Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти

(повна назва)

Кафедра Програмної інженерії

(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Веб-додаток для ведення документообігу контрактора

(тема)

Виконав:

студент 2 курсу, групи ПЗПІ-22-2

Литвинов Є. В.

(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення

(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія

(повна назва освітньої програми)

Керівник доц. каф. ПІ Русакова Н. Є.

(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В. Дудар

(підпис) (посада, прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти

Кафедра Програмної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми освітньо-професійна

Освітня програми Програмна інженерія

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____

(підпис)

«___»_____2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Литвинову Єгору Володимировичу.*

1. Тема роботи Веб-додаток для ведення документообігу контрактора

затверджена наказом університету від "17" червня 2024 р. № 588 Ст

2. Термін подання студентом роботи до екзаменаційної комісії "_21_"_07_2024 р.

3. Вихідні данні до роботи *вимоги до розроблюваної програми, вимоги до архітектури системи, електронні ресурси за обраною темою, мови програмування PHP та JavaScript, СКБД Oracle Mysql, середовища розробки Jetbrains DataGrip та Jetbrains PhpStorm*

4. Перелік питань, що потрібно опрацювати в роботі *вступ, аналіз предметної області, формування вимог до програмного забезпечення, архітектура та проектування розробленого програмного забезпечення, опис прийнятих програмних рішень, тестування програмного забезпечення, висновки, додатки.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Позначка про виконання
1	Аналіз проблемної області	06.05.24 – 08.05.24	Виконано
2	Розробка постановки задачі	08.05.24 – 10.05.24	Виконано
3	Аналіз та моделювання предметної області	10.05.24 – 11.05.24	Виконано
4	Проектування БД	11.05.24 – 12.05.24	Виконано
5	Розробка алгоритмів	11.05.24 – 14.05.24	Виконано
6	Проектування архітектури програмної системи	14.05.24 – 16.05.24	Виконано
7	Програмна реалізація системи	17.05.2024 - 27.06.2024	Виконано
8	Підготовка пояснювальної записки	28.06.2024 - 05.07.2024	Виконано
9	Підготовка презентації та доповіді	06.07.2024 - 10.07.2024	Виконано
10	Нормоконтроль	11.07.2024	Виконано
11	Рецензування	13.07.2024	Виконано
12	Занесення записки в електронний архів	16.07.2024	Виконано
13	Попередній захист	18.07.24	Виконано
14	Допуск до захисту у зав. кафедри	19.07.24	Виконано

Дата видачі завдання 6 травня 2024р.

Студент _____ Литвинов Є. В.

(підпис) (прізвище, ініціали)

Керівник роботи _____ доц.кафедри ПІ Русакова Н.Є

(підпис) (прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра містить: 82 с., 53 рис., 2 додатки, 14 джерел.

ДОКУМЕНТООБІГ, БАЗА ДАНИХ, ВЕБ-СИСТЕМА, ПОШУК, РЕЛЯЦІЙНА БАЗА ДАНИХ, СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ, PHP, JAVASCRIPT, DOCTRINE DBAL, ORACLE MYSQL

Об’єкт розробки – веб-система, яка реалізує ведення бухгалтерської звітності для індивідуального працівника-контактора, збережених документів, можливості їх перегляду, каталогізації, їх автоматизоване тегування, створення списків документів та контролювання їх додання до системи у потрібні терміни, а також за допомогою набору тегів, знайти потрібний документ.

Мета розробки – підвищення ефективності на контрольованості зберігання електроних копій документів, спрощення ведення бухгалтерії та розрахунок податків, автоматизація контролю за виконаннням умов документообігу.

Методи рішення завдання – аналіз та моделювання предметної області, концептуальне моделювання, UML-моделювання, об’єктно-орієнтоване програмування, використання СУБД Oracle MySQL, побудова сервісів на основі REST архітектури.

Результатом роботи є веб-система для ведення базової бухгалтерії для контрактора, а також збереження, автоматизоване каталогування, пошук документів, створення списків документів та контроль документообігу.

DOCUMENT MANAGEMENT, DATABASE, WEB SYSTEM, SEARCH, RELATIONAL DATABASE, DATABASE MANAGEMENT SYSTEM, PHP, JAVASCRIPT, DOCTRINE DBAL, ORACLE MYSQL

Object of Development is a web system that provides access to information on accounting for an individual contractor, stored documents, the

ability to view and catalog them, automated tagging, creation of document lists, and control of their addition to the system within required timeframes, as well as the ability to find the necessary document using a set of tags.

Purpose of Development is increasing the efficiency and controllability of storing electronic copies of documents, simplifying accounting and tax calculation, and automating the control of document workflow conditions.

Methods for Solving the Task is analysis and modeling of the subject area, conceptual modeling, UML modeling, object-oriented programming, the use of the Oracle MySQL DBMS, building services (server side, client side) based on REST architecture.

Result of the Work is a web system for basic accounting for a contractor, as well as for storing, automated cataloging, searching documents, creating document lists, and controlling document workflow.

Я, Литвинов Єгор Володимирович, студент гр. ПЗПІ-22-2, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Веб-додаток для ведення документообігу контрактора», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

У глобалізованому світі активно розвивається можливість віддаленої роботи там все більш актуальним стає робота у статусі незалежного спеціаліста який працює як незалежний консультант - контактор. Такий спеціаліст самостійно володіє та обслуговує свій інструментарій, виставляє рахунки замовнику на взаємодіє з фіскальними на регулюючими органами країни свого перебування.

Актуальність роботи полягає у створенні такої системи що орієнтовна на локальне та незалежне від стороніх сервісів, що може бути розгорнута на інфраструктурі приватного користувача. Проблема полягає в великій кількості цих даних і документів, та необхідності як їх централізованому зберіганню та пошуку, так і контролю, що усі необхідні документи є у наявності.

У ході виконання кваліфікаційної роботи бакалавра було проведено аналіз предметної галузі, ER-моделювання, концептуальне та логічне моделювання реляційної БД, проведена її фізичне реалізація.

В результаті розроблений додаток, який дозволяє зменшити обсяг додаткових зусиль по веденню договорів та банківських транзакцій, відстеженню відпрацьованого часу для створення рахунків та відстеження вимог до поданих документів до державних органів та збереження цих документів.

Під час розробки використовувалися мови програмування PHP та JavaScript, що дозволило створити динамічний та функціональний веб-додаток. Як база даних використовувався СКБД Oracle MySQL, що забезпечило надійне збереження необхідної інформації. Середовища розробки Jetbrains DataGrip та Jetbrains PhpStorm значно спростили процес написання та тестування коду.

Робота перевірена на унікальність тексту в мережі інтернет та базі ХНУРЕ (див. додаток А). За результатами кваліфікаційної роботи було розроблено презентацію (див. додаток Б).

1 АНАЛІЗ ПРОБЛЕМНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблемної галузі

Аналіз проблемної області є важливим етапом, який дозволяє сформувати розуміння про існуючі процеси та об'єкти в предметній області, а також їх взаємодію між собою. Завдяки цьому аналізу можна отримати цілісне уявлення про структуру та функціонування предметної області, що є необхідним для подальшої роботи та прийняття рішень.

У сучасному глобалізованому світі відбувається стрімкий розвиток технологій, що дозволяють працювати віддалено. З кожним роком все більше людей обирають статус незалежного спеціаліста, який працює як консультант або контактор. Ця модель роботи набирає популярності завдяки своїй гнучкості та можливості самостійного керування робочим процесом. Незалежні спеціалісти, або фрілансери, виконують широкий спектр завдань, починаючи від IT-послуг до консультативної підтримки в різних галузях.

Однак, така форма зайнятості має свої виклики. Однією з ключових проблем є необхідність самостійного управління великою кількістю даних та документів. Незалежні спеціалісти повинні:

- виставляти рахунки замовникам - забезпечувати точність у фінансових розрахунках, контролювати надходження платежів та вести бухгалтерію;
- взаємодіяти з фіскальними та регулюючими органами - забезпечувати відповідність діяльності чинному законодавству та своєчасно подавати необхідні документи;
- володіти та обслуговувати свій інструментарій - технічне обладнання та програмне забезпечення, що використовується у роботі, повинно бути корректно обслуговуватись та сплачуватись ліцензійні та амортизаційні платежі.

Ці та інші необхідні активності створюють потоки документації. Інвойси, акти виконаних робіт, договори, додаткові угоди, квитанції про сплати постачальникам обладнання та програмного забезпечення, звіти до фіскальних органів це не повний список документів які необхідно зберігати на мати можливість швидко знайти у разі потреби.

Іншим викликом є необхідність контролювати ці потоки документів. Наприклад, для того щоб здати квартальний налоговий звіт необхідно подати деларацію до налогової служби, отримати дві квітанції про отримання та обробку та зберегти квітанцію про сплату суму податку. Для разрахування необхідно мати усі банківські транзакції, курси валют на день їх отримання та, що важливо, зберегти інвойси та акти виконаних робіт які є первинною документацією для банку для зарахування грошей на рахунок [1].

Третьою складовою проблеми є масштаби системи. Зазвичай для керування бізнес-процесами використовують готові CRM системи з великим функціоналом. Але на розмірі одного індивідуального працівника чи мікро-бізнесу така система є дуже великою, складною та дорогою у розгортанні, налаштуванні, використанні та ліцензуванні.

Для того, щоб програмна система дійсно мала користь для користувача та допомагала у вищезгаданих питаннях, розробнику необхідно вирішити наступні проблеми:

- обрати архітектуру системи, яка забезпечить високу продуктивність і швидке взаємодію користувачів з нею;
- визначити відповідну базу даних та її структуру сутностей;
- розробити алгоритм для автоматизованої організації документів і контролю за дотриманням умов документообігу;
- створити зручний та інтерактивний інтерфейс для користувачів;
- забезпечити захист даних та їх надійне зберігання у системі.

Вибір архітектури додатка є одним з найважливіших етапів розробки програмного забезпечення. Від цього вибору залежить продуктивність,

масштабованість, надійність та зручність використання системи. Розглянемо основні принципи, підходи та критерії, що використовувалися для вибору архітектури додатка.

Перед вибором архітектури необхідно визначити вимоги до системи. Основними вимогами є висока продуктивність, яка забезпечує обробку великого обсягу даних без значних затримок, масштабованість, що дозволяє підтримувати збільшення навантаження та обсягу даних, надійність, яка гарантує стійкість до збоїв та забезпечує збереження даних, безпека, яка включає забезпечення конфіденційності, цілісності та доступності даних, а також зручність використання, де інтерфейс повинен бути інтуїтивно зрозумілим та зручним для користувачів.

Для розробки додатка було розглянуто кілька архітектурних підходів: монолітна архітектура, мікросервісна архітектура та сервіс-орієнтована архітектура (SOA). Монолітна архітектура має переваги у простоті розробки та розгортання, а також ефективному використанні ресурсів, але вона має значні недоліки, такі як складність підтримки та масштабування, а також висока залежність компонентів. Мікросервісна архітектура, навпаки, забезпечує легкість масштабування, незалежність компонентів та гнучкість у виборі технологій, але вона складніша в управлінні і має вищу вартість розробки та підтримки. Сервіс-орієнтована архітектура (SOA) надає високу масштабованість і можливість повторного використання сервісів, але її інтеграція є складною і потребує значних витрат на підтримку.

Для нашого додатку була обрана сервісно-орієнтовна архітектура яка дозволить розширювати основну систему за рахунок додаткових сервісів, які можуть реалізовувати окремі функції, такі як розрахунок податків чи служити проксі для інтеграції з зовнішніми системами.

Система буде реалізовувати підхід «клієнт-сервер» що дозволить реалізувати інтерфейс системи як веб-додаток. Це дозволить зробити

систему не залежною від операційної системи на якій вона буде розгорнута.

Основними компонентами обраної мікросервісної архітектури є API Gateway, який забезпечує єдину точку входу для всіх запитів до системи, розподіляє запити між мікросервісами та забезпечує безпеку, мікросервіси, які є незалежними компонентами, що відповідають за конкретні функціональні області (авторизація, зберігання документів тощо), бази даних, де кожен мікросервіс може використовувати власну базу даних, що забезпечує незалежність даних та їх ізоляцію, а також система моніторингу та логування, яка забезпечує контроль за станом системи, збір логів та аналітику.

Сервер оброблятиме запити від клієнтської частини, взаємодіючи з базою даних та виконуючи всі алгоритмічні обчислення. Зазвичай веб-інтерфейс відправляє HTTP-запити до сервера і відображає відповіді за допомогою UI/UX елементів системи.

REST (Representational State Transfer) архітектура є популярним підходом для створення веб-сервісів, який базується на принципах простоти та масштабованості. Основна ідея REST полягає у використанні стандартних HTTP методів (GET, POST, PUT, DELETE) для взаємодії з ресурсами, представленими у вигляді URL. Кожен ресурс має унікальний ідентифікатор і може бути запитаний або змінений за допомогою відповідних методів. REST архітектура забезпечує легкість інтеграції, оскільки вона використовує стандартні веб-протоколи та формати даних, такі як JSON або XML. Це дозволяє створювати гнучкі та розширювані системи, які легко масштабуються і підтримують високу продуктивність. Крім того, RESTful сервіси є безстанними, що означає, що кожен запит від клієнта містить всю необхідну інформацію для обробки, що спрощує управління сеансами та покращує надійність системи. [2]

Проектування системи збереження даних є ключовим етапом розробки інформаційних систем, оскільки від правильної організації даних залежить ефективність роботи всієї системи.

Першим кроком є аналіз вимог до даних, їх типам, структурам та кількості. Важливо також забезпечити цілісність даних, швидкий доступ до них і можливість масштабування системи.

В системі присутні два типи даних:

- самі документи, які зберігаються у вигляді файлів різного типу
- їх метадані, що описують інформацію про ці документи за звязки між ними.

У нашій системі є дані у вигляді файлів, які будуть зберігатися на локальному диску, оскільки це відповідає вимогам до локальності зберігання. Такий підхід дозволяє забезпечити швидкий доступ до файлів і зменшує залежність від зовнішніх сервісів. Для додаткової безпеки та збереження даних може бути передбачено можливість створення резервних копій у хмарних сховищах, що гарантує збереження файлів навіть у випадку збоїв локальної системи.

Метадані відіграють ключову роль у управлінні документами, фінансовими транзакціями та дотриманні правил документообігу. Для ефективного зберігання та обробки метаданих було вирішено використовувати реляційну базу даних (БД).

Реляційна БД була обрана для зберігання метаданих з кількох причин. По-перше, реляційні БД забезпечують високу цілісність даних через підтримку транзакцій та механізмів контролю цілісності, таких як первинні та зовнішні ключі. По-друге, вони підтримують складні запити та операції з даними за допомогою мови SQL, що полегшує маніпулювання метаданими. Реляційні БД підтримують ефективне виконання складних запитів і аналіз даних, що полегшує отримання необхідної інформації та прийняття рішень. По-третє, висока безпека

даних забезпечується вбудованими механізмами контролю доступу, шифрування та резервного копіювання.

Для розробки алгоритму автоматизованої організації документів і контролю за дотриманням умов документообігу необхідно детально дослідити предметну область, щоб врахувати всі специфічні вимоги та особливості документообігу в контексті даної системи. Це включає аналіз типів документів, які використовуються, їхні атрибути та метадані, умови зберігання, правила доступу, та обробки. Крім того, необхідно вивчити існуючі процеси та процедури документообігу, ідентифікувати ключові етапи та точки контролю, де потрібна автоматизація. Такий підхід дозволить створити алгоритм, який не тільки спростить управління документами, але й забезпечить дотримання всіх необхідних регуляторних вимог, підвищуючи ефективність та надійність системи.

Проектування користувацького інтерфейсу (UI) і взаємодії користувача (UX) є критичним аспектом розробки додатка, оскільки воно безпосередньо впливає на зручність і ефективність роботи користувачів з системою. В процесі проектування UI/UX необхідно врахувати потреби цільової аудиторії, забезпечити інтуїтивно зрозумілу навігацію та оптимізувати інтерфейс для різних пристроїв. Це включає створення макетів, прототипів і тестування користувацьких сценаріїв, щоб гарантувати, що інтерфейс є логічним і легко освоюваним. Особливу увагу слід приділити забезпеченню доступності інтерфейсу для користувачів з різними рівнями технічної підготовки, а також впровадженню візуальних і інтерактивних елементів, які полегшують взаємодію з системою. Таким чином, ретельне проектування UI/UX сприяє підвищенню загальної задоволеності користувачів і ефективності роботи з додатком.

Захист даних - ключовий аспект для веб-додатків, оскільки вони містять велику кількість конфіденційної інформації. Вони піддаються

різноманітним атакам, таким як перехоплення даних або несанкціонований доступ. Для забезпечення безпеки використовуються сучасні методи захисту, серед яких використання механізмів автентифікації на основі токенів. Це дозволяє ефективно ідентифікувати та контролювати доступ. Крім цього, використання протоколу HTTPS забезпечує захищену передачу даних між клієнтом та сервером, що запобігає можливості перехоплення інформації. Ці заходи сприяють мінімізації ризиків та забезпеченню високого рівня безпеки для користувачів веб-додатку.

1.2 Аналіз існуючих аналогів

CRM-системи стають невід’ємною частиною сучасного бізнесу, незалежно від його розміру і сфери діяльності. Вони спрощують і покращують управління взаємодією з клієнтами, документами та проектами. Проте, вибір правильної CRM-системи для задоволення вашим потребам великою мірою залежить від унікальних особливостей вашого бізнесу і можливих конкурентів. У цьому розширеному обзорі ми розглянемо конкурентів CRM-систем, і дізнаємося, чому деякі фірми вирішують віддати перевагу іншим рішенням.

Один з найпоширеніших конкурентів CRM-систем - це банківські додатки, такі як Privat24 [3] (див. рис. 1.1). Вони можуть служити постачальниками документів, таких як квитанції про сплату рахунків, або навіть надавати можливість генерувати податкові звіти. Проте, важливо відзначити, що ці документи зберігаються віддалено і вам не завжди дозволяється додавати свої власні документи, які CRM-система може не обробляти. Крім того, немає гарантії, що ці документи залишаться доступними наступні роки.

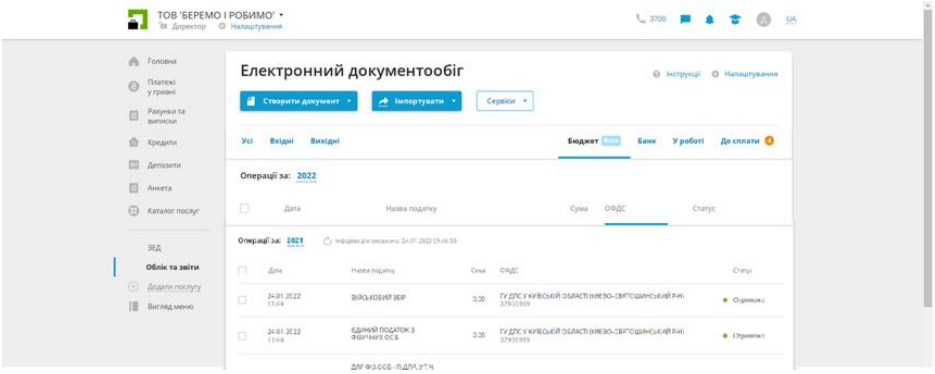


Рисунок 1.1 – Додаток Privat24 for business (за даними [3])

Іншою альтернативою CRM-системи є локальні або хмарні сховища, такі як Microsoft OneDrive, Google Drive, Dropbox та інші. Вони мають веб-інтерфейс та інтеграцію на рівні файловой системи у багатьох десктопних на мобільних операційних системах.

Приклад веб-додатку OneDrive наведено на рисунку 1.2

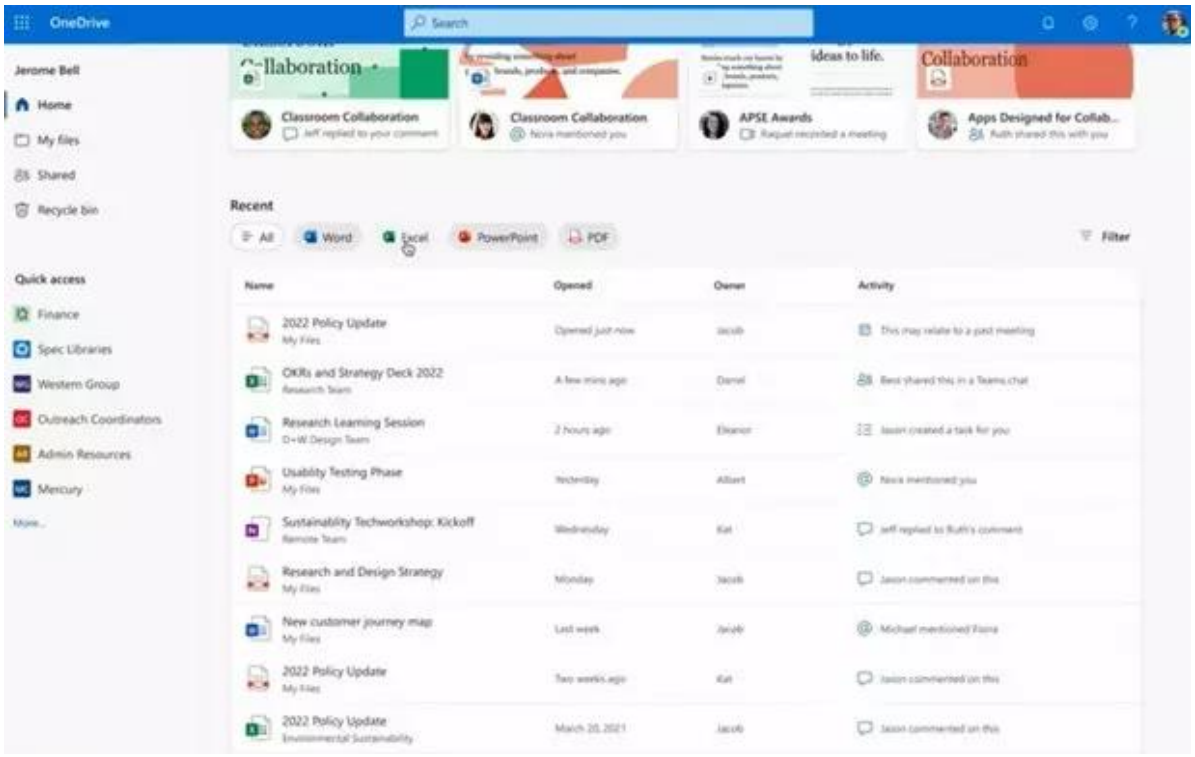


Рисунок 1.2 – Приклад веб-додатку OneDrive (за даними [4])

Також існує додаток ля мобільних операційних систем. Його приклад зображено на рисунку 1.3

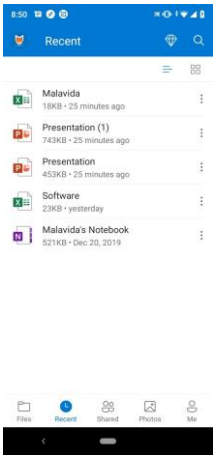


Рисунок 1.3 – Приклад мобільного додатку OneDrive (за даними [4])

Інший сервіс Google Drive – повністю повторює OneDrive крім того, що належиті інщій корпорації – Google та інтегрований у її екосистему.

