МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Інститут післядипломної освіти

Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Розробка інформаційної системи розпізнавання платіжних документів

Виконав студент групи КН-5
спеціальності 122 «Комп'ютерні науки»
Халілі Джаруллах Халіл огли

Керівник к.геогр.н., доцент
Кузніченко Світлана Дмитрівна

Консультант

Рецензент к.техн.н., доцент
Гнатовська Ганна Арнольдівна

Одеса 2023

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

ОС – операційна система

ПЗ – програмне забезпечення

ADT – Android Development Tools – Інструменти розробки в ОС Android

API – Application Programming Interface – програмний інтерфейс програми

AR – Augmented Reality – доповнена реальність

IDE – Integrated Development Environment – інтегроване середовище розробки

SDK – Software Development Kit – набір засобів розробки

ВСТУП

Розвиток сучасних комп'ютерних технологій призвів до використання електронного документообігу майже у всіх сфера людського життя. Але платіжні та фінансові документи у паперовому вигляді й досі залишаються обов'язковими за різних умов. Перш за все цього вимагає чинне законодавство. Звичайно, вже існують проекти про відсилання чеку за фінансову операцію у кабінеті користувача-клієнта банку, якщо операція була здійснена за допомогою картки. Тим не менш, паперові фінансові документи, будуть залишатися актуальними, ще значний час.

Тому актуальним є створення інформаційної системи, яка дозволить в автоматичному режимі, сканувати та аналізувати значну кількість типів платіжних документів, задля застосування інтелектуального аналізу та електронного документообігу. Подібна система може мати додаткові функції по систематизації та каталогізації подібних документів з метою упорядкування роботи з ними.

Метою кваліфікаційної роботи є розробка інформаційної системи для розпізнавання та аналізу платіжних документів.

Система зберігатиме документи: дозволятиме виконувати порівняння витрат за вибраний період часу та в обраній категорії, що значно спростить та поліпшить контроль фінансів.

Для досягнення мети необхідно виконати наступні задачі:

- аналіз аналогів та пошук функціоналу, що доцільно запозичити;
- вибір засобів та технологій розробки програмного забезпечення;
- проектування інформаційної системи розпізнання платіжних документів;
- реалізація програмної інформаційної системи;
- тестування програмного забезпечення.

Структура кваліфікаційної роботи складається вступу, трьох розділів, висновків, переліку посилань на 18 найменувань, додатків. Повний обсяг роботи становить 73 сторінки, містить 28 рисунків і 2 таблиці.

1 ОГЛЯД ІСНУЮЧИХ ІНФОРМАЦІЙНИХ СИСТЕМ ДЛЯ РОЗПІЗНАВАННЯ ПЛАТІЖНИХ ДОКУМЕНТІВ

1.1 Порівняння систем аналогів

Перед початком роботи над кваліфікаційною роботою бакалавра необхідно провести дослідження предметної області та знайти існуючі аналоги створюваної системи. Дослідження дозволить зрозуміти потреби, що існують на сучасному ринку, та зрозуміти як створити конкурентоспроможній продукт.

В результаті пошуку аналогів було знайдено три системи:

– Abbyy FlexiCapture;
– Regula Forensic Studio;
– Smart ReTypeDoc.

1.1.1 Система ABBYY FlexiCapture

ABBYY FlexiCapture – рішення для потокового введення даних і документів [1].

Продукт автоматизує вилучення інформації з паперових документів і зберігає дані в інформаційній системі підприємства. ABBYY FlexiCapture дозволяє різним організаціям, в тому числі великим корпораціям, урядовим структурам і освітнім установам, автоматизувати процес введення даних в інформаційні системи, знизити витрати і підвищити якість обслуговування клієнтів. Программа являє собою універсальнк платформ для інтелектуальної обробки інформації з будь-яких типів документів: відсканованих паперів, фотографій, електронних документів, текстів листів і вкладень. Рішення розпізнає, класифікує документи, отримує дані, перевіряє їх коректність і передає в корпоративні інформаційні системи. Застосування ABBYY FlexiCapture істотно оптимізує всі бізнес-процеси, пов'язані з документообігом.

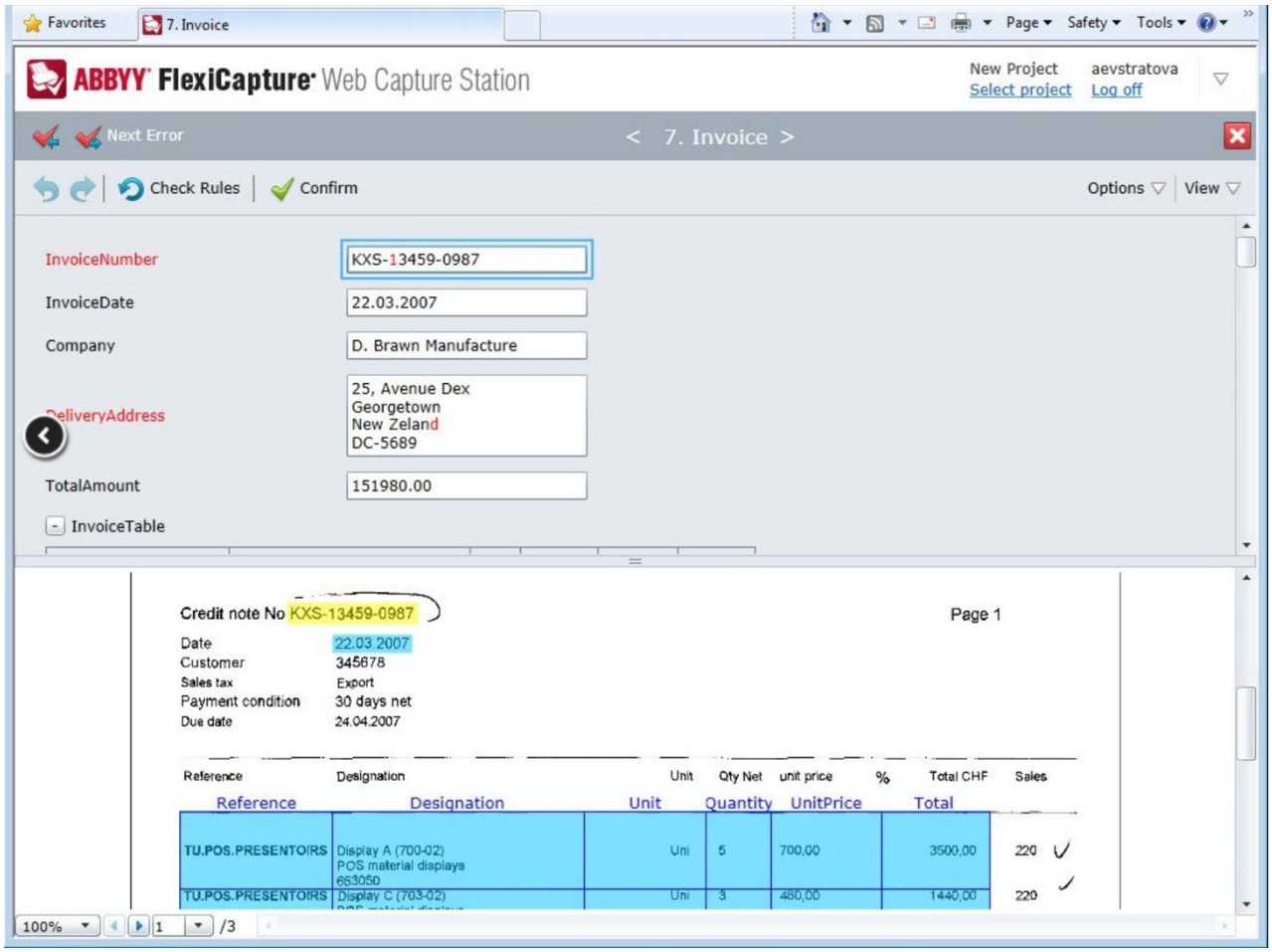

Рисунок 1.1 – Інтерфейс Abbyy FlexiCapture [2]

Платформа має широкі можливості, являє собою універсальний засіб: котрий легко розширюється та масштабується, з огляду на потреби користувача.

1.1.2 Regula Forensic Studio

Оперативний контроль справжності та експертне дослідження паспортів, ідентифікаційних карт і інших документів, що засвідчують особу та надають право на перетин кордону; візових марок та відбитків печатки, в тому числі для дозволу на в'їзд; банкнот; водійських посвідчень, сертифікатів на транспортні засоби, інших документів, пов'язаних з автотранспортом; підписів і коротких рукописних записів; акцизних і спеціальних марок;

цінних паперів та інших документів із засобами захисту від підробки [3].

