунок 3.5 – Сторінка «Ліцензії»

Фінансова аналітика, реалізована з використанням інформаційних блоків та діаграм різних видів. На рисунку 3.6 зображена сторінка «Фінанси».

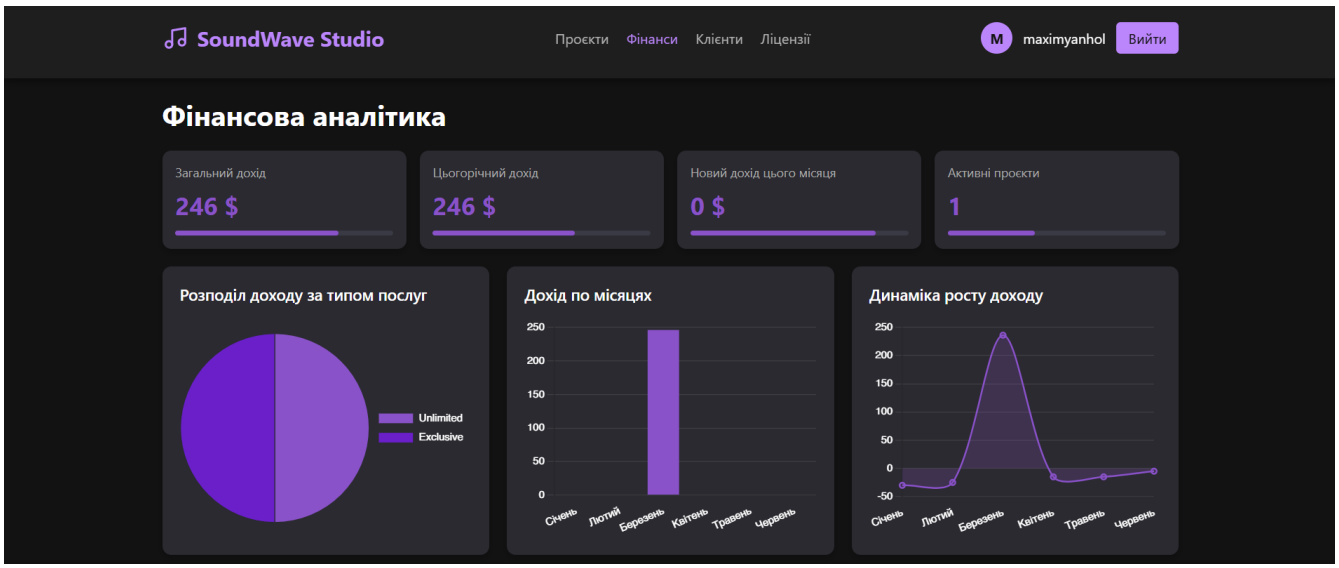


Рисунок 3.6 – Сторінка «Фінанси»

Розробка графічного інтерфейсу програмного застосунку була проведена у середовищі розробки IntelliJ IDEA з використанням фреймворків Spring Boot для серверної частини, TypeScript із Next.js для клієнтської частини, а також інтеграції з базою даних MongoDB для зберігання даних.

3.3 Розробка модуля управління проєктами

При розробці модуля управління проєктами в системі SoundWave необхідно враховувати ключові аспекти організації та обробки даних про проєкти, пов’язані з клієнтами. Основною задачею модуля є забезпечення можливості створення, редагування, видалення та перегляду проєктів, а також відстеження їхнього статусу та статистики активних проєктів [19]. Для цього було вирішено реалізувати REST API ендпоінти, які дозволяють взаємодіяти з проєктами через HTTP-запити, забезпечуючи зручність інтеграції з фронтендом.

Для реалізації модуля управління проєктами було створено контролер OrderController, який обробляє запити до API з базовим шляхом /api. У контролері визначено основні ендпоінти для створення (POST /api/order), оновлення (PUT /api/order/{id}), видалення (DELETE /api/order/{id}), перегляду всіх проєктів (GET /api/orders) та отримання статистики активних проєктів (GET /api/stats/active-projects). Кожен ендпоінт повертає відповідь у форматі JSON, що забезпечує

зручність обробки даних на фронтенді [21]. На рисунку 3.7 зображено запит для перегляду всіх проєктів.

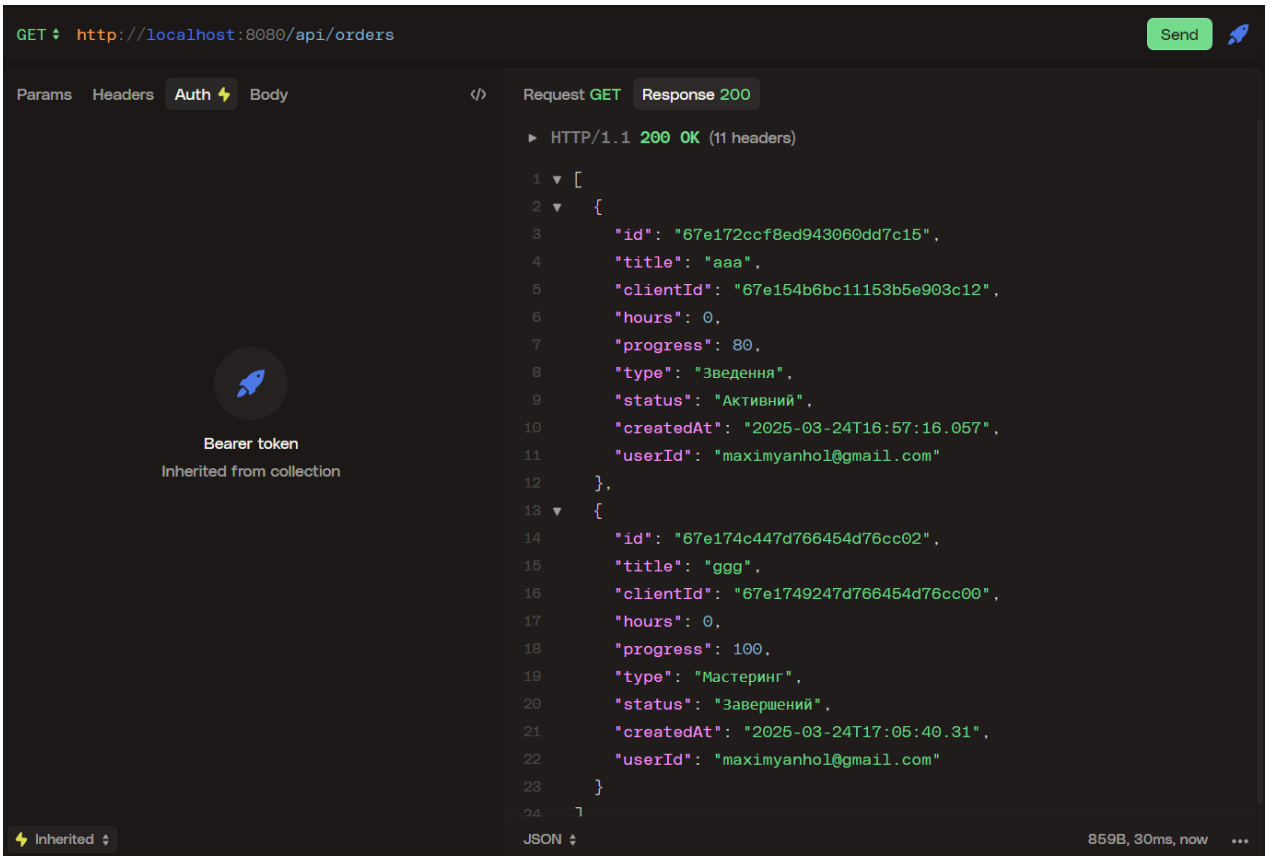


Рисунок 3.7 – Запит на отримання всіх проєктів

Наступним кроком стала розробка сервісного шару, який відповідає за бізнес-логіку обробки проєктів. Для цього було створено клас OrderServiceImpl, що реалізує інтерфейс OrderService. У цьому класі визначено методи для створення, оновлення, видалення та отримання проєктів, а також для підрахунку активних проєктів. Важливим аспектом є перевірка прав доступу: кожен запит до проєкту супроводжується перевіркою, чи належить проєкт або клієнт аутентифікованому користувачу, що забезпечує безпеку даних. Для створення нового проєкту використовується метод createOrder, який приймає об’єкт OrderDto із даними про проєкт, перевіряє наявність клієнта та його належність користувачу, після чого створює новий об’єкт Order із автоматично встановленою датою створення, якщо вона не вказана. Аналогічно, для оновлення проєкту метод updateOrder перевіряє

існування проєкту та права доступу, після чого оновлює його поля. На рисунку 3.8 зображено метод для створення проєкту.

```java
@Override
public Order createOrder(OrderDto orderDto) {
    String userId = getCurrentUserEmail();

    if (orderDto.getClientId() == null || orderDto.getClientId().isEmpty()) {
        throw new IllegalArgumentException("Client ID is required for creating an order");
    }

    // Перевіряємо, чи клієнт належить цьому користувачу
    Optional<Client> clientOptional = clientRepository.findByIdAndUserId(orderDto.getClientId(), userId);
    if (clientOptional.isEmpty()) {
        throw new IllegalArgumentException("Client with ID " + orderDto.getClientId() + " not found for user: " + userId);
    }

    Order order = Order.builder()
            .createdAt(orderDto.getCreatedAt() != null ? orderDto.getCreatedAt() : LocalDateTime.now())
            .title(orderDto.getTitle())
            .status(orderDto.getStatus())
            .clientId(orderDto.getClientId())
            .hours(orderDto.getHours())
            .progress(orderDto.getProgress())
            .type(orderDto.getType())
            .userId(userId)
            .build();
    return orderRepository.save(order);
}
```

Рисунок 3.8 – Створення проєкту

Таким чином, модуль управління проєктами забезпечує повноцінну роботу з проєктами, включаючи їх створення, редагування, видалення та перегляд, а також надає статистику активних проєктів, що дозволяє користувачам ефективно керувати своєю діяльністю в системі.

3.3 Розробка модуля фінансової аналітики

При розробці модуля фінансової аналітики, основна увага приділялася забезпеченню можливості аналізу даних, пов’язаних насамперед із ліцензіями (Lease), а також наданню користувачам зручного інструментарію для відстеження доходів за різні періоди часу. Модуль фінансів дозволяє отримувати фінансову статистику, таку як загальний дохід, річний дохід та місячний дохід, а також переглядати тип ліцензії за певний місяць або набір місяців. Для реалізації цього

функціоналу було створено набір REST API ендпоінтів, які забезпечують доступ до фінансових даних через HTTP-запити.

Для обробки фінансових запитів у системі було доповнено контролер OrderController, який містить ендпоінти для роботи з фінансовими даними. Зокрема, було додано ендпоінти для отримання ліцензій за місяць (GET /api/leases/month), за кілька місяців (GET /api/leases/months), а також для підрахунку загального доходу (GET /api/stats/total-revenue), річного доходу (GET /api/stats/yearly-revenue) та місячного доходу (GET /api/stats/monthly-revenue). Усі ендпоінти повертають відповідь у форматі JSON, що забезпечує зручність обробки даних на фронтенді. На рисунку 3.9 зображено запит отримання доходу за рік.

Усі вказані ендпоінти повертають дані у форматі JSON, що є стандартом у веб-розробці та значно спрощує обробку інформації на стороні клієнта. Це забезпечує не лише швидкий рендеринг інформації на інтерфейсі, а й можливість подальшого масштабування функціоналу без значних змін у логіці обробки запитів.

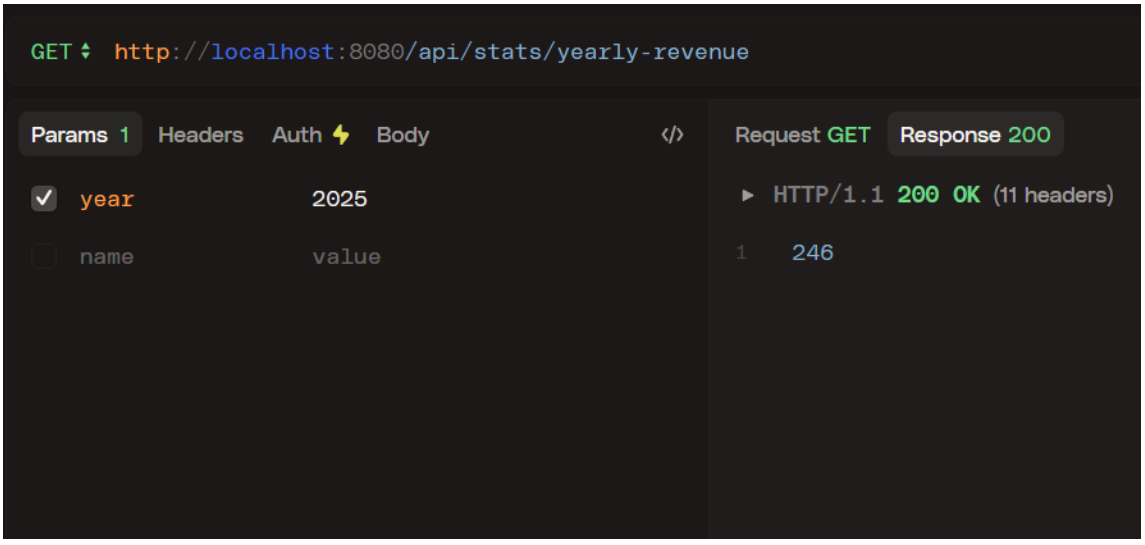


Рисунок 3.9 – Запит на отримання доходу за рік

Для підрахунку фінансових показників було реалізовано методи getTotalRevenue, getYearlyRevenue та getMonthlyRevenue. Метод getTotalRevenue обчислює загальний дохід, підсумовуючи ціни всіх ліцензій, що належать користувачу. Метод getYearlyRevenue обчислює дохід за вказаний рік, фільтруючи

ліцензії за діапазоном дат від початку до кінця року, а метод getMonthlyRevenue обчислює дохід за конкретний місяць, використовуючи відповідний діапазон дат. Усі методи використовують репозиторій для взаємодії з базою даних.

Отже, модуль фінансів забезпечує користувачам можливість аналізувати фінансові дані, пов’язані з ліцензіями, включаючи перегляд ліцензій за певні періоди та підрахунок доходів. Це дозволяє ефективно відстежувати фінансову діяльність у системі, надаючи зручний доступ до статистики через REST API.

3.4 Розробка модуля управління клієнтами

Модуль управління клієнтами призначений для створення, редагування, видалення та пошуку клієнтів. Для цього створено контролер ClientController із REST API ендпоінтами на базовому шляху /api, включаючи створення (POST /api/client), оновлення (PUT /api/client/{id}), видалення (DELETE /api/client/{id}), перегляд (GET /api/client) та пошук клієнтів за ім’ям (GET /api/clients/search). На рисунку 3.7 зображено ендпоінти контролера клієнтів.

Сервісний шар реалізовано у класі ClientServiceImpl, який забезпечує бізнес-логіку, а саме створення клієнтів із перевіркою унікальності email, оновлення з валідацією доступу, видалення та пошук за ім’ям. Кожен метод перевіряє права доступу через SecurityContextHolder, дозволяючи працювати лише з клієнтами автентифікованого користувача. На рисунку 3.8 зображено запит на додавання нового клієнта.

Модуль забезпечує зручне управління клієнтами, інтегруючись із фронтендом через REST API.

3.5 Розробка модуля управління ліцензіями

Модуль управління ліцензіями в системі StudioApp призначений для створення, редагування, видалення ліцензій, а також генерації PDF-документів і відправки їх на email. Для цього створено контролер LeaseController із REST API ендпоінтами на базовому шляху /api, включаючи створення (POST /api/lease), оновлення (PUT /api/lease/{id}), видалення (DELETE /api/lease/{id}), перегляд усіх

ліцензій (GET /api/leases), генерацію PDF (POST /api/lease/generate-pdf) та відправку email (POST /api/lease/send-email). На рисунку 3.9 зображено запит на створення ліцензії.

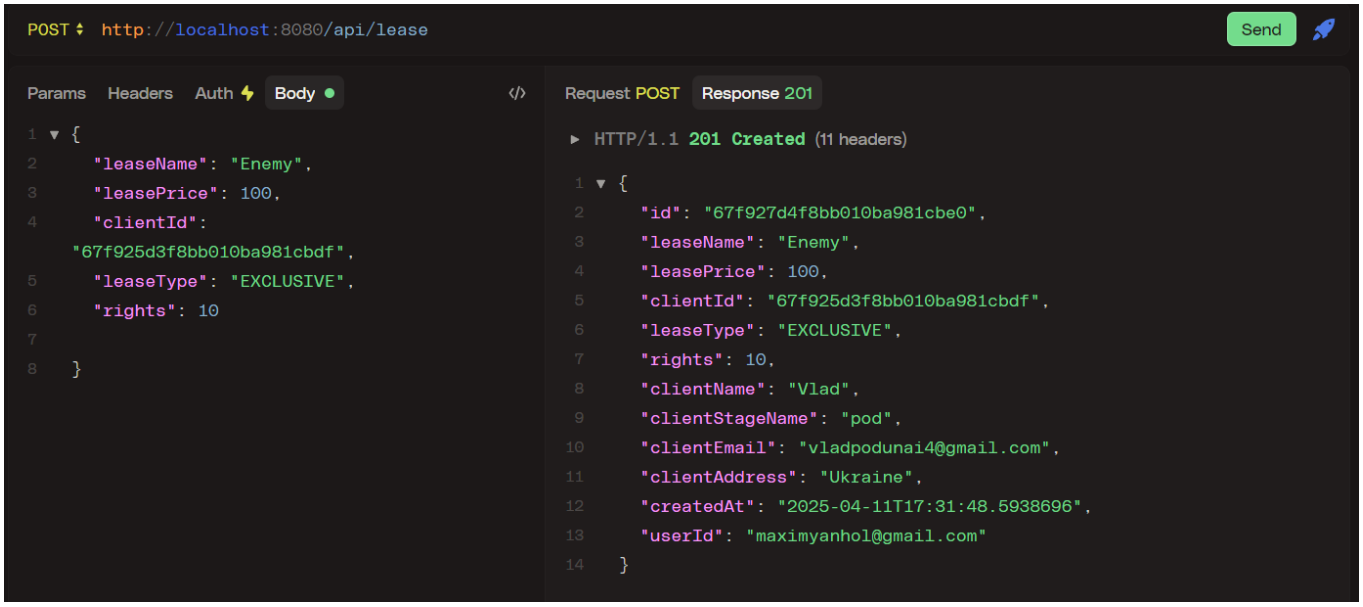


Рисунок 3.9 – Запит на створення ліцензії

Сервісний шар реалізовано у класі LeaseServiceImpl, який забезпечує бізнес-логіку: створення ліцензій із перевіркою доступу до клієнта, оновлення, видалення та генерацію PDF з використанням PdfGenerationService та інтеграцію з поштовим сервісом. На рисунку 3.10 зображено метод для створення листа для відправки ліцензії на email.

Метод sendLeaseEmail отримує параметри leaseId, emailTo та message, після чого викликає допоміжний метод generateLeasePDF, який формує PDF-файл ліцензії у вигляді масиву байтів. Для створення поштового повідомлення використовується бібліотека JavaMailSender, яка дозволяє зручно працювати з MIME-повідомленнями в Java. Зокрема, через MimeMessage та MimeMessageHelper налаштовуються відправник листа, одержувач, тема листа, текст повідомлення та додається вкладення у форматі PDF. Вкладення підписується динамічно на основі ідентифікатора ліцензії (license_<leaseId>.pdf), а MIME-тип файлу вказується як

application/pdf. Нарешті, сформоване повідомлення відправляється через поштовий сервер за допомогою javaMailSender.send(mimeMessage).