Приклад інтерфейсів приведено на рисунку 1.4 і 1.5.

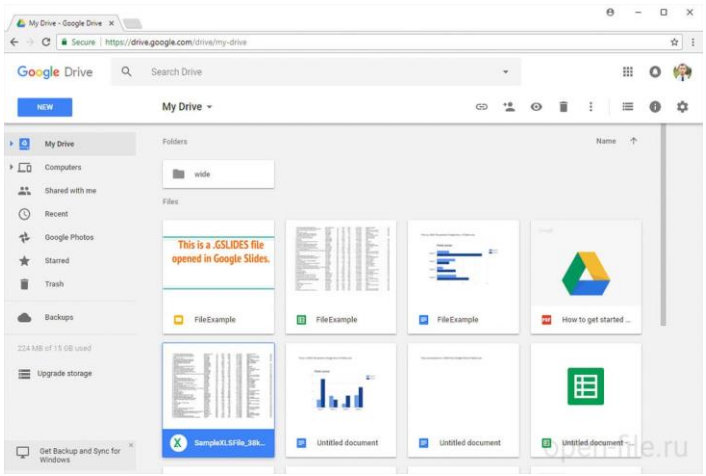


Рисунок 1.4 - Приклад веб-додатку Google Drive (за даними [5])

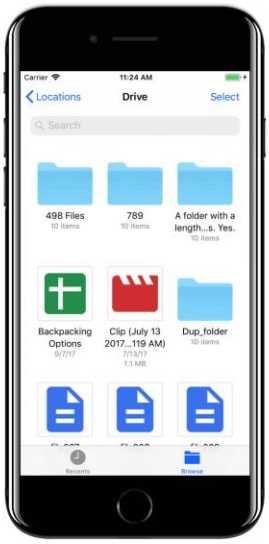


Рисунок 1.5 - Приклад мобільного додатку Google Drive (за даними [5])

Ці системи дають можливість збереження даних у хмарі без прив’язки до їх походження, там можуть зберігатися усі можливі документи. Також ці сервіси надаюсь базовий функціонал їх каталогізації та пошуку по ним. Також маючи інтеграцію у операційну систему на рівні файлової системи вона мають зручний функціонал додавання документів.

Але вони формують лише деревовидну файлову систему та не мають системи яка б контролювала наявність усіх необхідних файлів. Також відсутній модуль бухгалтерії у будь якому вигляді, як контроль транзакцій, так і розрахунок податків. Лише у вигляді окремих excel файлів з формулами які можуть бути засінхронізовані та використані для розрахування.

Вони добре підходять для зберігання і організації документів, проте їхній недолік полягає в жорсткій ієрархії файлів та відсутності можливості контролювати виконання умов документообігу. Крім того, вони не надають інтегрованих систем для ведення бухгалтерії.

Третьою альтернативою є повноцінні CRM-системи, такі як Zoho CRM [4] (див. рис. 1.6) чи Salesflare [5]. Вони мають весь необхідний функціонал для управління контактами, справами та проектами. Проте вони є хмарними системами і не завжди готові до кастомізації під українські податкові реалії.

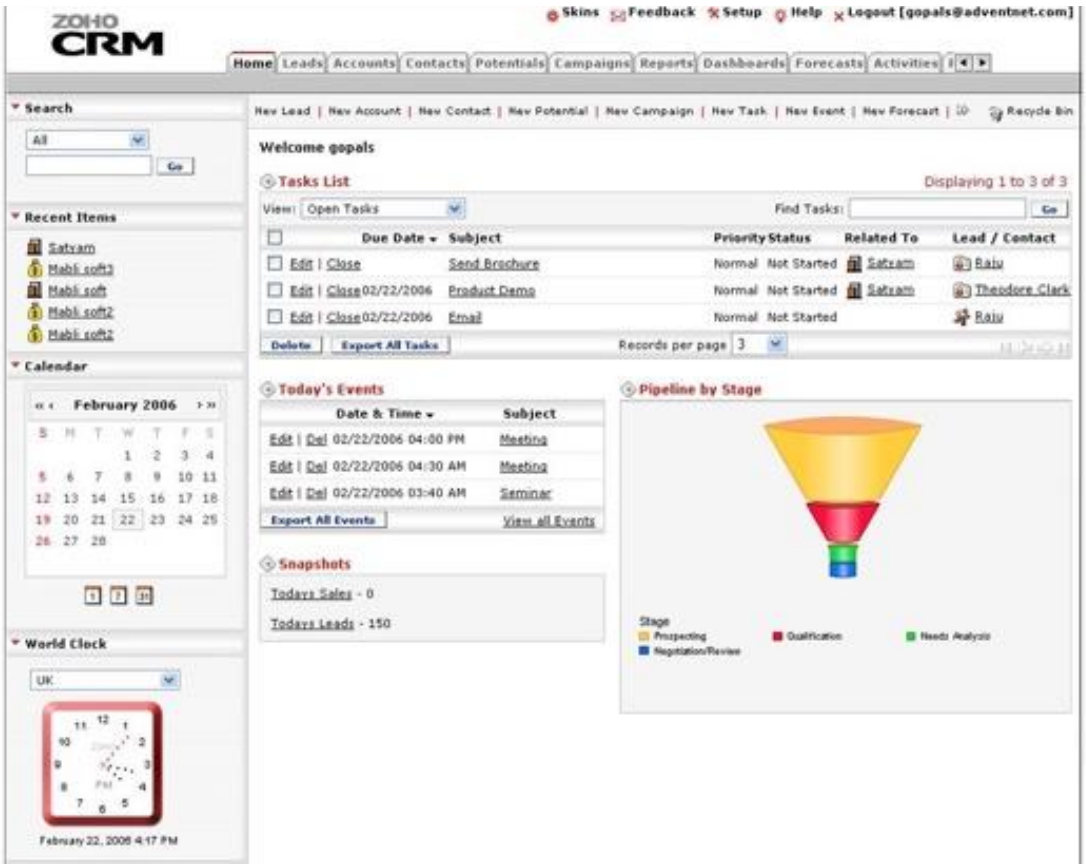


Рисунок 1.6 – Додаток Zoho CRM (за даними [6])

Зважаючи на розглянуті альтернативи, важливо визначити свої потреби і цілі для кращого вибору CRM-системи. Банківські додатки можуть бути простим і зручним варіантом для контролю за фінансами, але вони не забезпечують повний спектр управління взаємодією з клієнтами та проектами. Локальні та хмарні сховища підходять для зберігання документів, але вони не можуть служити CRM-системою. Нарешті, повноцінні CRM-системи надають найбільше функціоналу, але вони можуть не бути підходящими для певних податкових реалій.

Paperless-ngx є opern-source веб-додатком для зберігання та каталогзації документів. Приклад його інтерфейсу наведено на рисунку 1.7

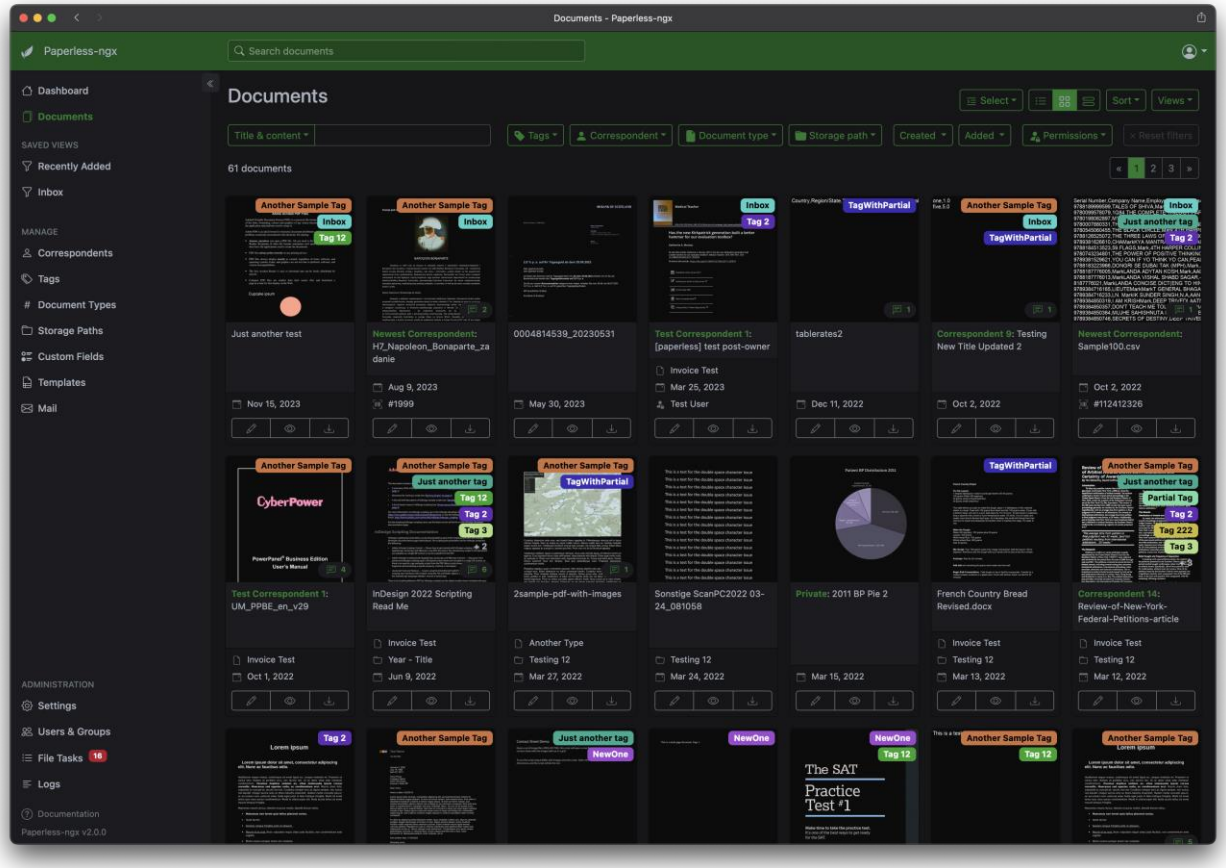


Рисунок 1.7 – Приклад веб-додатку Paperless-ngx (за даними [7])

Ця система має откритий код та може бути встановлена у клієнтському середовищі. Її головною особливістю є система каталогізації яка базується не на деревовидній системі, а на тегах, що дозволяє дуже гнучко шукати документи. Крім того система розпізнавання тексту та обєктів дозволяє системі автоматизовано ці теги протавляти.

Але система немає жодного функціоналу з керування проектами та бухгалтерії.

1.3 Постановка задачі

Метою цієї роботи є розробка програмної системи «Documentorro», яка допомагає організувати ведення базової бухгалтерії та документообігу для контрактора.

Основне завдання системи полягає в забезпеченні індивідуального працівника-контрактора інструментами для керування документообігом. Система повинна надавати можливість зберігання електронних копій документів, їх каталогізації. Має автоматизувати процеси каталогізуваня документів, контролювати їх додавання до системи у визначені терміни та забезпечувати розрахунку податків та ведення бухгалтерського обліку.

Система повинна мати можливість керувати даними про укладені договори, документи, відпрацьовані години, банківські транзакції, події та їх виконання. Підтимувати підготовку та друк бухгалтерської статистики, умови документообігу. Здійснює розрахунок інвойса по проекту та податкової декларації, а також автоматично розсилає листи з попередженнями про невиконання умов документообігу.

Використовуючи вищезгадану інформацію, постає такий список завдань, які слід виконати:

- провести аналіз та моделювання предметної області програмної системи;
- спроектувати базу даних для збереження інформації з предметної області;
- розробити алгоритм для автоматизованої організації документів і контролю за дотриманням умов документообігу;
- спроектувати архітектуру програмної системи;
- виконати програмну реалізацію системи та провести тестування створенного програмного продукту.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Система, яку ми розробляємо, повинна бути гнучкою, масштабованою та стійкою до будь-яких змін. Користувачам має бути зручно та легко працювати з нею, а внесення змін до системи повинно бути швидким і безпроблемним процесом.

Програмна система повинна мати клієнт-серверну сервісну архітектуру. Серверна частина повинна містити усю логіку, відповідати за збереження та обробку даних системи, а також бути максимально захищена від стороннього доступу зловмисників. Розробка серверної частини повинна бути реалізована за допомогою мови програмування PHP [9], яка є найбільш оптимальною для даної задачі.

Наступним кроком є створення відповідної системи зберігання даних в для цієї системи. Метадані до файлів та бухгалтерська інформація має зберігатися у реляційній базі даних. Оптимальним у даному випадку є реляційна база даних Oracle MySQL[10]. Ця база є фактичним стандартом для розробки веб-додатків та її API сумісний з іншими базами даних, таких як MariaDB чи Amazon RDS.

Сервер буде підключатись та взаємодіяти з базою даних за допомогою технології Doctrine DBAL [11]. Це бібліотека для роботи з базами даних в середовищі PHP, яка надає абстракцію бази даних із підтримкою різних реляційних систем. Вона дозволяє розробникам працювати з базою даних незалежно від конкретного типу СУБД, що полегшує розробку та підтримку додатків. Doctrine DBAL забезпечує безпечне взаємодію з базою даних через підготовлені запити, підтримуючи безпеку та враховуючи особливості кожної СУБД. Вона є невід'ємною частиною ORM Doctrine, але може використовуватися і самостійно для нижчого рівня роботи з базами даних

Документування API викликів серверної частини повинно бути реалізовано за підтримкою стандарту OpenAPI Specification [12], а

тестування API викликів за допомогою інструменту Swagger[13], який надає зручний графічний інтерфейс для роботи.

OpenAPI Specification (раніше відомий як Swagger Specification) - це стандарт для опису та документування RESTful API. Він дозволяє розробникам описувати ресурси, операції, параметри запитів та відповідей API у форматі, який легко зрозуміти та використовувати. Swagger, у свою чергу, є набором інструментів для роботи з OpenAPI Specification, який включає в себе редактори, генератори коду, тестувальні інструменти та інші корисні утиліти. Використання OpenAPI Specification та Swagger спрощує розробку, розгортання та співпрацю над RESTful API, дозволяючи розробникам створювати документацію API автоматично та ефективно.

Клієнтська частина нашої системи буде розроблятися за допомогою мови програмування JavaScript, що дозволить створити динамічний та інтерактивний інтерфейс користувача. Для швидкої і якісної побудови UI ми використовуватимемо бібліотеку Bootstrap, яка надає широкий набір готових компонентів та стилів. Це забезпечить адаптивність та сучасний вигляд нашого додатку на різних пристроях, полегшуючи процес розробки. Використання JavaScript у поєднанні з Bootstrap дозволить створити зручний та функціональний інтерфейс, який відповідатиме потребам користувачів.

У нашому проекті буде використовуватись архітектурний стиль REST з форматом даних JSON для обміну інформацією між клієнтською та серверною частинами системи. Використання REST дозволить забезпечити ефективну та легку взаємодію між різними компонентами додатку, спрощуючи процес розробки та підтримки. Формат JSON, у свою чергу, забезпечить легкість обробки та передачі даних завдяки його простоті та читабельності як для людини, так і для машини. Це поєднання дозволить створити гнучку, масштабовану і зручну у

використанні систему, яка задовольнить потреби користувачів та забезпечить надійну роботу додатку.

Наш проект буде мультиплатформеним, що забезпечить його доступність та коректну роботу на різних операційних системах, таких як Windows, macOS та Linux. Для розгортання додатку ми використовуватимемо Docker, який дозволяє створювати контейнеризовані середовища, що ізолюють програму та її залежності від основної системи. Це забезпечить високу портативність, спрощення процесу розгортання та підтримки додатку, а також мінімізує конфлікти з налаштуваннями середовища. Використання Docker гарантує, що наш додаток буде легко розгортатися і масштабуватися на будь-якій платформі, задовольняючи вимоги до продуктивності та стабільності.

Система повинна відображати та мати можливість редагувати данні:

- про основні елементи предметної області – укладені договори да первинні документи у діяльності контрактора;
- про повязані данні – відпрацьовані години, банківські транзакції, події та правила їх виконання.

Система повинна підтримувати сортування, пошук та фільтрацію даних:

- сортування договорів за рейтами, за датою початку та закінчення;
- сортування транзакцій за договором, датою, сумою;
- сортування подій за часом початку та закінчення. Фільтрація за інтервалом часу, договором, змістом опису.

Система повинна підтримувати додавання, редагування та видалення даних про договори, відпрацьовані години, банківські транзакції, документи, події та правила їх закриття.

Система повинна підтримувати підготовку та друк наступних статистик:

- отримання обороту за періоди часу (період, сума в валюті поступлення, сума в валюті юрисдикції);

- статистика витрат по договору за вказаний період (договір, тип трати, сума у валюті списання, сума у валюті юрисдикції);
- середній/мінімальний/максимальний часовий рейт за період часу (період, рейт у валюті рахунку);
- усі чи умови документообігу за виконанням умов (тип документу, кількість необхідних умов, кількість виконаних умов).

Система повинна підтримувати формування наступних звітів:

- розрахунок інвойса по проекту (проект, підсумкова сума, список затраченого часу + проектні витрати);
- розрахунок податкової квартальної/місячної податкової декларації для цього періоду часу.

Система повинна реалізовувати наступні задачи автоматизації - розсилання на пошту листів користувачам з поепередженнями про невиконная умов з документообігу. Реалізовувати автоматичне автодоповнення тегів до документу

.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз та моделювання предметної області

У глобалізованому світі активно розвивається можливість віддаленої роботи там все більш актуальним стає робота у статусі незалежного спеціаліста який працює як незалежний консультант - контактор. Такий спеціаліст самостійно володіє та обслуговує свій інструментарій, виставляє рахунки замовнику на взаємодіє з фіскальними на регулюючими органами країни свого перебування.

Крім надання безпосередньо послуг, робота у цьому статусі потребує додаткових зусиль по веденню договорів та банківських транзакцій, відстеженню відпрацьованого часу для створення рахунків та відстеження вимог до поданих документів до державних органів та збереження цих документів.

Розглянемо ці процеси ближче. Під час надання послуги етапи мають бути задокументовані.

По-перше кожна робота починається з підписання договору у якому приведені умови роботи та який має бути збереженим. При роботі по кожному договору кожний етап робіт має бути звітований. Для цього необхідно занести до звіту опис виконаної роботи, час початку та час закінчення роботи над цим етапом та, якщо необхідно, посилання на документ у системі роботодавця на це завдання.

Кількість договорів та кількість етапів у них потенційно необмежена, але виходячи з того, що це індивідуальний працівник наряде буде більше 50 договорів на рік та більше 1000 звітів у кожному з них.

Після завершення звітного періоду за кожним договором клієнту буде надано інвойс для сплати. Після сплати інформація по транзакції, така як назва контрагенту, розмір транзакції, валюта та, за необхідністю,

курс конвертації до валюти сплати податків, первинні документи по цій транзакції повинні бути збережені.

Також повинні бути збережені видаткові транзакції для включення їх до клієнтських інвойсів або використання для обчислювання податків. Для цього необхідно зберігати ці дані у вигляді який сприяє пошуку на сортуванню цієї інформації.

Додатково ця інформація є необхідною для створення звітів які допоможуть аналізувати стан бізнесу чи підраховувати видатки, за період часу узагалі чи на якогось клієнта.

Другою важливою частиною є ведення документообігу з фіскальними органами та контроль виконання його умов. Копії договорів, інвойсів, звіти до налогової, квитанцій з транзакцій та звітів мають бути збережені та постійно доступні для пошуку та перегляду. Важливим є контроль того, що усі необхідні документи було збережено. Тому мають бути створені чеклісти до кожної з подій – підписання договору, поява транзакції, відправка фіскального звіту та інших.

Однією з можливою задач автоматизації є автоматизація цього контролю та повідомлення коли якась умова не виконана або спливає час на її виконання.

Опис функціонального призначення ІС, що створюється наведено на рисунку 3.1.

Зазначимо перелік характеристик для кожного з наведених понять.

Базовим є поняття договору. Це документ якій підписується з замовником та є причиною виконання робіт. Його атрибути це номер договору, назва, дата початку, дата закінчення, часова вартість та вартість праці за підвищеною ціною, наприклад, у додатковий час.

Затрачений час це опис проведеного етапу робіт. Його атрибути це дата та час початку, дата та час закінчення, опис роботи, посилання на задачу у системі обліку клієнта та номер цього етапу робіт.