Рисунок 1.2 – Інтерфейс Regula Forensic Studio

До функціональних можливостей продукту належать: управління зовнішніми пристроями «Регула», отримання, обробка та аналіз даних. Програма підтримує російську, англійську, українську, китайську та польську мови.

1.1.3 Smart ReTypeDoc

Smart ReTypeDoc – вбудована технологія, яка дає можливість визначення типу документів (вибору одного з відомих типів документів) в відеопослідовність або на окремих зображеннях [4].

Дозволяє виконувати обробку довільного числа вхідних директорій з даними (об'єкти класифікації: окремі сторінки і послідовності сторінок багатосторінкових документів).

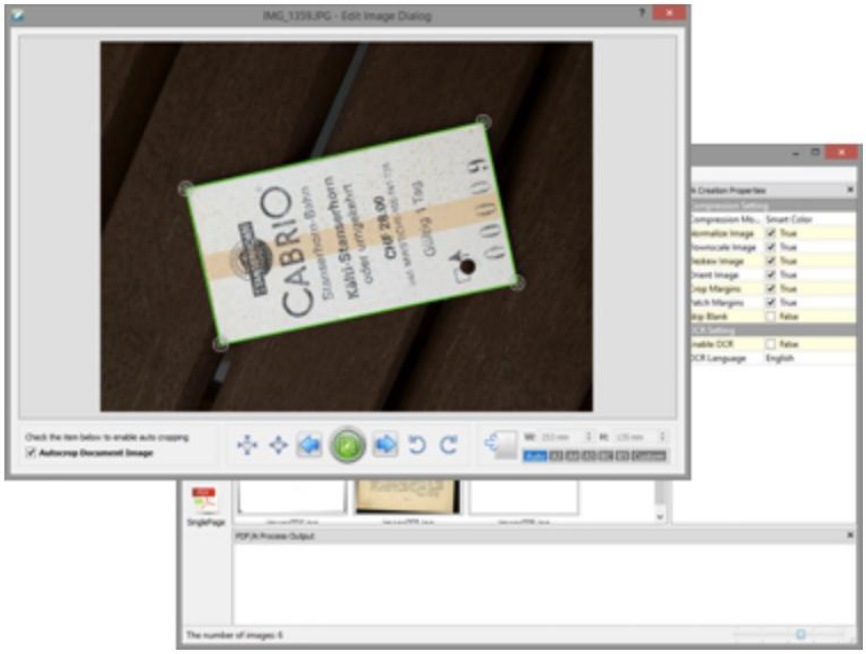

Рисунок 1.3 – Інтерфейс Smart ReTypeDoc

Виконує класифікацію за графічним ознаками для визначення окремих типів документів (що засвідчують особу і ін.), Пошуку логотипу, пошуку печаток, перевірки наявності підпису.

1.1.4 Результаті порівнянь

В результаті проведених досліджень та аналізу декількох аналогів можна прийти до висновку, що всі представлені аналогічні варіанти достатньо відрізняються один від одного.

Перша система (Abbyy FlexiCapture) являє собою веб-додаток, котрий дозволяє користувачеві сканувати та визначати тип сканованого документу [2]. Виконує розпізнавання типів документів, також може розподіляти документи за групами. Головною специфікацією є розпізнавання особистих документів особи. Можливості програми можуть бути розширені за. потребами користувача.

Друга система (Regula Forensic Studio) відповідає за перевірку від сканованих документів на дійсність [3]. Являє собою десктопний додаток: котрий може взаємодіяти з пристроями, що виготовляє компанія.

Третя система (Smart ReTypeDoc) являє собою вбудовану програму: котра виконує пошук документів у відео потоці, але не розпізнає тип документа [4].

Для більш наглядного порівняння використаємо таблицю в якій відзначимо основні схожості та розбіжності систем.

Таблиця 1.1 – Результат порівнянь аналогів

Ознаки	Назва системи			
	Abbyy FlexiCapture	Regula Forensic Studio	Smart ReTypeDoc	Власна система
Можливість збереження даних в системі	+	-	+	+
Можливість завантаження документів на зовнішній носій	+	-	-	+
Розпізнавання реквізитів платіжних документів	+	-	-	+
Групування за типом платіжного документа	-	-	-	+
Багатоплатформовий	+	-	-	+
Вільний доступ	-	-	-	+

Таким чином, створювана система містить в собі ідеї кожної з аналогічних, об’єднуючи в собі відкритість до покращення і додавання більш широкого функціоналу, але на відміну від існуючих аналогів є некомерційною та конкретно спрямованою на розпізнавання реквізитів платіжних документів, що робить її унікальною в своєму роді.

Створювана система відповідатиме потребам широкого спектру користувачів, котрі бажають стежити за витратами фінансів.

1.2 Вибір технологій для розробки системи

1.2.1 Вибір мови програмування

JavaScript – це мова програмування, що дозволяє зробити Web - сторінку інтерактивною, тобто такою що реагує на дії користувача [5].

JavaScript – об'єктно-орієнтована скриптова мова програмування і є діалектом мови ECMAScript. JavaScript зазвичай використовується як вбудована мова для програмного доступу до об'єктів додатків. Найбільш широке застосування знаходить у браузерах як мова сценаріїв для надання інтерактивності веб-сторінкам.

Основні архітектурні риси:

- динамічна типізація;
- автоматичне керування пам'яттю;
- прототипне програмування;
- функції як об'єкти першого класу.

JavaScript має низку властивостей об'єктно-орієнтованої мови, але реалізоване в мові прототипування обумовлює відмінності в роботі з об'єктами в порівнянні з традиційними об'єктно-орієнтованими мовами. Крім того, JavaScript має ряд властивостей, властивих функціональним мовам, - функції як об'єкти першого класу, об'єкти як списки, каррінг, анонімні функції, замикання – що додає мові додаткову гнучкість [6].

Синтаксис мови JavaScript дуже нагадує синтаксис C і Java, семантично ж мова набагато ближче до Self, Smalltalk або навіть Ліспу.

В JavaScript: всі ідентифікатори реєстрозалежні, в назвах змінних можна використовувати літери, підкреслення, символ долара, арабські цифри, назви змінних не можуть починатися з цифри, для оформлення однорядкових коментарів використовуються / /, багаторядкові і внутрішньорядкові коментарі починаються з / * і закінчуються * /.

Структурно JavaScript можна представити у вигляді об'єднання трьох частин, що чітко різняться одна від одної [7]: ядро (ECMAScript; об'єктна модель браузера (Browser Object Model або BOM); об'єктна модель документа (Document Object Model або DOM).

Об'єктну модель документа іноді розглядають як окрему від JavaScript сутність, що узгоджується з визначенням DOM як незалежного від мови інтерфейсу документа [5].

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією.

Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості.

Зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу. Мова JavaScript використовується для:

– написання сценаріїв веб-сторінок для надання їм інтерактивності;

– програмування на стороні сервера (Node.js); стаціонарних застосунків (Electron, NW.js);

– мобільних застосунків (React Native, Cordova);

– сценаріїв в прикладному ПЗ (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter);

– всередині PDF-документів тощо [8].

1.2.2 Вибір та обґрунтування front та backend технологій

React, або React.js, ReactJS – це відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка часткового оновлює вміст веб-сторінки, при розробці односторінкових застосунків [9]. React дозволяє розробникам створювати великі веб-додатки, які використовують дані, що змінюються з часом, без перезавантаження сторінки. Вимогами є: швидкість, простота та масштабованим. React обробляє тільки користувацький інтерфейс у додатках, що відповідає шаблону модель-вид-контролер (MVC), та використовується у поєднанні з іншими JavaScript бібліотеками або з великими фреймворками MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудів, щоб забезпечити веб-додатки, які не мають користувацького інтерфейсу. Бібліотеку React як інтерфейс користувача найчастіше використовують разом з іншими бібліотеками, такими як Redux.

Node.js –платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript [10].

Засновником платформи є Раян Дал (Ryan Dahl). Якщо раніше Javascript застосовувався для обробки даних в браузері користувача, то node.js надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їх виконання.

Платформа Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників.