Для генерації PDF-файлів було використано популярну бібліотеку iTextPDF, яка дозволяє створювати та обробляти документи у форматі PDF. Основні класи бібліотеки, задіяні в реалізації, включають:

- PdfWriter — клас для запису вмісту у файл PDF.
- PdfDocument — об'єкт моделі документу, що керує сторінками та контентом.
- Document — високорівневий клас, який забезпечує додавання тексту, таблиць, параграфів тощо у PDF.
- Paragraph — клас для створення та форматування параграфів тексту.
- Table — клас для побудови таблиць у PDF-документі.

У процесі генерації PDF файлу інформація про ліцензію (ім'я клієнта, дата, назва треку тощо) додається у структурованому вигляді. Також використовується стилізація шрифтів та базове форматування для забезпечення читабельності документа.

Завдяки використанню бібліотеки iTextPDF, рішення є гнучким, масштабованим та легко адаптується до змін у структурі ліцензійних документів у майбутньому.

Модуль забезпечує повноцінне управління ліцензіями, генерацію та їх відправку, інтегруючись із фронтендом через REST API.

3.6 Висновки

У третьому розділі було обґрунтовано вибір технологій та інструментів для розробки програмного забезпечення системи SoundWave, спрямованої на управління проєктами, фінансами, клієнтами та ліцензіями. Після аналізу вимог та задач проєкту було обрано мову програмування Java у поєднанні з фреймворком Spring Boot для створення серверної частини додатка, що забезпечило надійність, масштабованість та зручність інтеграції з базою даних MongoDB. Для розробки

клієнтської частини використано Next.js із TypeScript, що дозволило створити швидкий та інтерактивний інтерфейс користувача.

Також у розділі детально описано розробку інтерфейсу та основних програмних модулів. Усі модулі інтегровані через REST API, що забезпечує зручну взаємодію між фронтендом і бекендом, а також враховують безпеку завдяки перевірці прав доступу через Spring Security [22].

Розроблені модулі відповідають поставленим задачам, забезпечуючи ефективне управління даними та зручність використання системи SoundWave для користувачів.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

При розробці програмного забезпечення SoundWave необхідно застосовувати різні види тестування, щоб забезпечити його якість і відповідність вимогам. Функціональне тестування перевіряє, чи система виконує всі потрібні завдання, як-от реєстрація чи створення документів чи інших даних. Тестування відмов, дає змогу визначити, як система реагує на помилки у різних модулях програмного забезпечення. Тестування навантаження оцінює, як система працює під великим навантаженням, наприклад, при одночасній роботі багатьох користувачів. Тестування безпеки перевіряє захист даних і доступу, щоб уникнути вразливостей. Усі ці тестування важливі, щоб виявити помилки, забезпечити стабільність, безпеку і зручність використання, а також створити позитивний досвід для користувачів.

Тестування безпеки показало, що всі приватні маршрути надійно захищені JWT-токенами, а спроба підміни токена призводить до помилки 401 Unauthorized, що запобігає несанкціонованому доступу. Шифрування паролів за допомогою BCrypt унеможливлює їхнє зламування, забезпечуючи безпеку облікових даних користувачів. API стійке до введення шкідливих SQL-запитів завдяки використанню Mongo ORM та ретельній валідації вхідних даних на стороні серверу, що виключає ризик ін’єкцій, обмеження доступу до даних клієнтів і ліцензій через перевірку прав користувача додає додатковий рівень безпеки. Ці заходи забезпечують надійний захист системи від основних загроз. На рисунку 4.1 зображено аутентифікацію користувача та створений JWT токен [23].

Рисунок 4.1 – Аутентифікація користувача та отримання JWT токену

Використання BCrypt дає можливість зашифрувати пароль при реєстрації користувача, створюючи безпечний хеш, який зберігається в базі даних замість відкритого тексту. BCrypt автоматично додає сіль (salt) до хешу, що робить його стійким до атак. Під час входу введений пароль хешується тим самим способом, і отриманий хеш порівнюється з збереженим у базі даних, що дозволяє перевірити правильність пароля без його розшифрування, забезпечуючи високий рівень безпеки. На рисунку 4.2 зображено використання BCrypt для хешування паролю користувача.

Рисунок 4.2 – Використання BCrypt для хешування паролю

Функціональне тестування проводилось з метою перевірки того, що всі модулі системи працюють згідно з технічним завданням.

Функціональні перевірки у модулі «Проєкти»:

- створення проєкту можливе лише для існуючого клієнта;
- зміна статусу;
- коректне збереження даних.

На рисунку 4.3 зображено валідацію помилки при створенні проєкту лише для існуючого клієнта. Ця помилка виникає, коли користувач намагається створити новий проєкт у системі, вказавши ім’я клієнта, якого не існує в базі даних або який не належить поточному аутентифікованому користувачу. Сервер робить перевірку на ім’я користувача у полі для вводу «Артист», він постійно перевіряє при зміні кожного символу, коли дані з поля зрівняються з даними, які є у базі, повідомлення зникне та користувачу буде доступно створити проєкт.

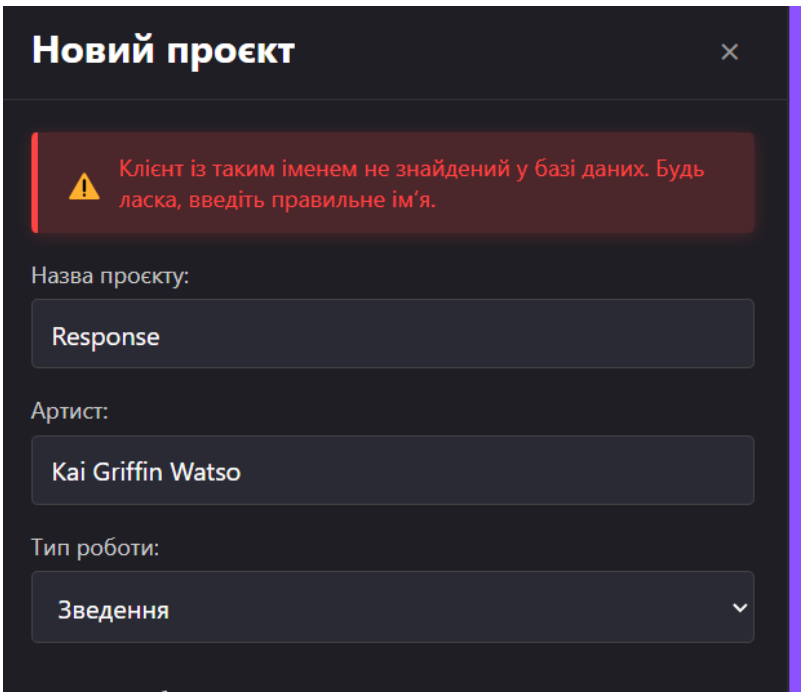


Рисунок 4.3 – Помилка при створенні проєкту

Функціонал завершення проєкту реалізований у системі таким чином, що кожен проєкт має атрибут прогресу, який відображає ступінь виконання завдань у відсотках. Коли значення прогресу досягає 100%, система автоматично змінює статус проєкту на «Завершений». Після цього проєкт переміщується у відповідну категорію «Завершені Проекти», яка відображається окремим блоком у користувацькому інтерфейсі.

На рисунку 4.4 продемонстровано вигляд завершених проєктів у системі. Для кожного завершеного проєкту відображається його назва, ім'я відповідального користувача, та індикатор прогресу, який фіксовано встановлений на рівні 100%.

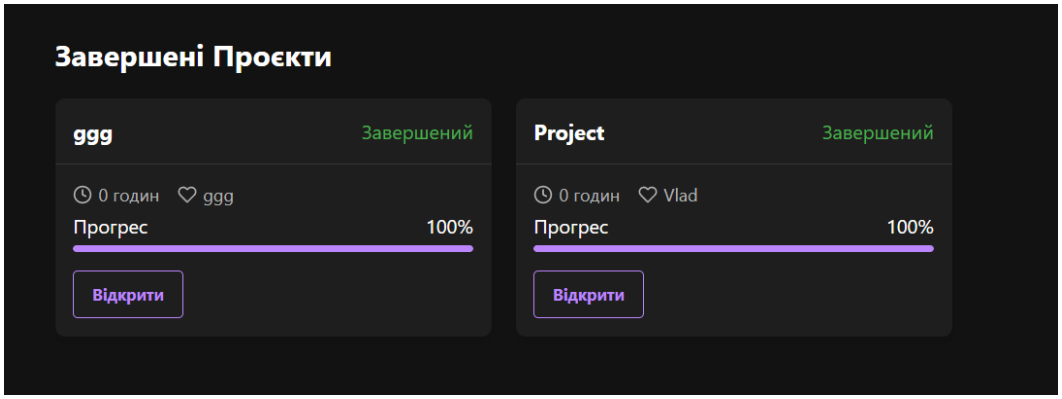


Рисунок 4.4 – Категорія «Завершені Проекти»

На рисунку 4.5 зображено успішно створений проєкт. Інтеграція сервера та клієнтської частини працює успішно, прогрес проєкту зберігається у базі даних.

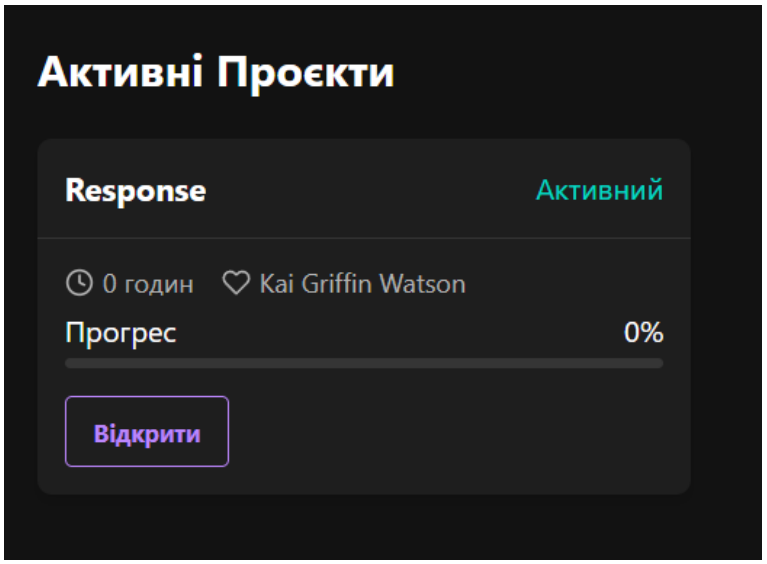


Рисунок 4.5 – Створений проєкт

Функціонал збереження проєкту реалізовано таким чином, що користувач надсилає оновлені дані проєкту через спеціальне модальне вікно редагування, яке відкривається з інтерфейсу управління проєктами. У цьому вікні можна змінити основні атрибути проєкту, зокрема: назву, статус (Активний / Завершений), кількість витрачених годин, рівень прогресу (у відсотках) та тип проєкту (наприклад, Аранжування, Міксування тощо).

Після внесення змін і натискання кнопки збереження, дані формуються у вигляді JSON-об’єкта і надсилаються на сервер через REST API. Сервер приймає ці дані та зберігає їх у базі даних.

Таким чином, реалізована система дозволяє гнучко та безпечно управляти даними проєктів, швидко оновлювати їх відповідно до реального прогресу, а також автоматично відображати актуальний стан завдань у загальному переліку проєктів.

На рисунку 4.6 показано приклад збереженого документа у базі даних. Поле progress зберігає інформацію про поточний рівень виконання роботи над проєктом і може бути змінено в будь-який момент відповідно до фактичного стану задач.

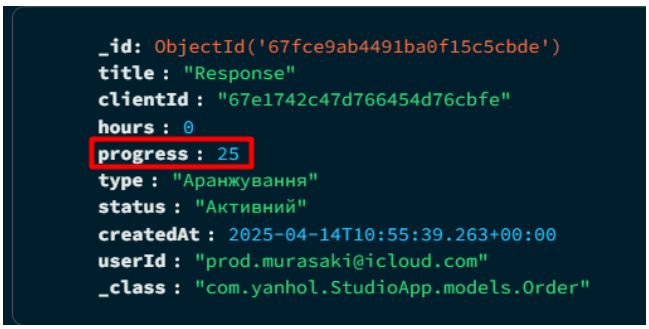


Рисунок 4.6 – Коректне збереження даних

Функціональні перевірки у модулі «Проєкти»:

- додавання нового клієнта з унікальним email;
- при спробі створити клієнта з уже існуючим email система повертала повідомлення про помилку;
- можливість редагування та видалення клієнта.

При створенні клієнта, на сервері реалізована валідація email, сервер поверне помилку 404 Bad Request, якщо email не є унікальним. На рисунку 4.7 зображено запит та валідацію помилки.

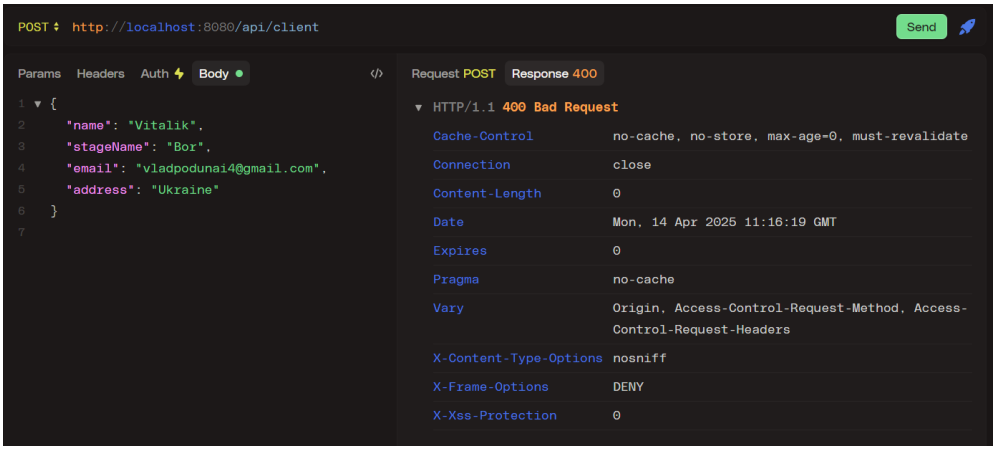


Рисунок 4.7 – Валідація помилки

Функціональні перевірки у модулі «Ліцензії»:

- створення та ліцензії працює з обов’язковою перевіркою наявності клієнта;
- генерація PDF-файлу;
- надсилання PDF ліцензії на email клієнта через поштовий сервіс.

Якщо клієнт не існує або не належить аутентифікованому користувачу, генерується виняток із повідомленням про помилку, що запобігає створенню некоректних ліцензій.

Було проведено навантажувальне тестування системи. У ході тестування симульовано 100 одночасних користувачів, які одночасно створюють нові проєкти через та переглядають існуючі проєкти, що відображає реальний сценарій використання системи. Результати показали, що система змогла обробити 95% запитів менш ніж за 1 секунду, не втрачаючи даних, що свідчить про високу швидкодію та стабільність. Під час тестування не спостерігалося витоку пам’яті або збою сервера, що підтверджує ефективність управління ресурсами Spring Boot та MongoDB у бекенді. Кешування відповідей для GET-запитів, реалізоване за допомогою Spring Cache, дозволило суттєво знизити навантаження на сервер, зменшивши час обробки повторюваних запитів на 40%. Додатково було перевірено поведінку системи при піковому навантаженні в 150 користувачів: час відгуку зріс до 1,2 секунди, але система залишилася стабільною, а помилки склали менше 1%, що свідчить про її готовність до масштабування. Ці результати підтверджують високу продуктивність SoundWave і її здатність ефективно працювати в умовах реального використання.

4.2 Розробка інструкції користувача

Інструкція користувача є невід’ємною частиною документації програмного забезпечення. Вона призначена для надання користувачам чітких вказівок щодо встановлення, налаштування, запуску та ефективної роботи із системою. Інструкція описує порядок встановлення програмного забезпечення, запуску сервісів, налаштування облікового запису користувача, а також правила взаємодії з основними модулями системи, такими як «Проєкти», «Клієнти», «Ліцензії» та «Профіль».

Для забезпечення стабільної, безпечної та продуктивної роботи системи необхідно дотримуватись мінімальних або рекомендованих технічних вимог до обладнання та програмного середовища, які наведено у таблиці 4.1. Відповідність

цим вимогам дозволить уникнути критичних помилок, втрати даних і забезпечить коректну роботу всіх функціональних модулів.

Крім того, інструкція містить рекомендації щодо регулярного оновлення програмного забезпечення та опис можливих помилок при роботі системи з відповідними способами їх усунення.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i7 або Apple M1/M2	Intel Core i3 або AMD Ryzen 3
Оперативна пам’ять	16 ГБ RAM	4 ГБ RAM
Вільний простір на диску	15 ГБ	4 ГБ
Операційна система	Windows 10+, macOS Big Sur або Linux Ubuntu	Windows 7, macOS Mojave
Браузер	Google Chrome (v100+), Firefox, Safari	Будь-який сучасний браузер із підтримкою JavaScript

Для бекенд розгортання необхідна Java 17+, MongoDB 6.0, доступ до інтернету та поштовий SMTP-сервер.

Після розгортання системи на сервері або запуску локального варіанту, користувач переходить на головну сторінку. Відображається форма входу (аутентифікації). Користувач вводить свої облікові дані та переходить до головного інтерфейсу системи. Якщо обліковий запис ще не створено, користувач може його створити на сторінці реєстрація. На рисунку 4.8 зображено форму реєстрації користувача.

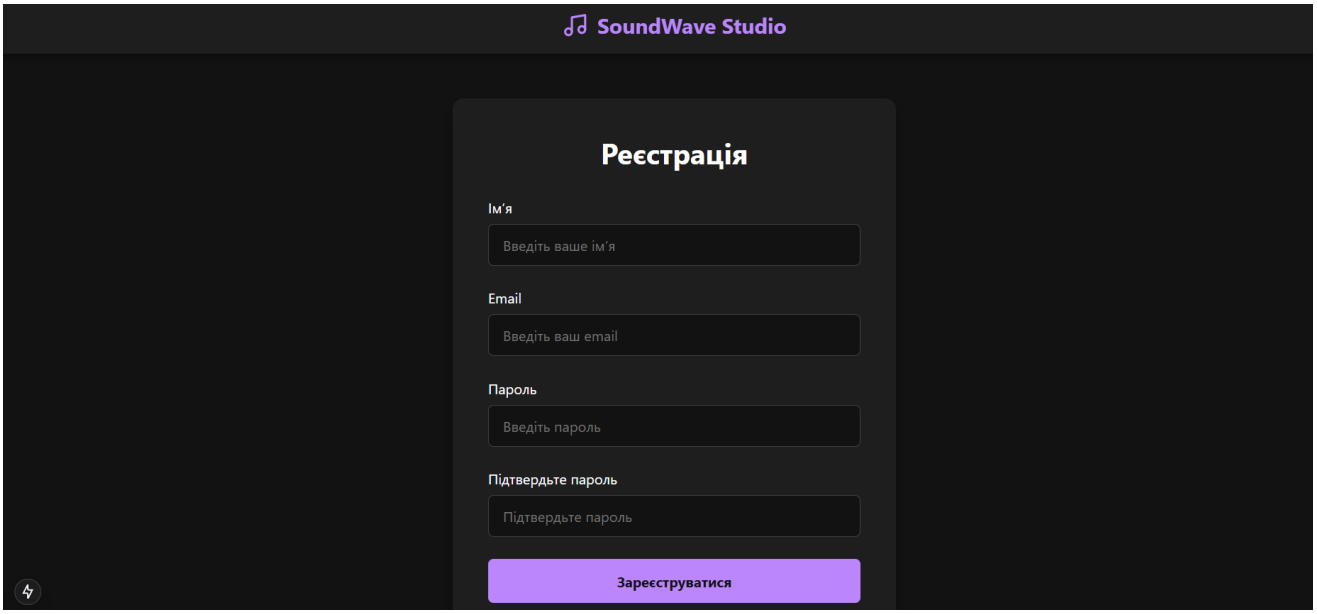


Рисунок 4.8 – Форма реєстрації користувача

Після авторизації відкривається сторінка «Клієнти». Інформацію про всіх клієнтів знаходиться у таблиці. Також в кожному рядку є кнопка «Редагувати», яка дозволяє відредагувати дані клієнта. На рисунку 4.9 зображено кнопку «Редагувати».