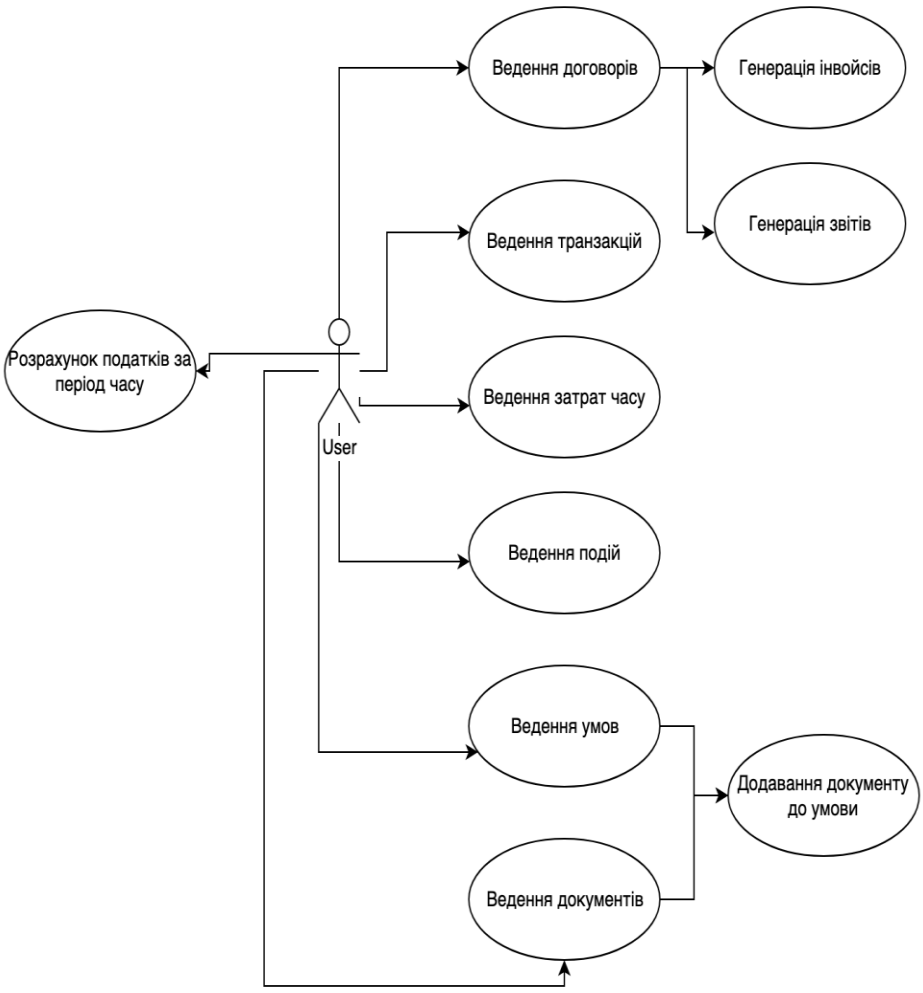


Рисунок 3.1 – Загальна USE-CASE діаграма (рисунок виконано самостійно)

Опис концептів програмної області та зв’язків між ними зобразимо у вигляді загальної діаграми класів представлено на рисунку 3.2.

Банківська транзакція є описом транзакції на банківському рахунку. Описується атрибутами – номер транзакції, номер транзакції у банку, дата та час надходження, сума у оригінальній валюті, валюта надходження, курс до валюти сплати податків, отримання чи виплата, чи потрібно додавати суму з транзакції до інвойсу, опис транзакції.

Подія це набор документів. Описується номером, назвою, описом, датою початку, датою кінця та списком вимог.

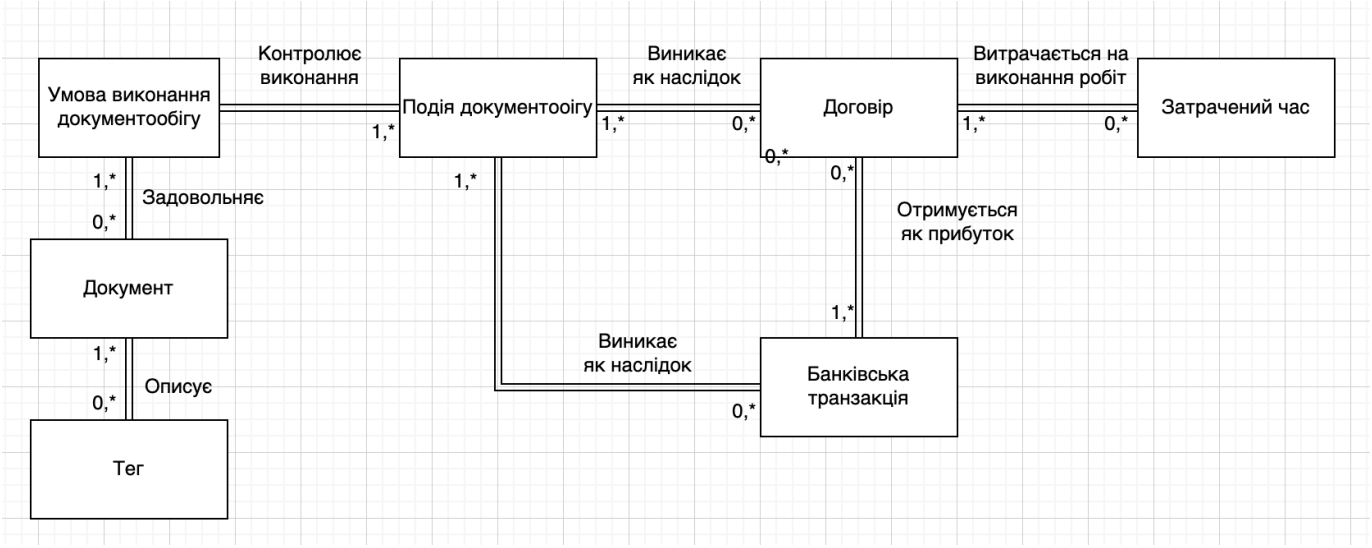


Рисунок 3.2 – Загальна діаграма класів (рисунок виконано самостійно)

Вимоги це список документів які обов’язково мають бути у події.

Документ – це документ завантажений до системи. Описується датою додавання, типом, шляхом до місця збереження, списком тегів які до нього приєднані.

Для спрощення користувача під час роботи з інформацією він повинен мати можливість:

- сортувати договорів за часом початку, назвою, рейтами та валютою, затрачений час за договором, подій за назвою та датами початку чи закінчення, документів за датою додавання та типом, транзакції за часом додавання та типом;
- здійснювати пошук інформації про договір за його повною або частковою назвою, часом початку чи припинення Транзакцію за договором та за інтервалом часу надходження. Подію за договором чи транзакцією. Документ за типом, назвою та списком тегів.

Користувач для спрощення аналізу інформації з робочого процесу повинен мати можливість отримання наступної статистики:

- отримання обороту за періоди часу (період, сума в валюті надходження, сума в валюті юрисдикції);
- статистика витрат по договору за вказаний період (договір, тип трати, сума у валюті списання, сума у валюті юрисдикції);
- середній/мінімальний/максимальний часовий рейт за період часу (період, рейт у валюті рахунку, рейт у валюті юрисдикції);
- усі чи умови документообігу за виконанням умов (споживання, кількість необхідних умов, кількість виконаних умов).

Невиконання вимог документообігу може створити критичні або коштовні проблеми у майбутньому, тому необхідна автоматизація яка буде слідкувати за їх виконанням та попереджати про строки завершення виконання чи про просрочені вимоги.

Також існує проблема в детальному описі яких конкретно документів та у якій події не вистачає. Система повинна це знаходити та формувати попередження, яке повинно проходити автоматично на електрону пошту користувача.

Запропоновану автоматизацію зображено за допомогою діаграми послідовностей даних (див. рис. 10).

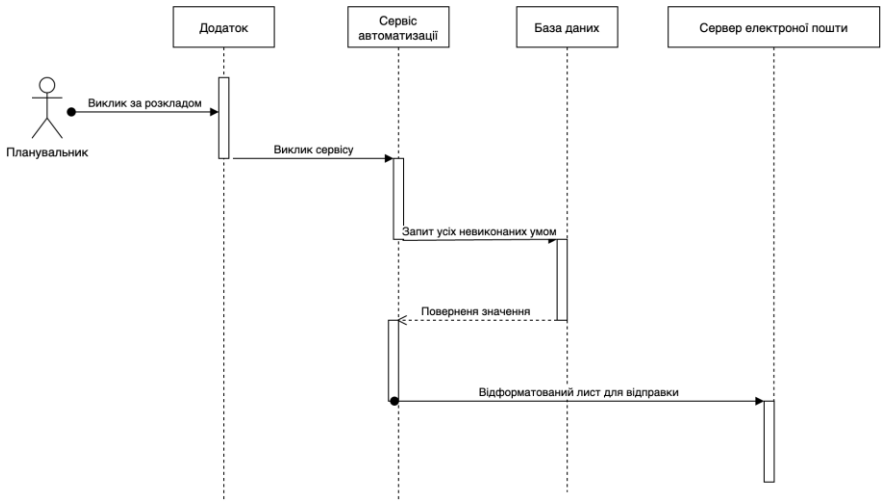


Рисунок 3.3 – Діаграма послідовностей (рисунок виконано самостійно)

Інша проблема це необхідніть самостійно вводити теги для кожного доданого документу. Система повинна вміти самостійно аналізувати документ та давати користувачу автоматично сформований набір тегів для вводу та редагування.

Опис існуючого документообігу складається з наступних документів:

- звіт «Розрахунок інвойсу по проетку»: проект, підсумкова сума, список затраченого часу + проектні витрати;
- «Податкова декларація»: Назва періоду, сума надходжень у валюті сплати податків, розрахований податок за цей період, сума податку сплачений з початку року періоду.

В проблемній області існує рід обмежень, які можна віднести до обмежень цілісності стосовно ідентифікації:

- усі валюти ідентіфікуються їх кодом згідно стандарту ISO 4217;
- інші сутності ідентифікуються унікальним сурогатним ключем – автоматично генеруючимся позитивним числом.

В проблемній області існує рід обмежень, які можна віднести до обмежень цілісності стосовно зв’язків:

- кожний договір може мати скільки завгодно етапів та скільки завгодно транзакцій;
- транзакція може мати лише один договір чи не мати його взагалі;
- затрачений робочий час може бути приєднаний лише до одного договору;
- подія може бути приєднана чи не приєднана взагалі лише до одного договору;
- подія може бути приєднана чи не приєднана взагалі лише до однієї транзакції;
- умова повинна бути приєднана лише до однієї подій;
- документ може бути приєднаний до скількох завгодно умов;

– документ може мати лише один тип;

– документ може мати скільки завгодно тегів.

З урахуванням ліцензійних умов необхідно дотримуватися наступних обмежень:

– повинно використовуватись відкрите програмне забезпечення;

– використовувати СКБД MySQL.

3.2 Розробка алгоритму рішення бізнес-задачі

У сучасному світі обробка великих обсягів документів вимагає ефективних автоматизованих рішень. Для нашої веб-системи, що забезпечує індивідуальних працівників-контракторів інструментами для ведення бухгалтерської звітності та керування документообігом, важливо впровадити механізм автоматизованої обробки документів. Одним із таких рішень є інтеграція з нейронними мережами, наприклад ChatGPT, що дозволяє аналізувати документи та отримувати список тегів, необхідних для їхньої каталогізації та швидкого пошуку. У цьому розділі ми розглянемо процес розробки алгоритму для автоматизованої відправки документів до ChatGPT через API та отримання списку тегів.

Ручна обробка документів займає значний час та є схильною до помилок, що може призвести до затримок у роботі та збільшення витрат. Автоматизація процесу обробки документів дозволяє значно підвищити ефективність та точність, скоротити час на обробку та забезпечити зручний механізм пошуку необхідних документів. Важливо, щоб така система була гнучкою та легко інтегрувалася в існуючу інфраструктуру, забезпечуючи безперебійний робочий процес.

На першому етапі процесу необхідно підготувати документ до відправки. Це включає перевірку формату документу та забезпечення його відповідності вимогам API ChatGPT. Документ повинен бути у текстовому форматі (наприклад, TXT або PDF) та містити всю необхідну

інформацію для аналізу. Крім того, потрібно забезпечити коректне кодування даних, щоб уникнути помилок під час передачі через API.

Схематично алгоритм має наступний вигляд (див. рис. 3.4).

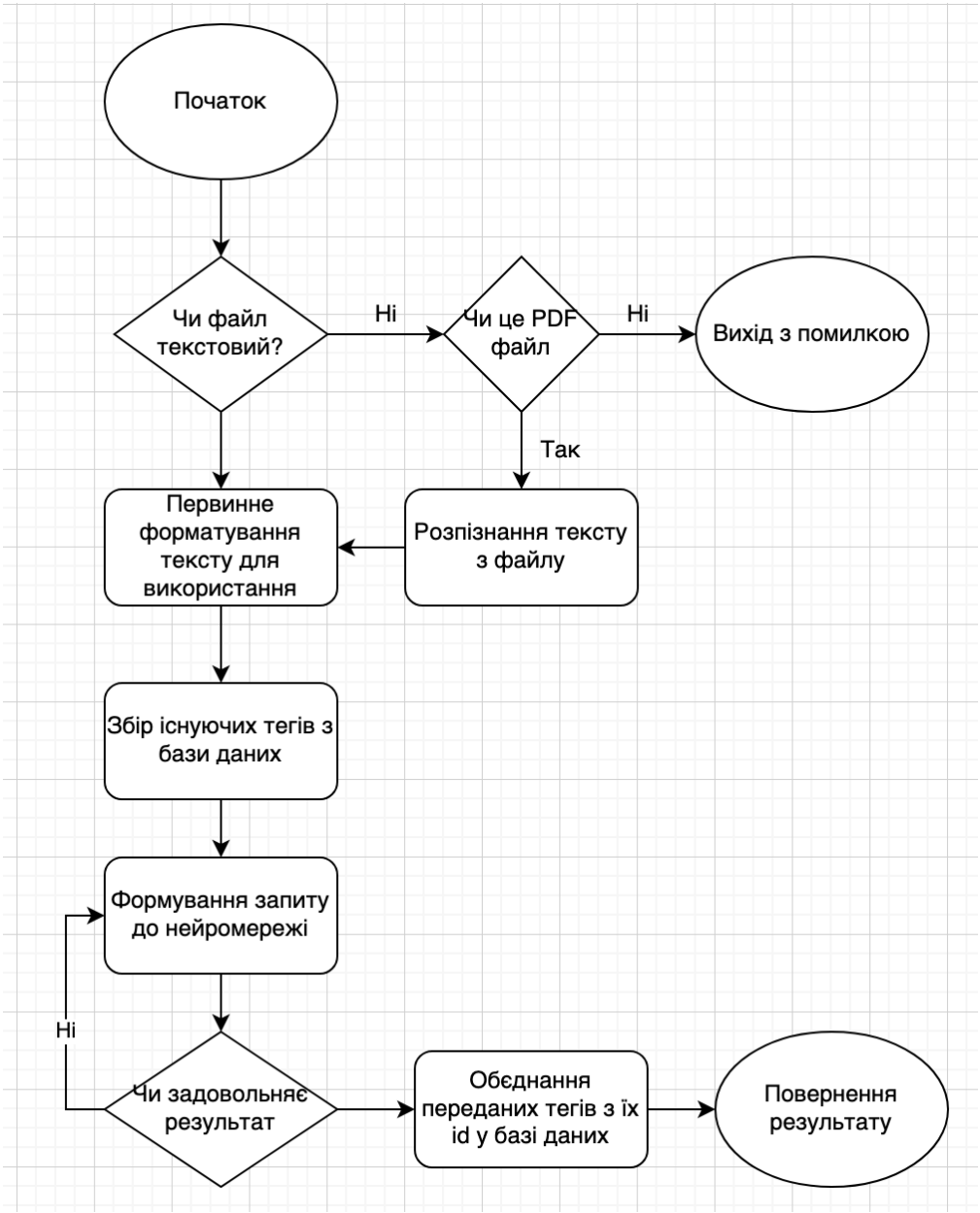


Рисунок 3.4 – Алгоритм системи автоматизованого тегування документів (рисунок виконано самостійно)

Для інтеграції з ChatGPT ми використовуватимемо REST API, який дозволяє надсилати HTTP запити до серверу моделі. Основні кроки для створення запиту включають вибір відповідного методу (POST),

додавання необхідних заголовків (Content-Type та Authorization) та вкладення документу у тіло запиту у форматі JSON.

Після підготовки запиту необхідно надіслати його до сервера ChatGPT. Сервер обробить запит та поверне відповідь у форматі JSON, що міститиме результати аналізу, зокрема список тегів для документу.

Отримавши відповідь від сервера, необхідно розпарсити її, щоб витягти потрібні дані. У нашому випадку це список тегів, який ми використовуватимемо для каталогізації документу. Алгоритм повинен забезпечити коректну обробку відповіді, враховуючи можливі помилки або несподівані формати даних.

Отримані теги потрібно зберегти у базі даних для подальшого використання. Це дозволить легко знаходити документи за тегами та забезпечить швидкий доступ до необхідної інформації.

Алгоритм відправки документів та обробки відповідей від ChatGPT повинен бути інтегрований у загальну систему. Це може бути реалізовано через створення тригерів у базі даних, що автоматично викликатимуть алгоритм при додаванні нових документів, або через безпосередні виклики у відповідних місцях коду.

При роботі з документами та їхнім відправленням до зовнішніх сервісів важливо забезпечити безпеку та конфіденційність даних. Використання HTTPS для передачі даних гарантує захист від перехоплення інформації. Крім того, необхідно дотримуватися стандартів безпеки при зберіганні та обробці даних у системі.

Наступна автоматизация це формування списків очікуваних та просрочених документів, формування повідомлення та відправка його користувачу.

Схематично алгоритм має наступний вигляд (див. рис. 3.5).

Для даного проекту важливо вибрати оптимальний канал зв'язку для ефективної комунікації. Існують різні канали зв'язку, які можна використовувати для цього. Серед них можна виділити електронну

пошту, мессенджери та інші засоби. Проте, кожен з цих каналів має свої переваги та недоліки.

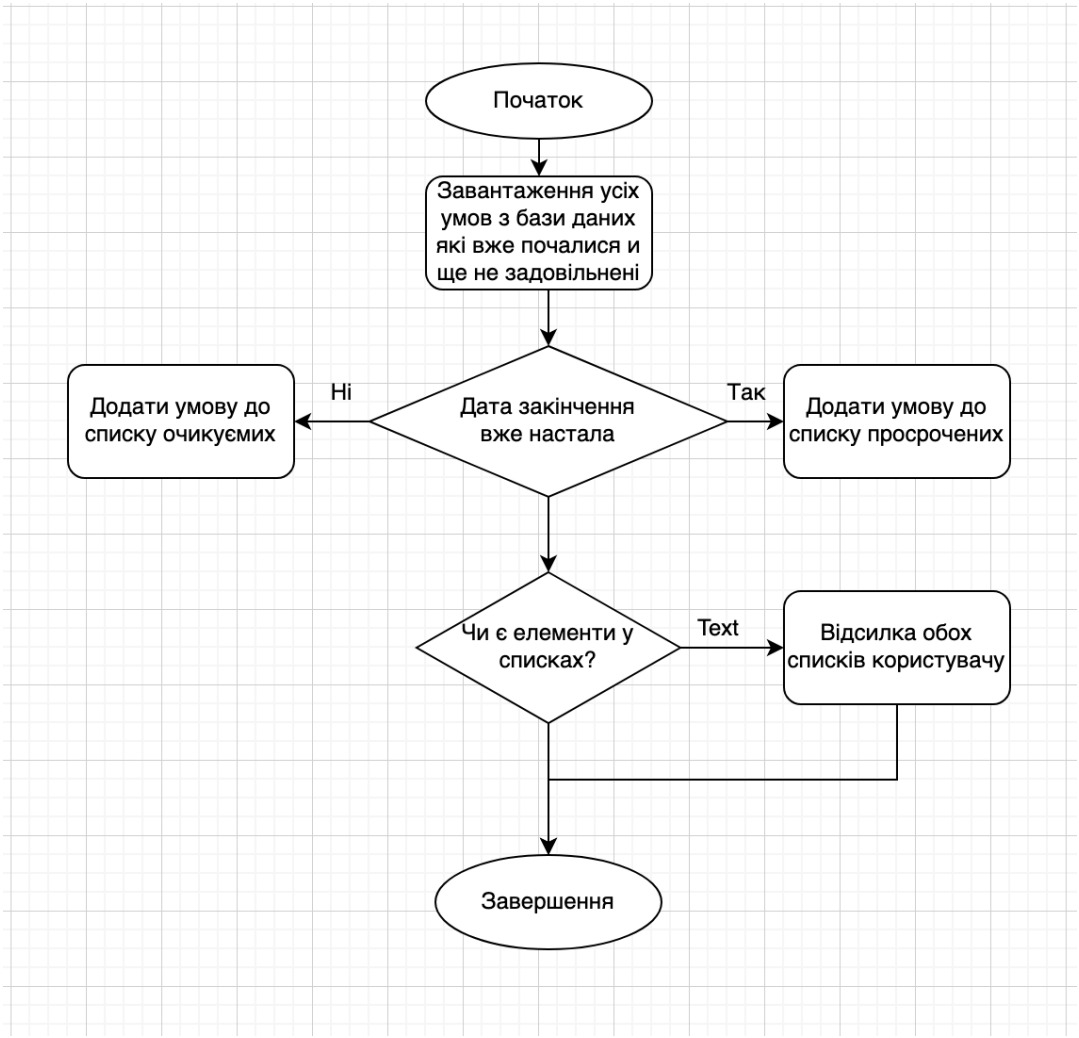


Рисунок 3.5 – Алгоритм системи контролю виконная умов документообігу (рисунок виконано самостійно)

Одним з популярних та зручних варіантів є використання мессенджера Telegram. Він набув значної популярності завдяки своїй зручності та функціоналу. Однією з основних переваг Telegram є те, що він безкоштовний для відправки та отримання повідомлень, що робить його доступним для широкого кола користувачів.

Для інтеграції Telegram з іншими системами будемо використовувати REST API. Це дозволить автоматизувати процеси

відправки та отримання повідомлень, що значно спрощує роботу з мессенджером. Використання REST API забезпечує високу гнучкість та надійність інтеграції, а також дозволяє масштабувати систему залежно від потреб проекту.

Завдяки використанню мессенджера Telegram та його інтеграції через REST API, ми зможемо забезпечити швидку, зручну та надійну комунікацію для проекту.

3.3 Проектування бази даних

На даному етапі проектування БД доцільно за основу взяти загальну діаграму класів. Було розроблена ER-діаграма за нотацією Баркера (див. рис. 3.6).

Реляційна база даних – це база даних, яка використовується для зберігання та організації доступу до взаємопов’язаних елементів інформації. Реляційні бази даних ґрунтуються на реляційній моделі – інтуїтивно зрозумілому, наочному поданні інформації у вигляді таблиць. Кожен рядок у таблиці такої бази даних являє собою запис унікальним ідентифікатором – ключем. Стовпці таблиць мають атрибути даних, що дає змогу встановлювати зв’язки між елементами даних.