Node.js має наступні властивості [10]:

– асинхронна одно-нитева модель виконання запитів;
– неблокуючий ввід/вивід;
– система модулів CommonJS;
– рушій JavaScript Google V8;

Для керування модулями використовується пакетний менеджер npm (node package manager).

Як платформенна реалізація Node.js заснована на двигуні V8, що трансляє JavaScript в машинний код. Це перетворює JavaScript з вузькоспеціалізованого мови в мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями введення-виведення через свій API, написаний на C ++, підключати інші зовнішні бібліотеки, які можуть бути написані на різних мовах [11]. Node.js застосовується переважно на сервері, виконуючи роль веб-сервера, але є можливість розробляти на Node.js і десктопні віконні додатки через NW.js, AppJS або Electron для Linux, Windows і macOS.

Крім того є можливість програмувати мікроконтролери, такі як наприклад, tessel і espruino. В основі Node.js лежить подієво-орієнтоване і асинхронне, чи реактивне програмування з неблокуючим введенням та виведенням.

1.2.3 Вибір та обґрунтування бази даних

Існують два найбільш поширених типу систем управління даними: реляційні БД (РБД) [12, 13] та NoSQL БД, різні в багатьох аспектах роботи.

В останні роки було розроблено велику кількість нових БД, які надають гарне горизонтальне масштабування для простих операцій читання, запису для БД, розподілених на безлічі серверів, орієнтованих на Big Data [14]. На відміну від них, традиційні БД дають менше можливостей до масштабування.

Безліч нових БД визначаються терміном «NoSQL ». Термін «NoSQL», що розшифровується як «Not Only SQL» або «Not Relational». Зазвичай такі системи задовольняють такими ознаками: наявність засобів розподілу навантаження на безліч серверів; можливість розподілу даних на безліч серверів; простий протокол виклику операцій; більш слабка модель паралелізму, ніж ACID-транзакції; ефективне використання розподілених індексів й оперативної пам'яті для зберігання даних; відсутність фіксованої схеми даних.

NoSQL-системи зазвичай не задовольняють властивостям ACID-транзакцій: допускається узгодженість в кінцевому рахунку. Пропонується модель «BASE» (Basically Available, Soft state, Eventually consistent, узгодженість в кінцевому рахунку) в протилежність ACID.

Ідея полягає в тому, що, відмовившись від обмежень ACID, можна домогтися набагато кращої продуктивності й масштабованості. Більшість систем різняться в ступенях відмови від ACID. Прихильники NoSQL часто посилаються на CAP-теорему, яка стверджує, що система може задовольняти лише двом з трьох таких властивостей: узгодженість - дані завжди однакові для всіх реплік; доступність - дані завжди доступні користувачеві; стійкість до поділу - система БД продовжує коректну роботу не дивлячись на відмову мережі або вузлів.

У NoSQL системах зазвичай опускається узгодженість, але припущення можуть бути складніше. Найчастіше моделі даних в NoSQL-системах розбиваються на наступні категорії [15–18]: сховища типу «ключ-значення» (Key-value stores): зберігають значення й ідентифікатор для пошуку, заснований на заданому ключі; документно-орієнтовані сховища (Document stores): система зберігає дані в формі документів, які індексуються; сховища, що розширюються записом (Extensible records stores): записи в таких сховищах можуть бути розподілені вертикально та горизонтально по вузлах.

Для середнього обсягу даних РБД та NoSQL дають схожі часові відгуки. Розбіжності проявляються при аналізі основних операцій.

Операції запису, оновлення, видалення даних потребують найбільшого часу на виконання для всіх типів БД. Найбільший час виконання операції запису, оновлення записів у DynamoDB при всіх обсягах даних. При середньому обсязі даних PostgreSQL та MySQL показує найкращий час виконання, ніж Cassandra, MongoDB. На великому обсязі даних швидше працює MongoDB.

Операції читання записів в БД PostgreSQL та MySQL виконуються швидше ніж NoSQL системах для середнього обсягу даних. DynamoDB знач-

но програє в продуктивності ніж Cassandra, MongoDB та показує рівний результат при великому обсязі даних. Cassandra, MongoDB працюють значно шивдше при великому обсязі даних для простих операцій читання.

Операція поновлення записів. Згідно результатів тестування продуктивність БД DynamoDB нижче інших систем. Найкращий показник продуктивності має БД MongoDB та Cassandra для великого обсягу даних. Операції Update в БД PostgreSQL та MySQL виконуються швидше ніж в NoSQL системах для середнього обсягу даних.

Операції підрахунку з угрупованням. У зв'язку з тим, що дана операція виконується в PostgreSQL, MySQL на стороні сервера, а для NoSQL вона реалізована на стороні користувача. PostgreSQL демонструє кращу продуктивність, ніж NoSQL системи. В свою чергу, продуктивність Cassandra, MongoDB вище, ніж продуктивність DynamoDB.

Таким чином, у першому розділі було розглянуто три системи, що мають схожий функціонал. За результатами проведеного аналізу аналогів, на підставі їх переваг і недоліків було розроблено бачення нової системи, що дозволяє забезпечити користувачів надійним, швидким та зручним інструментом, щодо розпізнавання документів різних типів.

Система являтиме собою веб-додаток, який дозволить користувачеві визначати тип платіжного документу, визначати його атрибути та групувати документи за типом та призначенням.

Сформоване бачення системи, дозволило визначитися із технологіями, що будуть використані при розробці. За застосування вирішено використовувати мову програмування JavaScript, для frontend частини буде використана технологія ReactJs, для частини backend – Node.Js. Для забезпечення коректної та швидкої роботи в системі буде реалізована база даних MongoDB.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Мета та задачі інформаційної системи

Інформаційна система призначена для роботи з платіжними документами, а саме розпізнавання платіжних документів стандартного переліку за їх атрибутами для спрощення взаємодії людини з такими документами та їх подальшої автоматизованої обробки. Система дозволить перетворювати паперові чеки та квитанції в електронні, що допоможе користувачеві систематизувати фінансовий облік.

Мета інформаційної системи – за зображенням паперового платіжного документу перетворити його в електронний з активними полями. Система має бути зручною для кожного користувача. Додаток зберігатиме документи користування в його особистому кабінеті, дозволятиме виконувати порівняння витрат за вибраний період часу та в обраній категорії, що значно спростить та поліпшить контроль його фінансів.

Цільова аудиторія: громадські організації, приватні та державні установи, приватні особи. Система дозволить зручно контролювати витрати. На основі електронних документів проводити аналіз витрат.

Продукт являє собою веб-додаток, в якому присутній особистий кабінет користувача.

Головними функціями веб-додатку є:

– розпізнавання фотокопії паперового документу;

– визначення типу платіжного документа, шляхом розпізнавання його реквізитів;

– аналіз зображення, пошук реквізитів по зображенню;

– направлення електронної версії платіжного документу до особистого кабінету користувача.

Інформаційні потоки системи зображено на рисунку 2.1.

Рисунок 2.1 – Діаграма інформаційних потоків

2.2 Типи користувачів

У веб-ресурсі заплановано два типи користувачів – це адміністратор та користувач ресурсу.

Користувач – може вибрати мову, переглянути свої платіжні документи, створювати / видаляти / переглядати групи документів, додати нові платіжні документи.

Адміністратор – може видалити користувача. Обслуговування сервісу, допомога з помилками.

2.3 Функціональні вимоги

1. Авторизація в додатку

F1. Функція прямої авторизації користувача

F1.1. Користувач при авторизації повинен ввести логін (номер телефону / email) і пароль. Логін не повинен повторяться, пароль повинен бути надійний. Авторизація виконується для зручності використання системи, дає

можливості створювати особисті профілі користувача і зберігати результати роботи з продуктом кожного користувача.

F1.2. У разі некоректного введення логіна і пароля система виведе повідомлення про помилку і підказку. «Невірний логін / пароль, перевірте коректність введення».

F1.3. Для коректності виконується перевірка, логін повинен бути введений за структурою: user@email.com/+380 XX XXX XX XX, пароль не повинен перевищувати 8 символів, припустимі символи латинського алфавіту і цифри.

F1.4. У разі успішної авторизації користувач потрапляє в свій «кабінет» (робоче вікно).

2. Робота в режимі авторизованого користувача.

F2. Додати новий документ.

F2.1. Вибір опції додавання документа.

F2.2 Завантаження файлу, відповідних вимог (розмір, формат).

F2.3 Розпізнавання документу та збереження результатів.

F2.3.1 Розпізнавання документу.