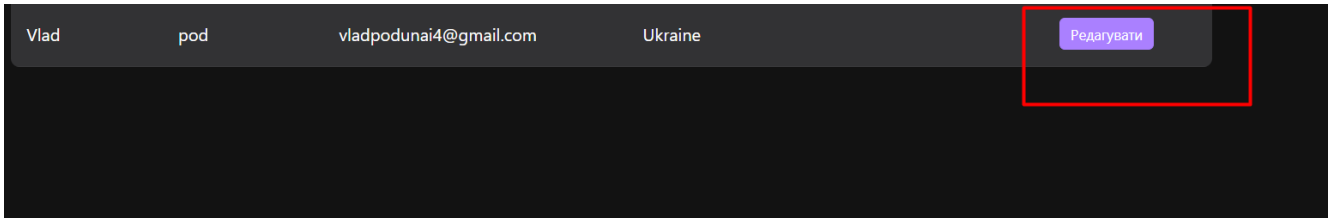


Рисунок 4.9 – Кнопка «Редагувати»

На рисунку 4.10 зображено модальне вікно для редагування клієнта. Модальне вікно дозволяє редагувати всі дані клієнта, також воно містить три кнопки. Модальне вікно містить три основні кнопки, кожна з яких виконує конкретну дію. Кнопка «Скасувати» дозволяє користувачу скасувати зміни, не зберігаючи внесених правок, і закрити модальне вікно без впливу на існуючі дані в базі. Це зручно у випадках, коли редагування було розпочато помилково або користувач передумав вносити зміни. Кнопка «Видалити» надає можливість повністю видалити клієнта з бази даних, причому передбачено механізм

підтвердження дії, що запобігає випадковому видаленню. Нарешті, кнопка «Зберегти» дозволяє записати всі внесені зміни до клієнтських даних у базу даних. Після збереження система оновлює інформацію на сторінці без потреби у повному перезавантаженні інтерфейсу, що забезпечує швидкий і плавний досвід користувача.

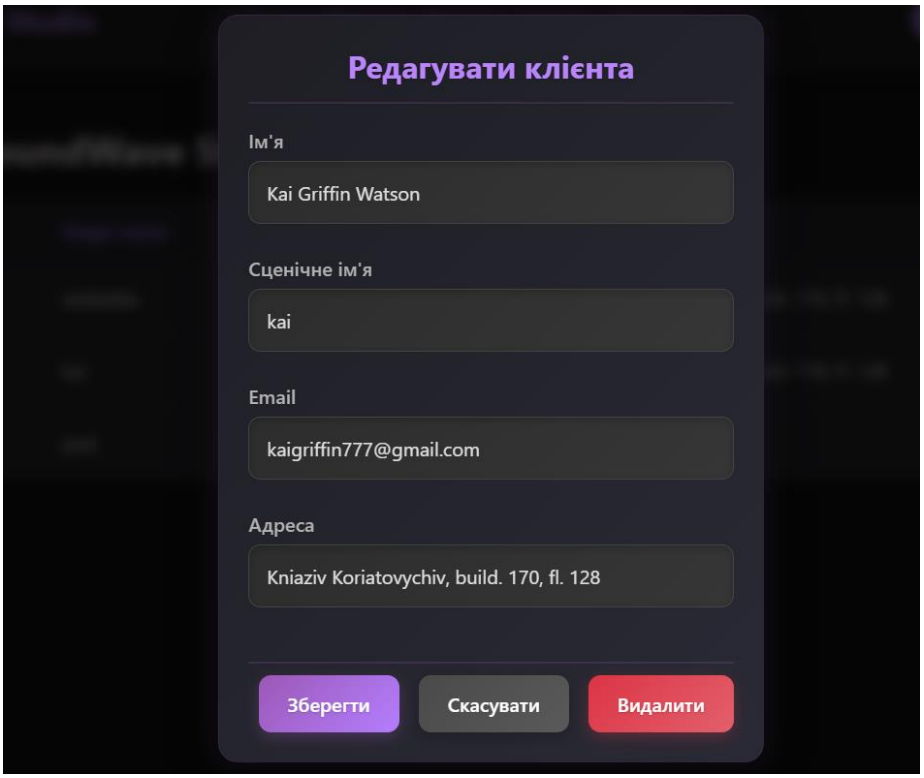


Рисунок 4.10 – Вікно для редагування клієнта

Сторінку створення ліцензії зображено на рисунку 4.11. Вона містить набір полів для заповнення, перелік доступних типів ліцензій, а також інформаційні повідомлення про успішність чи помилки у процесі створення. Таким чином, реалізація цієї частини інтерфейсу є важливим кроком до повної автоматизації документообігу в межах системи.

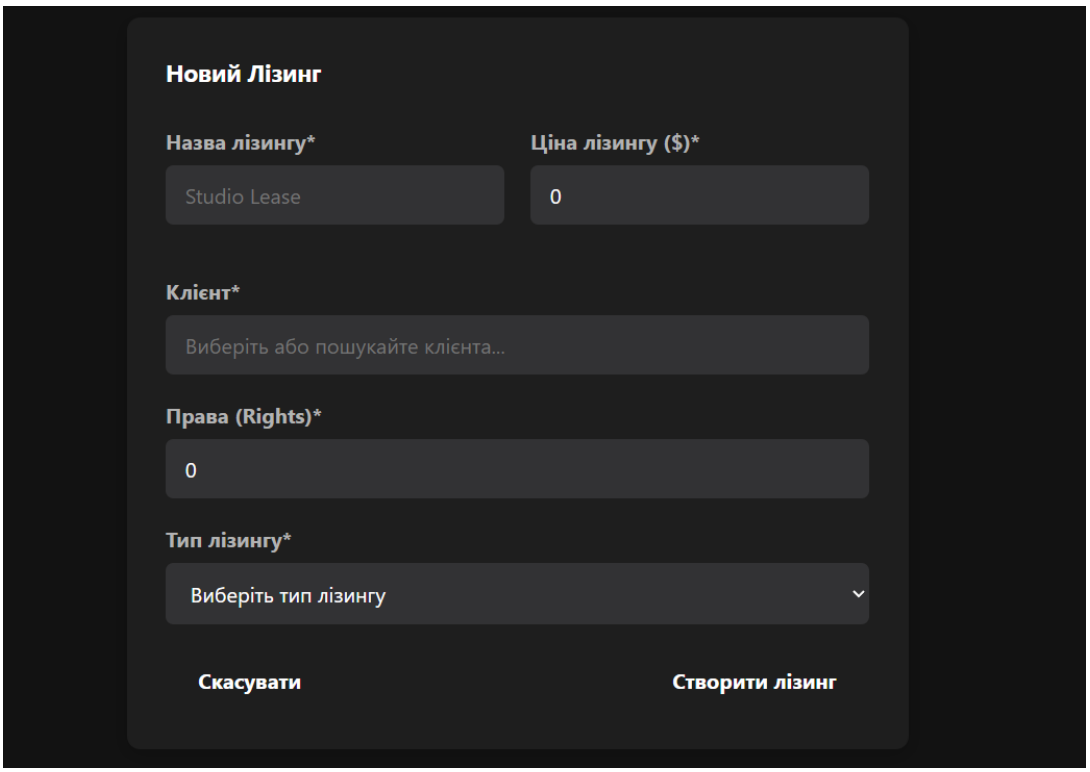


Рисунок 4.11 – Сторінка створення ліцензії

Користувачу потрібно заповнити всі дані форми перед тим як створити ліцензію. Форма містить такі дані:

- назва лізингу – назва ліцензії, назва композиції чи музичного твору;
- ціна лізингу – вартість ліцензії, у валюті USD;
- клієнт – клієнт, до якого буде прив’язана ліцензія;
- права – скільки студія матиме роялті з музичної композиції, вказується у відсотках;
- тип лізингу – тип ліцензії, який залежить від потреб користувачів, ліцензія має такі типи: «Premium», «Unlimited», «Exclusive». Всі ліцензії відрізняються за умовами, які студія дає користувачеві.

Користувач має можливість додати клієнта до ліцензії за допомогою пошуку за іменем. На рисунку 4.12 зображено випадаюче меню з вибором клієнта.

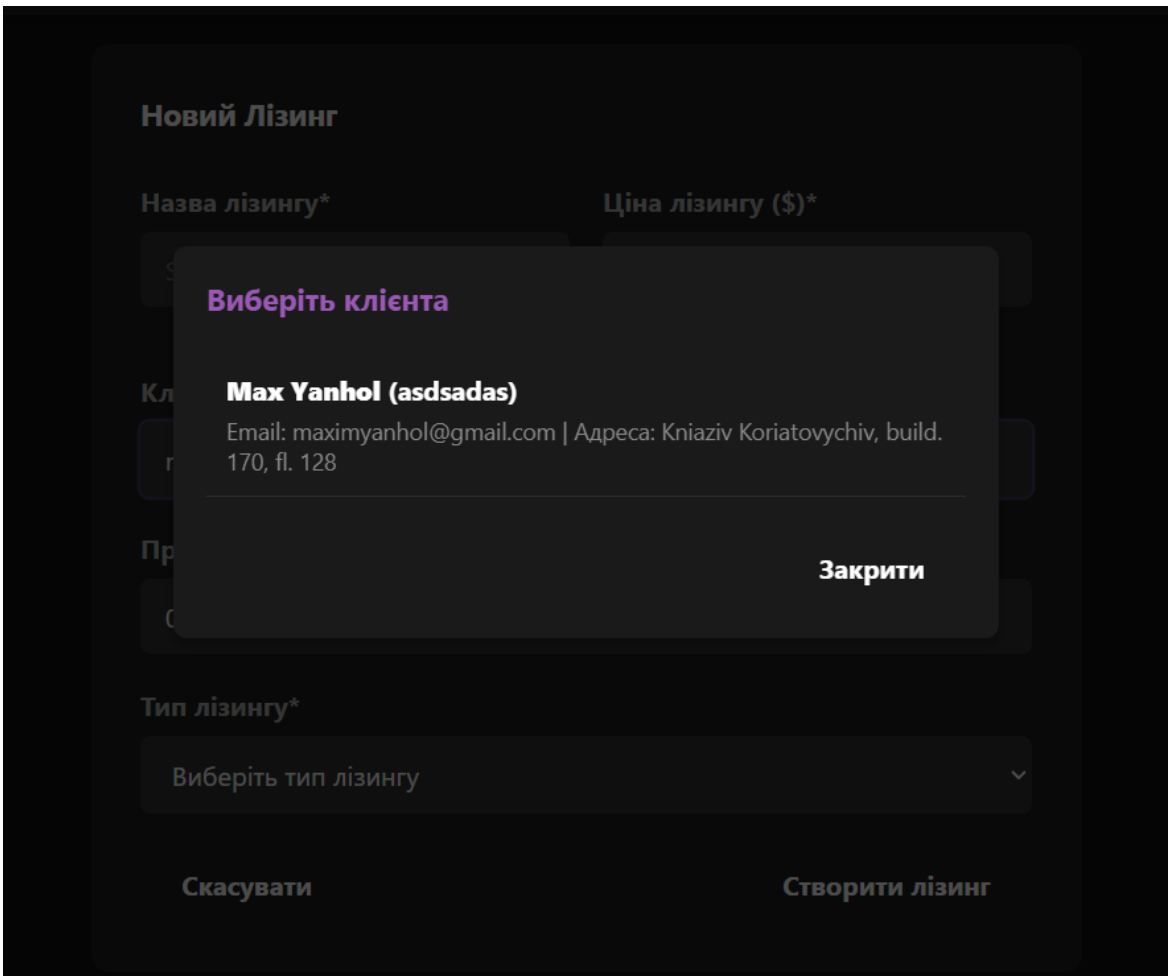


Рисунок 4.12 – Випадаюче меню

На рисунку 4.13 зображено табличку для відображення ліцензій. Користувач може переглянути ліцензію, створити, відредагувати, завантажити або відправити безпосередньо на email клієнта.

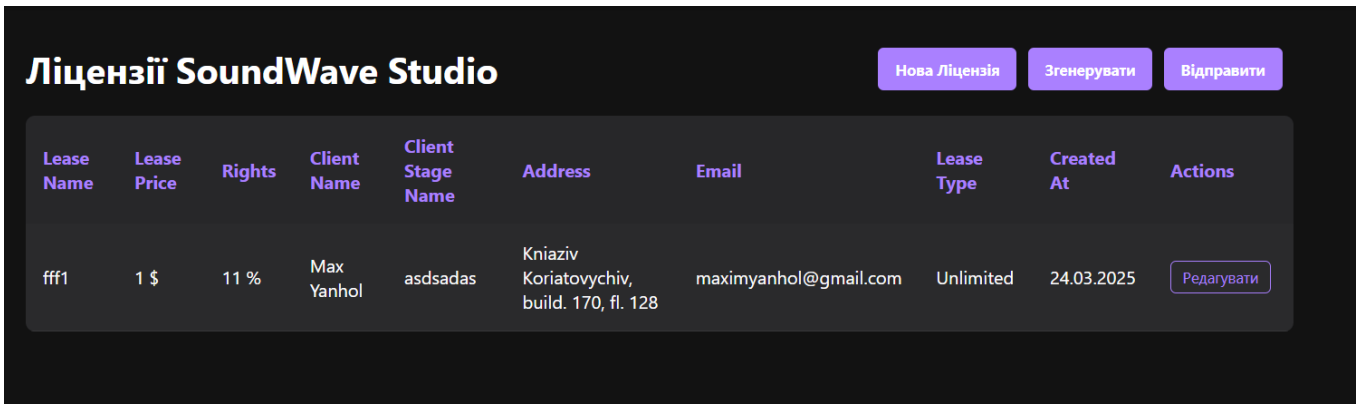


Рисунок 4.13 – Табличка з ліцензіями

При натисканні на кнопку «Згенерувати» у модулі управління ліцензіями відкривається спеціальне модальне вікно для створення нової ліцензії. У цьому вікні користувач вибирає ліцензію зі списку. На рисунку 4.14 зображено приклад такого вікна для ліцензії для генерації.

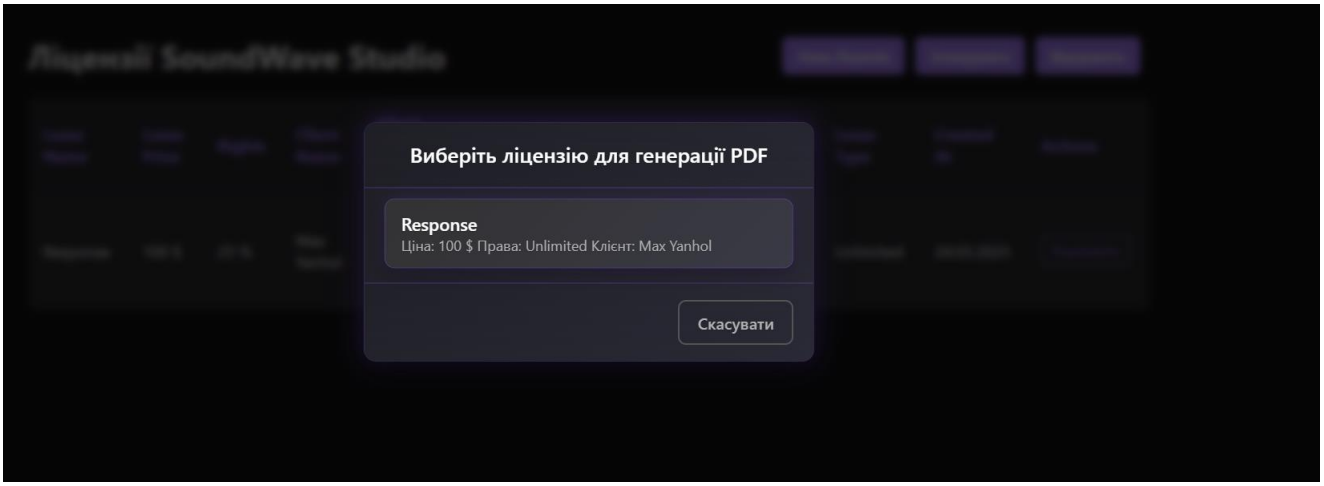


Рисунок 4.14 – Вікно для вибору ліцензії

Після заповнення всіх обов’язкових полів користувач натискає кнопку підтвердження, після чого система автоматично формує PDF-файл ліцензії за допомогою бібліотеки iTextPDF. Згенерований файл містить всю важливу інформацію про права користування продуктом і автоматично прикріплюється до відповідного клієнта в базі даних.

На рисунку 4.15 зображено приклад згенерованої ліцензії, яка укладається між продюсером та клієнтом.

Ліцензія містить такі основні розділи:

- предмет угоди, зазначається, яку саме музичну композицію надає продюсер артисту, в якій формі і для яких цілей. Вказується, що композиція є попередньо створеною продюсером, а виконання артистом вважається похідним твором;
- права і використання прописується, що майстер-запис є "роботою за наймом" і належить артисту. Також дозволяються невеликі редагування

композиції для соцмереж, але комерційне використання змінених версій потребує додаткової згоди.

- оплата визначено фіксовану оплату за ліцензію, яка не підлягає поверненню.
- маркетинг та кредити, артист зобов'язується відмітити продюсера у соціальних мережах під час релізу композиції.
- винагороди і права на пісню уточнюється, що створення інструменталу саме по собі не вважається написанням пісні, і основні права на мелодію та текст залишаються за артистом, але базова композиція поділяється між артистом і продюсером у певних пропорціях.
- інші умови встановлюються правові аспекти — наприклад, юрисдикція угоди, порядок внесення змін тощо.

Важливим є те, що договір фіксує юридичну відповідальність сторін за виконання своїх зобов'язань та містить умови щодо внесення змін або припинення дії угоди за згодою обох сторін. Такий підхід забезпечує прозорість співпраці, захист інтересів продюсера і клієнта, а також гарантує правову визначеність у питаннях використання музичного контенту.

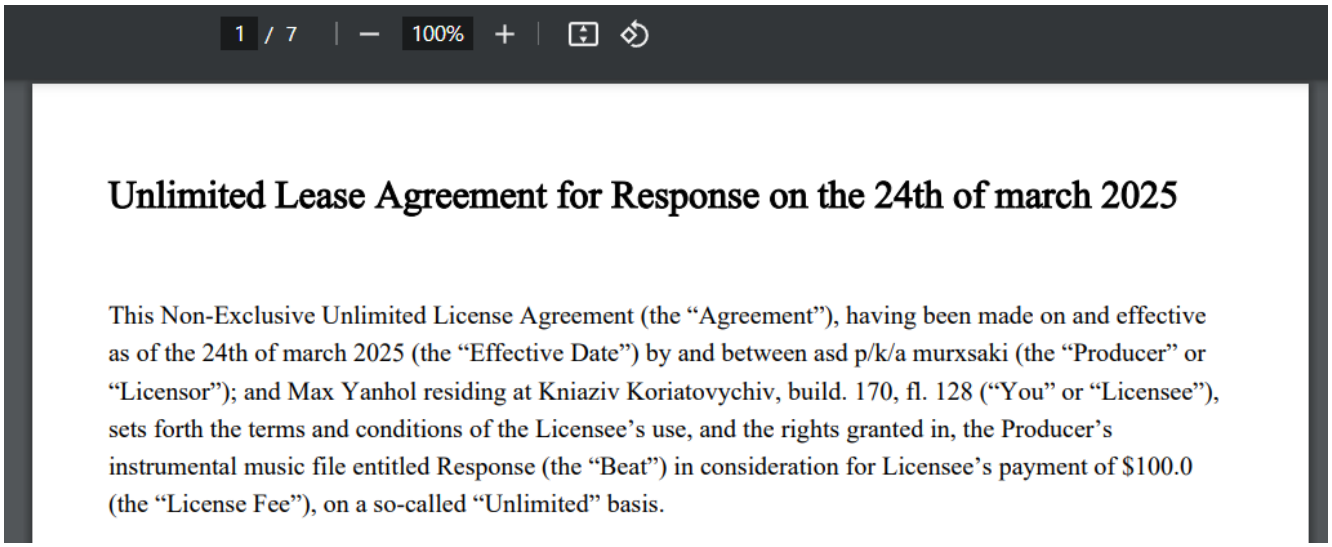


Рисунок 4.15 – Згенерована ліцензія

Для відправки ліцензії на електронну пошту клієнта, користувачу необхідно натиснути на кнопку «Відправити». У формі, що відкриється, потрібно ввести ім’я клієнта, адресу електронної пошти, а також вибрати відповідну ліцензію з доступного списку. Система автоматично підставляє всю наявну інформацію про клієнта та ліцензію. Користувач має змогу перевірити правильність усіх введених даних перед відправкою. У разі необхідності форму можна скасувати, натиснувши кнопку «Скасувати», або підтвердити дію, натиснувши «Відправити ліцензію». Після натискання кнопки, система формує PDF-документ із вказаними даними, прикріплює його до листа та надсилає клієнту за вказаною адресою.