Саме цьому, реляційна модель бази даних є найбільш ефективною для вирішення задачі даної кваліфікаційної роботи.

Нормальна форма – це вимога, що висувається до структури таблиць в теорії реляційних баз даних для усунення з бази зайвих функціональних залежностей між атрибутами. В нашому проекті при нормалізації виконується приведення таблиць до третьої нормальної форми. Відношення знаходиться у третій нормальній формі, коли воно знаходиться у першій та другій нормальній формі, тобто всі атрибути атомарні, кожний не ключовий атрибут повністю функціонально залежить від первинного ключа відношення та між неключовими атрибутами немає транзитивних залежностей.

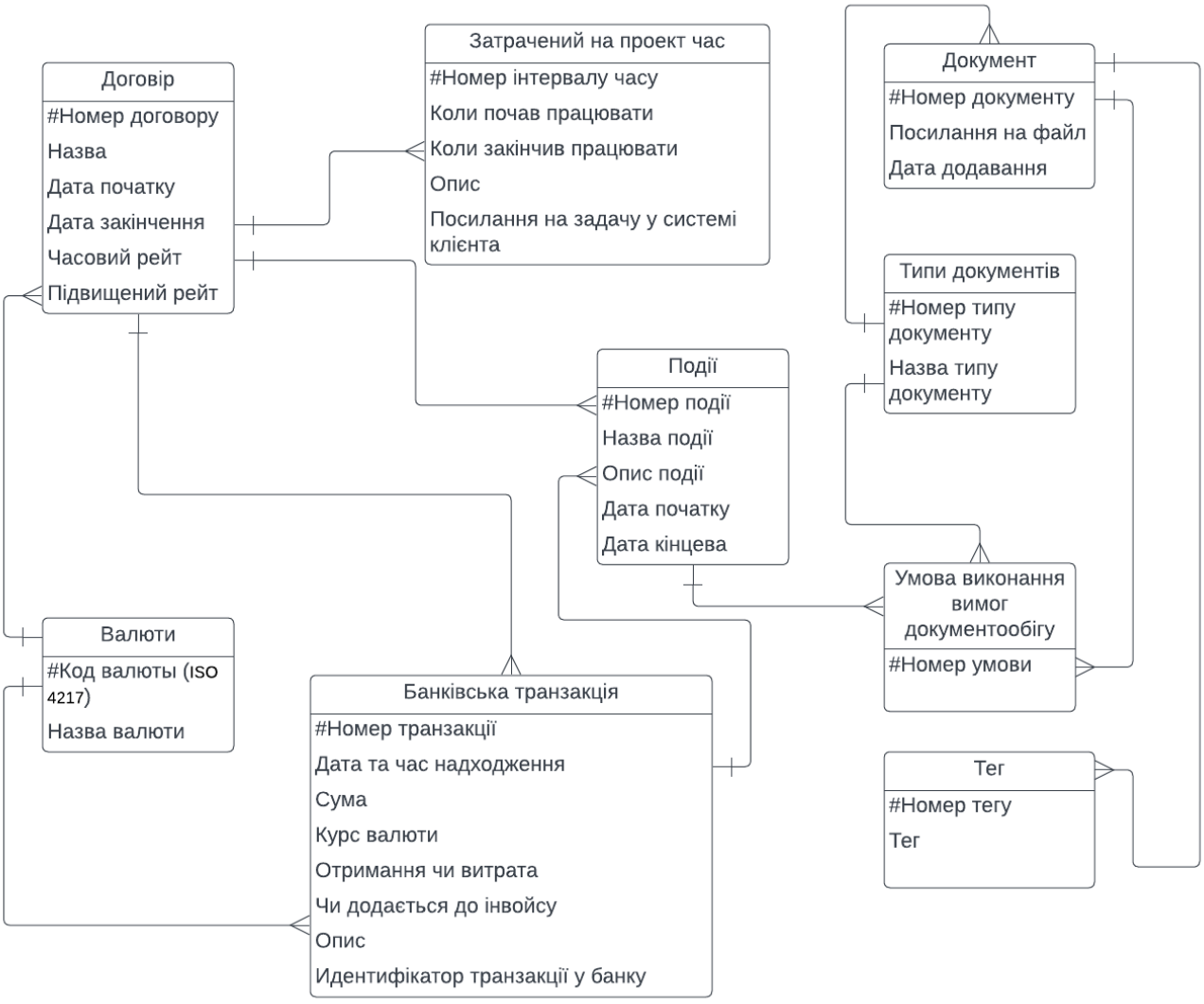


Рисунок 3.6 – ER-діаграма за нотацією Баркера (рисунок виконано самостійно)

В усіх відношеннях всі атрибути є атомарними, тому вони знаходяться в 1 НФ.При зведенні до другої нормальної форми було доведено, що всі відношення не містять неповних функціональних залежностей, тобто в складі потенційного ключа відсутня менша підмножина атрибутів від якої можна також вивести дану функціональну залежність.При зведенні до третьої нормальної форми була проведена перевірка на відсутність транзитивних функціональних залежностей. Отже, отримана схема БД знаходиться в 3НФ (див. рис. 3.7).

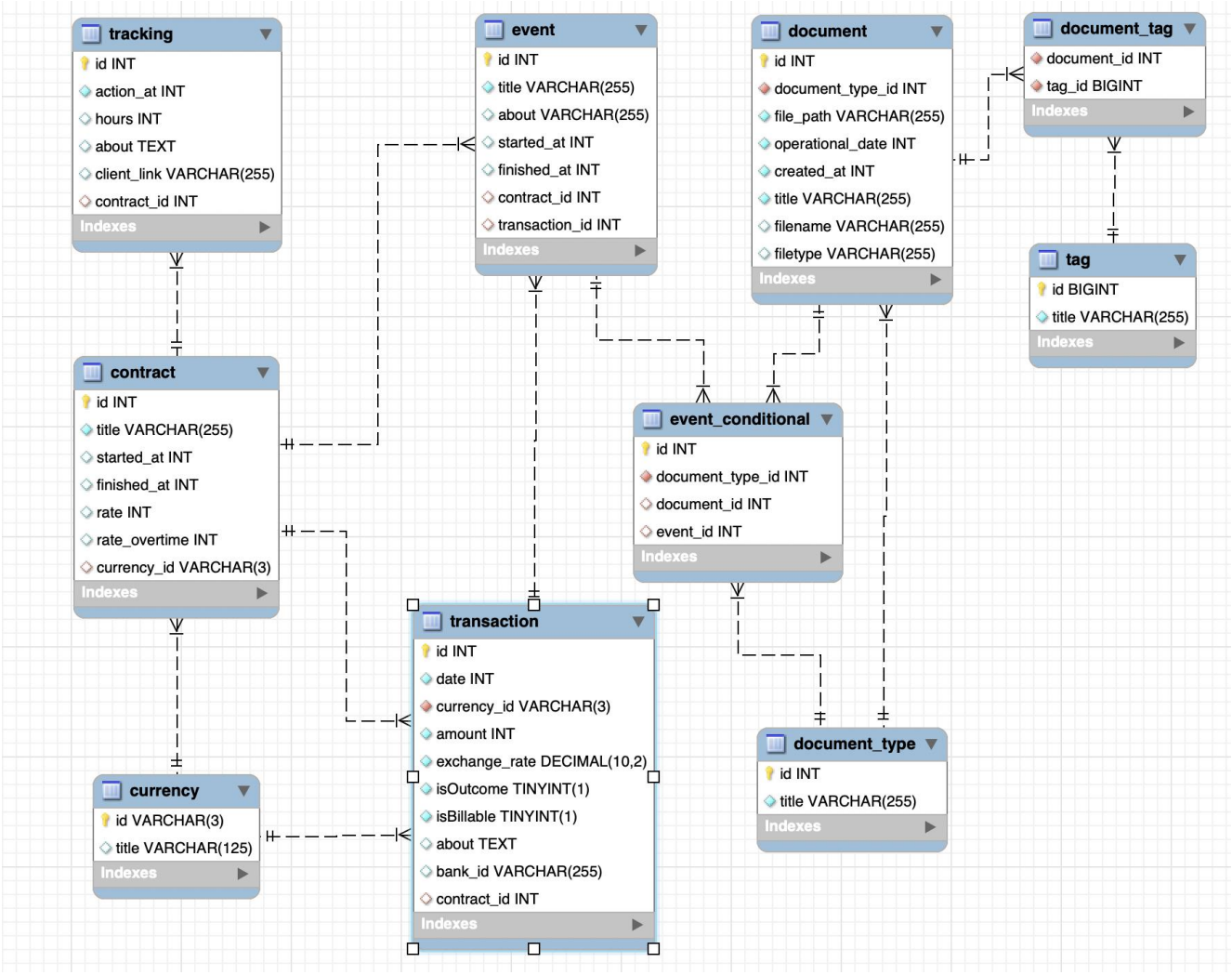


Рисунок 3.7 – Схема реляційної бази даних (рисунок виконано самостійно)

3.4 Розробка архітектури системи

В сучасній веб-розробці ключовим елементом успішного проекту є правильно спроектована архітектура системи. Особливо це стосується веб-додатків з фронтендом на JavaScript, які відображають сучасні вимоги до швидкості, масштабованості та користувацької зручності. У даному розділі роботи розглядається процес розробки архітектури системи для веб-додатків з фронтендом на JavaScript, зосереджуючись на використанні передових підходів та технологій для створення надійних та ефективних програмних рішень.

Загальна схема розробляємого додатку зображена на рисунку 3.8

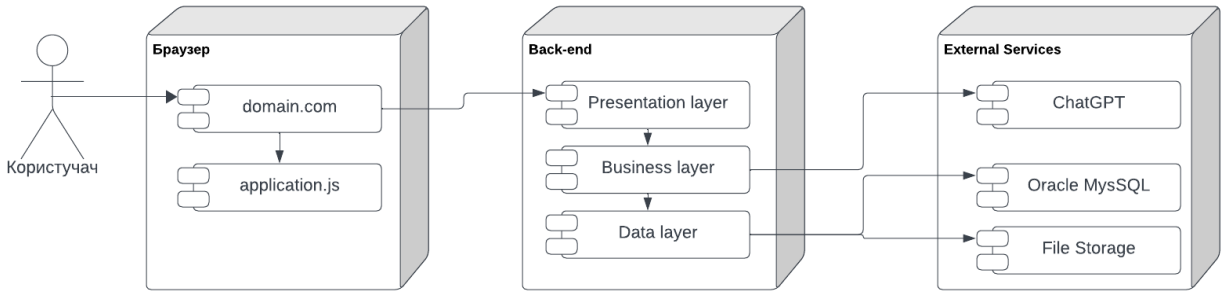


Рисунок 3.8 – Загальна схема архітектури програмної системи (рисунок виконано самостійно)

Таким чином Front-end частина дордатку забезпечую презентацію даних для користувача та відображення інтерфейсу. У майбутньому це дозволить створити окремі додатки для мобільних платформ та клієнти для різних операційних систем.

Beck-end, серверна частина, реалізую усю логіку додатку, реалізує API методи для коммунікації з front-end частиною додатку. Також реалізує зв'язок в шаром збереження даних – файлове сховище та базою даних.

Слід зазначити, що бекенд не є монолітною структурою. Він складається з деяких окремих артефактів-контейнерів, що дозводяє вгнучко додавати функціонал до системи та, за необхідністю, масштабувати.

Загальна схема усіх артефактів зображена на рисунку 3.9.

Як вхідна точка додатку через яку користувач отримує доступ до системи буде використовуватись веб сервер Nginx [11] який часто використовується в архітектурі веб-додатків як зворотний проксі, балансувальник навантаження та HTTP-сервер. Він ефективно розподіляє вхідні запити між кількома вузлами сервера, забезпечуючи високу доступність і відмовостійкість програми. Завдяки високій продуктивності та низькому споживанню ресурсів, Nginx також може

кешувати статичний вміст, знижуючи навантаження на сервери та прискорюючи доставку сторінок користувачам. Цей сервіс дає користувачу досту до контейнеру з головним додатком. Доступу до інших модулів з зовнішнього середовища не має.

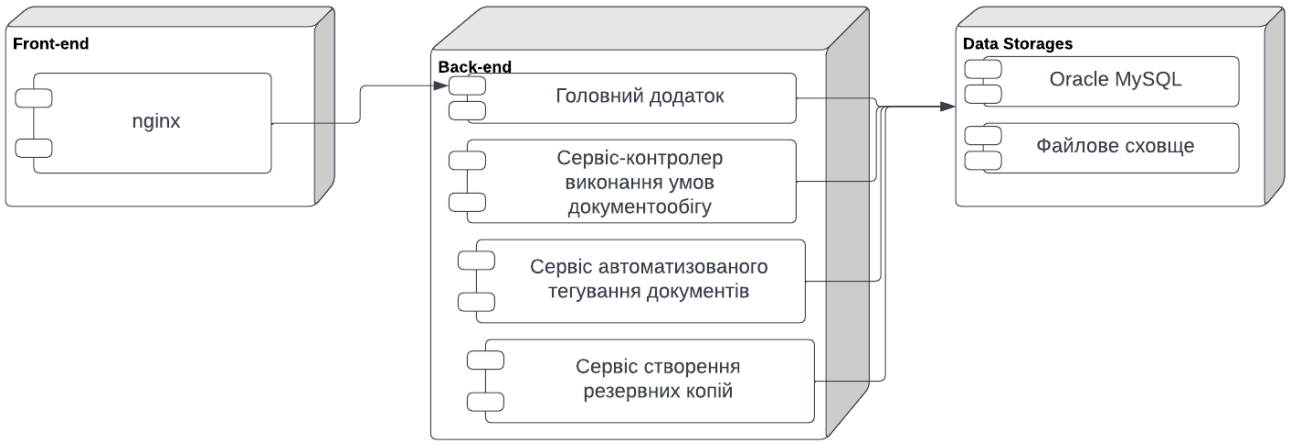


Рисунок 3.9 – Загальна схема розгортання програмної системи (рисунок виконано самостійно)

Серверна сторона веб-додатку складається з кількох автономних сервісів, які взаємодіють між собою за допомогою бази даних або API-викликів. Це забезпечує модульність, гнучкість та масштабованість системи. Ключовим компонентом є головний додаток, що відповідає за реалізацію інтерфейсу користувача та основного функціоналу керування даними. Він також забезпечує доступ до результатів роботи інших модулів за потреби, що дозволяє користувачам отримувати необхідну інформацію в режимі реального часу.

Одним із важливих модулів є сервіс-контролер виконання умов документообігу, який автономно контролює виконання встановлених умов документообігу та надсилає повідомлення користувачам про їх порушення. Це дозволяє забезпечити своєчасне реагування на будь-які відхилення від нормального процесу документообігу, підвищуючи ефективність та надійність системи.

Сервіс автоматизованого тегування документів реалізує функціонал автоматичного додавання тегів до документів, які користувач завантажує в систему. Це спрощує подальший пошук та організацію документів, що значно покращує зручність роботи користувачів із системою.

Сервіс створення резервних копій автоматично виконує резервне копіювання системи у визначений час, що забезпечує збереження даних та можливість їх відновлення у випадку непередбачених ситуацій. Регулярне резервне копіювання є критично важливим для забезпечення надійності та безпеки даних.

Кожен із цих сервісів реалізується як окремий Docker-контейнер, що надає можливість широкої конфігурованості додатку у майбутньому. Використання Docker-контейнерів дозволяє ізолювати сервіси один від одного, що спрощує їх оновлення, масштабування та обслуговування. Це також забезпечує сумісність сервісів з різними середовищами та полегшує розгортання додатку на різних платформах.

3.5 Створення UI / UX або іншого дизайну системи

Для структури сторінок була обрана ґратчаста структура. Необхідні для сценаріїв сторінки розташовані у різних розділах, але дані пов’язані і повинен існувати спосіб швидко до цих даних переходити з усіх розділів.

Нижче додана повна схематична структура сторінок (див. рис. 3.10).

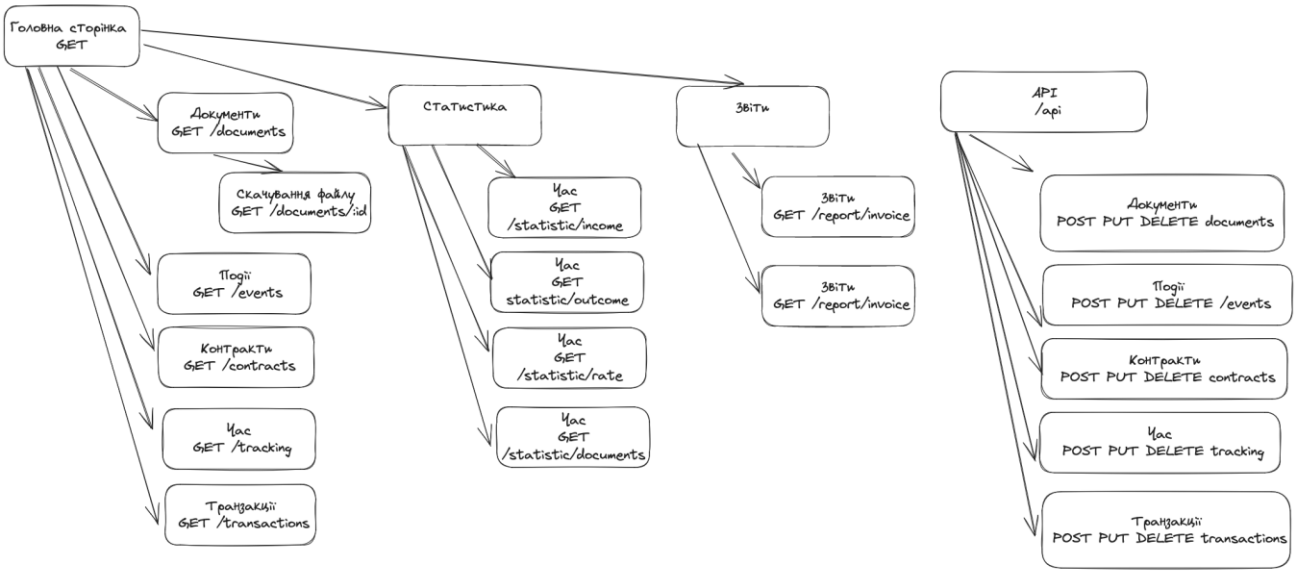


Рисунок 3.10 – Дерево сторінок сайту (рисунок виконано самостійно)

Структура розділена на дві незалежні частини:

– відображення функціональних сторінок з інтерфейсом для користувача та API. Відображення сторінок забезпечує інтерактивний інтерфейс для користувача, який дозволяє взаємодіяти з додатком та виконувати різні дії. Відображення сторінок може бути здійснене з використанням різноманітних технологій, таких як HTML, CSS та JavaScript, які забезпечують різноманітні функції, такі як відображення даних, введення даних користувачем, відображення графічних елементів тощо.
– API - який є іншою важливою складовою частиною web-додатків. Він забезпечує користувачеві інтерфейс для доступу до даних та операцій з ними. API може бути доступним з різних причин, таких як обмін даними між додатками, підтримка мобільних додатків та більш складних операцій з даними. API забезпечує різноманітні можливості для роботи з даними та дозволяє розробникам створювати складні та інноваційні додатки, які

можуть забезпечувати значну користь для користувачів та бізнесу в цілому.

Сторінки користувацького інтерфейсу доступні через GET HTTP запити, на які система відправляє html сторінки який згенеровано на сервері.

У випадку з сторінкою скачування буде відправлено файл

Сторінки API віддають дані у JSON форматі та доступні по через GET | POST | PUT | DELETE HTTP запити в залежності від операції з даними які потрібно виконати.

У випадку запиту на неіснуючу сторінку буде відправлено 404 помилку.

Кожна сторінка надає можливість виконувати CRUD операції над однією з сутностей. Усі операції виконуються на одній сторінці.

Сторінки з групи «Статистика» генерують csv файли з результатами роботи запитів.

Сторінки з групи «Звіти» ведуть на сторінки звітів.

Для прототипування використовуються wireframes. Wireframes забезпечують візуальне представлення макета та структури веб-сторінки, допомагаючи зрозуміти користувальницький інтерфейс та користувальницький досвід веб-програми. Вони служать планом для остаточного дизайну, забезпечуючи просте представлення кінцевого продукту. Використовуючи їх можна виявити потенційні проблеми з макетом, потоком та функціональністю свого веб-додатка, перш ніж переходити до більш детальних проектів. Це може заощадити час і ресурси в довгостроковій перспективі, оскільки зміни можуть бути внесені на початку процесу. В цілому, вайрфрейми є потужним інструментом для створення ефективних, зручних веб-додатків, які відповідають потребам своєї цільової аудиторії.

У системі присутні три основні типи сторінок.

Перший тип – сторінка відображення даних. Цей тип сторінки використовується для відображення, редагування та видалення даних з системи. Основний макет такої сторінки включає в себе таблиці, форми для редагування та кнопки для виконання дій, таких як збереження змін або видалення записів. Детальний макет сторінки відображення даних представлено на рисунку 3.11.