F2.3.2 Збереження у особистому кабінеті.

F2.3.3 Можливість вивантажити інформацію в спеціальному файлі.

F2.3.4 Можливість відправити інформацію з використанням поштових клієнтів.

F2.3.5 При невдалому збереження користувачеві буде повідомлено про помилку.

3. Додаткові можливості в режимі авторизованого користувача.

F3. Створення груп документів.

F3.1. Виконання групування розпізнаних документів по типу, документи з певним типом знаходяться в окремій групі.

F3.2 Редагування документів.

F3.3. Виконання видалення документів або груп документів.

4. Робота в режимі адміністратора

F4. Видалення неактивних користувачів (користувач не відвідував особистий кабінет 6 місяців).

Наведені вимоги дозволяють побудувати діаграму прецедентів, яка подана на рисунку 2.2.

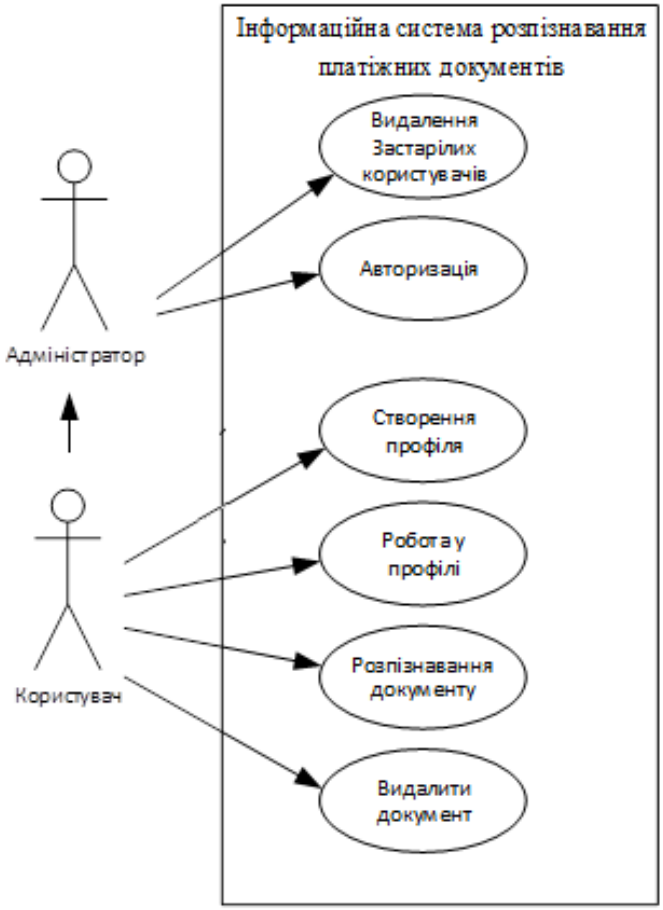

Рисунок 2.2 – Діаграма прецедентів

На вхід системи подається документ, який підлягає розпізнанню. Важливим етапом розпізнання є класифікація символів. Класифікація символів здійснюється за допомогою нейронного кола, кожен з блоків відповідає за певний етап розпізнання. Схема блоків розпізнавання подана на рисунку 2.3.

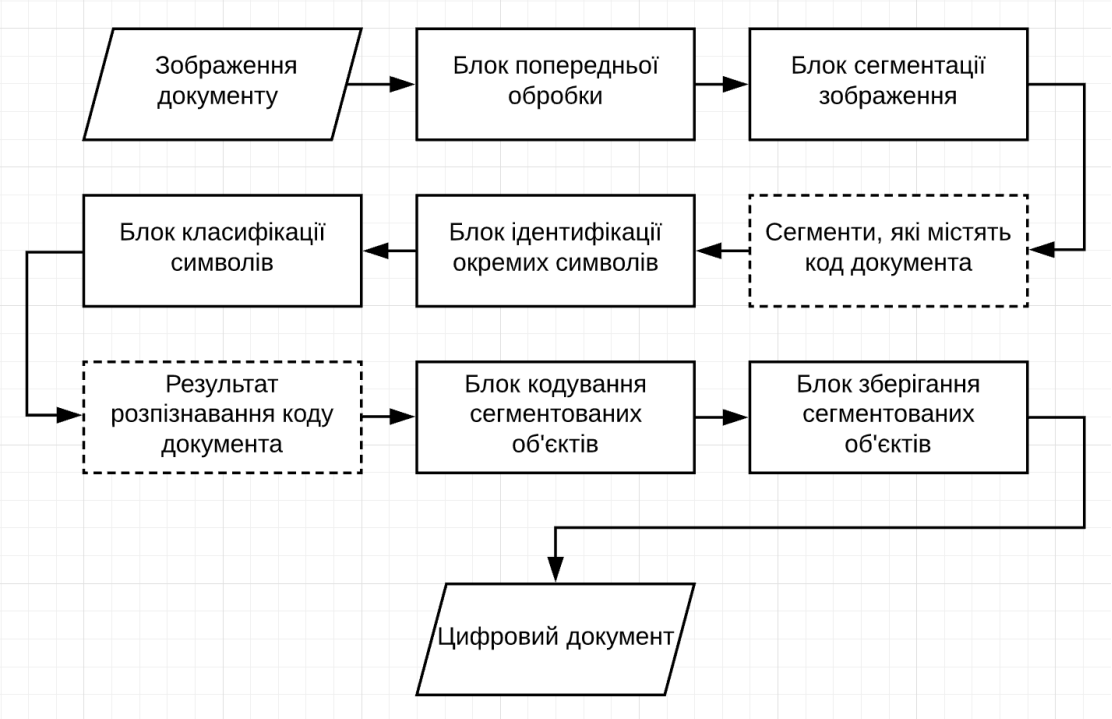

Рисунок 2.3 – Структура схеми розпізнавання

Головним етапом розпізнавання є робота блоку сегментації зображення. На рисунку 2.4 наведено приклад документу з виділеним сегментом, котрий містить код документу.

Рисунок 2.4 – Документ з виділеним сегментом

2.4 Нефункціональні вимоги

NF1. Додаток має працювати з активним підключенням до інтернету.

NF 2. Додаток має працювати на операційній системі Windows 7/8/10

NF 3. Система повинна зберігати дані про адміністратора та користувачів в БД.

NF 4. Додаток не повинен закриватися при наявності помилок, а вміти реагувати на них і сповіщати при цьому користувача.

2.5 Користувацький інтерфейс

Для наглядної візуалізації майбутньої інформаційної системи було розроблено ряд макетів.

Вікно реєстрації / авторизації – вікно, де знаходяться форми заповнення для входу або реєстрації на сайті. Зареєструвавшись або увійшовши, можна потрапити в особистий кабінет (рисунок 2.5).