Крім того, перед надсиланням користувач може переглянути попередній вигляд згенерованої ліцензії, щоб переконатися у правильності заповнення всіх реквізитів. Якщо виявлені помилки або неточності, система дозволяє оперативно внести зміни без необхідності повторного заповнення всієї форми. Після успішного надсилання ліцензії, користувач отримує відповідне повідомлення про доставку листа, що забезпечує контроль за станом обробки замовлень. Функціонал відправки також передбачає автоматичне збереження історії надісланих ліцензій у базі даних системи. Це дозволяє у будь-який момент відстежити, кому, коли та яку саме ліцензію було надіслано, що підвищує рівень обліку та прозорості роботи з клієнтами. На рисунку 4.16 зображено форму відправлення ліцензії.

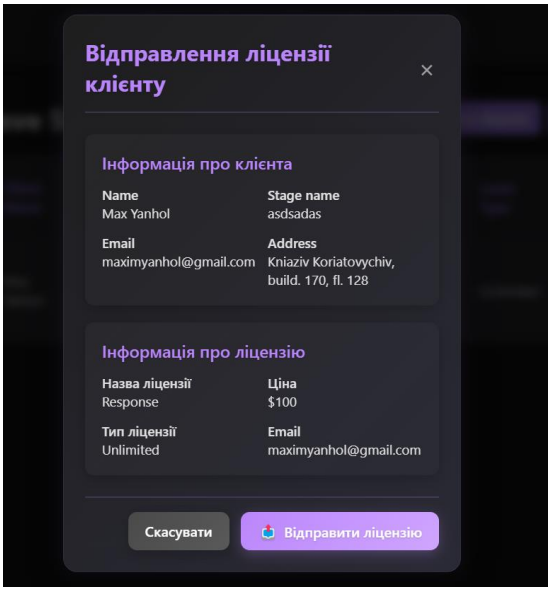


Рисунок 4.16 – Форма для відправлення ліцензії

На рисунку 4.17 зображено отримана ліцензія.

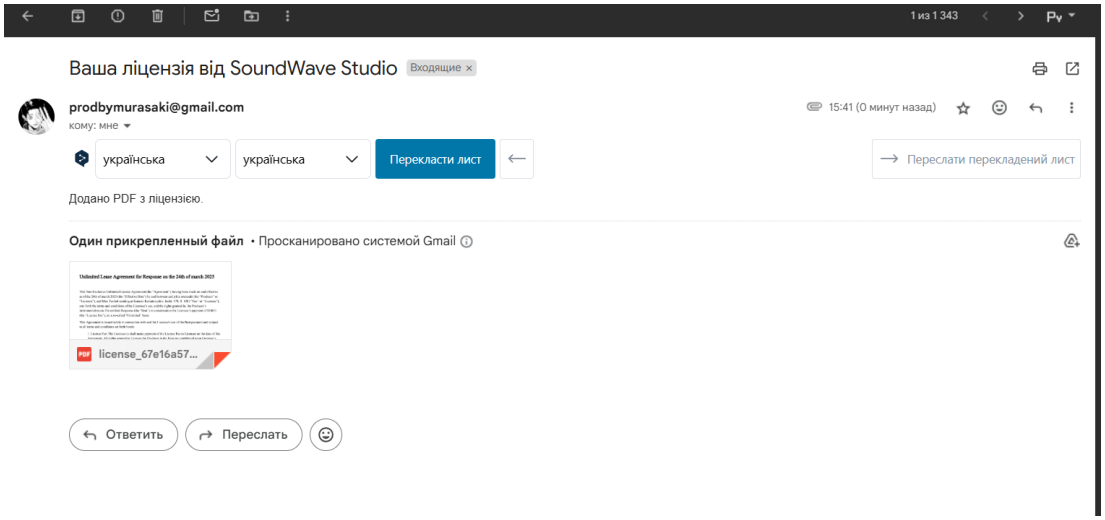


Рисунок 4.17 – Отримана ліцензія

Отримана ліцензія надходить на електронну пошту клієнта у вигляді листа від студії. Лист містить вкладений файл у форматі PDF, у якому збережена повна версія ліцензійної угоди між продюсером та клієнтом. У темі листа зазвичай вказується назва студії та короткий опис змісту, що дозволяє клієнту одразу ідентифікувати призначення повідомлення.

У тілі листа відображається коротке повідомлення про доданий файл з ліцензією. Користувач може безпосередньо переглянути прикріплений документ або завантажити його на свій пристрій для подальшого ознайомлення або збереження. Система також автоматично сканує вкладення для безпеки, що гарантує цілісність і захист переданих даних.

У розділі «Фінанси» доступна фінансова статистика: місячний, річний та загальний дохід. Також можна фільтрувати інформацію за датою, переглядати кількість проданих ліцензій та їхні типи. Система дозволяє побачити розподіл доходу за категоріями ліцензій (наприклад, Unlimited, Exclusive), що допомагає проаналізувати, який тип послуг користується найбільшим попитом.

На рисунку 4.18 представлений приклад сторінки фінансової аналітики з візуалізацією основних показників діяльності.

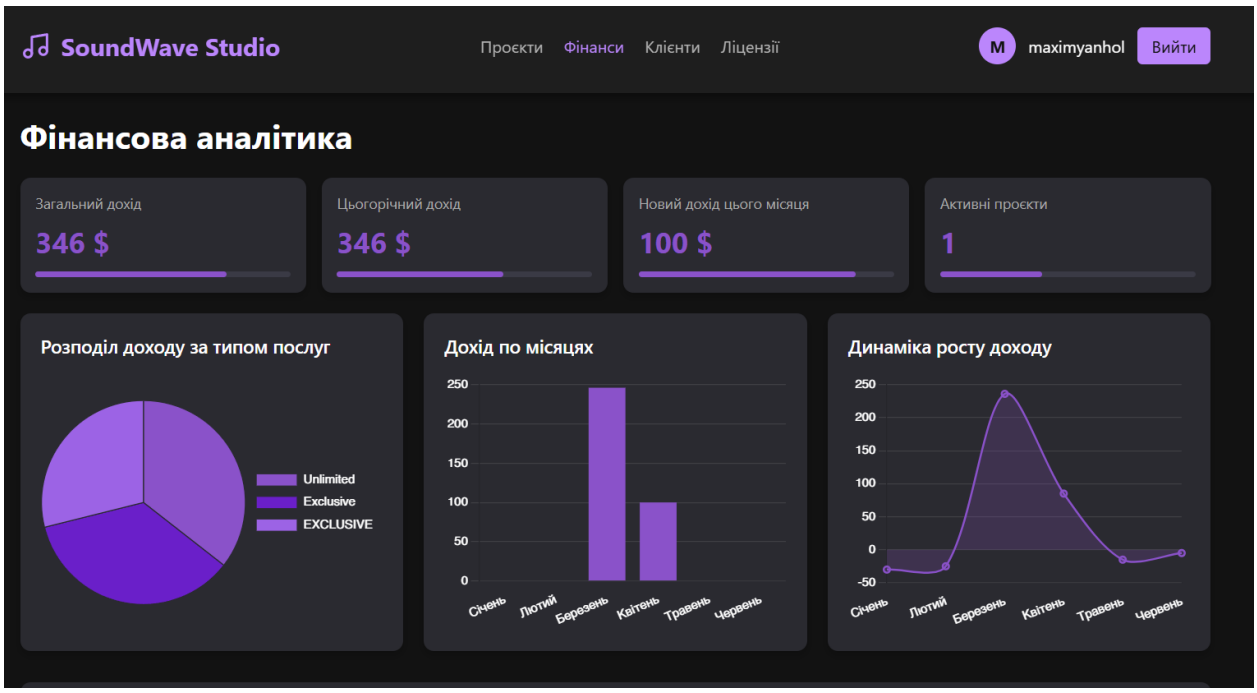


Рисунок 4.18 – Фінансова аналітика

Додатково на сторінці представлена інформація про дохід по місяцях у вигляді стовпчастої діаграми, що дозволяє легко оцінити фінансові показники в динаміці за обраний період. Графік динаміки росту доходу відображає тренди зміни доходів за місяцями та дозволяє швидко виявляти періоди активного зростання або спаду.

Інтерфейс системи побудований таким чином, щоб забезпечити максимально швидкий доступ до важливої інформації та сприяти зручному прийняттю рішень щодо подальшого розвитку студії.

4.3 Практичне застосування програмного застосунку

Розроблений програмний застосунок, є комплексним інструментом для автоматизації бізнес-процесів студії звукозапису, об’єднуючи модулі управління клієнтами, проєктами, ліцензіями, фінансової аналітики та документообігу. Завдяки модульній структурі він підходить для студій різного масштабу, від невеликих незалежних колективів до великих комерційних студій, які прагнуть оптимізувати робочі процеси. Застосунок забезпечує зручний інтерфейс для виконання складних завдань, таких як генерація ліцензій, аналіз доходів і

планування проєктів, і є доступним як для менеджерів студій, так і для продюсерів із мінімальним досвідом роботи з подібними системами.

Однією із основних функцій програмного забезпечення, є генерація та документообіг ліцензій. Модуль ліцензії містить основні елементи як таблиця всіх ліцензії та основні функції модуля. Генерація та документообіг реалізовано на серверній частині застосунку. Було наведено порівняльний аналіз створення ліцензій з використанням традиційних методів та з використанням програмного застосунку.

Основні кроки для створення ліцензії з використанням традиційних методів:

1. Запис даних про ліцензію, менеджер чи продюсер вручну записує дані про ліцензію (назва, тип, ціна, права) у блокнот, таблицю Excel (див рис. 4.19).

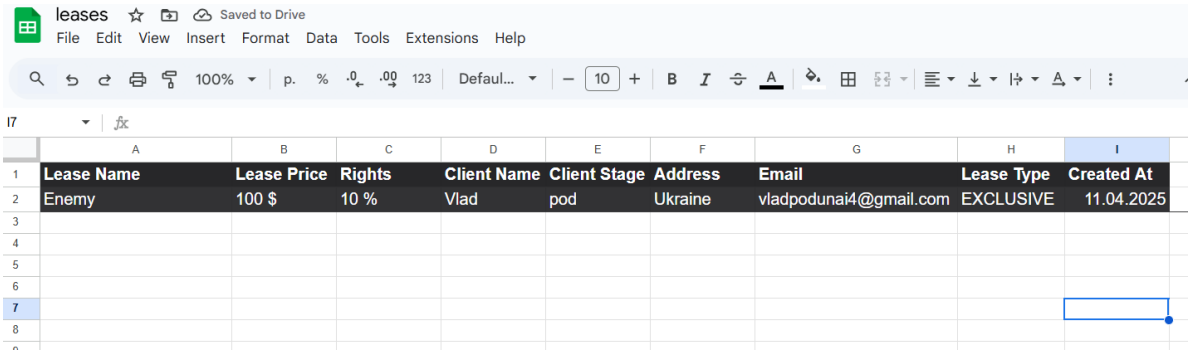


Рисунок 4.19 – Створення ліцензії з використання Google Table

2. Створення чистого документу, відкривається текстовий редактор (наприклад, Microsoft Word) для створення нового документа.
3. Вибір угоди в залежності від типу ліцензії, обирається відповідний шаблон угоди із попередньо підготовлених файлів.
4. Вставка документа іншого клієнта як шаблон, використовується документ іншого клієнта як основа, що потребує адаптації (див. рис. 4.20).

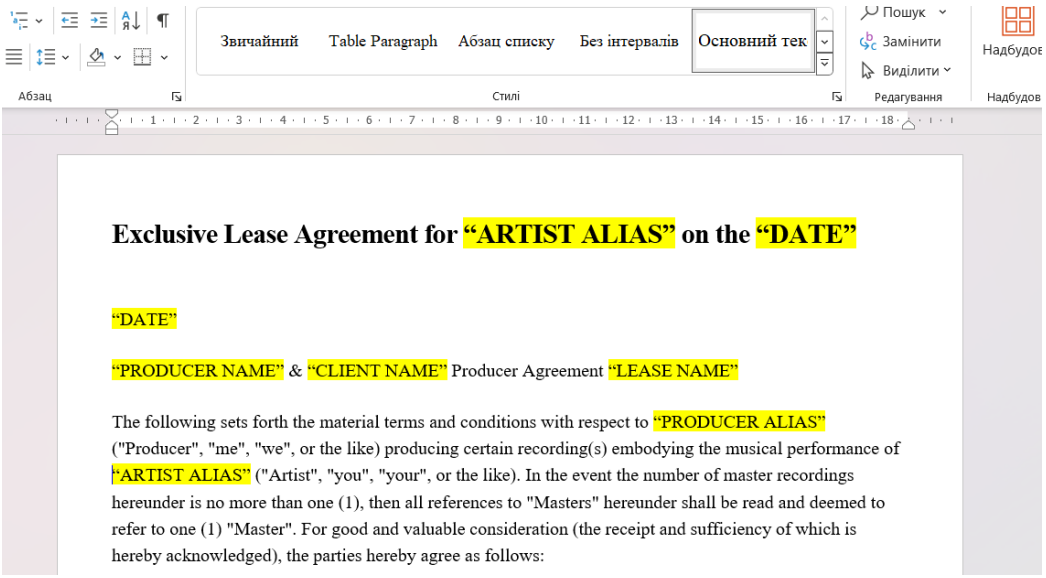


Рисунок 4.20 – Використання шаблону

5. Ввід даних клієнта з поправкою шаблону документа, вручну вносяться дані клієнта (ім’я, email, деталі угоди), коригується шаблон, що часто призводить до помилок через людський фактор, зображено на рисунку 4.21.

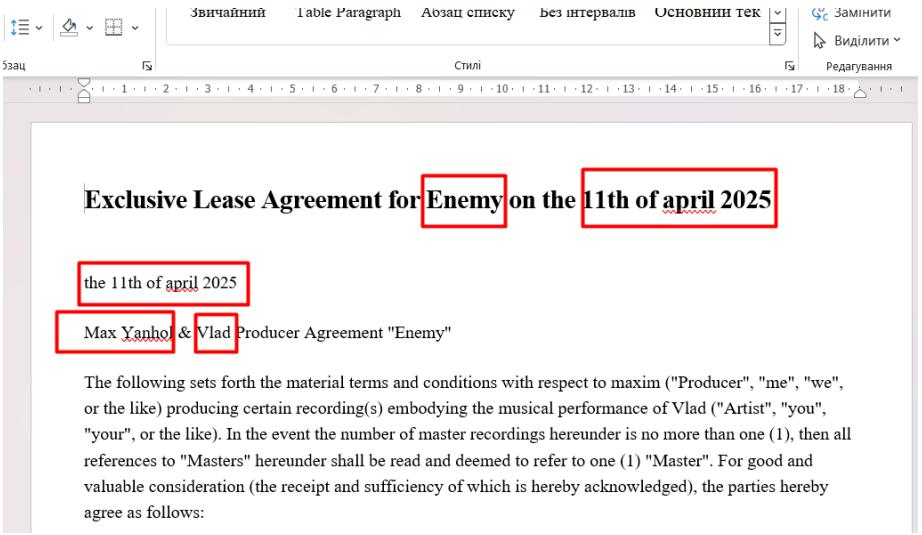


Рисунок 4.21 – Редагування документа вручну

6. Експорт в PDF-документ. Документ експортується у формат PDF через функцію збереження редактора.

7. Надсилання клієнту на його пошту. Менеджер вручну прикріплює файл до листа і надсилає його клієнту через email-клієнт, зображено на рисунку 4.22.

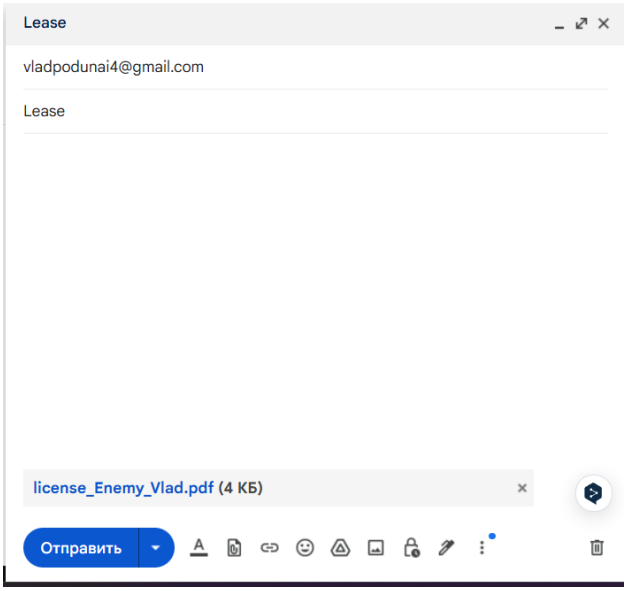


Рисунок 4.22 – Відправлення ліцензії клієнту вручну

Основна проблема традиційного методу полягає у високій імовірності помилок через ручне введення даних, що може призводити до некоректних угод (наприклад, неправильне ім’я клієнта чи сума). Процес займає значний час – від 5 до 7 хвилин на одну ліцензію, що стає критичним при великій кількості клієнтів. Крім того, відсутність автоматизації ускладнює відстеження історії ліцензій, а ручне надсилання документів через email не забезпечує належного захисту даних і може призвести до витоку інформації.

Процес створення ліцензії з використанням програмного застосунку:

1. Вибір клієнта та введення даних, у модулі «Ліцензії» користувач обирає клієнта з випадаючого меню (з інтеграцією з модулем «Клієнти»), вводить дані ліцензії (назва, ціна, права, тип) у зручній формі. Форма має інтуїтивно зрозумілий інтерфейс, що дозволяє швидко та зручно вводити всю необхідну інформацію. Валідація полів забезпечує правильність заповнення, а всі введені дані автоматично зберігаються у відповідну колекцію бази даних MongoDB після натискання кнопки «Створити лізинг». Це значно спрощує процес

оформлення ліцензій та мінімізує кількість помилок, порівняно з ручним введенням, зображено на рисунку 4.23.