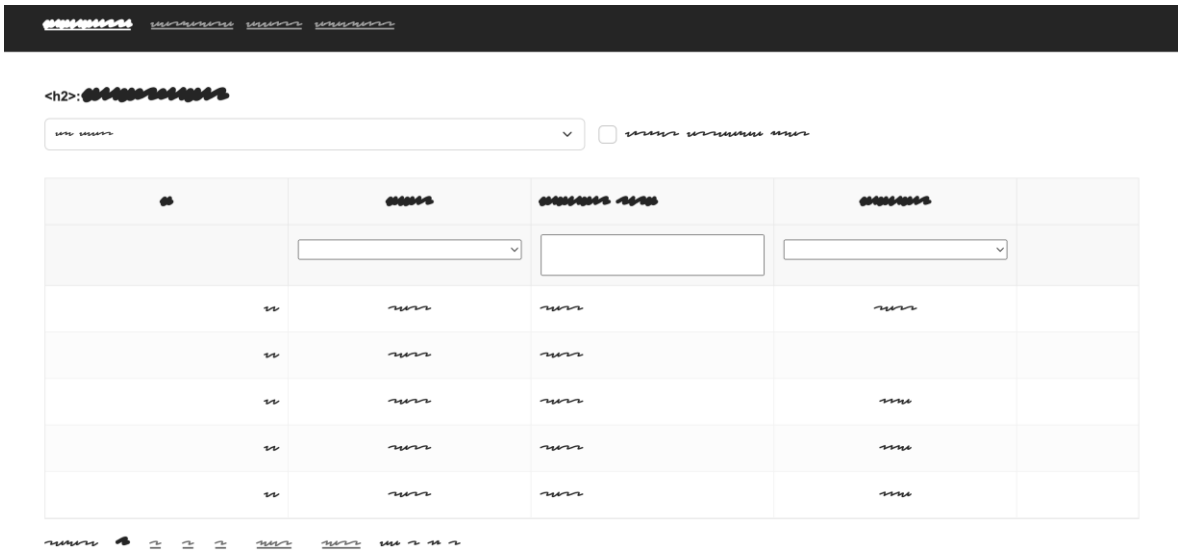


Рисунок 3.11– Загальний макет сторінки керування даними (рисунок виконано самостійно)

Другий тип – сторінка згенерованого звіту. На цій сторінці відображаються результати генерації звіту в системі. У верхній частині сторінки розміщені поля для вводу параметрів, які користувач може налаштовувати для отримання потрібного звіту. Після введення параметрів, нижче відображається результат розрахунку у вигляді таблиць, графіків або текстових даних. Загальний макет сторінки згенерованого звіту представлено на рисунку 3.12.

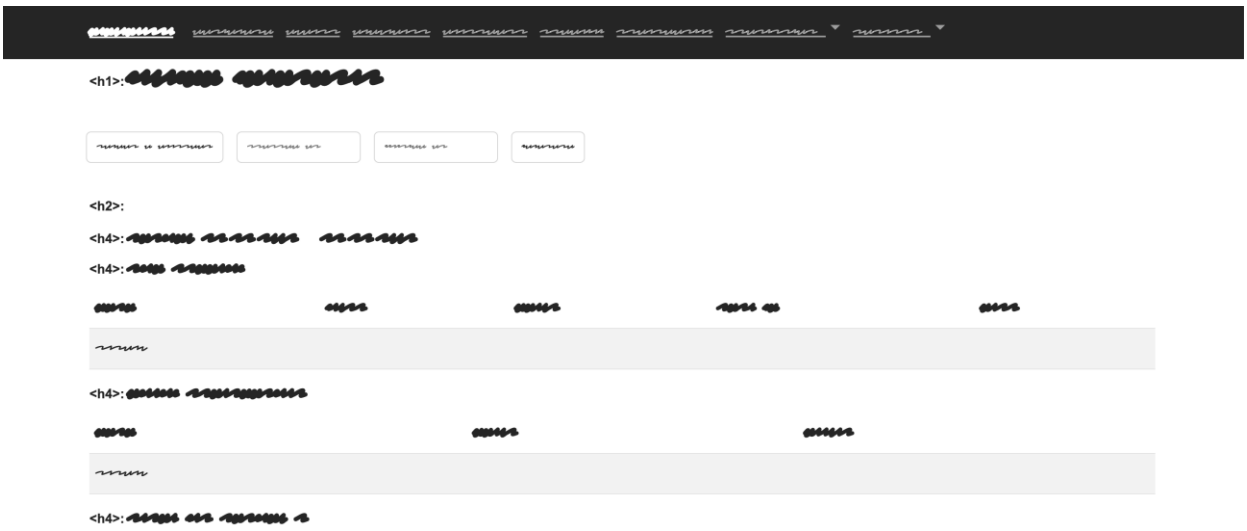


Рисунок 3.12 – Загальний макет сторінки звіту (рисунок виконано самостійно)

Третій тип сторінок – сторінка згенерованої статистики. Цей тип сторінки призначений для виведення статистичних даних, які генеруються системою у вигляді CSV файлів. Оскільки виведення здійснюється у форматі CSV, для таких сторінок не потрібен окремий графічний макет. Користувач може завантажити згенерований файл для подальшого аналізу або обробки за допомогою інших програм.

Ці три типи сторінок забезпечують повний функціонал для роботи з даними в системі, включаючи їх відображення, редагування, генерацію звітів та виведення статистичних даних. Така структура дозволяє користувачам зручно і ефективно взаємодіяти з системою, забезпечуючи високу продуктивність і точність обробки інформації.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Вибір засобів програмної реалізації

Для розробки веб-застосунку було обрано ряд інструментів та технологій, що забезпечують ефективність, гнучкість та надійність у створенні програмного забезпечення.

Враховуючи вимоги проекту та сучасні практики розробки, було прийнято рішення використовувати наступні засоби програмної реалізації: серверну частину буде реалізовано на PHP, що є потужною та популярною мовою програмування для створення веб-застосунків, яка забезпечує широкий спектр функціональних можливостей, гарну продуктивність та велике співтовариство розробників.

В якості бази даних використовується MySQL, яка є надійною та продуктивною реляційною СКБД з широкими можливостями та активною підтримкою.

Для роботи з базою даних використовується також Doctrine DBAL, що забезпечує абстракцію над різними СКБД та дозволяє працювати з базами даних на високому рівні, використовуючи об'єктно-орієнтовані підходи, спрощуючи маніпуляції з даними та забезпечуючи прозорість і безпеку операцій.

Для ведення логів у застосунку використовується бібліотека Monolog, яка дозволяє зберігати логи в різних форматах і місцях, таких як файли, бази даних, системи відстеження помилок та інші, забезпечуючи гнучку конфігурацію та легкість інтеграції в проект.

Бібліотека Dotenv використовується для зберігання конфігураційних параметрів у вигляді середовищних змінних, що дозволяє розробникам легко керувати налаштуваннями застосунку без необхідності змінювати код; конфігураційні файли .env зберігаються окремо та можуть бути різними для різних середовищ (розробка, тестування, продакшн).

Для генерації HTML-шаблонів використовується шаблонізатор Twig, що забезпечує розділення логіки бізнес-процесів та представлення даних,

роблячи код чистішим та легшим для підтримки; Twig підтримує різні фільтри та функції для зручного опрацювання даних у шаблонах.

На фронтенді використовується бібліотека jQuery, яка значно спрощує роботу з DOM, подіями, ефектами та AJAX-запитами, маючи зручний API та велику кількість плагінів для розширення функціональності. Для покращення користувацького досвіду при роботі з випадаючими списками використовується плагін Select2, який дозволяє створювати зручні та функціональні списки з підтримкою пошуку, автозаповнення та інших функцій. Для створення інтерактивних інтерфейсів застосовується бібліотека jQuery UI, яка надає широкий набір компонентів, таких як діалоги, календарі, слайдери та інші, що дозволяє швидко та зручно створювати багатофункціональні користувацькі інтерфейси. Для забезпечення стабільного та відтворюваного середовища розробки та деплойменту використовується Docker, який дозволяє упаковувати застосунок з усіма залежностями у контейнери, що значно спрощує процеси налаштування та розгортання проекту.

Система контролю версій Git використовується для відстеження змін у коді, співпраці між розробниками та керування версіями проекту, забезпечуючи надійність зберігання історії змін та можливість швидкого відновлення попередніх станів проекту.

Для документування API використовується Swagger, який дозволяє створювати інтерактивну документацію для RESTful API, що полегшує розробку, тестування та інтеграцію різних компонентів системи. Таким чином, вибір цих засобів програмної реалізації забезпечує надійність, гнучкість та зручність у розробці, підтримці та масштабуванні веб-застосунку.

Для розробки використовуються середовища JetBrains PhpStorm та DataGrip, які забезпечують розширений функціонал для роботи з кодом і базами даних, зручні інструменти для налагодження та підвищення

продуктивності розробників. Таким чином, вибір цих засобів програмної реалізації забезпечує надійність, гнучкість та зручність у розробці, підтримці та масштабуванні веб-застосунку.

4.2 Опис фізичної моделі бази даних

В якості СКБД для зберігання даних було обрано Oracle MySQL. З'єднання бази даних та серверної частини забезпечується бібліотекою Doctrine DBAL. Ця бібліотека забезпечую низькорівневу роботу з базою даних на рівні конструювання SQL запитів. Це дозволяє провести оптимізацію запитів, наприклад розрахунки статистики чи звітів, та з іншого боку мати певний рівень абстракції від використаної бази даних та мати можливість у майбутньому її змінити, наприклад, на локальну SQLite.

При реалізації фізичної моделі бази даних було створено вісім таблиць. Нижче наведено приклад SQL команди на їх створення.

```
create table event_conditional
(
    id                int auto_increment comment '#Номер умови'
primary key,
    document_type_id int not null comment 'Тип документу',
    document_id       int null,
    event_id          int null,
    constraint FK_event_conditional_document_id
        foreign key (document_id) references document (id),
    constraint FK_event_conditional_document_type_id
        foreign key (document_type_id) references document_type
(id),
    constraint FK_event_conditional_event_id
        foreign key (event_id) references event (id)
)
    comment 'Умова виконання вимог документообігу';
```

```
create index FK_document_type_id_idx
    on event_conditional (document_type_id);

create index FK_doxument_id_idx
    on event_conditional (document_id);

create index FK_event_conditional_event_id_idx
    on event_conditional (event_id);

create index FK_transaction_contract_id_idx
    on transaction (contract_id);

create index FK_transaction_currency_id_idx
    on transaction (currency_id);
```

4.3 Опис програмної реалізації алгоритму автоматичного тегування

Алгоритм автоматичного тегування складається з декількох послідовних етапів.

Перший з них – валідація відного файлу. На даний момент тільки .txt чи pdf файли можуть бути оброблені. Програмна реалізація першого етапу є наступною:

```
protected function isTypeSupported(string $fileType) : bool
{
    $supportedTypes = [
        'application/pdf',
        'text/plain',
        'text/rtf',
    ];

    return in_array($fileType, $supportedTypes);
}
```

Друий етап це отримання списку усіх тегів які є у системі та які повинні бути використані при генерації тегів нейромережею. Приоритетними є теги які вже існують в системі та відповідають хоч одному файлу у системі.

Для цього викростовується запит:

```
select document_tag.tag_id, tag.title from document_tag
join tag ON document_tag.tag_id = tag.id
GROUP BY document_tag.tag_id
```

Якщо немає жодного, то ми беремо усі теги які є у системі

```
$existedTags = [];
$tagList = $this->tagService->get([]);
if (!empty($tagList['itemsCount'])) {
    foreach ($tagList['data'] as $item) {
        $existedTags[] = $item['title'];
    }
}
```

Наступний етап це конфігування запиту до моделі ChatGPT з якої ми отримаємо згенеровані теги

```
$tags = $this->openApiClient->chat()->create([
    'model' => 'gpt-4o',
    'messages' => [
        ['role' => 'user', 'content' => "Вот текст из файла с именем {$fileName}:\n\n{$fileContent}\n\nТеперь, пожалуйста, сделай следующее: Напиши список тегов которыми можно описать назначение и содержимое этого файла. Напиши не более {$tagLimit} тегов.

        По возможности используй теги из этого списка, но не ограничивайся ими: {$existedTags}. Используй {$delimiter} в качестве разделителя"],
    ],
]);
```

Після отримання тегів необїідно перевірити чи ічнують вони у базі даних та зєднати з їх id у базі

```
$list = "";
if (!empty($tags)) {
    $list = $tags->choices[0]->message->content;
}

$tags = explode($delimiter, $list);
$tags = array_map(fn($value): string => trim($value), $tags);

foreach ($tags as $tagRow) {
    $tagData = $this->tagService->get(['title' => $tagRow]);
    $result[] = [
        'id' => !empty($tagData['itemsCount']) ?
$tagData['data'][0]['id'] : null,
        'text' => $tagRow,
    ];
}
```

Після цього дані готові для відправки на сторінку для використання.

4.4 Опис програмної реалізації алгоритму контролю документообігу

Невиконання вимог документообігу може створити критичні або коштовні проблеми у майбутньому, тому необхідна автоматизація яка буде слідкувати за їх виконанням та попереджати про строки завершення виконання чи про просрочені вимоги.

Також існує проблема в детальному описі яких конкретно документів та у якій події не вистачає. Система повинна це знаходити та формувати попередження, яке повинно проходити автоматично до користувача.

Автоматизація виконана у вигляді консольної задачі яка повинна виконуватися за допомогою планувальника cronjob кожного дня, аналізувати вимогу та якщо необхідно відправляти повідомлення користувачу.

Задача може бути виконана у ручному режимі за допомогою команди

```
make alert
```

Як транспорт для цих повідомлень було обрано Telegram як поширену систему, бескоштовну та доступним API.

Існує 2 типи документів про які треба повідомлення:

– документи які вже повинні бути додані та час яких ще не сплинув;

– документи які вже повинні бути додані та час яких ще сплинув.

Якщо перевірка знайшла хоч один з цих типів документів то буде відправлене повідомлення приклад повідомлення можна побачити на рисунку 4.1.

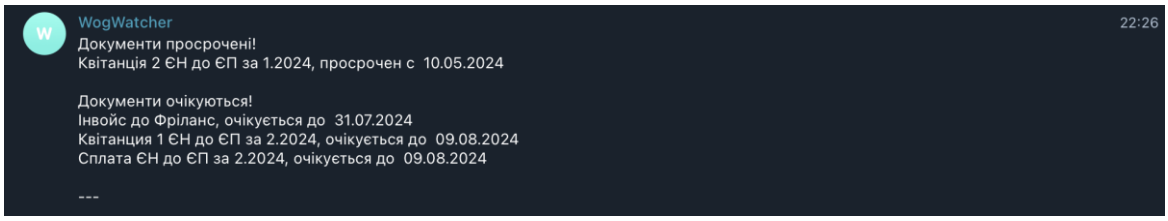


Рисунок 4.1 – Приклад повідомлення про невиконная умов документообігу (рисунок виконано самостійно)

Перший етап – завантаження з бази даних усіх невиконних умов для подій які вже стартували.

```
$sql = "select e.title, e.started_at, e.finished_at, dt.title as
document_type, e.title as event
from event_conditional ec
JOIN event e on ec.event_id = e.id
JOIN document_type dt on dt.id = ec.document_type_id
WHERE ec.document_id is NULL AND e.started_at <  ? AND
e.finished_at <  ?
ORDER BY finished_at ASC ;
";
$query = $databaseConnection->prepare($sql);
```

```
$result = $query->executeQuery([$current_time, $limit_time])-
>fetchAllAssociative();
```

Після цього наступним етапом потрібно розсортувати умови на ті які очікуються – їх дата закінчення ще не настала, та прострочені – дата закінчення яких вже пройшла.

```
$waited = [];
$failed = [];
foreach ($result as $row) {
    if ($row['finished_at'] < $current_time) {
        $failed[] = $row;
    } else {
        $waited[] = $row;
    }
}
```

Третій етап – формування повідомлень для відправки

```
if (!empty($failed)) {
    $content .= PHP_EOL."Данные документы просрочены!".PHP_EOL;
    foreach ($failed as $row)
    {
        $content .= $row['document_type']." для
".$row['title'].", просрочен с  ".date("d.m.Y",
$row['finished_at']).PHP_EOL;
    }
}

if (!empty($waited)) {
    $content .= PHP_EOL."Данные документы ожидаются!".PHP_EOL;
    foreach ($waited as $row)
    {
        $content .= $row['document_type']." для
".$row['title'].", ожидается до  ".date("d.m.Y",
$row['finished_at']).PHP_EOL;
    }
}
```

Четвертий етап - відправка до Telegram якщо є дані.

```
if (!empty($content)) {
    $content .= PHP_EOL . '---' . PHP_EOL;

    $response =
file_get_contents("https://api.telegram.org/bot$botApiKey/sendMes
sage?" . http_build_query([
            'chat_id' => $botChatId,
            'text' => $content
        ]));
}
```

4.5 Опис інтерфейсу користувача

Головною сторінкою є сторінка з переліком документів які потрібно завантажити до системи щоб задовольнити всі вимоги до документообігу.

На початку роботи немає жодних вимог и цей стан є тим, якого користувач повинен досягати. Приклад зображено на рисунку 4.2.

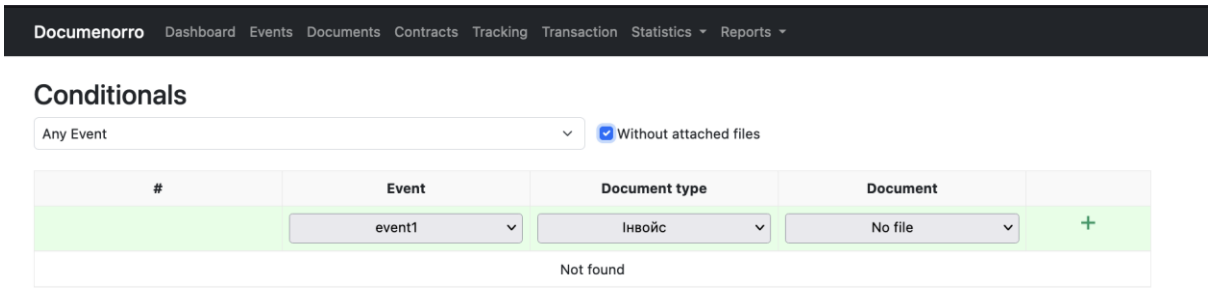


Рисунок 4.2 – Головна сторінка без незадовільнених умов (рисунок виконано самостійно)

Користувацький сценарій у проекті Documentorro починається із створення нової події. Це означає, що користувач ініціює процес збору нового пакету документів. У цьому контексті подія представляє собою

структуру, яка містить усі необхідні документи, що мають бути зібрані для досягнення конкретної мети.

Для цього треба перейти у розділ з подіями. Для того, щоб додати нову подію необхідно ввести дані у поля форми на натиснути кнопку додавання. Приклад можна побачити на рисунку 4.3

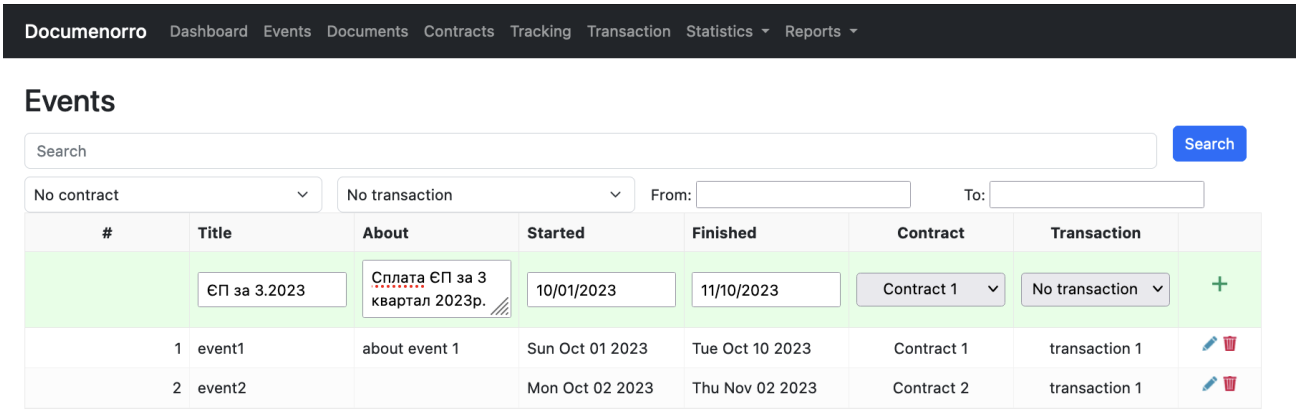


Рисунок 4.3 – Приклад заповнення форми додання події (рисунок виконано самостійно)

Зауважимо, що на цій та іншій формах працює система перевірки даних які ми вводимо. Наприклад, якщо не ввести назву події, то система не дасть її створити та виділить це поле кольором (див. рисунок 4.4).

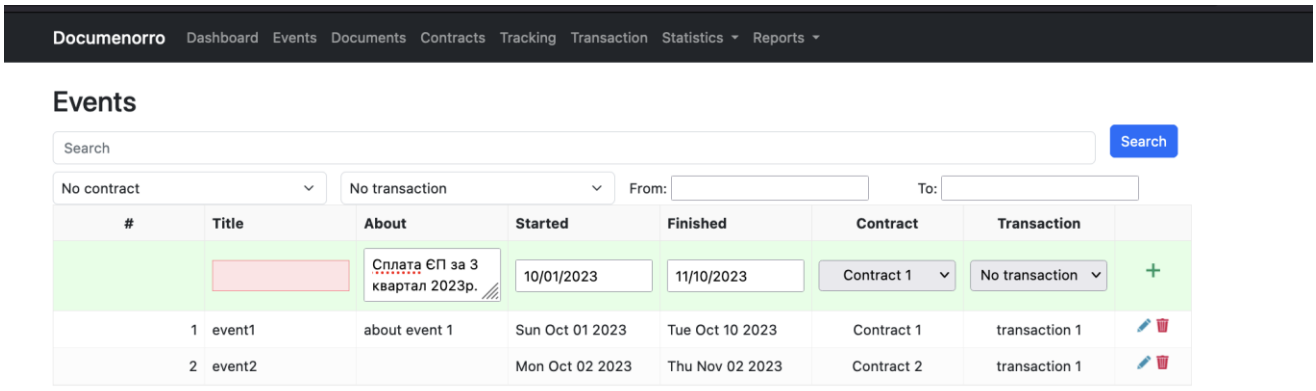


Рисунок 4.4 – Приклад заповнення форми додання події з помилкою (рисунок виконано самостійно)

Після успішного створення події вона буде додана до таблиці (див. рис. 4.5).