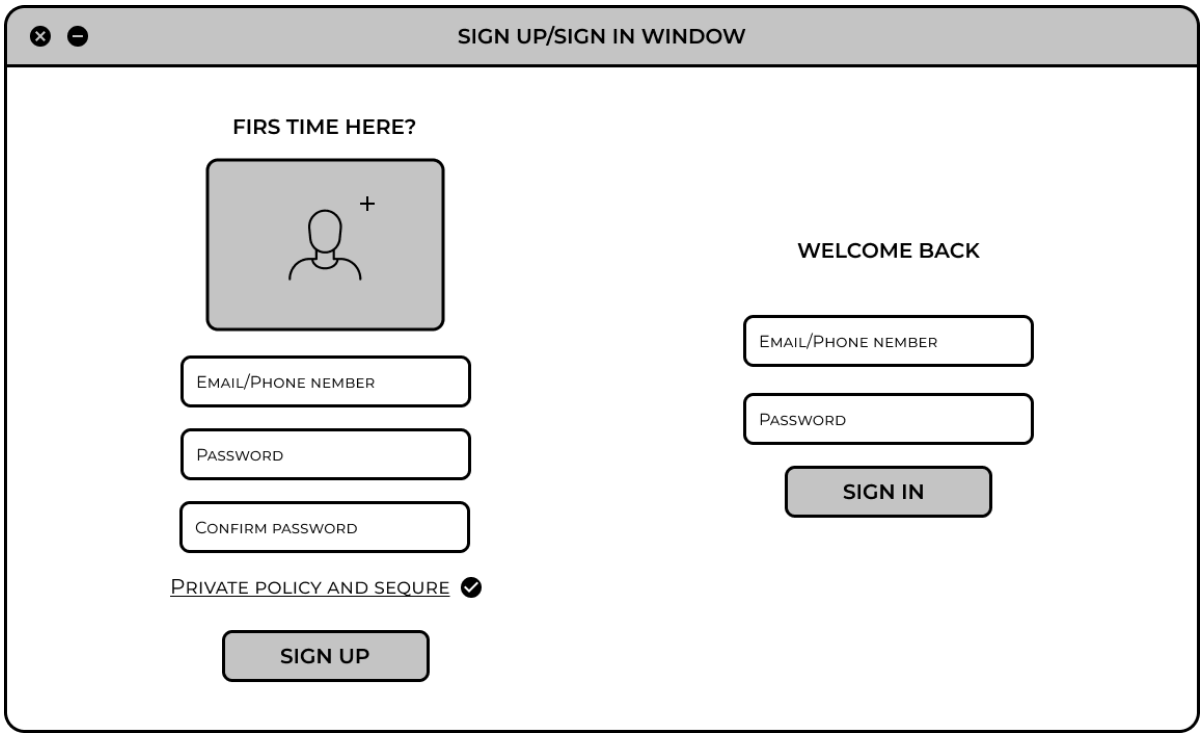

Рисунок 2.5 – Макет інтерфейсу екрану реєстрації/авторизації

Особистий кабінет – вікно, де можна переглядати інформацію про платіжні документи. З особистого кабінету можна перейти до в вікна «групи документів» і «додати новий документ» (рисунок 2.6).

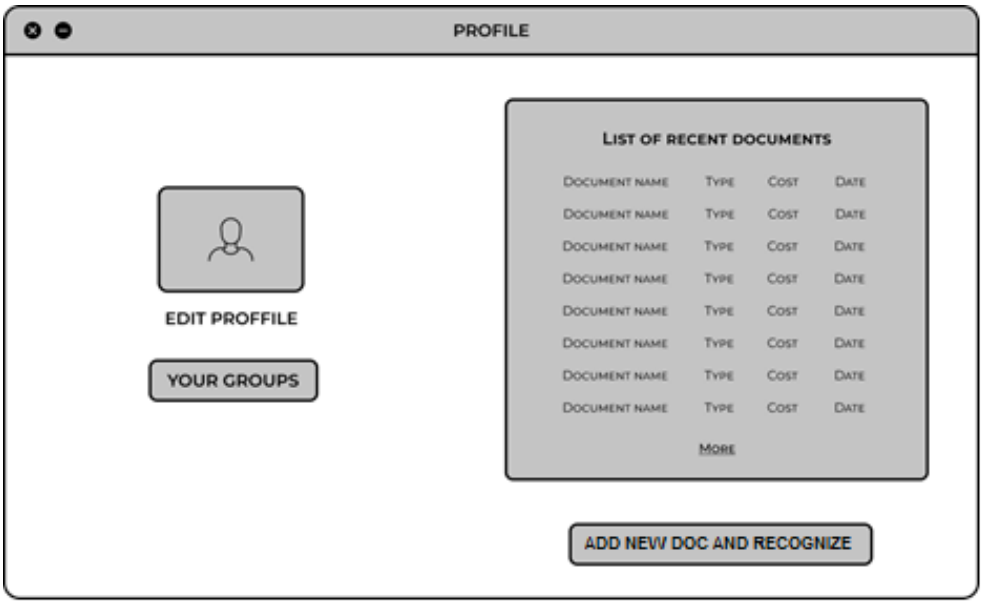

Рисунок 2.6 – Макет особистого кабінету користувача

Додати платіжний документ та розпізнати його – дозволяє додати новий платіжний документ додавши його зображення або ввівши код документа та розпізнати його (рисунок 2.7).

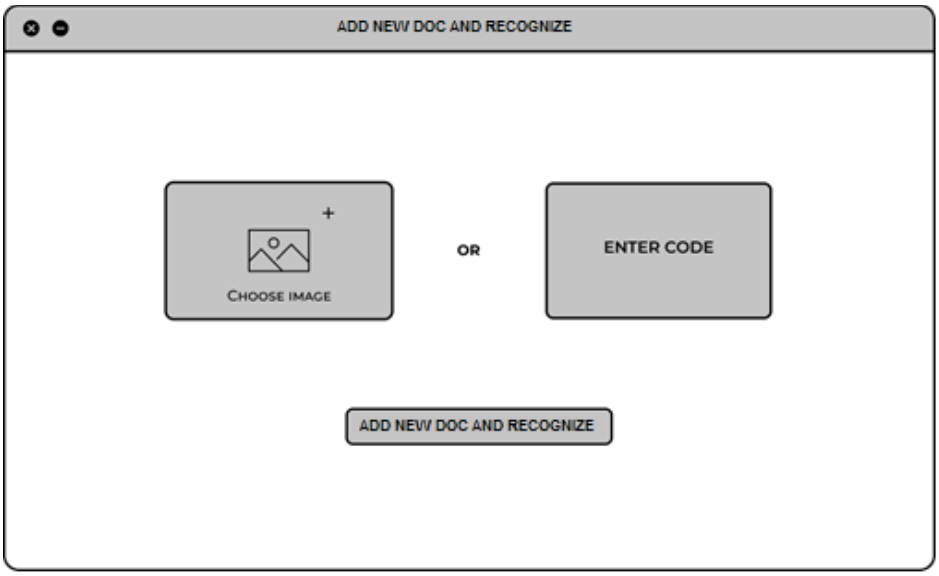

Рисунок 2.7 – Макет «Додати новий документ та розпізнати»

Групи – дозволяють подивитися групи користувача, перейти до додавання нового документу для його розпізнавання, перейти до групи або видалити її (рисунок 2.8).

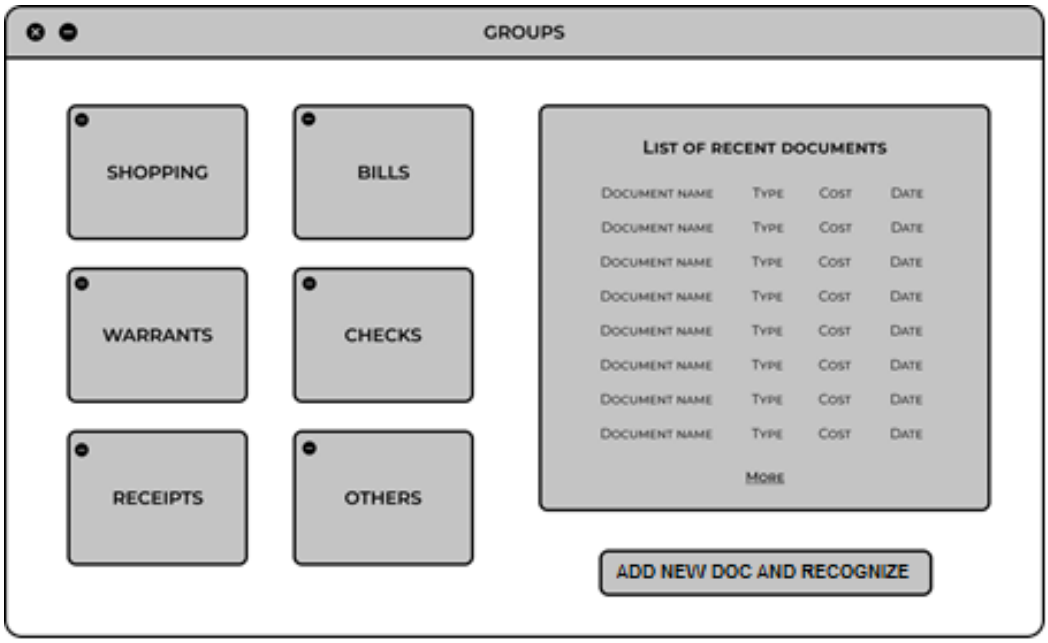

Рисунок 2.8 – Макет «Мої групи»

Перегляд групи – дозволяє переглянути групу, суму коштів групи, видалити / додати платіж з групи (рисунок 2.9).