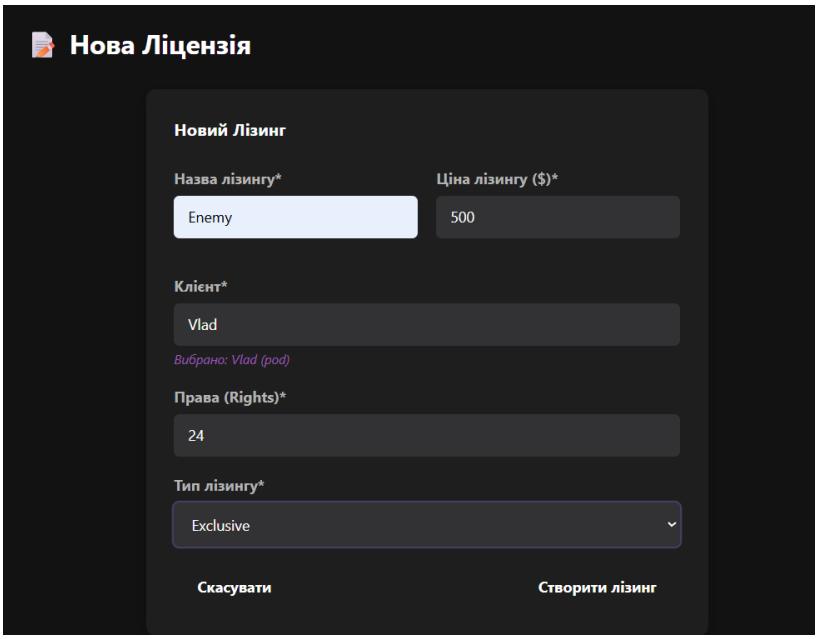


Рисунок 4.23 – Створення ліцензії у програмному застосунку

2. Автоматична генерація PDF. Після натискання кнопки «Згенерувати» система створює PDF-документ на основі динамічного шаблону, враховуючи тип ліцензії та дані клієнта, зображено на рисунку 4.24.

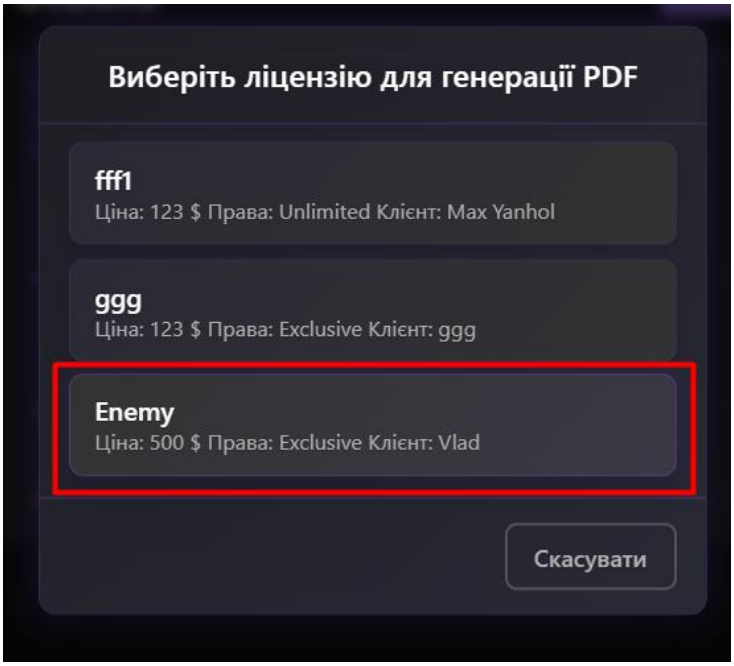


Рисунок 4.24 – Генерація ліцензії локально

3. Надсилання на email, користувач натискає кнопку «Відправити», вводить email клієнта, SoundWave через поштовий сервіс надсилає ліцензію клієнту з прикріпленим PDF-файлом.

Також з однією із основних функцій програмного забезпечення є управління проєктами. Модуль проєктів містить основні елементи, такі як таблиця всіх проєктів, а також функції створення, редагування, видалення та відстеження прогресу.

Основні кроки для управління проєктами з використанням традиційних методів:

1. Запис даних про проєкт, менеджер вручну записує дані про проєкт (назва, клієнт, статус) у паперовому журналі або таблиці Excel, зображено на рисунку 4.25.

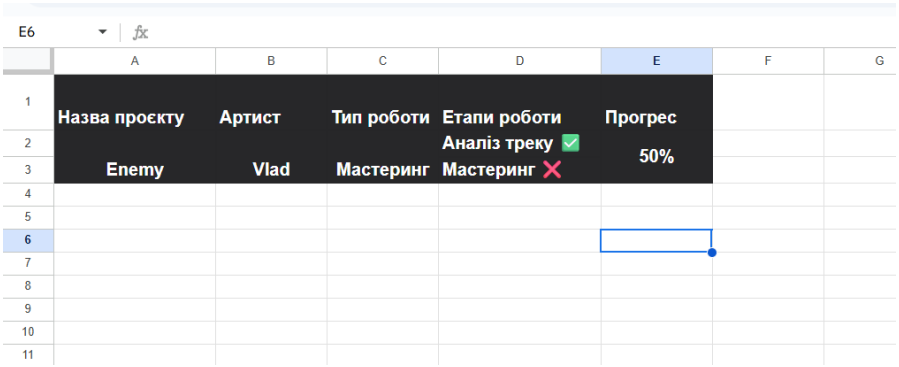


Рисунок 4.25 – Використання Google Table для менеджменту проєкту

2. Відстеження прогресу, ручне оновлення статусу проєкту (наприклад, «Активний» або «Завершений») проводиться через періодичні перевірки або телефонні дзвінки, що займає час.
3. Облік завершених проєктів, завершені проєкти переносяться до окремого списку вручну, що часто призводить до помилок або пропущених записів.

Основна проблема традиційного методу полягає у високій трудоємності та ймовірності помилок через ручне ведення. Процес відстеження прогресу і повідомлення клієнтів може займати від 10 до 15 хвилин на проєкт, особливо при великій кількості замовлень. Відсутність централізованої системи ускладнює

контроль, а ручне перенесення даних призводить до невідповідностей між записами.

Процес управління проєктами з використанням SoundWave:

1. Створення проєкту, у модулі «Проєкти» користувач обирає клієнта з інтегрованого списку, вводить дані проєкту (назва, статус, прогрес) у зручній формі, а система автоматично прив’язує його до профілю клієнта, зображено на рисунку 4.26.

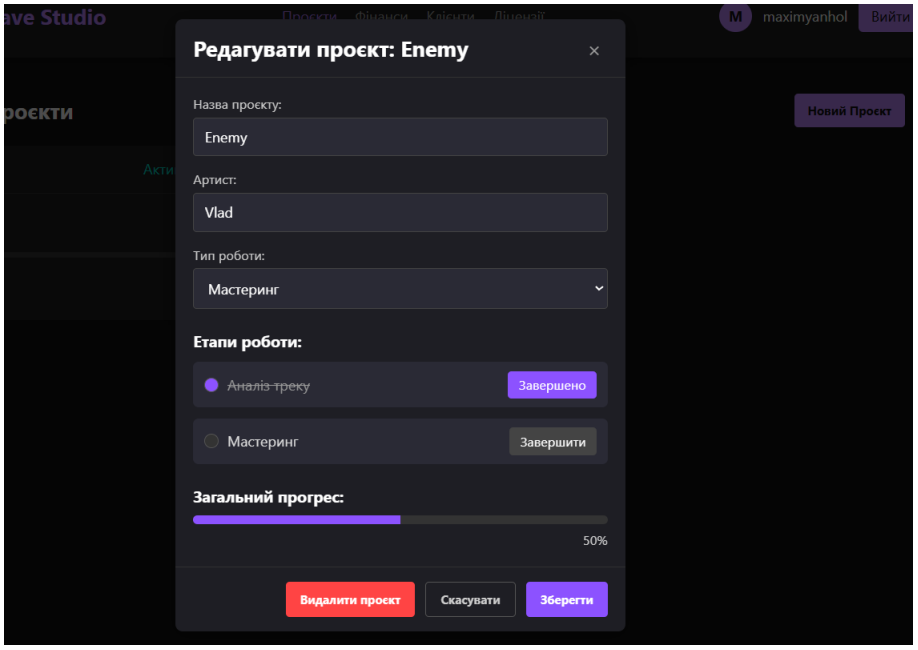


Рисунок 4.26 – Використання SoundWave для менеджменту проєкту

2. Відстеження прогресу, користувач оновлює прогрес у відсотках через інтерфейс, а при досягненні 100% проєкт автоматично змінює статус на «Завершений» і переноситься до відповідної категорії.

Таким чином, програмний застосунок значно перевершує традиційні методи управління проєктами за швидкістю, точністю і контролем, оптимізуючи робочі процеси студії, зменшуючи рутинну роботу і підвищуючи продуктивність.

Для підтвердження досягнення поставленої мети роботи та практичного застосування, у таблиці 4.1 наведено порівняння між застосунком SoundWave та аналогами, розглянутими в таблиці 1.1. Таблиця демонструє, що програмний

застосунок реалізує унікальні функції, яких немає в інших рішеннях, або покращує їхню реалізацію, забезпечуючи вищу ефективність і зручність для користувачів.

Таблиця 4.1 – Порівняння наявності унікальних функцій

Критерії	Sound Studio Management	Trello	StudioCloud	SoundWave
Автоматизований документообіг	-	-	-	+
Управління проєктами	-	-	-	+
Управління клієнтами	+	-	+	+
Фінансова аналітика	+	-	+	+

Отже, розроблений застосунок перевершує існуючі рішення завдяки реалізації унікальних функцій, таких як автоматизована генерація та надсилання PDF-документів ліцензій, управління проєктами, захищений доступ до даних та наявність повноцінного REST API.

4.4 Висновки

У четвертому розділі було проведено всебічне тестування програмного забезпечення, розробленого для автоматизації роботи студії звукозапису. Зокрема, перевірялися основні модулі системи: модуль обробки та зберігання інформації про клієнтів, модуль управління проєктами, модуль створення та відправки ліцензій, тощо.

У ході тестування була змодельована низка типових та критичних сценаріїв взаємодії користувача з системою. Особливу увагу приділено перевірці стійкості до помилкових даних, підтримці коректної логіки у випадку порушення

послідовності дій, а також перевірці взаємозв’язку між модулями. Завдяки цьому вдалося переконатися в надійності функціонування системи, її готовності до експлуатації в умовах реальної студійної діяльності, з урахуванням потреб клієнтів, обробки великої кількості замовлень та формування ліцензій.

Окремо було розроблено детальну інструкцію користувача, яка описує порядок інсталяції програмного забезпечення, технічні вимоги, а також послідовність дій для ефективної роботи з програмою. Це дозволяє користувачу швидко адаптуватися до інтерфейсу системи та забезпечити її ефективне використання без додаткових витрат часу на навчання.

Таким чином, тестування підтвердило відповідність програмного забезпечення заявленим функціональним вимогам, а також його практичну придатність для використання в реальних умовах роботи студії звукозапису.

ВИСНОВКИ

Програмний продукт SoundWave забезпечує комплексну автоматизацію ключових аспектів діяльності студії, включаючи управління клієнтами, проєктами, ліцензіями та фінансами. Реалізація роботи відповідає методичним вказівкам [24]. Під час виконання роботи було проведено аналіз сучасного стану технологій автоматизації бізнес-процесів у студіях звукозапису. Розглянуто основні аналоги, виявлено їхні недоліки та проведено порівняння з розробленим продуктом.

У процесі аналізу технологій розробки було обґрунтовано вибір мов програмування та було вибрано Java для бекенду та JavaScript для фронтенду, також було обґрунтовано вибір бібліотек та фреймворків, які були використані.

У бакалаврській кваліфікаційній роботі виконано наступне:

- розроблено зручний та інтуїтивно зрозумілий інтерфейс для ефективного управління замовленнями, проектами, клієнтами, фінансами і ліцензіями;
- розроблено модуль фінансової системи, забезпечуючи точність обліку, генерацію звітів;
- розроблено модуль для планування та організації роботи, відстеження прогресу і управління проєктами;
- реалізовано модуль для зберігання інформації про проєкти, фінанси, клієнтів, фінансів, ліцензій та взаємодії користувача між ними;
- об'єднано всі модулі в єдину платформу для ефективної автоматизації всіх бізнес-процесів студії.
- проведено всебічне тестування для забезпечення стабільності та відповідності вимогам системи.

Було створено схеми алгоритму генерації ліцензії та схему загальної роботи застосунку. Крім того, було розроблено модель системи з використанням UML-діаграми. Було проведено тестування та було підтверджено працездатність програмного застосунку та створено інструкцію користувача. Отже поставлену мету бакалаврської кваліфікаційної роботи досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. P. White, The Studio SOS Book: Solutions and Techniques for the Project Recording Studio, 1st Edition, Routledge, 2020, 302 p.

2. B. Ruecker, Practical Process Automation: Orchestration and Integration in Microservices and Cloud Native Architectures, 1st Edition, O'Reilly Media, 2021, 292 p.

3. G. Blokdyk, Business Process Automation BPA: A Clear and Concise Reference, 5STARCooks, 2021, 311 p.

4. Янголь М.О., Розробка програмного забезпечення для автоматизації бізнеспроцесів студії звукозапису. Вінниця: ВНТУ, 2025. 2с. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24375/20074 [Електронний ресурс] – Режим доступу до ресурсу: https://ist.kpi.ua/uk/konferencziya-infocom/ (дата звернення 10.03.2025)

5. Янголь М.О. Програмний застосунок для аналізу та підвищення продуктивності роботи комп’ютера. XIII Міжнародна науково-практична конференція з інформаційних систем та технологій Infocom Advanced Solutions 2025. Київ: КПІ, 2025. 3с. [Електронний ресурс] – Режим доступу до ресурсу: https://ist.kpi.ua/uk/konferencziya-infocom/ (дата звернення 10.03.2025)

6. StudioCloud [Електронний ресурс] – Режим доступу до ресурсу: https://studiocloud.com/ (дата звернення 14.03.2025).

7. Trello [Електронний ресурс] – Режим доступу до ресурсу: https://trello.com/en (дата звернення 14.03.2025).

8. Sound Studio Management [Електронний ресурс] – Режим доступу до ресурсу: https://mysonido.com/ (дата звернення 14.03.2025).

9. G. Kim, K. Beht, G. Spafford, The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win, 2014, IT Revolution Press, 2014, 376 p.

10. M. Heckler, Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications, 1st Edition, O'Reilly Media, 2021, 325 p.

11. B. Cherny, Programming TypeScript: Making Your JavaScript Applications Scale, 1st Edition, O'Reilly Media, 2019, 322 p.

12. S. Bradshaw, MongoDB: The Definitive Guide: Powerful and Scalable Data Storage, 3rd Edition, O'Reilly Media, 2020, 511 p.

13. IntelliJ IDEA [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/idea/ (дата звернення 25.03.2025).

14. D. Reeves, UX/UI Graphic Design For Beginners Made Simple: The Comprehensive Guide How to Master Essential Tools, Create Stunning Interfaces, & Improve User Experience for Your Clients Easily & Effectively, Ahava Drew Publishing LLC, 2025, 142 p.

15. ItextPdf [Електронний ресурс] – Режим доступу до ресурсу: https://itextpdf.com/ (дата звернення 25.03.2025).

16. A. Dennis, B. Wixom, D. Tegarden, Systems Analysis and Design: An Object-Oriented Approach with UML, 6th Edition, Wiley, 544 p.

17. J. Gough, D. Bryant, M. Auburn, Mastering API Architecture: Design, Operate, and Evolve API-Based Systems, 1st Edition, O'Reilly Media, 286 p.

18. Spring Dependency Injection [Електронний ресурс] – Режим доступу до ресурсу: https://www.baeldung.com/spring-dependency-injection (дата звернення 28.03.2025).

19. Cursor IDE [Електронний ресурс] – Режим доступу до ресурсу: https://www.cursor.com/ (дата звернення 28.03.2025).

20. B. Pollard, HTTP/2 in Action, 1st Edition, Manning, 416 p.

21. JSON [Електронний ресурс] – Режим доступу до ресурсу: https://www.json.org/json-en.html (дата звернення 29.03.2025).

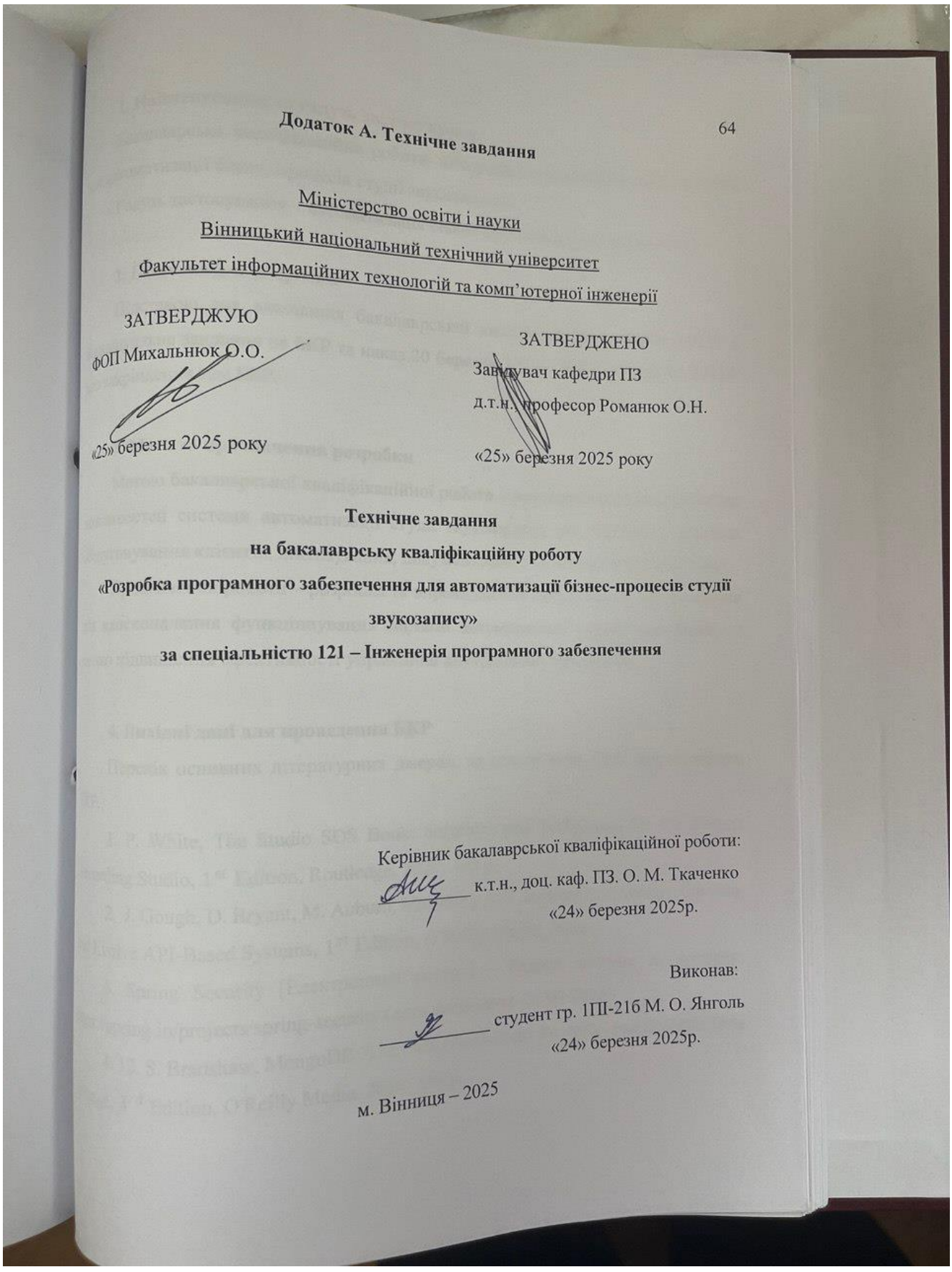
22. Spring Security [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-security (дата звернення 31.03.2025).

23. C. Domoney, Defending APIs: Uncover advanced defense techniques to craft secure application programming interfaces, 1st Edition, Packt Publishing, 384 p.