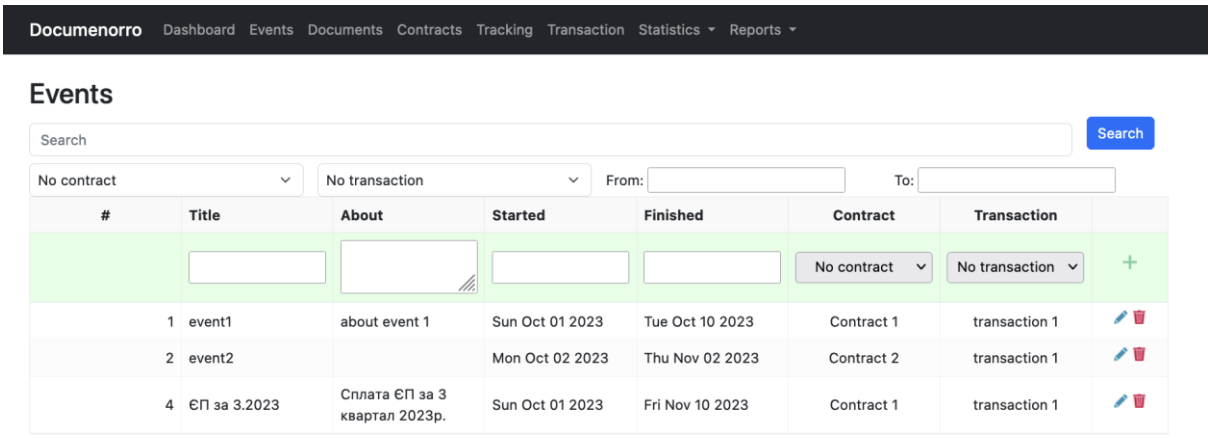


Рисунок 4.5 – Приклад сторінки списку подій (рисунок виконано самостійно)

Тепер можемо повернутися до головної сторінки та створити умову для цієї події – нам потрібно відправити звіт до налогової служби. Приклад заповнення можна побачити на рисунку 4.6

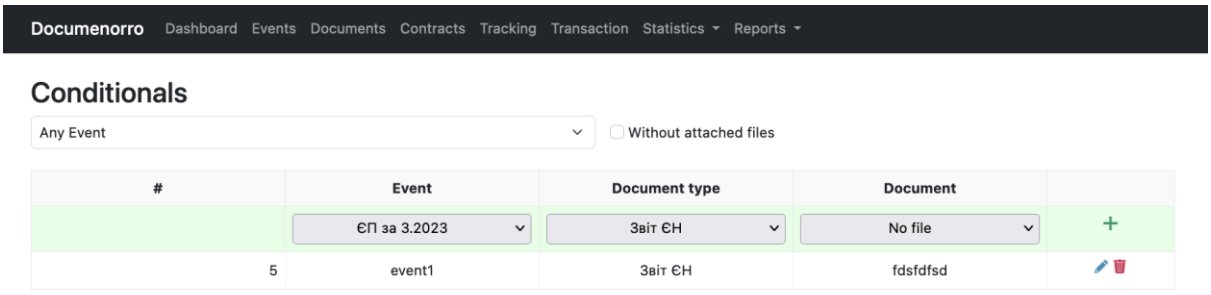


Рисунок 4.6 – Приклад заповнення форми додання умови до події (рисунок виконано самостійно)

Після додавання запис буде додано до списку (див. рис 4.7)

Зазначимо, що неможливо видалити подію яка має умови, задовільнені чи ні. Буде виведено помилку.

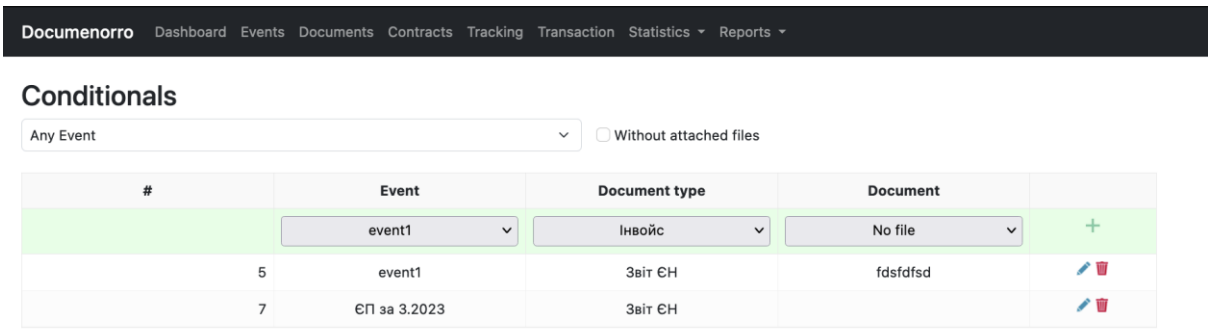


Рисунок 4.7 – Приклад сторінки списку умов (рисунок виконано самостійно)

Для перегляду списку документів у системі необходіно перейти за посиланням Documents у верхньому меню. Зовнішній вигляд сторінки з фільтрами та списком наведено на рисунку 4.8.

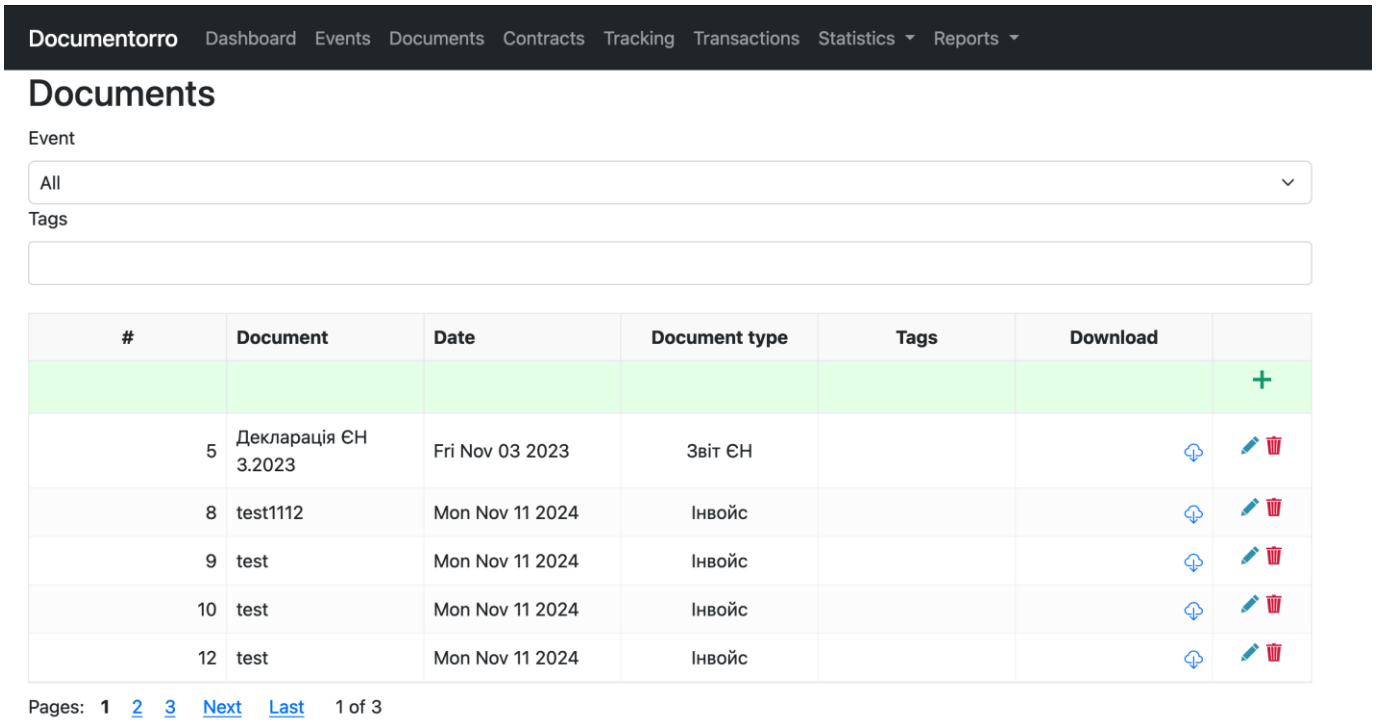


Рисунок 4.8 – Приклад сторінки списку документів (рисунок виконано самостійно)

Щоб додати новий документ необхідно натиснути на кнопку «Додати документ» у верхній частині сторінки.

Приклад заповнення зображено на рисунку 4.9.

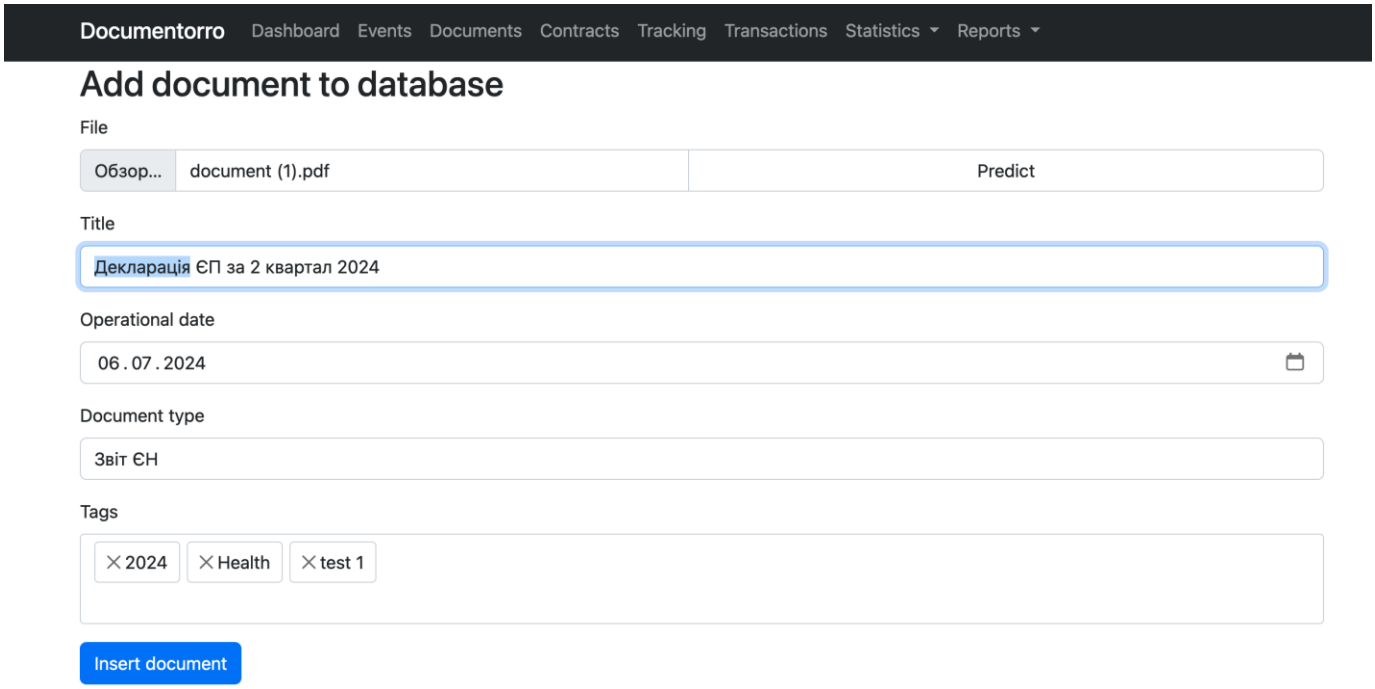


Рисунок 4.9 – Приклад заповнення форми додавання документа (рисунок виконано самостійно)

Зауважимо, що після додавання файл не може бути змінений, може бути виправлена лише його назва. Зазначимо, що неможливо видалити документ який доданий до хоча б однієї умови. Буде виведено помилку.

Останнім шагом сценарію є створення зв’язку між доданим документом та умовою, ку він повинен задовольнити.

Для цього на головному екрані потрібно перейти в режим редагування запису та в полі Document обрати доданий файл.

Таким чином користувач повністю виконав основний користувацький сценарій – була створена нова подія, створені умови для документообігу для неї, завантажений новий документ у систему та цей документ був доданий до умови, задовольнивши її.

Друга частина системи це робота с контрактами (див. рис 4.10).

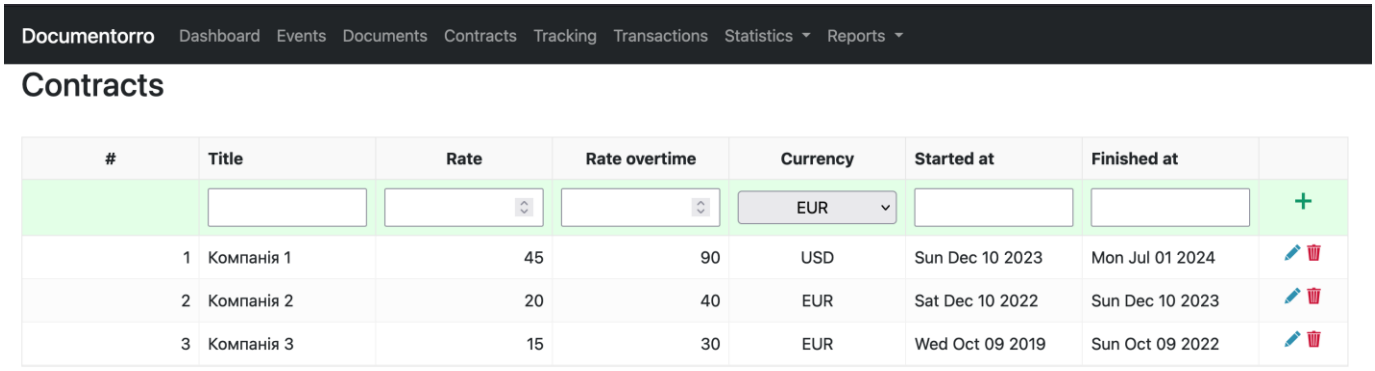


Рисунок 4.10 – Приклад сторінки списку контрактів (рисунок виконано самостійно)

На сторінці «Час» користувач може керувати часом який був витрачений на роботу на клієнта (див. рис 4.11).

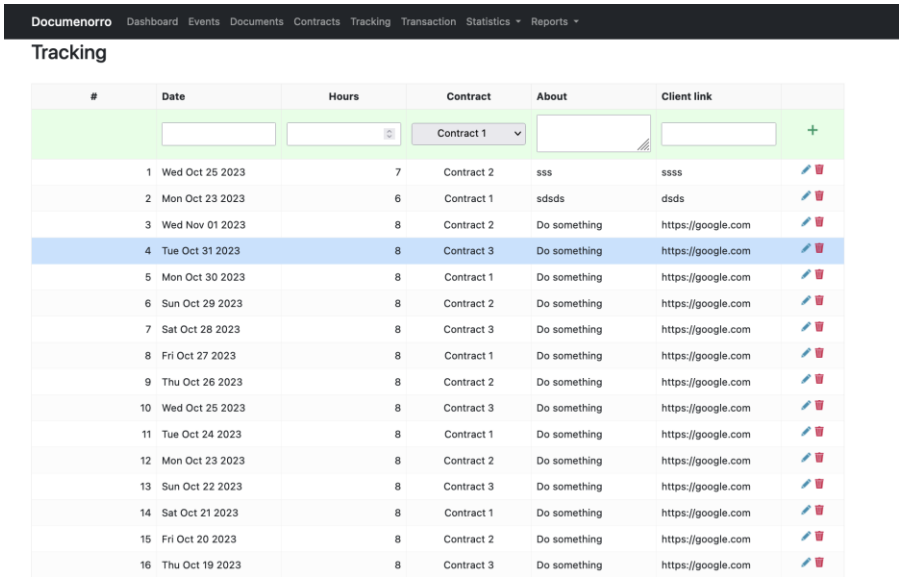


Рисунок 4.11 – Приклад сторінки списку витрат часу (рисунок виконано самостійно)

На сторінці «Транзакції» користувач може керувати переліком фінансових транзакцій які відносяться до його бізнесу За усіма полями доступно сортування (див. рис 4.12).

Transactions

id	Date	Currency	Amount	Exchange rate	is Outcome	is Billable	Contract	About	Bank ID	
		EUR					No contra			+
5	Tue Oct 24 2023	EUR	1000	33.60			Contract 2	transaction 1		
32	Sat Nov 04 2023	USD	4240	36.60			Contract 2	Transaction #bankid0	#bankid0	
33	Wed Oct 04 2023	USD	4240	36.60			Contract 3	Transaction #bankid1	#bankid1	
34	Fri Oct 27 2023	USD	3600	36.60			Contract 1	Transaction #bankid2	#bankid2	
35	Fri Aug 04 2023	USD	3600	36.60			Contract 2	Transaction #bankid3	#bankid3	
36	Tue Jul 04 2023	USD	4320	36.60			Contract 3	Transaction #bankid4	#bankid4	
37	Sun Jun 04 2023	USD	4400	36.60			Contract 1	Transaction #bankid5	#bankid5	

Рисунок 4.12 – Приклад сторінки списку транзакцій (рисунок виконано самостійно)

Алгоритм роботи усіх цих сторінок однаковий. Розглянемо його на основі сторінки умов подій.

Точка виклики api методу та обробка помилок

```
$router->get('/', function() use ($eventConditionalController)
{
    echo json_encode($eventConditionalController->apiGet($_GET));
});
```

Обробляємо дані та передаємо до сервісу

```
public function apiGet(array $input) : array
{
    $filter = [];

    $pageIndex = $input['pageIndex'] ?? null;
    $pageSize = $input['pageSize'] ?? null;
    $sortField = $input['sortField'] ?? "id";
    $sortOrder = $input['sortOrder'] ?? "asc";

    $filter = $this->fillFilter($filter, $input);

    if (empty($pageIndex)) {
```

```
        $result = $this->service->get([]);
    } else {
        $result = $this->service->getPage($filter, $pageIndex,
$pageSize, $sortField, $sortOrder);
    }

    return $result;
}
```

Формуємо SQL запит та виконуємо його

```
public function getPage(array $filter = [], int $page = 1, int
$pageSize = 20, string $sortField = "id", string $sortOrder =
"asc") : array
{
    $query = $this->databaseConnection->createQueryBuilder();

    $query = $query->select($this->getSelectFields())-
>from($this->table, 't');

    $query = $this->getAttachJoins($query);

    $query = $this->applyFilters($query, $filter);



    $query = $query->select('count(*)');

    $itemCount = $query->executeQuery()->fetchOne();

    $query = $query->select($this->getSelectFields());
    $query = $query->orderBy('t.'.$sortField, $sortOrder);
    $query = $query->setFirstResult(($page - 1) * $pageSize)-
>setMaxResults($pageSize);

    $result = $query->executeQuery()->fetchAllAssociative();

    return ['data' => $result, 'itemsCount' => $itemCount];
```

```
}
```

Обробляємо критерії пошуку

```
protected function fillFilter(array $filter, array $input) :
array
{
    if (!empty($input['event_id'])) {
        $filter['event_id'] = (int)$input['event_id'];
    }

    if (!empty($input['withoutFile'])) {
        $filter['document_id'] = ['function' => "isNull", 'value'
=> !empty($input['withoutFile'])];
    }

    return $filter;
}
```

Додаємо до нього фільтри за полями

```
protected function applyFilters($query, array $filter)
{
    foreach ($filter as $index => $value)
    {
        if (is_string($value))
        {
            $query = $query->andWhere($index.' LIKE :'.$index)-
>setParameter($index, $value);
        }
        else if (is_array($value)) {
            if ($value['function'] == static::FUNCTION_IS_NULL)
            {
                $query = $query->andWhere($index . ' IS NULL')-
>orWhere($index. ' = 0');
            }
            if ($value['function'] ==
static::FUNCTION_MOREOREQUAL)
```

```
            {
                $query = $query->andWhere($index . ' >=
:'.$index)->setParameter($index, $value['value']);
            }
            if ($value['function'] ==
static::FUNCTION_LESSOREQUAL)
            {
                $query = $query->andWhere($index . ' <=
:'.$index)->setParameter($index, $value['value']);
            }
            if ($value['function'] == static::FUNCTION_BETWEEN)
            {
                $query = $query->andWhere($index . ' BETWEEN
:'.$index.'_start AND :'.$index.'_finish')
                    ->setParameter($index.'_start',
$value['value'][0])
                    ->setParameter($index.'_finish',
$value['value'][1]);
            }
        }
        else {
            $query = $query->andWhere($index.' = :'.$index)-
>setParameter($index, $value);
        }
    }

    return $query;
}
```

У результаті отримаємо SQL запит який обере усі невиконані умови

```
SELECT t.* FROM event_conditional t WHERE (document_id IS NULL)
OR (document_id = 0) ORDER BY t.id asc LIMIT 50
```

Система передбачає генерацію статистик. Статистики генеруються у вигляді csv файла та скачуються на ПК копистувача. Для їх перегляну користувач може використати такі програми як Microsoft Excel чи Open

Office. Генерація статистик доступна у пункті меню «Статистики» (див. рис 4.13).

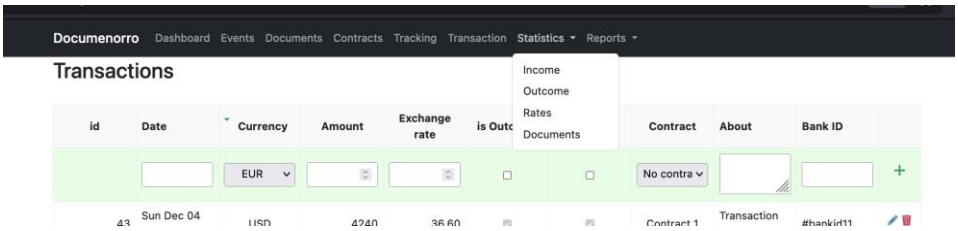


Рисунок 4.13 – Меню доступних статистик (рисунок виконано самостійно)

Перша статистика це розрахунок доходу за місяцями. Для генерації цієї статистики використовується SQL запит:

```
Select YEAR(FROM_UNIXTIME(date)) as year,
MONTH(FROM_UNIXTIME(date)) as month, c.title as currency,
count(*) as count, SUM(amount * IF(isOutcome = 1, -1, 1)) as
amount, SUM(amount * IF(isOutcome = 1, -1, 1) * exchange_rate) as
amount_local  from transaction
JOIN currency c on transaction.currency_id = c.id
group by YEAR(FROM_UNIXTIME(date)), MONTH(FROM_UNIXTIME(date)),
currency_id
ORDER BY YEAR(FROM_UNIXTIME(date)) DESC,
MONTH(FROM_UNIXTIME(date)) DESC
```

Як результат буде згенеровано наступну статистику (див. рис 4.14).

Друга статистика це статистика рейтів за періодами часу. Розраховується на основі трекінга часу за договорами (див. рис 4.15).