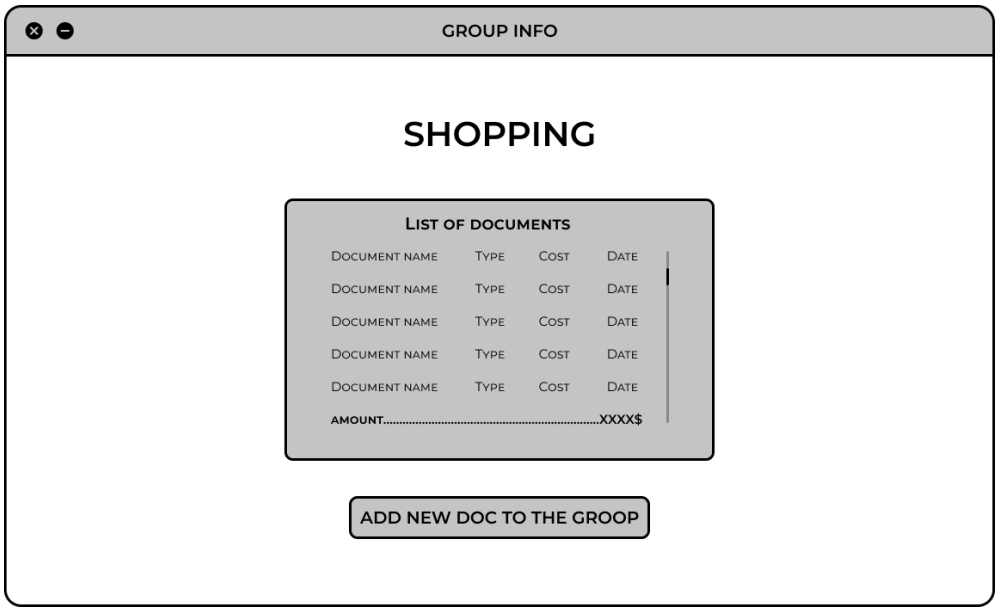

Рисунок 2.9 – Макет «Перегляд групи»

Додавання в групу – дозволяє додати новий документ в групу, документ має бути заздалегідь внесений до системи (рисунок 2.10).

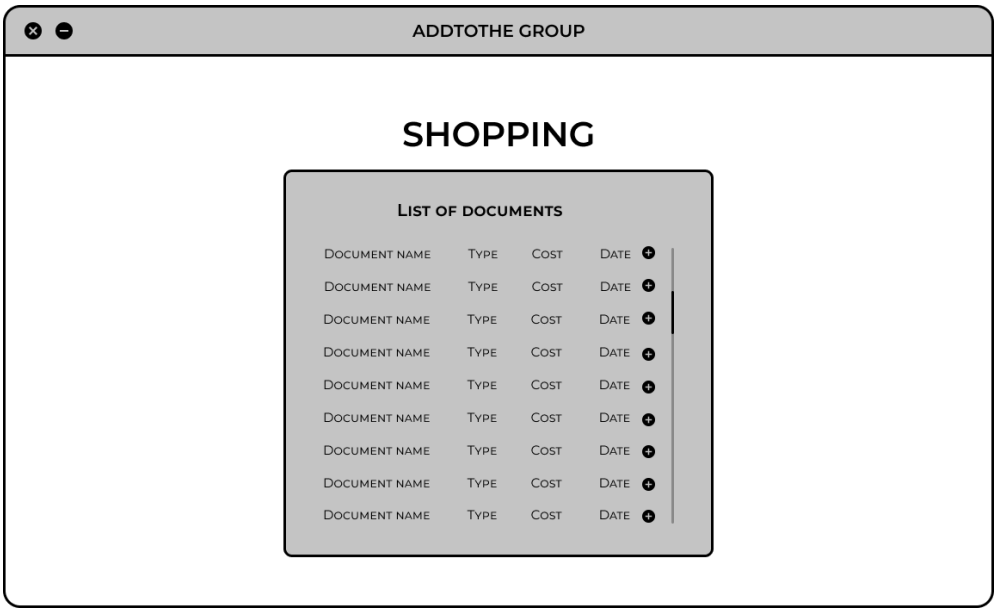

Рисунок 2.10 – Макет «Додати документ до групи»

Базуючись на створених макетах побудуємо діаграму станів додатку (рисунок 2.11).

2.6 Логічне уявлення про ІС (Logical View)

Для створення абстрактної уяви про майбутню систему, була розроблена діаграма логіки роботи з платіжними документами (ПД). На рисунку 2.12 подана логіка роботи системи та напрямки взаємодії користувача з додатком та БД.

В базі даних зберігається інформація про клієнтів.

Адміністратор має усі можливості користувача, та ще має змогу через додаток взаємодіяти з базою даних користувачів. Видаляти, редагувати та ін. Логіка адміністратора передбачає усю логіку користувача з розширеними можливостями.

Рисунок 2.11 – Діаграма станів додатку

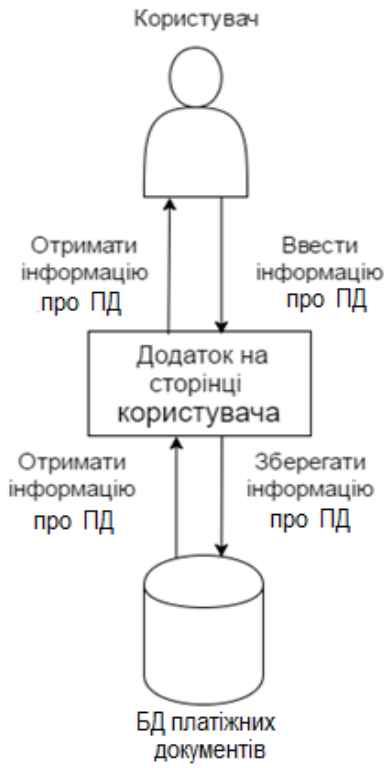

Рисунок 2.12 – Схема логіки роботи системи та зв’язку компонентів для користувача

2.7 Ідентифікація архетипу ІС

Веб-додаток розроблено для декстопних ком’ютерів з операційною системою Windows. Тобто дана ІС відноситься до типів Rich Client Application (RCA) і Calculating Application (CA). Також система потребує постійного підключення до інтернету.

2.8 Уявлення компонентів ІС (Components View)

Важливою частиною інформаційної системи є зв’язок між її компонентами та зовнішніми модулями чи інструментами. Для візуалізації зв’язку між компонентами програми побудуємо діаграму компонентів (рис. 2.13).

У даній ІС зовнішніми джерелами є база даних та файли, що зберігаються для роботі нейронної мережі. У базі даних зберігаються дані про клієнтів та користувачів. А у допоміжних файлах нейронної мережі зберігається інформація про нейронну мережу, а саме її навчання та тестування.

Рівень View зберігає компоненти відповідних вікон. У цих вікнах відображаються усі елементи керування системою та вводу виводу інформації.

На рівні Application зберігаються компоненти контролерів відповідних вікон. Компоненти контролери оброблюють отримані дані із форм вікон та проводять відповідні розрахунки.

Саме у цих компонентах реалізується майже уся логіка інформаційної системи, починаючи від авторизації та закінчуючи оцінкою кредитоспроможності та виводом кредитної рейтингової таблиці.

Компоненти вікон та компоненти контролерів зв’язані напряму, без використання інтерфейсів.

Діаграма компонентів надає повну інформацію про статичну модель

інформаційної системи, зв’язків між ними та зовнішніми елементами.