24. Романюк О. Н., Чорноволик Г. О., Методичні вказівки до виконання бакалаврських дипломних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення»: методичні – вказівки Вінниця : ВНТУ, 2022. 51 с.

ДОДАТКИ

Додаток А. Технічне завдання



Додаток А. Технічне завдання 64

Міністерство освіти і науки
Вінницький національний технічний університет
факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
ФОП Михальнюк О.О.
«25» березня 2025 року

ЗАТВЕРДЖЕНО
Завідувач кафедри ПЗ
д.т.н., професор Романюк О.Н.
«25» березня 2025 року

Технічне завдання
на бакалаврську кваліфікаційну роботу
«Розробка програмного забезпечення для автоматизації бізнес-процесів студії звукозапису»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. каф. ПЗ. О. М. Ткаченко
«24» березня 2025р.

Виконав:
студент гр. 1ПІ-21б М. О. Янголь
«24» березня 2025р.

м. Вінниця – 2025

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмного забезпечення для автоматизації бізнес-процесів студії звукозапису»

Галузь застосування – автоматизація бізнес-процесів студії звукозапису.

2. Підстава для розробки

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ 20 березня 2025р. №97 ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки

Метою бакалаврської кваліфікаційної роботи є розширення функціональних можливостей системи автоматизації студії звукозапису, яка оптимізує процеси обслуговування клієнтів, ліцензування, документообігу та управління проєктами.

Призначення роботи – розробка та впровадження програмного забезпечення для вдосконалення функціонування системи автоматизації студії звукозапису з метою підвищення ефективності управління внутрішніми процесами.

4. Вихідні дані для проведення БКР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. P. White, The Studio SOS Book: Solutions and Techniques for the Project Recording Studio, 1^{st} Edition, Routledge, 2020, 302 p.

2. J. Gough, D. Bryant, M. Auburn, Mastering API Architecture: Design, Operate, and Evolve API-Based Systems, 1^{st} Edition, O'Reilly Media, 286 p.

3. Spring Security [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-security (дата звернення 31.03.2025).

4. 12. S. Bradshaw, MongoDB: The Definitive Guide: Powerful and Scalable Data Storage, 3^{rd} Edition, O'Reilly Media, 2020, 511 p.

5. Технічні вимоги

Методи обробки баз даних для створення сховища для збереження даних; методи проектування програмного забезпечення з застосуванням принципів SOLID для розробки REST-API та формування структури клієнт-серверної взаємодії; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт

1. Пояснювальна записка до БКР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ

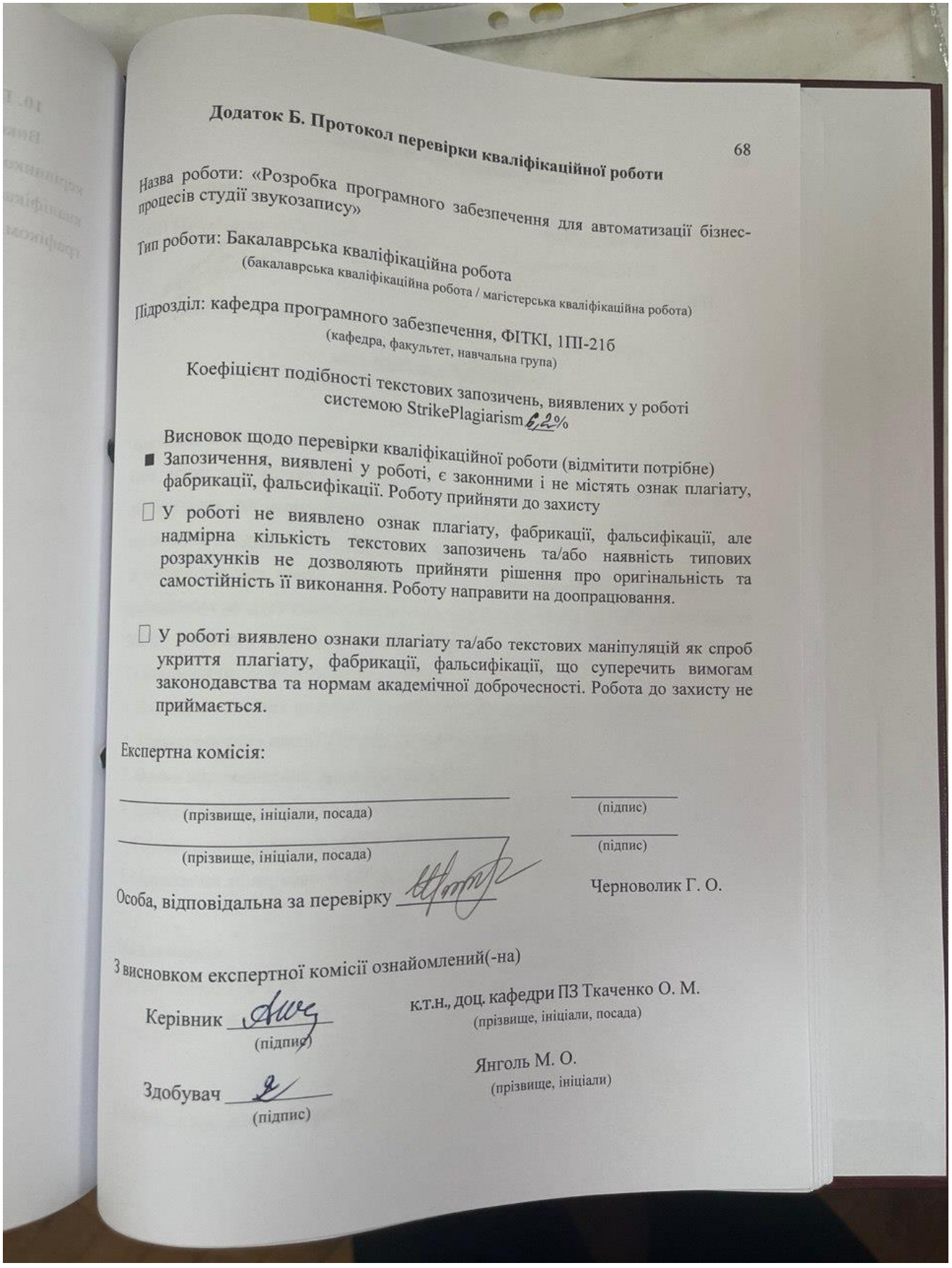
9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1.	Аналіз стану технологій автоматизації бізнес процесів у студіях звукозапису	25.03.25-02.04.25
2.	Розробка моделі системи та методів програмного застосунку	03.04.25-14.04.25
3.	Розробка програмних модулів програмного застосунку	15.04.25-02.05.25
4.	Тестування і практичне застосування системи	03.05.25-26.05.25
5.	Оформлення матеріалів до захисту БДР	27.05.25-30.05.25

10. Порядок контролю та прийняття

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б. Протокол перевірки кваліфікаційної роботи



Додаток Б. Протокол перевірки кваліфікаційної роботи

68

Назва роботи: «Розробка програмного забезпечення для автоматизації бізнес-процесів студії звукозапису»

Тип роботи: Бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: кафедра програмного забезпечення, ФІТКІ, 1ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 6,2%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

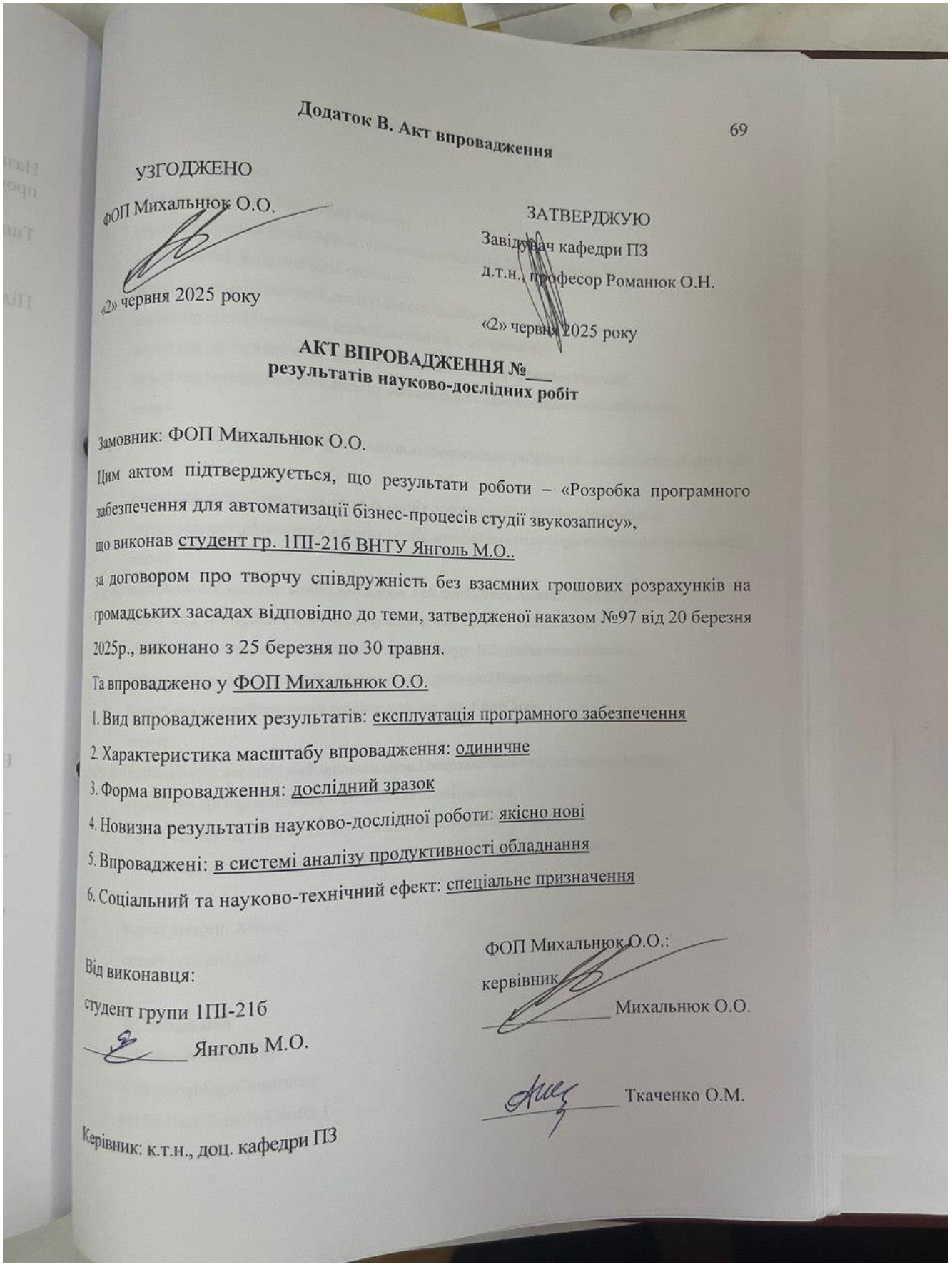
Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ к.т.н., доц. кафедри ПЗ Ткаченко О. М.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Янголь М. О.
(підпис) (прізвище, ініціали)

Додаток В. Акт впровадження



Додаток В. Акт впровадження 69

УЗГОДЖЕНО
ФОП Михальнюк О.О.
«2» червня 2025 року

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор Романюк О.Н.
«2» червня 2025 року

АКТ ВПРОВАДЖЕННЯ №___
результатів науково-дослідних робіт

Замовник: ФОП Михальнюк О.О.

Цим актом підтверджується, що результати роботи – «Розробка програмного забезпечення для автоматизації бізнес-процесів студії звукозапису»,

що виконав студент гр. 1ПІ-21б ВНТУ Янголь М.О..

за договором про творчу співдружність без взаємних грошових розрахунків на громадських засадах відповідно до теми, затвердженої наказом №97 від 20 березня 2025р., виконано з 25 березня по 30 травня.

Та впроваджено у ФОП Михальнюк О.О.

1. Вид впроваджених результатів: експлуатація програмного забезпечення
2. Характеристика масштабу впровадження: одиничне
3. Форма впровадження: дослідний зразок
4. Новизна результатів науково-дослідної роботи: якісно нові
5. Впроваджені: в системі аналізу продуктивності обладнання
6. Соціальний та науково-технічний ефект: спеціальне призначення

Від виконавця:
студент групи 1ПІ-21б
_____ Янголь М.О.

ФОП Михальнюк О.О.:
керівник
_____ Михальнюк О.О.

_____ Ткаченко О.М.

Керівник: к.т.н., доц. кафедри ПЗ

Додаток Г. Лістинг програми

SecurityConfig.java:

```
package com.yanhol.StudioApp.security;
import com.yanhol.StudioApp.service.implemintation.CustomUserDetailsService;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfigurat
ion;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.util.Arrays;
import java.util.List;

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig {
```

```
    private final JwtRequestFilter jwtRequestFilter;
    private final CustomUserDetailsService userDetailsService;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .cors(cors -> cors.configurationSource(corsConfigurationSource()))
            .csrf(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .anyRequest().authenticated()
            )
            .sessionManagement(session -> session
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            );

        http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(List.of("http://localhost:3000"));
        configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE",
"OPTIONS"));
        configuration.setAllowedHeaders(Arrays.asList("Authorization", "Content-Type"));
        configuration.setAllowCredentials(true);
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }

    @Bean
```

```
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration
authenticationConfiguration) throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }
}
```

AuthController.java:

```
package com.yanhol.StudioApp.controllers;

import com.yanhol.StudioApp.dto.UserDto;
import com.yanhol.StudioApp.exceptions.ValidationException;
import com.yanhol.StudioApp.models.User;
import com.yanhol.StudioApp.service.AuthService;
import com.yanhol.StudioApp.service.UserService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.context.support.DefaultMessageSourceResolvable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.BindingResult;
```

```
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
public class AuthController {

    private final UserService userService;
    private final AuthService authService;

    @PostMapping("/register")
    public ResponseEntity<?> createUser(@Valid @RequestBody UserDto userDto,
BindingResult bindingResult) {
        if (bindingResult.hasErrors()) {
            List<String> errors = bindingResult.getAllErrors()
                    .stream()
                    .map(DefaultMessageSourceResolvable::getDefaultMessage)
                    .collect(Collectors.toList());
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errors);
        }
        List<String> errors = userService.checkUniqueFields(userDto);
        if (!errors.isEmpty()) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errors);
        }
        User user = userService.saveUser(userDto);
        return ResponseEntity.status(HttpStatus.CREATED).body(user);
    }

    @PostMapping("/login")
    public ResponseEntity<?> loginUser(@RequestBody UserDto userDto) {
        try {
            String user = authService.login(userDto.getEmail(), userDto.getPassword());
```

```
            return ResponseEntity.ok(user);
        } catch (ValidationException ex) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(ex.getMessage());
        }
    }
}
```

ClientController.java

```
package com.yanhol.StudioApp.controllers;

import com.yanhol.StudioApp.dto.ClientDto;
import com.yanhol.StudioApp.exceptions.UpdateClientException;
import com.yanhol.StudioApp.models.Client;
import com.yanhol.StudioApp.service.ClientService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.context.support.DefaultMessageSourceResolvable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.server.ResponseStatusException;

import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/api")
@RequiredArgsConstructor
public class ClientController {

    private final ClientService clientService;

    @PostMapping("/client")
```

```
    public ResponseEntity<?> saveClient(@RequestBody @Valid ClientDto clientDto,
BindingResult bindingResult) {
        if (bindingResult.hasErrors()) {
            List<String> errors = bindingResult.getAllErrors()
                    .stream()
                    .map(DefaultMessageSourceResolvable::getDefaultMessage)
                    .collect(Collectors.toList());
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errors);
        }

        try {
            Client client = clientService.saveClient(clientDto);
            return ResponseEntity.status(HttpStatus.CREATED).body(client);
        } catch (UpdateClientException ex) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(ex.getMessage());
        } catch (Exception ex) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body("Помилка при створенні клієнта: " + ex.getMessage());
        }
    }

    @GetMapping("/client")
    public ResponseEntity<List<Client>> getClientsForCurrentUser() {
        try {
            List<Client> clients = clientService.getClientsForCurrentUser();
            return ResponseEntity.ok(clients);
        } catch (Exception ex) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body(null);
        }
    }

    @GetMapping("/client/{id}")
    public ResponseEntity<Client> getClientById(@PathVariable String id) {
        Optional<Client> client = clientService.findById(id);
```

```
        return client.map(ResponseEntity::ok)
                .orElseThrow(() -> new ResponseStatusException(
                        HttpStatus.NOT_FOUND,
                        "Клієнт з ID " + id + " не знайдений або не належить вам"
                ));
    }

    @PutMapping("/client/{id}")
    public ResponseEntity<?> updateClient(@PathVariable String id, @RequestBody @Valid
ClientDto clientDto, BindingResult bindingResult) {
        // Перевірка валідації DTO
        if (bindingResult.hasErrors()) {
            List<String> errors = bindingResult.getAllErrors()
                    .stream()
                    .map(DefaultMessageSourceResolvable::getDefaultMessage)
                    .collect(Collectors.toList());
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errors);
        }

        try {
            Client updatedClient = clientService.updateClient(id, clientDto);
            return ResponseEntity.ok(updatedClient);
        } catch (UpdateClientException ex) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(ex.getMessage());
        } catch (Exception ex) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body("Помилка при оновленні клієнта: " + ex.getMessage());
        }
    }

    @DeleteMapping("/client/{id}")
    public ResponseEntity<?> deleteClient(@PathVariable String id) {
        try {
            clientService.deleteClient(id);
            return ResponseEntity.noContent().build();
```

```
    } catch (IllegalArgumentException ex) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND)
                .body("Клієнт з ID " + id + " не знайдений або не належить вам");
    } catch (Exception ex) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("Помилка при видаленні клієнта: " + ex.getMessage());
    }
}

@GetMapping("/client/id/{id}")
public ResponseEntity<Client> getClientByIdForOrder(@PathVariable String id) {
    Optional<Client> client = clientService.findById(id);
    return client.map(ResponseEntity::ok).orElseThrow(() -> new ResponseStatusException(
            HttpStatus.NOT_FOUND,
            "Клієнт з ID " + id + " не знайдений або не належить вам"
    ));
}

@GetMapping("/clients/search")
public ResponseEntity<?> searchClients(@RequestParam String name) {
    try {
        List<Client> clients = clientService.searchClients(name);
        return ResponseEntity.ok(clients);
    } catch (Exception ex) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("Помилка при пошуку клієнтів: " + ex.getMessage());
    }
}

@GetMapping("/client/name/{name}")
public ResponseEntity<?> findIdByName(@PathVariable String name) {
    try {
        String clientId = clientService.findIdByName(name);
        return ResponseEntity.ok(clientId);
    } catch (ResponseStatusException ex) {
```

```
            return ResponseEntity.status(HttpStatus.NOT_FOUND)
                    .body("Клієнт з ім’ям " + name + " не знайдений або не належить вам");
        } catch (Exception ex) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body("Помилка при пошуку клієнта за ім’ям: " + ex.getMessage());
        }
    }
}
```