	A	B	C	D	E	F
1	Year	Month	Currency	Count	Amount	Amount in local currency
2	2023	11	USD	1	4240	155184.00
3	2023	10	EUR	1	1000	33600.00
4	2023	10	USD	2	-7840	-286944.00
5	2023	8	USD	1	3600	131760.00
6	2023	7	USD	1	-4320	-158112.00
7	2023	6	USD	1	-4400	-161040.00
8	2023	5	USD	1	-3840	-140544.00
9	2023	4	USD	1	3960	144936.00
10	2023	3	USD	1	3920	143472.00
11	2023	2	USD	1	3880	142008.00
12	2023	1	USD	1	3840	140544.00
13	2022	12	USD	1	-4240	-155184.00
14	2022	11	USD	1	-3720	-136152.00
15	2022	10	USD	1	3680	134688.00
16	2022	9	USD	1	-3640	-133224.00
17	2022	8	USD	1	-3960	-144936.00
18	2022	7	USD	1	3720	136152.00
19	2022	6	USD	1	-4080	-149328.00
20	2022	5	USD	1	3760	137616.00
21	2022	4	USD	1	-4040	-147864.00
22	2022	3	USD	1	-4320	-158112.00
23	2022	2	USD	1	4200	153720.00
24	2022	1	USD	1	-4000	-146400.00
25	2021	12	USD	1	-4320	-158112.00
26						
27						

Рисунок 4.14 – Приклад статистики надходжень (рисунок виконано самостійно)

	A	B	C	D	E
1	Year	Month	Currency	Min rate	Max rate
2	2023	11	EUR	25	25
3	2023	10	USD	15	15
4	2023	10	EUR	25	25
5	2023	10	USD	20	20
6	2023	9	USD	15	15
7	2023	9	EUR	25	25
8	2023	9	USD	20	20
9	2023	8	USD	15	15
10	2023	8	EUR	25	25
11	2023	8	USD	20	20
12	2023	7	USD	15	15
13	2023	7	EUR	25	25
14	2023	7	USD	20	20
15	2023	6	USD	15	15
16	2023	6	EUR	25	25

Рисунок 4.15 – Приклад статистики рейтів за договорами (рисунок виконано самостійно)

Наступним важливим елементом є генерація звітів. Перший це генератор інвойсів. Для його створення потрібно обрати договір, період часу за який ми його розраховуємо та відправити на сервер. Результат генерації можна побачити на рисунку 4.15.

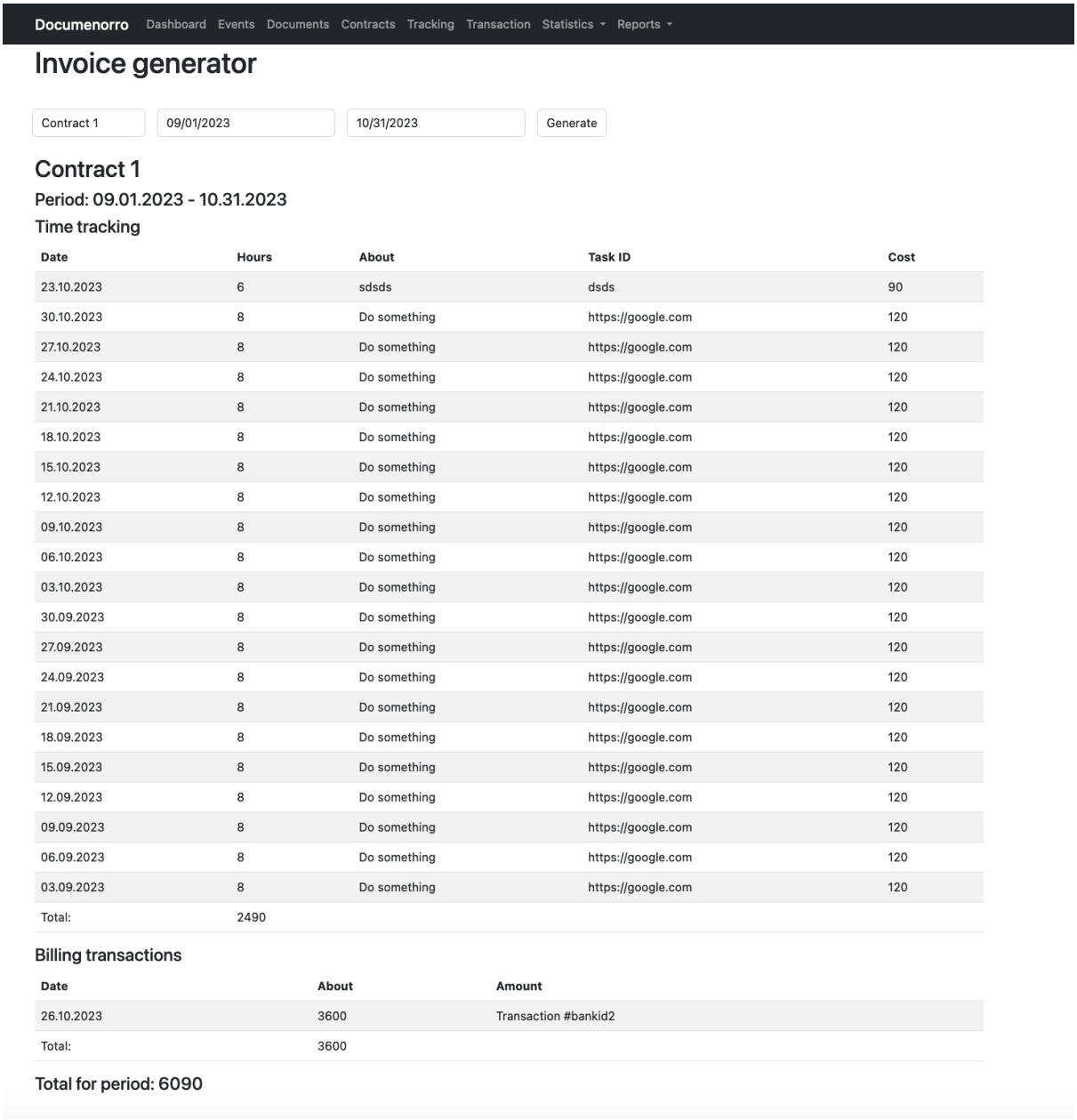


Рисунок 4.16 – Результат генерації інвойса (рисунок виконано самостійно)

У звіті рахується час який був використаний для работи на цьому проекті на у окремій секції додаються вихідгні транзакції які мають бути додані до клієнтського рахунку.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Обґрунтування вибору виду тестування

Тестування є невід'ємною частиною процесу розробки веб-застосунків, оскільки воно забезпечує виявлення помилок та недоліків у програмному забезпеченні, покращує якість коду та гарантує, що всі функціональні та нефункціональні вимоги виконуються належним чином. Вибір виду тестування залежить від специфіки проекту, вимог замовника та етапу розробки. Для нашого веб-застосунку було обрано декілька основних видів тестування, кожен з яких виконує свою важливу роль у забезпеченні якості продукту.

Модульне тестування є базовим видом тестування, який виконується на рівні окремих модулів або функцій застосунку. Мета модульного тестування полягає у верифікації коректності роботи кожного окремого компонента в ізоляції від інших. Це дозволяє виявляти та виправляти помилки на ранніх етапах розробки, що значно знижує витрати на їх виправлення. Для модульного тестування у проекті використовуються такі інструменти, як PHPUnit.

Прикладом реалізації одного з тест-кейсів метода який реалізує сортування подій на очікувані и просрочені є:

```
class AlertServiceTest extends TestCase
{
    protected $alertService = null;
    protected function setUp(): void
    {
        $this->alertService = new \Lab\Service\AlertService();
    }

    public function testConditionSorting(): void
    {
```

```
        $correctResult = '{"failed":[{"title":"Event
1","started_at":1696118400,"finished_at":1696896000,"document_typ
e":"DT1","event":"Event 1 "},{"title":"Event
1","started_at":1696118400,"finished_at":1696896000,"document_typ
e":"DT2","event":"Event 1"}],"waited":[{"title":"Event
3","started_at":1696118400,"finished_at":1701302400,"document_typ
e":"DT3","event":"Event 3"}]}';

        $data = '[{"title":"Event
1","started_at":1696118400,"finished_at":1696896000,"document_typ
e":"DT1","event":"Event 1 "},{"title":"Event
1","started_at":1696118400,"finished_at":1696896000,"document_typ
e":"DT2","event":"Event 1"},{"title":"Event
3","started_at":1696118400,"finished_at":1701302400,"document_typ
e":"DT3","event":"Event 3"}]';

        $input = json_decode($data, true);

        $currentDate = 1701302400;

        $result = $this->alertService->sortConditions($input,
$currentDate);

        $this->assertSame(json_encode($result), $correctResult);

    }

}
```

Для запуску тестів використовується консольна команда

```
./vendor/bin/phpunit --bootstrap vendor/autoload.php --testdox
tests
```

Функціональне тестування перевіряє відповідність системи її функціональним вимогам. Метою цього тестування є забезпечення того, що всі функції веб-застосунку працюють відповідно до специфікацій. Функціональне тестування зазвичай включає перевірку користувацьких сценаріїв та бізнес-логіки. Для автоматизації функціонального тестування можуть використовуватися такі інструменти, як Selenium.

У проекті для тестування API використовується інструмент Postman. Postman є потужним інструментом, який дозволяє розробникам створювати, тестувати та документувати API-запити у зручному графічному інтерфейсі. Він підтримує автоматизацію тестів, організацію тестових колекцій та інтеграцію з системами CI/CD. Використання Postman дозволяє швидко і ефективно перевіряти коректність роботи

API, знаходити помилки та забезпечувати відповідність специфікаціям, що значно підвищує загальну якість та надійність веб-застосунку.

Прикладом базового тестування API метода отримання списку валют у системі є скрипт який перевіряє доступніть виклику та формат отриманих даних.

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response is a json object", function () {
    pm.expect(pm.response.json()).to.be.an('object');
});

pm.test("Response to have 'data' object", function () {
    pm.expect(pm.response.json()).to.have.property('data');
});