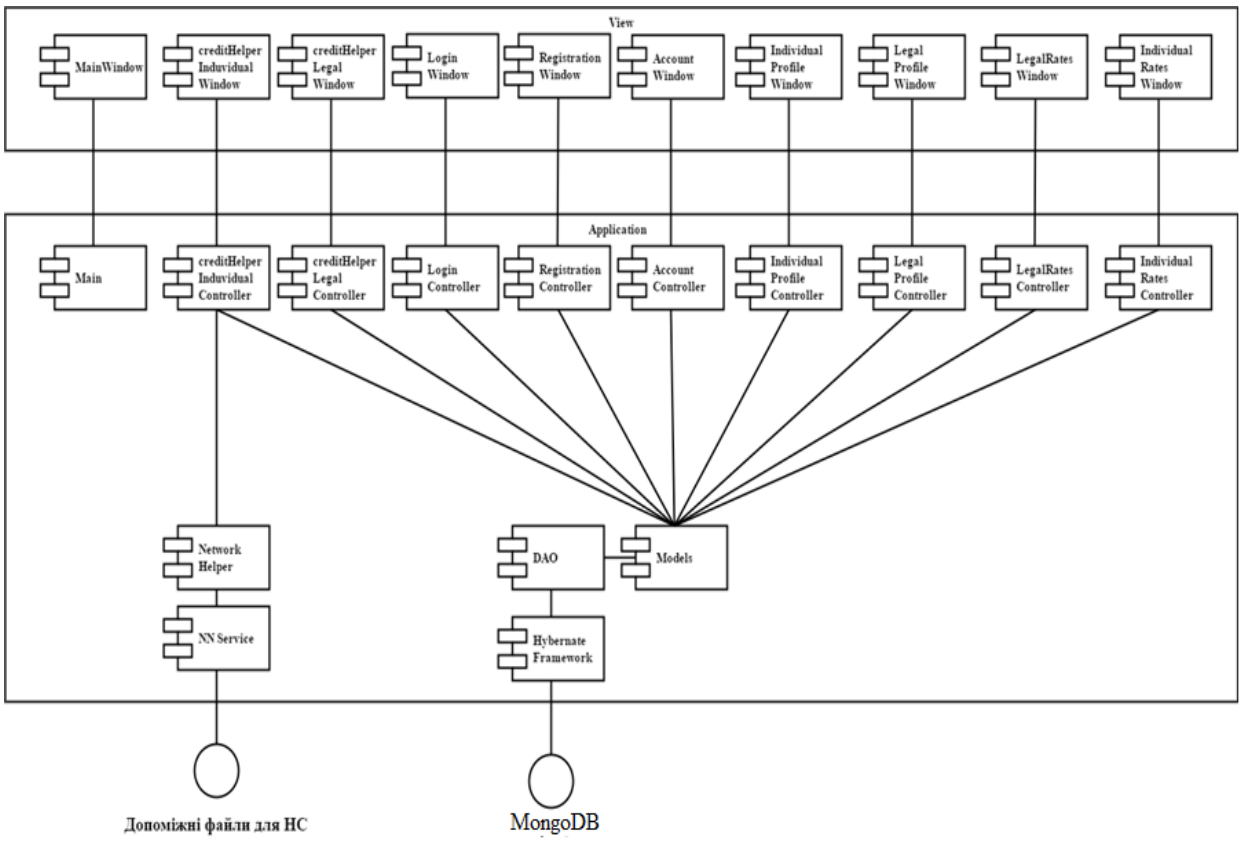

Рисунок 2.13 – Діаграма компонентів ІС

2.9 Уявлення процесів ІС (Process View)

В інформаційній системі застосовується багато різних алгоритмів роботи, але майже всі вони виконуються системою чи окремими бібліотеками. Та все ж в програмі є декілька алгоритмів, які мають найбільшу цінність і виділяють систему з ряду інших: авторизація, розпізнавання платіжного документу, оцінка витрат користувача за його попередніми платіжними документами. Розглянемо процеси, якиі є нетривіальними у порівнянні з іншими.

Сценарій 1. Розпізнавання доданого до ІС документу. Користувач може розпізнати фотокопію паперового платіжного документу, для подальшої обробки та аналізу.

На рисунку 2.14 подано діаграму активності сценарію розпізнавання платіжного документу.

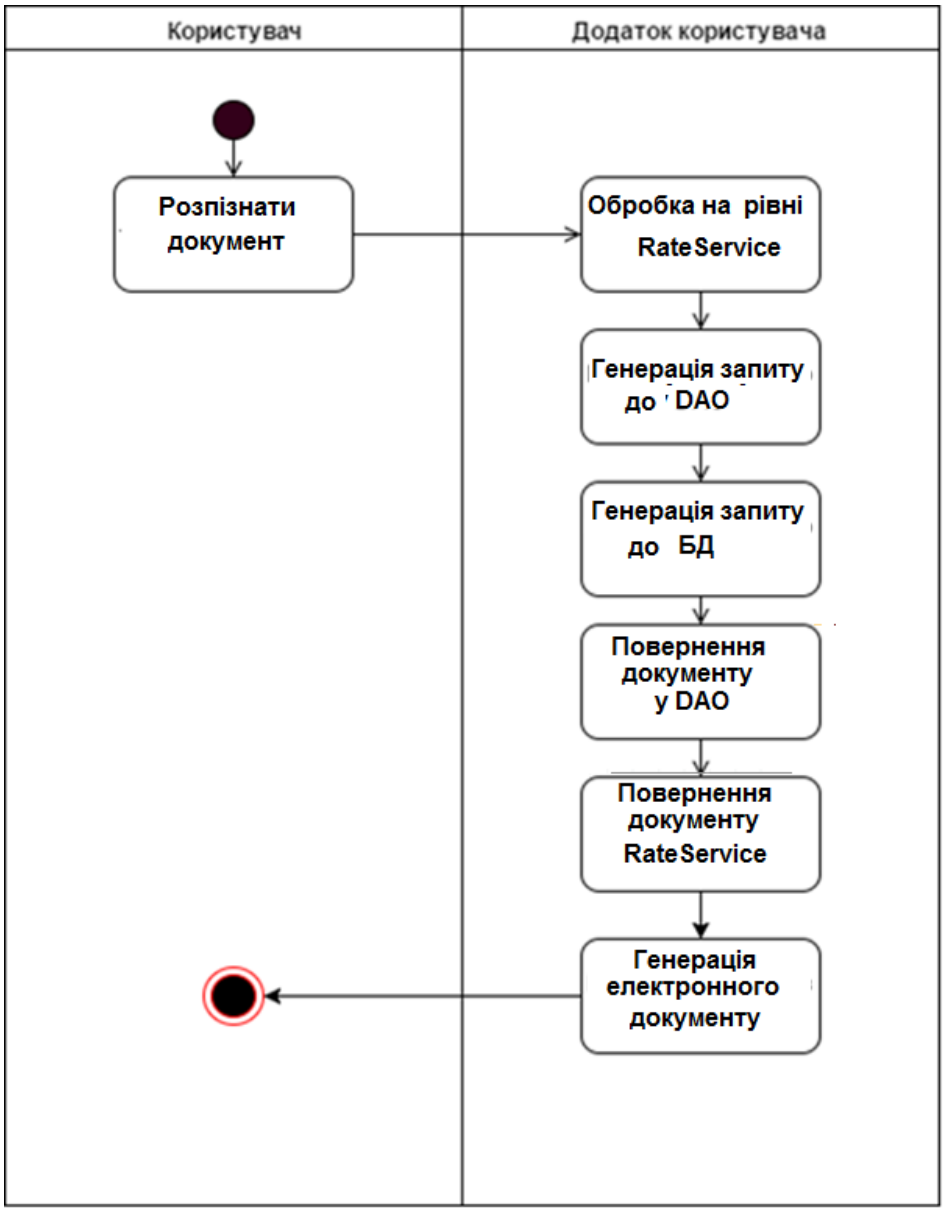

Рисунок 2.14 – Діаграма активності для розпізнавання платіжного документу

Сценарій 2. Перегляд переліку витрат користувача за попередньо обробленими платіжними документами є одним з головних сценаріїв програми. Користувач може отримати таблицю із витратами за попередній період, яка буде сформована за даними, що зберігаються у його кабінеті. На рисунку 2.15 подано діаграму активності сценарію виводу списку витрат, а на рисунку 2.16 діаграму послідовності цього сценарію.