LeaseController.java:

```
package com.yanhol.StudioApp.controllers;

import com.yanhol.StudioApp.dto.LeaseDto;
import com.yanhol.StudioApp.models.Lease;
import com.yanhol.StudioApp.service.LeaseService;
import com.yanhol.StudioApp.service.PdfGenerationService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.context.support.DefaultMessageSourceResolvable;
import org.springframework.core.io.ByteArrayResource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

import java.io.IOException;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/api")
@RequiredArgsConstructor
```

```
public class LeaseController {

    private final LeaseService leaseService;
    private final PdfGenerationService pdfGenerationService;



    @PostMapping("/lease")
    public ResponseEntity<?> saveLease(@RequestBody @Valid LeaseDto leaseDto,
BindingResult bindingResult) {
        if (bindingResult.hasErrors()) {
            List<String> errors = bindingResult.getAllErrors()
                    .stream()
                    .map(DefaultMessageSourceResolvable::getDefaultMessage)
                    .collect(Collectors.toList());
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errors);
        }
        Lease lease = leaseService.saveLease(leaseDto);
        return ResponseEntity.status(HttpStatus.CREATED).body(lease);
    }

    @GetMapping("/leases")
    public ResponseEntity<List<Lease>> getAllLeases() {
        List<Lease> leases = leaseService.getAllLeases();
        return ResponseEntity.ok(leases);
    }

    @PutMapping("/lease/{id}")
    public ResponseEntity<?> updateLease(@PathVariable String id, @RequestBody @Valid
LeaseDto leaseDto, BindingResult bindingResult) {
        if (bindingResult.hasErrors()) {
            List<String> errors = bindingResult.getAllErrors()
                    .stream()
                    .map(DefaultMessageSourceResolvable::getDefaultMessage)
                    .collect(Collectors.toList());
```

```
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errors);
        }
        try {
            Lease updatedLease = leaseService.updateLease(id, leaseDto);
            return ResponseEntity.ok(updatedLease);
        } catch (IllegalArgumentException ex) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
        }
    }

    @DeleteMapping("/lease/{id}")
    public ResponseEntity<Void> deleteLease(@PathVariable String id) {
        try {
            leaseService.deleteLease(id);
            return ResponseEntity.ok().build();
        } catch (IllegalArgumentException ex) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        }
    }

    @PostMapping("/lease/generate-pdf")
    public ResponseEntity<ByteArrayResource> generateLeasePDF(@RequestBody
Map<String, String> request) throws IOException {
        String leaseId = request.get("leaseId");
        byte[] pdfBytes = pdfGenerationService.generateLeasePDF(leaseId);

        ByteArrayResource resource = new ByteArrayResource(pdfBytes);

        HttpHeaders headers = new HttpHeaders();
        headers.add("Content-Disposition", "attachment; filename=license_" + leaseId + ".pdf");
        headers.add("Cache-Control", "no-cache, no-store, must-revalidate");
        headers.add("Pragma", "no-cache");
        headers.add("Expires", "0");

        return ResponseEntity.ok()
```

```
                .headers(headers)
                .contentLength(pdfBytes.length)
                .contentType(MediaType.APPLICATION_PDF)
                .body(resource);
        }

        @PostMapping("/lease/send-email")
        public ResponseEntity<String> sendLeaseEmail(@RequestBody Map<String, String>
request) {
            String leaseId = request.get("leaseId");
            String emailTo = request.get("emailTo");
            String message = request.get("message");

            try {
                pdfGenerationService.sendLeaseEmail(leaseId, emailTo, message);
                return ResponseEntity.ok("Ліцензію успішно відправлено на email " + emailTo);
            } catch (IOException e) {
                return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                        .body("Помилка при відправці email: " + e.getMessage());
            } catch (jakarta.mail.MessagingException e) {
                throw new RuntimeException(e);
            }
        }

        @GetMapping("/leases/month")
        public List<Lease> getLeasesByMonth(@RequestParam int year, @RequestParam int month)
{
            return leaseService.getLeasesByMonth(year, month);
        }

        @GetMapping("/leases/months")
        public List<Lease> getLeasesByMonths(@RequestParam int year, @RequestParam
List<Integer> months) {
            return leaseService.getLeasesByMonths(year, months);
        }
```

```
    @GetMapping("/stats/total-revenue")
    public double getTotalRevenue() {
        return leaseService.getTotalRevenue();
    }

    @GetMapping("/stats/yearly-revenue")
    public double getYearlyRevenue(@RequestParam int year) {
        return leaseService.getYearlyRevenue(year);
    }

    @GetMapping("/stats/monthly-revenue")
    public double getMonthlyRevenue(@RequestParam int year, @RequestParam int month) {
        return leaseService.getMonthlyRevenue(year, month);
    }
}
```

OrderController.java:

```
package com.yanhol.StudioApp.controllers;

import com.yanhol.StudioApp.dto.OrderDto;
import com.yanhol.StudioApp.models.Order;
import com.yanhol.StudioApp.service.OrderService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api")
@RequiredArgsConstructor
public class OrderController {

    private final OrderService orderService;
```

```
    @PostMapping("/order")
    public ResponseEntity<Order> createOrder(@RequestBody OrderDto orderDto) {
        return ResponseEntity.ok(orderService.createOrder(orderDto));
    }

    @GetMapping("/orders")
    public ResponseEntity<List<Order>> getAllOrders() {
        return ResponseEntity.ok(orderService.getAllOrders());
    }

    @PutMapping("/order/{id}")
    public ResponseEntity<Order> updateOrder(@PathVariable String id, @RequestBody
OrderDto orderDto) {
        return ResponseEntity.ok(orderService.updateOrder(id, orderDto));
    }

    @DeleteMapping("/order/{id}")
    public ResponseEntity<Void> deleteOrder(@PathVariable String id) {
        orderService.deleteOrder(id);
        return ResponseEntity.ok().build();
    }

    @GetMapping("/stats/active-projects")
    public ResponseEntity<Integer> getActiveProjects() {
        return ResponseEntity.ok(orderService.getActiveProjects());
    }

}
```

UserController.java:

```
package com.yanhol.StudioApp.controllers;

import com.yanhol.StudioApp.dto.UserDto;
import com.yanhol.StudioApp.models.User;
```

```
import com.yanhol.StudioApp.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/users")
@RequiredArgsConstructor
public class UserController {

    private final UserService userService;

    @PutMapping("/{id}")
    public ResponseEntity<User> updateUser(
            @PathVariable String id,
            @RequestBody UserDto userDto,
            Authentication authentication) {
        String email = authentication.getName();
        User currentUser = userService.findByEmail(email);
        if (!currentUser.getId().equals(id)) {
            return ResponseEntity.status(403).build();
        }

        User updatedUser = userService.updateUser(id, userDto);
        return ResponseEntity.ok(updatedUser);
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> getUserById(
            @PathVariable String id,
            Authentication authentication) {
        String email = authentication.getName();
        User currentUser = userService.findByEmail(email);
        if (!currentUser.getId().equals(id)) {
```

```
            return ResponseEntity.status(403).build();
        }

        User user = userService.findById(id);
        return ResponseEntity.ok(user);
    }

    @GetMapping("/by-email")
    public ResponseEntity<String> getUserIdByEmail(Authentication authentication) {
        String email = authentication.getName();
        User user = userService. findByEmail(email);
        if (user == null) {
            return ResponseEntity.status(404).build();
        }
        return ResponseEntity.ok(user.getId());
    }
}
```

ValidationControllerDevice.java:

```
package com.yanhol.StudioApp.controllers;

import com.yanhol.StudioApp.exceptions.UpdateClientException;
import com.yanhol.StudioApp.exceptions.ValidationException;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestControllerAdvice
public class ValidationControllerAdvice {
```

```
    @ExceptionHandler(ValidationException.class)
    public ResponseEntity<List<String>> handleValidationExceptions(ValidationException ex)
{
        return new ResponseEntity<>(List.of(ex.getMessage()), HttpStatus.BAD_REQUEST);
    }
    @ExceptionHandler(UpdateClientException.class)
    public ResponseEntity<Map<String, String>>
handleUpdateClientException(UpdateClientException ex) {
        Map<String, String> error = new HashMap<>();
        error.put("error", ex.getMessage());
        return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);
    }
}
```

Client.java:

```
package com.yanhol.StudioApp.models;


import lombok.*;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.data.mongodb.core.mapping.Field;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Builder(toBuilder = true)
@Document(collection = "clients")
public class Client {
    @Id
    private String id;
    private String name;
    @Field(name = "stage_name")
    private String stageName;
```

```
    private String email;
    private String address;
    private String userId;
}
```

Lease.java:

```
package com.yanhol.StudioApp.models;

import lombok.*;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import java.time.LocalDateTime;

@Document(collection = "leases")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class Lease {
    @Id
    private String id;

    private String leaseName;
    private double leasePrice;
    private String clientId;
    private String leaseType;

    private Integer rights;

    private String clientName;
    private String clientStageName;
    private String clientEmail;
    private String clientAddress;
```

```
    private LocalDateTime createdAt;
    private String userId;
}
```

ClientServiceImpl.java:

```
package com.yanhol.StudioApp.service.implemintation;

import com.yanhol.StudioApp.dto.ClientDto;
import com.yanhol.StudioApp.exceptions.UpdateClientException;
import com.yanhol.StudioApp.models.Client;
import com.yanhol.StudioApp.repositories.ClientRepository;
import com.yanhol.StudioApp.service.ClientService;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;
import org.springframework.web.server.ResponseStatusException;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Service
@RequiredArgsConstructor

public class ClientServiceImpl implements ClientService {

    private final ClientRepository clientRepository;
    private static final Logger logger = LoggerFactory.getLogger(ClientServiceImpl.class);
```

```
    @Override
    public Client saveClient(ClientDto clientDto) {
        String userId = getCurrentUserEmail();
        checkUniqueFields(clientDto, userId);

        Client client = Client.builder()
                .name(clientDto.getName())
                .stageName(clientDto.getStageName())
                .email(clientDto.getEmail())
                .address(clientDto.getAddress())
                .userId(userId)
                .build();

        return clientRepository.save(client);
    }

    private void checkUniqueFields(ClientDto clientDto, String userId) {
        Optional<Client> existingClient = clientRepository.findByEmailAndUserId(clientDto.getEmail(), userId);
        if (existingClient.isPresent()) {
            throw new UpdateClientException("Email is already in use by another client of this user");
        }
    }

    @Override
    public List<Client> getClientsForCurrentUser() {
        String userEmail = getCurrentUserEmail();
        return clientRepository.findByUserId(userEmail);
    }

    @Override
    public Client updateClient(String id, ClientDto clientDto) {
        Optional<Client> optionalClient = clientRepository.findById(id);
```

```
        if (optionalClient.isEmpty()) {
            throw new UpdateClientException("Client not found with ID: " + id);
        }

        Client client = optionalClient.get();

        String userEmail = getCurrentUserEmail();
        String clientUserId = client.getUserId();
        if (clientUserId == null || !clientUserId.equals(userEmail)) {
            throw new UpdateClientException("You do not have access to this client");
        }

        String newEmail = clientDto.getEmail();
        Optional<Client> existingClient = clientRepository.findByEmail(newEmail);
        if (existingClient.isPresent() && !existingClient.get().getId().equals(id)) {
            String existingClientUserId = existingClient.get().getUserId();
            if (existingClientUserId != null && existingClientUserId.equals(userEmail)) {
                throw new UpdateClientException("Email already in use by another client of this
user");
            }
        }

        client.setName(clientDto.getName());
        client.setStageName(clientDto.getStageName());
        client.setEmail(newEmail);
        client.setAddress(clientDto.getAddress());

        return clientRepository.save(client);
    }

    @Override
    public void deleteClient(String id) {
        Optional<Client> optionalClient = clientRepository.findById(id);
        if (optionalClient.isEmpty()) {
            throw new IllegalArgumentException("Client not found with ID: " + id);
```

```
        }

        Client client = optionalClient.get();

        String userEmail = getCurrentUserEmail();
        String clientUserId = client.getUserId();
        if (clientUserId == null || !clientUserId.equals(userEmail)) {
            throw new IllegalArgumentException("You do not have access to this client");
        }

        clientRepository.deleteById(id);
    }

    @Override
    public String findIdByName(String name) {
        String userEmail = getCurrentUserEmail();
        logger.info("Finding client ID by name: {} for user: {}", name, userEmail);

        Optional<Client> clientOptional = clientRepository.findByNameAndUserId(name.trim(),
userEmail);
        if (clientOptional.isEmpty()) {
            logger.warn("Client not found with name: {} for user: {}", name, userEmail);
            throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Client not found with
name: " + name + " or does not belong to you");
        }

        return clientOptional.get().getId();
    }

    @Override
    public List<Client> searchClients(String name) {
        String userEmail = getCurrentUserEmail();
        return clientRepository.findByNameContainingIgnoreCase(name).stream()
                .filter(client -> {
                    String clientUserId = client.getUserId();
```

```
                return clientUserId != null && clientUserId.equals(userEmail);
            })
            .toList();
    }

    @Override
    public Optional<Client> findById(String id) {
        String userEmail = getCurrentUserEmail();
        return clientRepository.findById(id)
                .filter(client -> {
                    String clientUserId = client.getUserId();
                    return clientUserId != null && clientUserId.equals(userEmail);
                });
    }

    private String getCurrentUserEmail() {
        Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        if (principal instanceof UserDetails) {
            return ((UserDetails) principal).getUsername();
        } else {
            throw new RuntimeException("Користувач не автентифікований");
        }
    }
}
```

LeaseServiceImpl.java:

```
package com.yanhol.StudioApp.service.implemintation;

import com.yanhol.StudioApp.dto.LeaseDto;
import com.yanhol.StudioApp.models.Client;
import com.yanhol.StudioApp.models.Lease;
import com.yanhol.StudioApp.repositories.ClientRepository;
import com.yanhol.StudioApp.repositories.LeaseRepository;
import com.yanhol.StudioApp.service.LeaseService;
import lombok.RequiredArgsConstructor;
```

```
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class LeaseServiceImpl implements LeaseService {

    private final LeaseRepository leaseRepository;
    private final ClientRepository clientRepository;

    @Override
    public List<Lease> getLeasesByMonths(int year, List<Integer> months) {
        String userEmail = getCurrentUserEmail();
        return leaseRepository.findByCreatedAtBetween(
                    LocalDateTime.of(year, months.get(0), 1, 0, 0),
                    LocalDateTime.of(year, months.get(months.size() - 1), 1, 23, 59, 59)
                            .plusMonths(1).minusSeconds(1)
                ).stream()
                .filter(lease -> months.contains(lease.getCreatedAt().getMonthValue()))
                .filter(lease -> {
                    String leaseUserId = lease.getUserId();
                    return leaseUserId != null && leaseUserId.equals(userEmail);
                })
                .collect(Collectors.toList());
    }

    @Override
    public List<Lease> getLeasesByMonth(int year, int month) {
        String userEmail = getCurrentUserEmail();
```

```
    LocalDateTime start = LocalDateTime.of(year, month, 1, 0, 0);
    LocalDateTime end = start.plusMonths(1).minusSeconds(1);
    return leaseRepository.findByCreatedAtBetween(start, end).stream()
            .filter(lease -> {
                String leaseUserId = lease.getUserId();
                return leaseUserId != null && leaseUserId.equals(userEmail);
            })
            .collect(Collectors.toList());
}

@Override
public double getTotalRevenue() {
    String userEmail = getCurrentUserEmail();
    List<Lease> userLeases = leaseRepository.findByUserId(userEmail);
    return userLeases.stream()
            .mapToDouble(Lease::getLeasePrice)
            .sum();
}

@Override
public double getYearlyRevenue(int year) {
    String userEmail = getCurrentUserEmail();
    LocalDateTime start = LocalDateTime.of(year, 1, 1, 0, 0);
    LocalDateTime end = LocalDateTime.of(year, 12, 31, 23, 59, 59);
    List<Lease> leases = leaseRepository.findByCreatedAtBetween(start, end).stream()
            .filter(lease -> {
                String leaseUserId = lease.getUserId();
                return leaseUserId != null && leaseUserId.equals(userEmail);
            })
            .toList();
    return leases.stream()
            .mapToDouble(Lease::getLeasePrice)
            .sum();
}
```

```
    @Override
    public double getMonthlyRevenue(int year, int month) {
        String userEmail = getCurrentUserEmail();
        LocalDateTime start = LocalDateTime.of(year, month, 1, 0, 0);
        LocalDateTime end = start.plusMonths(1).minusSeconds(1);
        List<Lease> leases = leaseRepository.findByCreatedAtBetween(start, end).stream()
                .filter(lease -> {
                    String leaseUserId = lease.getUserId();
                    return leaseUserId != null && leaseUserId.equals(userEmail);
                })
                .toList();
        return leases.stream()
                .mapToDouble(Lease::getLeasePrice)
                .sum();
    }

    @Override
    public Lease saveLease(LeaseDto leaseDto) {
        Client client = clientRepository.findById(leaseDto.getClientId())
                .orElseThrow(() -> new IllegalArgumentException("Client not found with ID: " +
leaseDto.getClientId()));

        String userEmail = getCurrentUserEmail();

        String clientUserId = client.getUserId();
        if (clientUserId == null || !clientUserId.equals(userEmail)) {
            throw new IllegalArgumentException("You do not have access to this client");
        }

        Lease lease = Lease.builder()
                .leaseName(leaseDto.getLeaseName())
                .leasePrice(leaseDto.getLeasePrice())
                .rights(leaseDto.getRights())
                .clientId(leaseDto.getClientId())
                .leaseType(leaseDto.getLeaseType())
```

```
                .clientName(client.getName())
                .clientStageName(client.getStageName())
                .clientEmail(client.getEmail())
                .clientAddress(client.getAddress())
                .createdAt(leaseDto.getCreatedAt() != null ? leaseDto.getCreatedAt() :
LocalDateTime.now())
                .userId(userEmail)
                .build();

        return leaseRepository.save(lease);
    }

    @Override
    public List<Lease> getAllLeases() {
        String userEmail = getCurrentUserEmail();
        return leaseRepository.findByUserId(userEmail);
    }

    @Override
    public Lease updateLease(String id, LeaseDto leaseDto) {
        Optional<Lease> optionalLease = leaseRepository.findById(id);
        if (optionalLease.isEmpty()) {
            throw new IllegalArgumentException("Lease not found with ID: " + id);
        }

        Lease lease = optionalLease.get();
        String userEmail = getCurrentUserEmail();
        String leaseUserId = lease.getUserId();
        if (leaseUserId == null || !leaseUserId.equals(userEmail)) {
            throw new IllegalArgumentException("You do not have access to this lease");
        }

        Client client = clientRepository.findById(leaseDto.getClientId())
                .orElseThrow(() -> new IllegalArgumentException("Client not found with ID: " +
leaseDto.getClientId()));
```

```
        String clientUserId = client.getUserId();
        if (clientUserId == null || !clientUserId.equals(userEmail)) {
            throw new IllegalArgumentException("You do not have access to this client");
        }

        lease.setLeaseName(leaseDto.getLeaseName());
        lease.setRights(leaseDto.getRights());
        lease.setLeasePrice(leaseDto.getLeasePrice());
        lease.setClientId(leaseDto.getClientId());
        lease.setLeaseType(leaseDto.getLeaseType());
        lease.setClientName(client.getName());
        lease.setClientStageName(client.getStageName());
        lease.setClientEmail(client.getEmail());
        lease.setClientAddress(client.getAddress());
        lease.setCreatedAt(lease.getCreatedAt() != null ? lease.getCreatedAt() :
LocalDateTime.now());
        lease.setUserId(userEmail);

        return leaseRepository.save(lease);
    }

    @Override
    public void deleteLease(String id) {
        Optional<Lease> optionalLease = leaseRepository.findById(id);
        if (optionalLease.isEmpty()) {
            throw new IllegalArgumentException("Lease not found with ID: " + id);
        }

        Lease lease = optionalLease.get();
        String userEmail = getCurrentUserEmail();
        String leaseUserId = lease.getUserId();
        if (leaseUserId == null || !leaseUserId.equals(userEmail)) {
            throw new IllegalArgumentException("You do not have access to this lease");
        }
```

```
        leaseRepository.deleteById(id);
    }

    @Override
    public Optional<Lease> findById(String id) {
        String userEmail = getCurrentUserEmail();
        return leaseRepository.findById(id)
                .filter(lease -> {
                    String leaseUserId = lease.getUserId();
                    return leaseUserId != null && leaseUserId.equals(userEmail);
                });
    }

    private String getCurrentUserEmail() {
        Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        if (principal instanceof UserDetails) {
            return ((UserDetails) principal).getUsername();
        } else {
            throw new RuntimeException("Користувач не автентифікований");
        }
    }
}
```