pm.test("Response to have 'itemsCount' object", function () {
    pm.expect(pm.response.json()).to.have.property('itemsCount');
});
```

Для тестування додатку було створено ряд інтеграційних та модульних тестів, які допомогли знайти існуючі помилки, а також швидко отримувати попередження коли зявлялися нові.

ВИСНОВКИ

Проведена робота спрямована на вирішення проблем, пов'язаних з управлінням документами та інформацією для незалежних спеціалістів у глобалізованому світі. В результаті аналізу предметної галузі, ER-моделювання, концептуального та логічного моделювання реляційної бази даних, була створена система, яка може бути розгорнута на інфраструктурі приватного користувача без залежності від сторонніх сервісів.

Розроблена специфікація додатка дозволяє розробити програмне забезпечення що дає можливість суттєво зменшити обсяг додаткових зусиль, пов'язаних з веденням договорів та банківських транзакцій, відстеженням відпрацьованого часу, створенням рахунків та контролем за відповідністю вимог до поданих документів державним органам. Система передбачає централізоване зберігання та зручний пошук документів, що особливо важливо на дистанції у декілька років.

Наступним етапом було проєктування та розробка архітектури програмної системи, фізичної структури бази даних, алгоритмів автотегування, сповіщення про очікувані дії, а також розрахунку статистик та звітів.

Програмна система була розроблена та протестована за допомогою мови програмування PHP та JavaScript. В якості СУБД була обрана MySQL.

Результатом роботи є досягнення поставленої задачі, розроблено та протестовано програмну систему.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Єдиний податок. Державна податкова служба України. URL: https://tax.gov.ua/baneryi/onlayn-navchannya/ediniy-podatok/fizichni-osobi (дата звернення 25.10.23)

2. Thomas Erl, Benjamin Carlyle, Cesare Pautasso, Raj Balasubramanian. 5.1 // SOA with REST. — Prentice Hall, 2013. — ISBN 978-0-13-701251-0.

3. Privat24 for business. URL: https://privatbank.ua/business (дата звернення 25.10.23)

4. Особисте хмарне сховище – Microsoft OneDrive. URL: https://www.microsoft.com/uk-ua/microsoft-365/onedrive/online-cloud-storage (дата звернення 25.10.23)

5. Google Drive. URL: https://www.google.com/intl/ru_uA/drive/#overview (дата звернення 25.10.23)

6. Zoho CRM. URL: https://www.zoho.com/ru/crm/lp/crm-software.html (дата звернення 25.10.23)

7. Paperless-ngx. URL: https://docs.paperless-ngx.com/ (дата звернення 25.10.23)

8. Salesflare. URL: https://salesflare.com/feature1 (дата звернення 25.10.23)

9. PHP documentation. URL: https://www.php.net/docs.php (дата звернення 25.10.23)

10. Oracle MySQL documentation. URL: https://docs.oracle.com/cd/E17952_01/index.html (дата звернення 27.10.23)

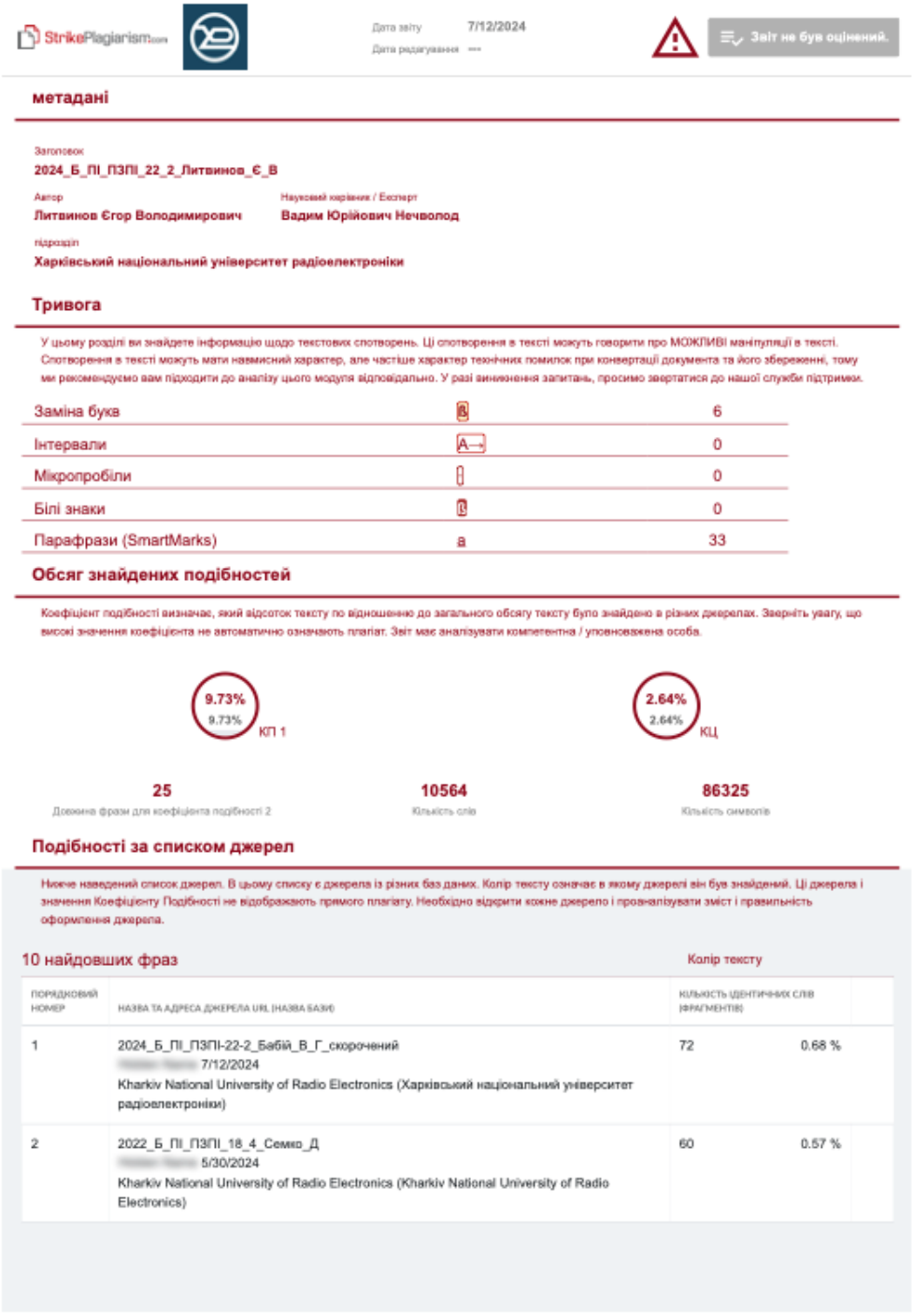
11. Doctrine DBAL documentation. URL: https://www.doctrine-project.org/projects/doctrine-dbal/en/3.5/reference/ (дата звернення 27.10.23)

12. OpenAPI Specification. URL: https://swagger.io/specification/ (дата звернення 27.10.23)

13. Swagger. URL: https://swagger.io/ (дата звернення 27.10.23)
14. Nginx. URL: https://nginx.org (дата звернення 27.10.23)

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



StrikePlagiarism.com

Дата звіту 7/12/2024
Дата редагування ---

Звіт не був оцінений.

метадані

Заголовок
2024_Б_ПІ_ПЗПІ_22_2_Литвинов_Є_В

Автор
Литвинов Єгор Володимирович

Науковий керівник / Експерт
Вадим Юрійович Нечволод

підрозділ
Харківський національний університет радіоелектроніки

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	B	6
Інтервали	A→	0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)	a	33

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

9.73% 9.73% КП 1

2.64% 2.64% КЦ

25	10564	86325
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	2024_Б_ПІ_ПЗПІ-22-2_Бабій_В_Г_скорочений [redacted] 7/12/2024 Kharkiv National University of Radio Electronics (Харківський національний університет радіоелектроніки)	72	0.68 %
2	2022_Б_ПІ_ПЗПІ_18_4_Семко_Д [redacted] 5/30/2024 Kharkiv National University of Radio Electronics (Kharkiv National University of Radio Electronics)	60	0.57 %

ДОДАТОК Б

Слайди презентації

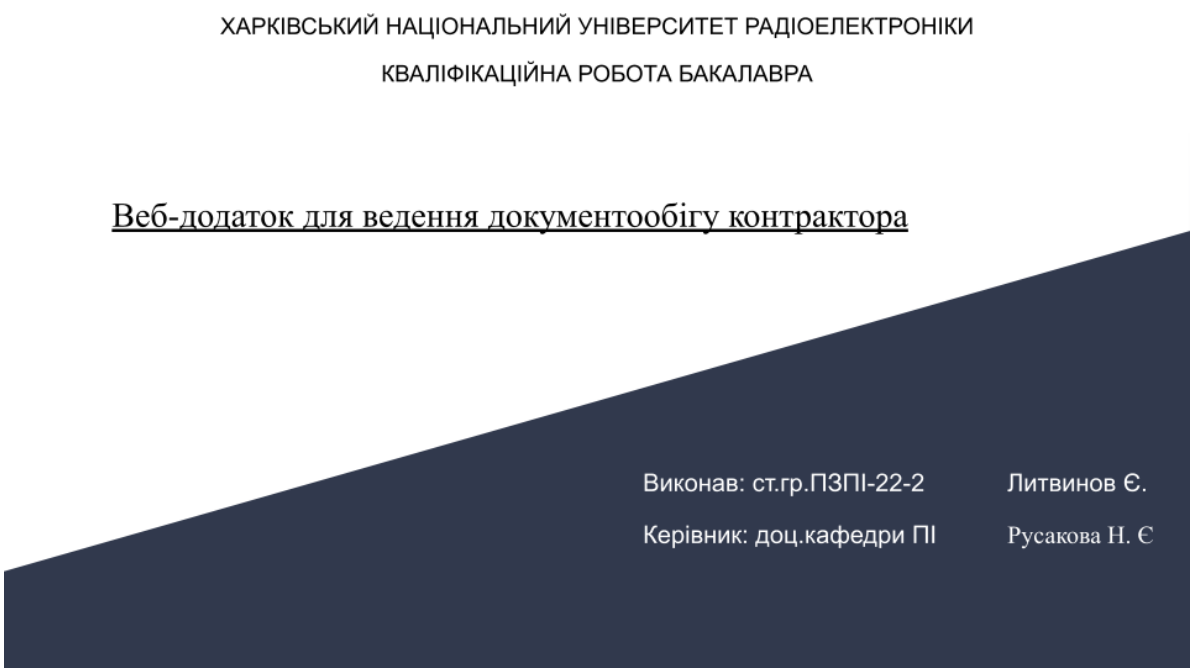


Рисунок Б.1 – Титульний слайд (рисунок виконано самостійно)

Рисунок Б.2 – Слайд «Актуальність розробки» (рисунок виконано самостійно)

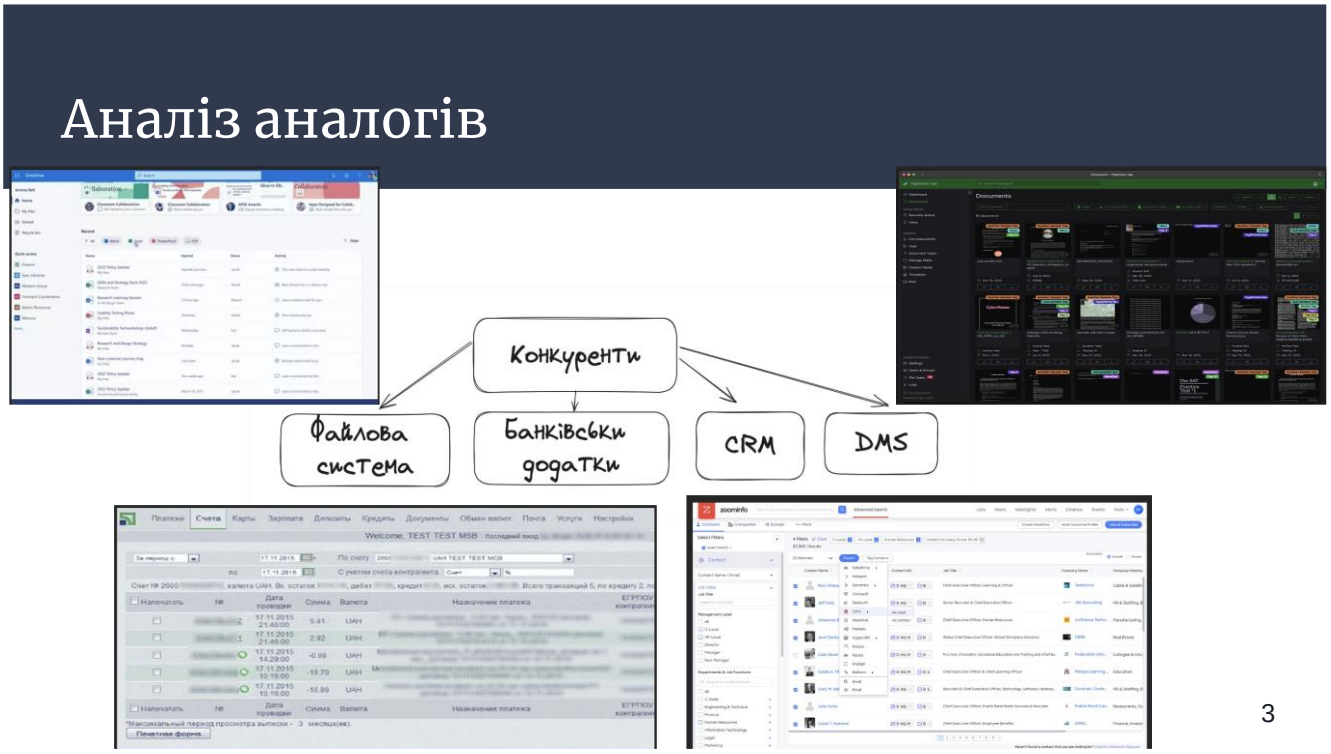


Рисунок Б.3 – Слайд «Аналіз аналогів» (рисунок виконано самостійно)

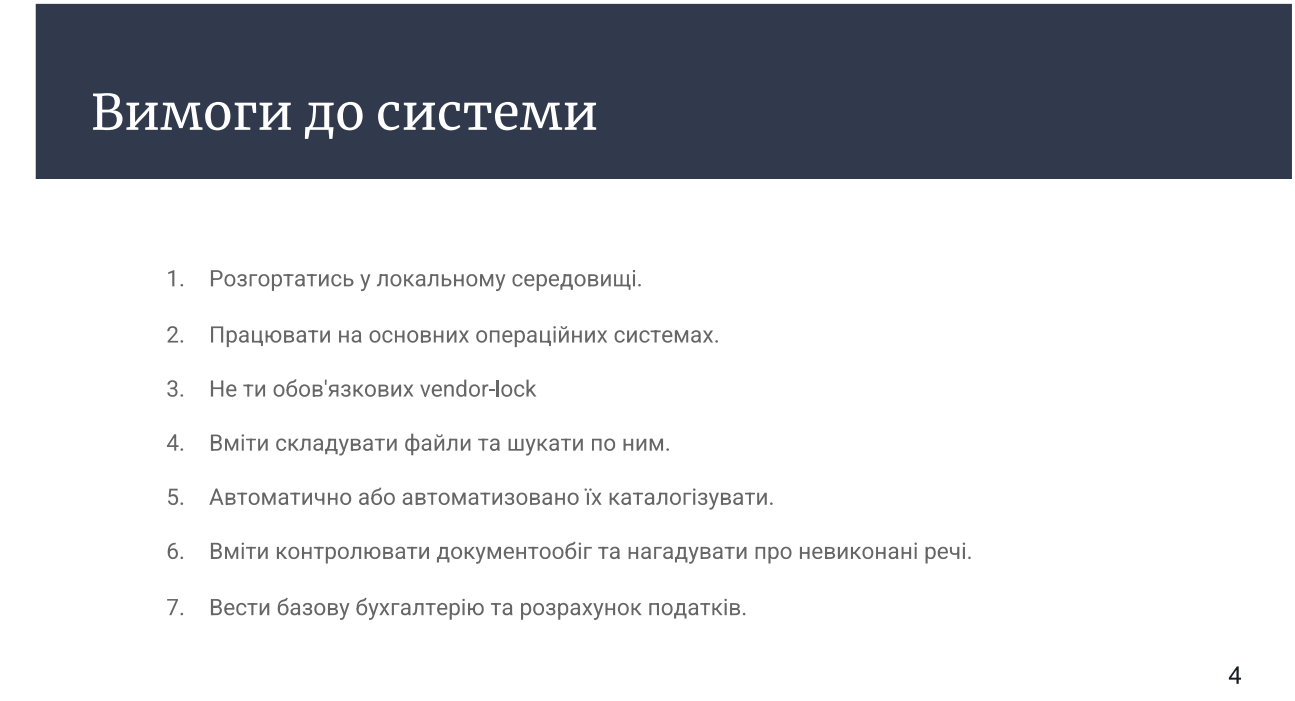


Рисунок Б.4 – Слайд «Вимоги до системи» (рисунок виконано самостійно)

Постановка задачі

1. Провести аналіз та моделювання предметної області програмної системи

2. Спроектувати базу даних для збереження інформації з предметної області

3. Розробити алгоритм для автоматизованої організації документів і контролю за дотриманням умов документообігу

4. Спроектувати архітектуру програмної системи;

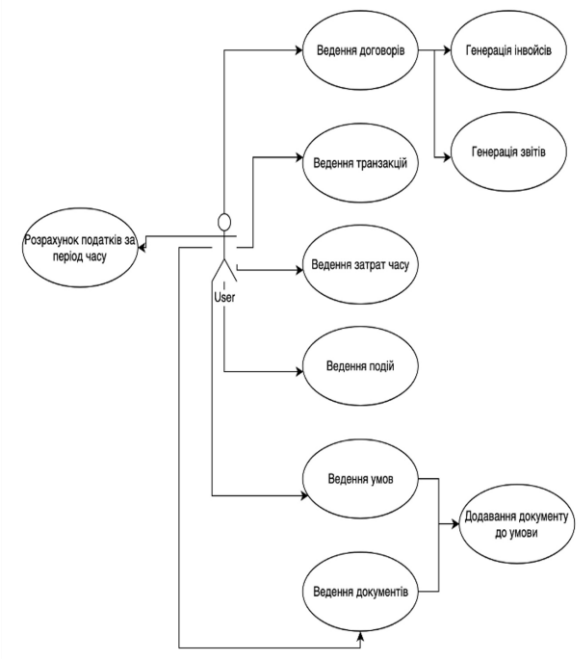
5. Виконати програмну реалізацію системи та провести тестування створеного програмного продукту.

5

Рисунок Б.5 – Слайд «Постановка задачі» (рисунок виконано самостійно)

Аналіз та моделювання предметної області

Загальна Use-Case діаграма



6

Рисунок Б.6 – Слайд «Аналіз та моделювання предметної області» (рисунок виконано самостійно)

Рисунок Б.7 – Слайд «Концептуальне моделювання» (рисунок виконано самостійно)

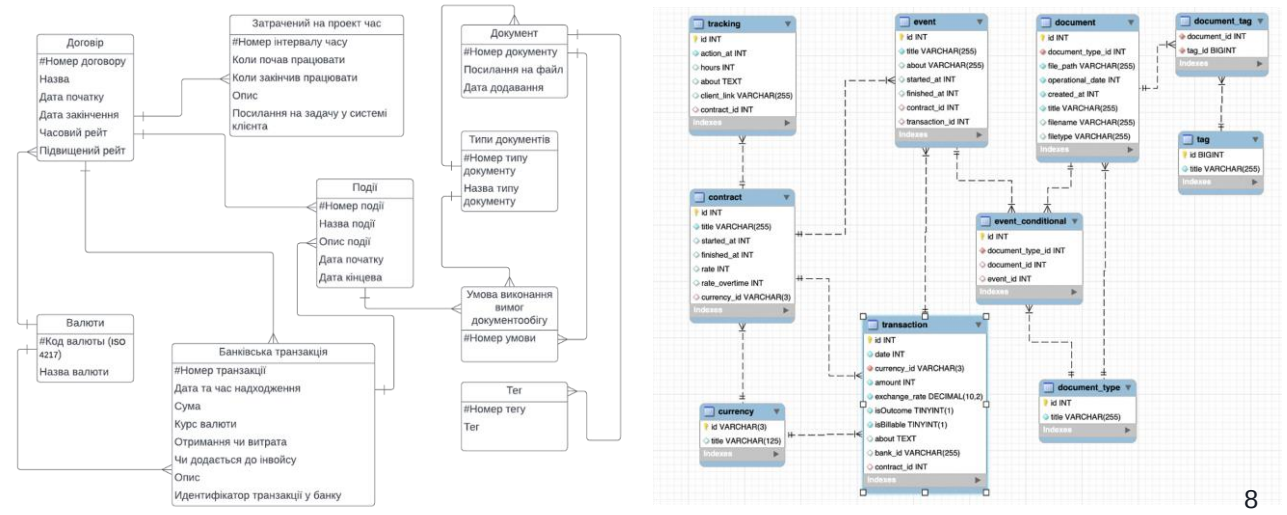
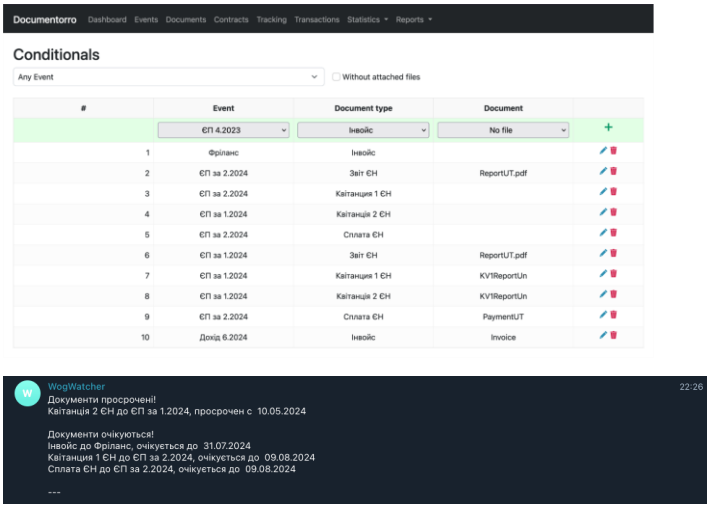


Рисунок Б.8 – Слайд «Проектування бази даних» (рисунок виконано самостійно)

Алгоритм автоматизації контролю документообігу



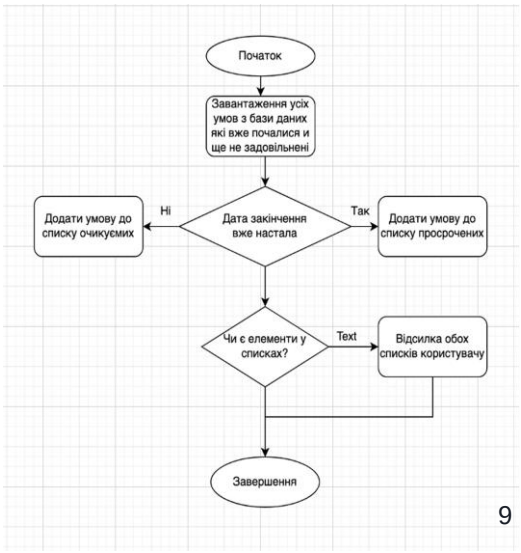


Рисунок Б.9 – Слайд «Алгоритм автоматизації контролю документообігу» (рисунок виконано самостійно)

Алгоритм автотегування документів

Отриманий результат:

"Квитанція | Оплата | ХНУРЕ | 2 семестр | Іванов Іван Петрович | Післядипломна освіта | ПриватБанк | 2023 | Вища освіта | Платіжна інструкція"

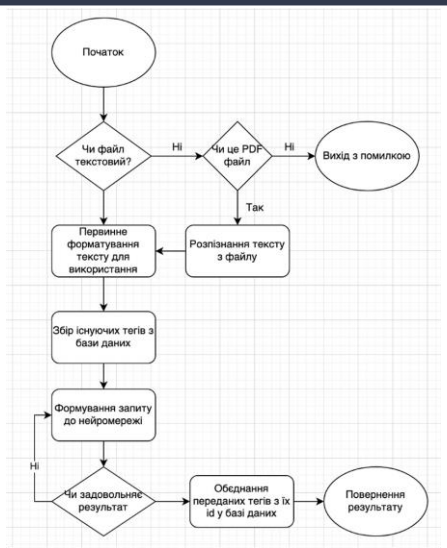
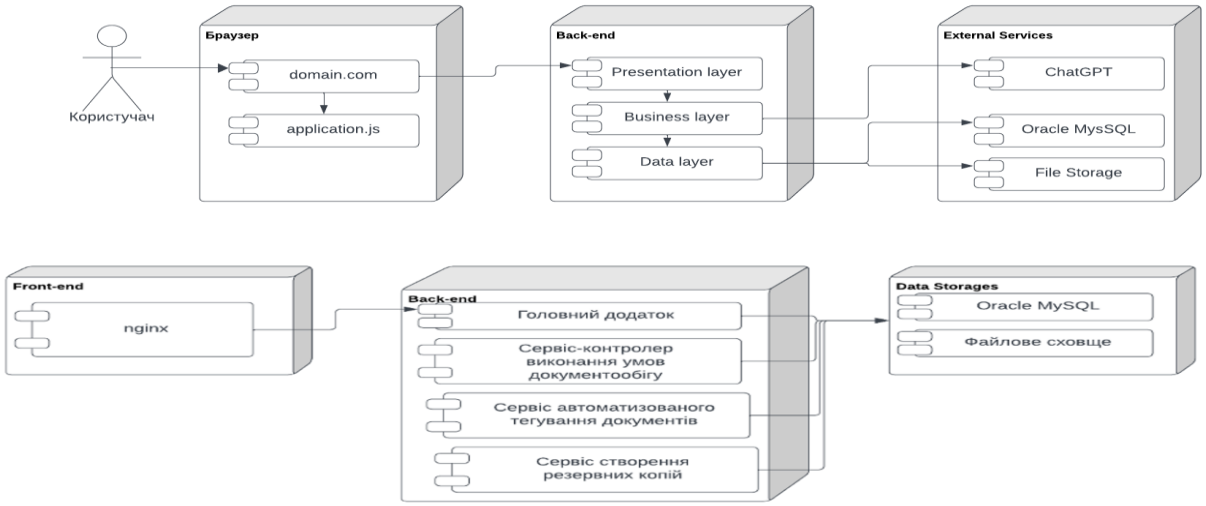


Рисунок Б.10 – Слайд «Алгоритм автотегування документів» (рисунок виконано самостійно)

Проектування архітектури



11

Рисунок Б.11 – Слайд «Проектування архітектури» (рисунок виконано самостійно)

Технології та мови програмування

Тип додатку	Веб-додаток
Environment	Docker
Back-end	PHP 8, Symfony components
Front-end	Javascript
Database	Oracle MySQL
LLM	ChatGPT 4o
OCR	pdftotext

12

Рисунок Б.12 – Слайд «Технології та мови програмування» (рисунок виконано самостійно)

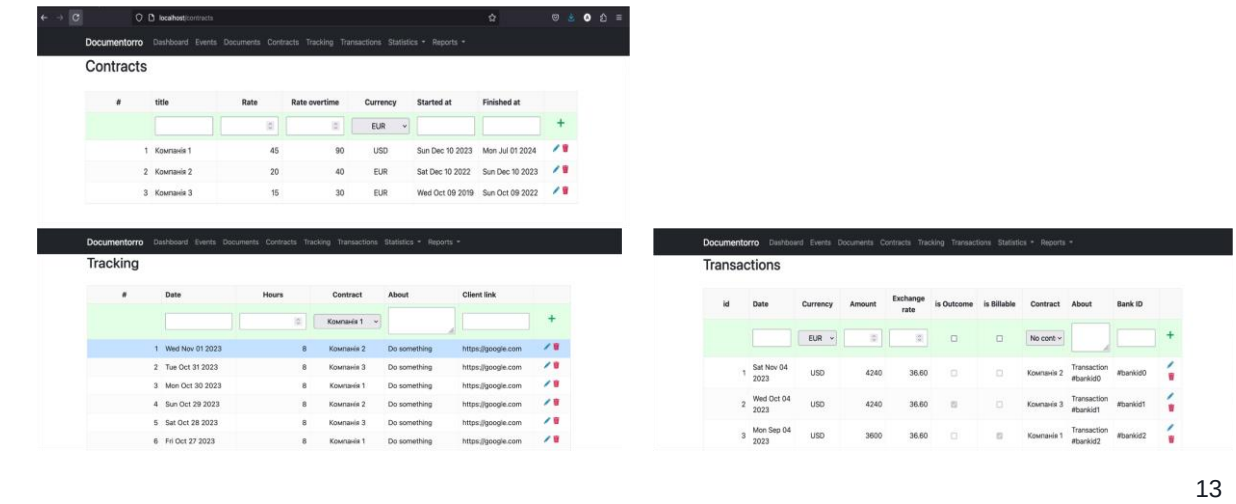


Рисунок Б.13 – Слайд «UI: Договори, трекінг часу та транзакції» (рисунок виконано самостійно)

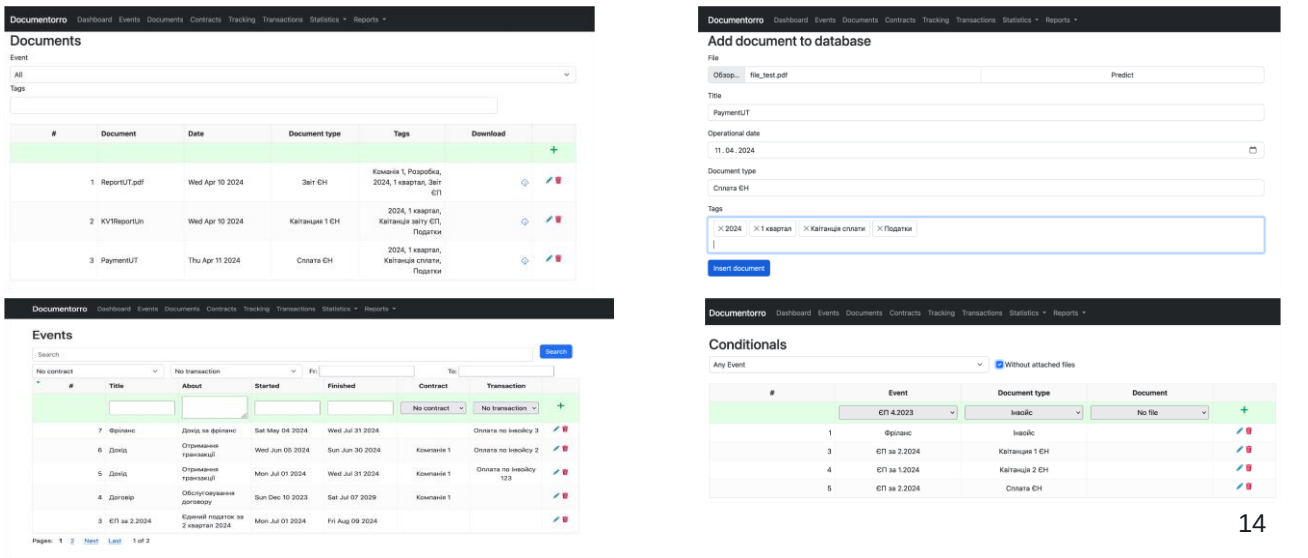
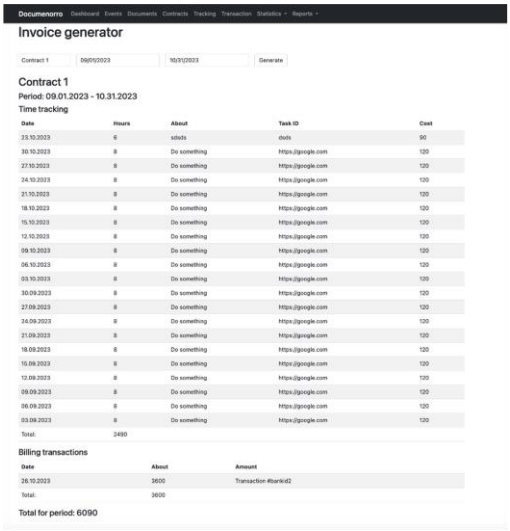
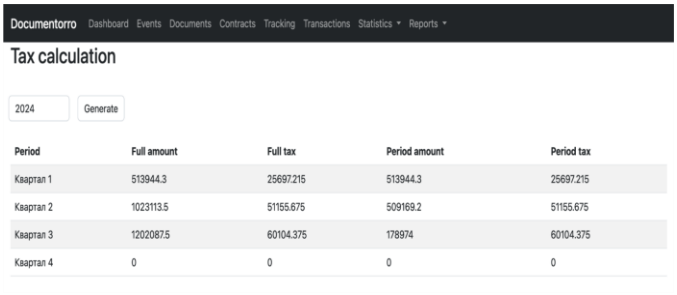


Рисунок Б.14 – Слайд «UI: Події та умови виконання подій» (рисунок виконано самостійно)

UI: Звіти





15

Рисунок Б.15 – Слайд «UI: Звіти» (рисунок виконано самостійно)

Тестування програми

Методика тестування			
Налаштування	Не проводиться		N/A *
Крок	**Дія**	**Очікуваний результат**	**Відмітка (V)***
1.	Запустити функцію.	Ніяких повідомлень не приходить	(V)
2.	Створити запис Умова 3 - дата активації менша за сьогодні, дата закінчення більше за сьогодні, не виконана.	Приходить повідомлення «Умова 3 чекає на виконання»	(V)
3.	Умова 4 - дата активації менша за сьогодні, дата закінчення менша за сьогодні, не виконана	Приходить повідомлення «Умова 3 чекає на виконання. Умова 4 просрочена»	(V)

16

Рисунок Б.16 – Слайд «Тестування програми» (рисунок виконано самостійно)

1. Проведено аналіз та моделювання предметної області програмної системи
2. Спроектовано базу даних для збереження інформації з предметної області
3. Розроблено алгоритм для автоматизованої організації документів і контролю за дотриманням умов документообігу
4. Спроектовано архітектуру програмної системи
5. Виконано програмну реалізацію системи та проведено тестування створеного програмного продукту

17

Рисунок Б.17 – Слайд «Висновки» (рисунок виконано самостійно)