OrderServiceImpl.java:

```
package com.yanhol.StudioApp.service.implemintation;

import com.yanhol.StudioApp.dto.OrderDto;
import com.yanhol.StudioApp.models.Client;
import com.yanhol.StudioApp.models.Order;
import com.yanhol.StudioApp.repositories.ClientRepository;
import com.yanhol.StudioApp.repositories.OrderRepository;
import com.yanhol.StudioApp.service.OrderService;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.context.SecurityContextHolder;
```

```
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;

@Service
@RequiredArgsConstructor
public class OrderServiceImpl implements OrderService {

    private final OrderRepository orderRepository;
    private final ClientRepository clientRepository;

    @Override
    public Order createOrder(OrderDto orderDto) {
        String userId = getCurrentUserEmail();

        if (orderDto.getClientId() == null || orderDto.getClientId().isEmpty()) {
            throw new IllegalArgumentException("Client ID is required for creating an order");
        }

        Optional<Client> clientOptional = clientRepository.findByIdAndUserId(orderDto.getClientId(), userId);
        if (clientOptional.isEmpty()) {
            throw new IllegalArgumentException("Client with ID " + orderDto.getClientId() + " not found for user: " + userId);
        }

        Order order = Order.builder()
                .createdAt(orderDto.getCreatedAt() != null ? orderDto.getCreatedAt() : LocalDateTime.now())
                .title(orderDto.getTitle())
                .status(orderDto.getStatus())
                .clientId(orderDto.getClientId())
```

```
                .hours(orderDto.getHours())
                .progress(orderDto.getProgress())
                .type(orderDto.getType())
                .userId(userId)
                .build();
        return orderRepository.save(order);
    }

    @Override
    public Order updateOrder(String id, OrderDto orderDto) {
        String userId = getCurrentUserEmail();
        Optional<Order> existingOrderOptional = orderRepository.findByIdAndUserId(id,
userId);
        if (existingOrderOptional.isEmpty()) {
            throw new IllegalArgumentException("Order not found with ID: " + id + " for user: " +
userId);
        }

        Order existingOrder = existingOrderOptional.get();

        // Перевіряємо, чи новий clientId належить цьому користувачу
        if (orderDto.getClientId() != null && !orderDto.getClientId().isEmpty()) {
            Optional<Client> clientOptional =
clientRepository.findByIdAndUserId(orderDto.getClientId(), userId);
            if (clientOptional.isEmpty()) {
                throw new IllegalArgumentException("Client with ID " + orderDto.getClientId() + "
not found for user: " + userId);
            }
        }

        existingOrder.setStatus(orderDto.getStatus());
        existingOrder.setTitle(orderDto.getTitle());
        existingOrder.setClientId(orderDto.getClientId());
        existingOrder.setHours(orderDto.getHours());
        existingOrder.setProgress(orderDto.getProgress());
```

```
        existingOrder.setType(orderDto.getType());
        return orderRepository.save(existingOrder);
    }

    @Override
    public void deleteOrder(String id) {
        String userId = getCurrentUserEmail();
        Optional<Order> existingOrder = orderRepository.findByIdAndUserId(id, userId);
        if (existingOrder.isEmpty()) {
            throw new IllegalArgumentException("Order not found with ID: " + id + " for user: " +
userId);
        }
        orderRepository.deleteById(id);
    }

    @Override
    public Optional<Order> getOrderById(String id) {
        String userId = getCurrentUserEmail();
        return orderRepository.findByIdAndUserId(id, userId);
    }

    @Override
    public List<Order> getAllOrders() {
        String userId = getCurrentUserEmail();
        return orderRepository.findByUserId(userId);
    }

    @Override
    public List<Order> getOrdersByClientId(String clientId) {
        String userId = getCurrentUserEmail();
        // Перевіряємо, чи клієнт належить цьому користувачу
        Optional<Client> clientOptional = clientRepository.findByIdAndUserId(clientId, userId);
        if (clientOptional.isEmpty()) {
            throw new IllegalArgumentException("Client with ID " + clientId + " not found for user:
" + userId);
```

```
        }
        return orderRepository.findByClientIdAndUserId(clientId, userId);
    }

    @Override
    public int getActiveProjects() {
        String userId = getCurrentUserEmail();
        return (int) orderRepository.countByStatusAndUserId("Активний", userId);
    }

    private String getCurrentUserEmail() {
        Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        if (principal instanceof UserDetails) {
            return ((UserDetails) principal).getUsername();
        } else {
            throw new RuntimeException("Користувач не автентифікований");
        }
    }
}
```

Додаток Д. Ілюстративна частина

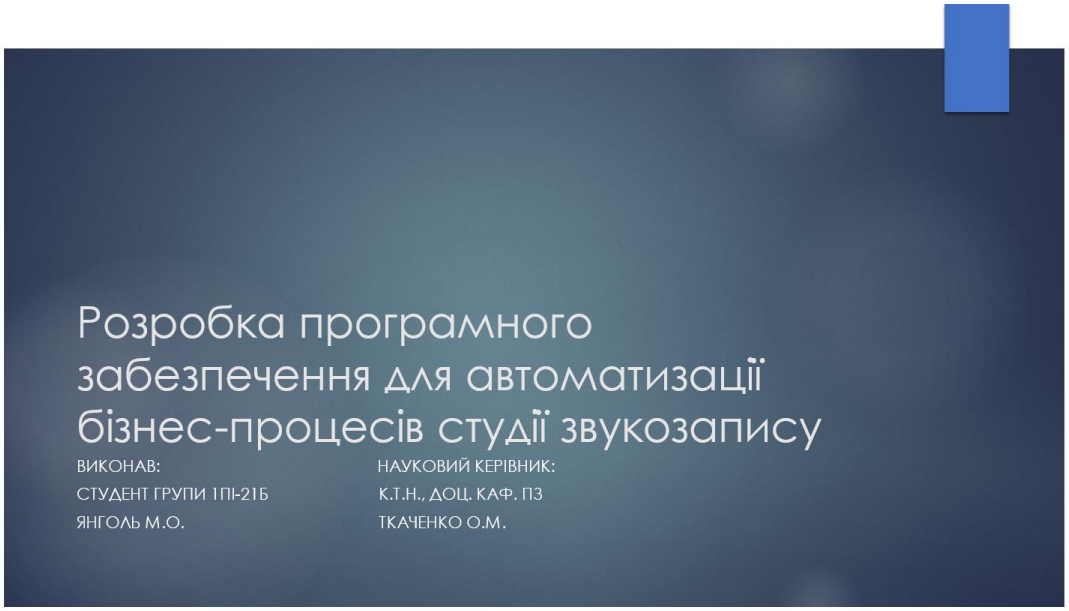


Рисунок Д.1 – Титульний слайд

Рисунок Д.2 – Актуальність розробки

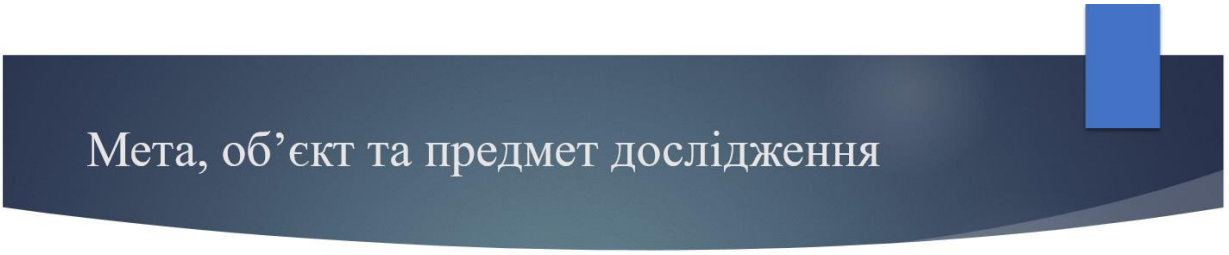


Рисунок Д.3 – Мета, об’єкт та предмет дослідження

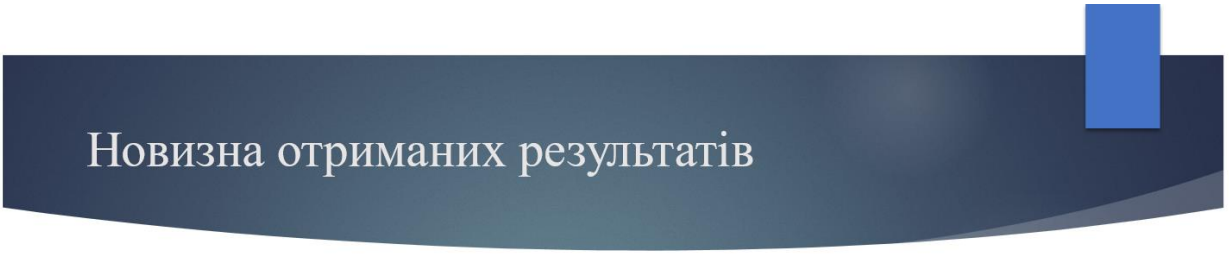


Рисунок Д.4 – Новизна отриманих результатів

Практичне значення отриманих результатів

- Розроблений програмний застосунок, є комплексним інструментом для автоматизації бізнес -процесів студії звукозапису, об'єднуючи модулі управління клієнтами, проєктами, ліцензіями, фінансової аналітики та документообігу. Завдяки модульній структурі він підходить для студій різного масштабу, від невеликих незалежних колективів до великих комерційних студій, які прагнуть оптимізувати робочі процеси.

Рисунок Д.5 – Практичне значення отриманих результатів

Аналіз аналогів

На рисунку зображено таблицю порівняльної характеристики аналогів, яка містить критерії функціональних можливостей

Критерії	Sound Studio Management	Trello	StudioCloud	SoundWave
Управління клієнтами	+	-	+	+
Фінансова аналітика	+	-	+	+
Управління проєктами	+	+	+	+
Генерація ліцензій	-	-	-	+
Звітність	+	-	-	+
Загальна оцінка	80%	40%	60%	100%

Рисунок Д.6 – Аналіз аналогів

UML-діаграма діяльності програмного застосунку

UML-діаграма демонструє основні етапи роботи користувача з програмним застосунком: аутентифікацію, вибір модуля, показує основні функції програмного застосунку для кращого розуміння його роботи.

Рисунок Д.7 – UML-діаграма діяльності

Схема алгоритму роботи програмного застосунку

На представленій схемі, зображено алгоритм взаємодії користувача із програмним забезпеченням після реєстрації та входу в застосунок. Після авторизації користувач потрапляє до головного інтерфейсу, де має змогу обирати різні функціональні вкладки, що відповідають за основні модулі системи.

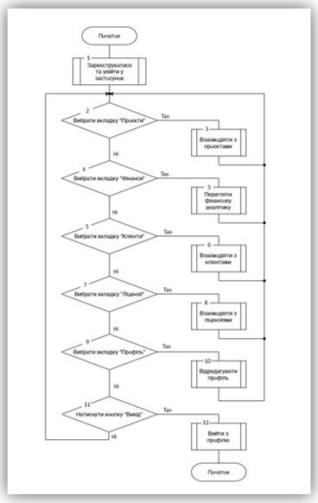
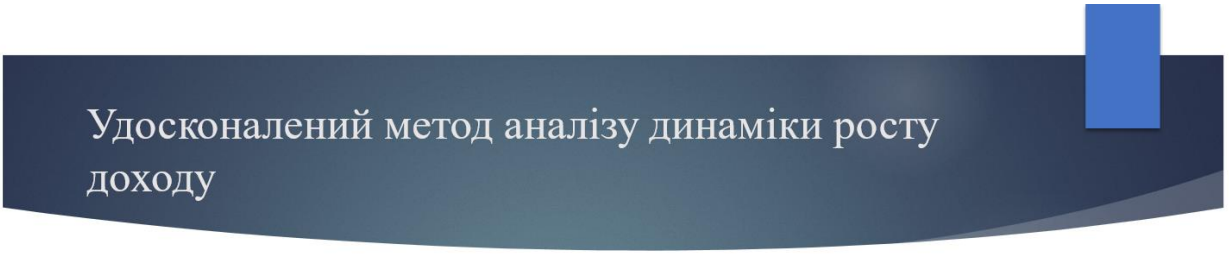


Рисунок Д.8 – Схема алгоритму роботи застосунку

- До впровадження вдосконаленого методу автоматизації документообігу в студіях звукозапису процес створення та обробки ліцензійних угод був переважно ручним. Зазвичай студії використовували статичні шаблони документів, які заповнювалися вручну на основі даних клієнтів і проєктів. Після цього готові документи надсилалися клієнтам через електронну пошту або месенджери, що вимагало значних затрат часу. Удосконалений метод значно оптимізував процес створення, зберігання документообігу ліцензійний угод.
- Основні етапи вдосконалення:

1. проведення аналізу вимог документів;
2. розробка модуля генерації документів;
3. інтеграція з поштовим сервісом;
4. валідація та безпека даних.

Рисунок Д.9 – Удосконалений метод автоматизації документообігу



- До впровадження запропонованого методу аналізу динаміки росту доходу в студіях звукозапису фінансовий аналіз зазвичай проводився вручну або з використанням базових інструментів, таких як електронні таблиці. Аналіз доходів часто обмежувався підрахунком загальної суми доходів за місяць без урахування сезонних коливань. Запропонований метод аналізу динаміки росту доходу, реалізований у системі, значно покращив процес фінансового аналізу в студіях звукозапису.
- Основні етапи реалізації методу:

1. збір даних через REST API;
2. побудова сезонного індексу;
3. корекція доходів і витрат із урахуванням сезонності.

Рисунок Д.10 – Удосконалений метод аналізу динаміки росту доходу

Інтерфейс модуля створення ліцензіїї

Модуль створення ліцензії містить сторінку, в якій користувач вводить дані про ліцензію, вибирає клієнта до якого ліцензія буде прив'язана та заповнює інші дані.

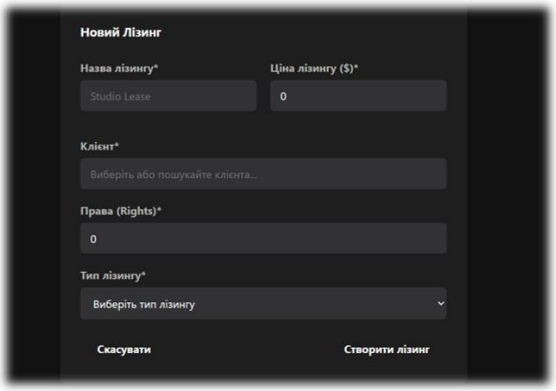


Рисунок Д.11 – Інтерфейс модуля створення ліцензіїї

Інтерфейс модуля відправки ліцензіїї

Модуль відправки ліцензії містить поля про клієнта та ліцензії, вони додаються шляхом вводу через рядок пошуку та автодоповнення.

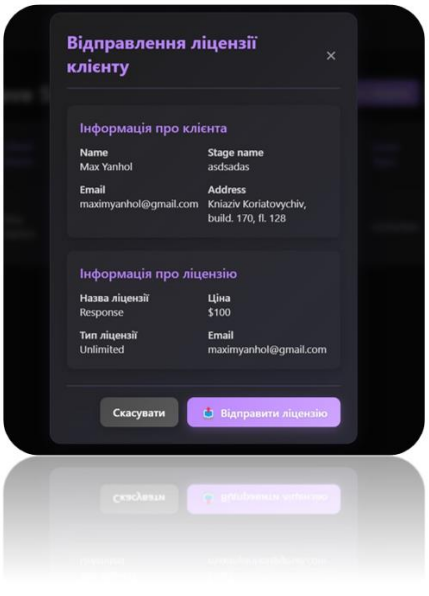
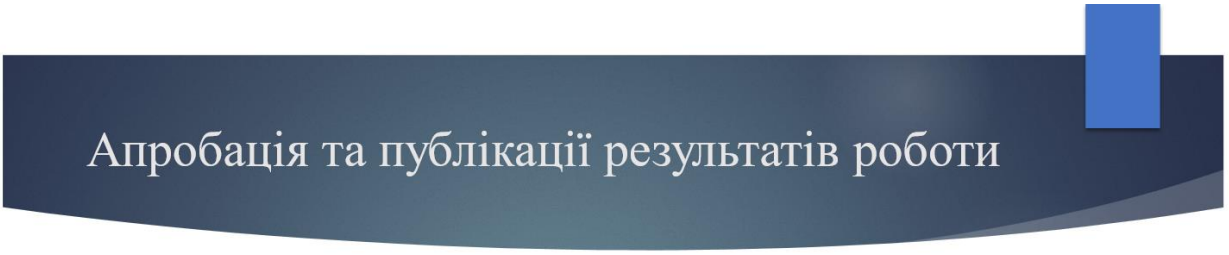


Рисунок Д.12 – Інтерфейс модуля відправки ліцензіїї

Рисунок Д.13 – Інтерфейс модуля фінансової аналітики

Рисунок Д.14 – Візуалізація API за допомогою документації Swagger UI



- Наукові результати, представлені в бакалаврській кваліфікаційній роботі, були висвітлені на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (м. Вінниця, 2025 р.) та на XIII Міжнародній науково-практичній конференції з інформаційних систем і технологій Infocom Advanced Solutions 2025 (м. Київ, 2025 р.).
- Публікації :

1. Янголь М.О., Розробка програмного забезпечення для автоматизації бізнеспроцесів студії звукозапису. Вінниця : ВНТУ, 2025. 2с.
2. Янголь М.О. Програмний застосунок для аналізу та підвищення продуктивності роботи комп'ютера. XIII Міжнародна науково-практична конференція з інформаційних систем та технологій Infocom Advanced Solutions 2025. Київ: КПІ, 2025. 3с.

Рисунок Д.15 – Апробації та публікації результатів роботи

Рисунок Д.16 – Акт впровадження

Рисунок Д.17 – Висновки

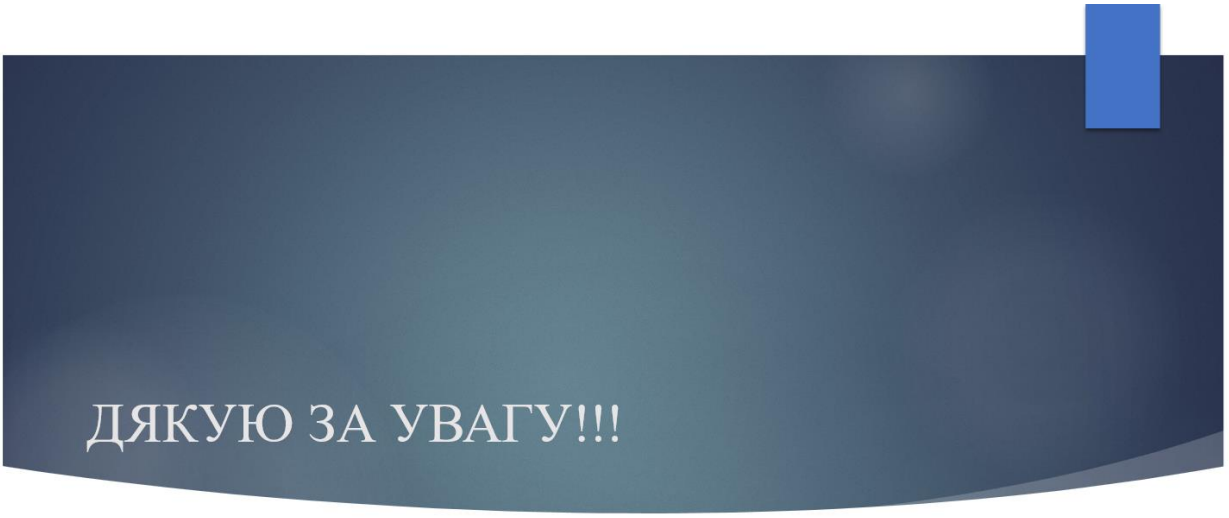


Рисунок Д.18 – Дякую за увагу