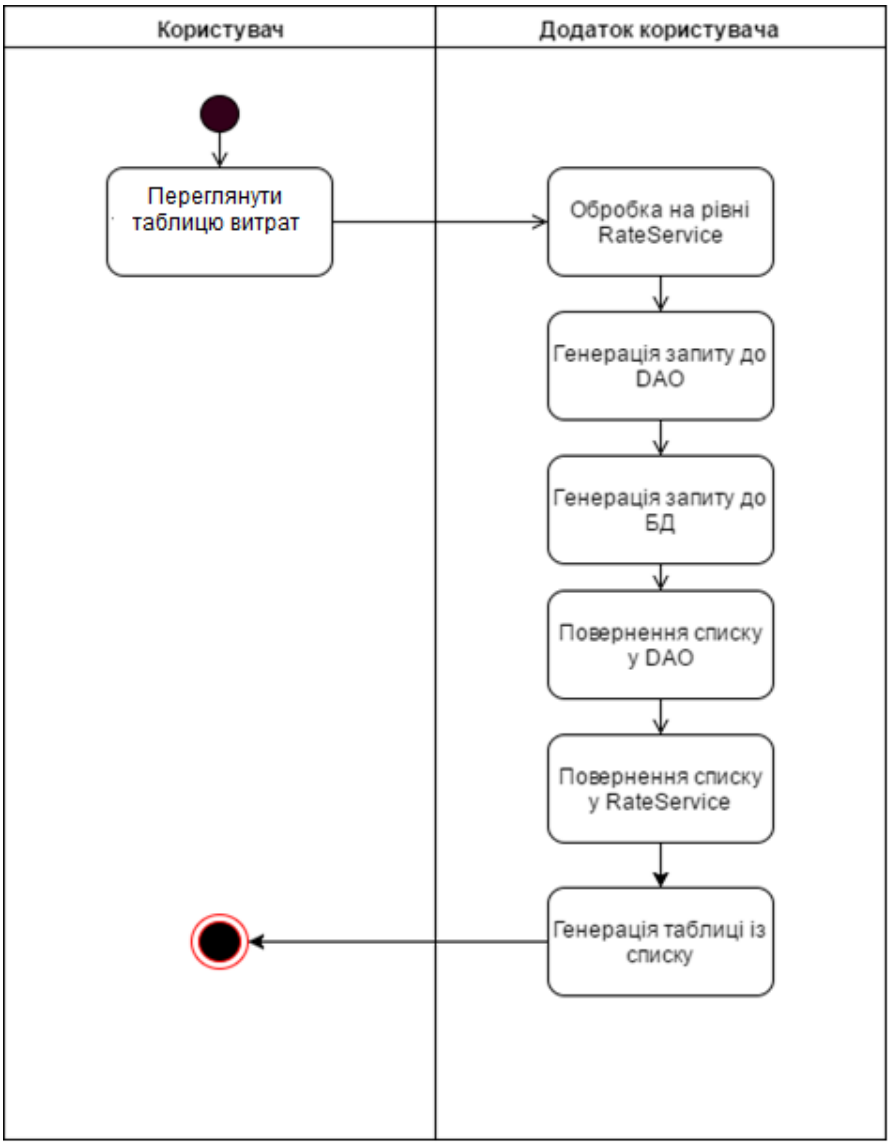

Рисунок 2.15 – Діаграма активності для виводу списку витрат

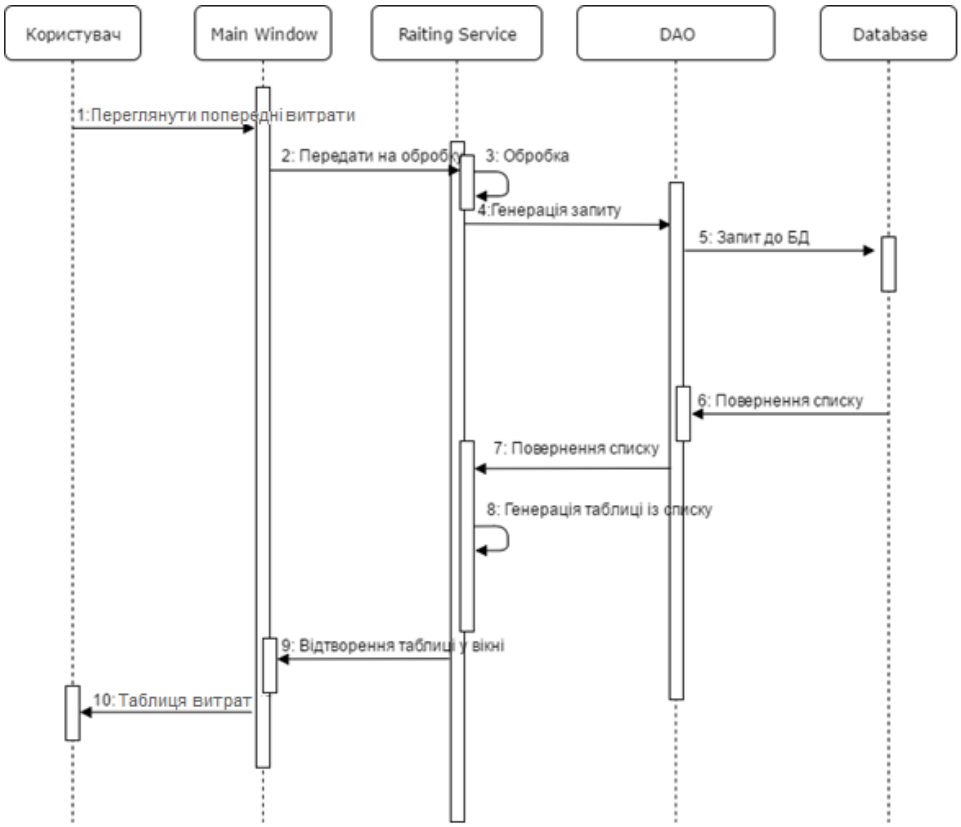

Рисунок 2.16 – Діаграма послідовності для виводу списку витрат

2.10 Уявлення даних IC (Data View)

Список класів, що використовуються при роботі з БД:

– IndividualModel (містить інформацію про користувачів);

– CreditModel (містить інформацію про платіжні документи користувачів);

– AccountModel (містить інформацію про користувачів системою).

Для наглядного представлення про будову локальної бази даних побудовано схеми БД. Схема бази даних таблиць користувачів (рис. 2.17). У базі зберігається інформація про користувачів. Може бути лише один користувач з унікальним ідентифікатором. Він може мати безліч платіжних документів, тому він пов’язаний із сутністю як один до багатьох.

Hibernate у свою чергу має логіку, завдяки якої створюється таблиця, що зберігає ідентифікатори платіжних документів та користувачів.

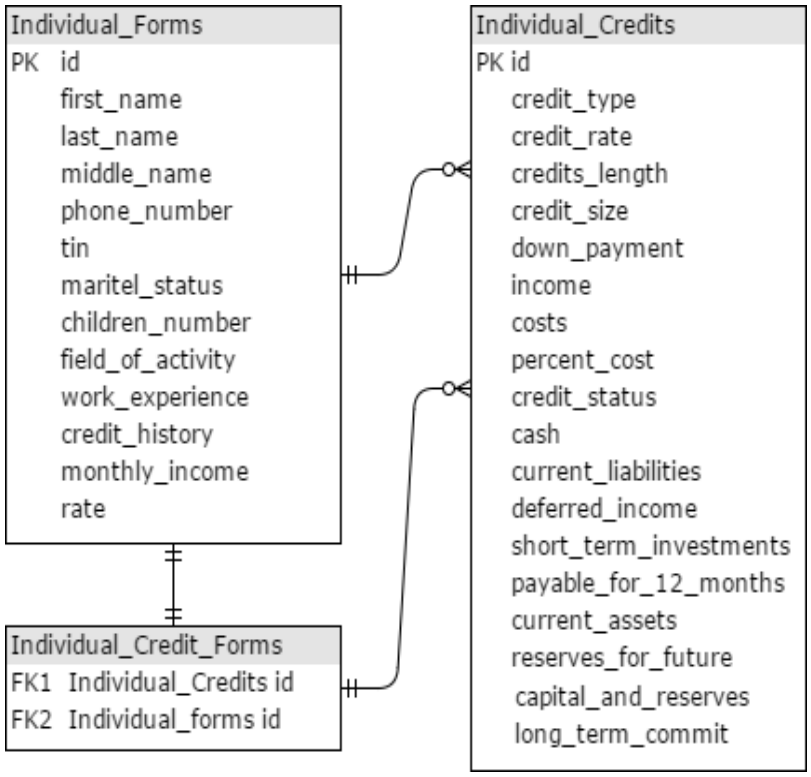

Рисунок 2.17 – Схема бази даних

Тип зберігання даних буде база даних – MongoDB. Для роботи з базою даних будуть використовуватись додаткові класи – класи DAO, що дозволить додавати, редагувати, створювати записи у базі та встановлювати безпосередній зв’язок з БД.

2.11 Опис стеку технологій

Технології, що використовувались:

- Javascript – мова програмування для веб-додатків [5];
- React.js – відкрита JavaScript бібліотека для створення інтерфейсів користувача [9];

− Node.js – платформа для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript [10].

− MongoDB – система управління базами даних [18].

2.12 Інфраструктурне уявлення ІС (Infrastructure View)

В інформаційній системі використовуються лише пристрої користувачів – персональні комп’ютери. При виконанні кваліфікаційної роботи бакалавра, було протестовано навантаження системі під час роботи ІС На рисунку 2.18 подано результати тестування.

Рисунок 2.18 – Тестування навантаження системи під час роботи ІС

Характеристики пристрою:

– 32–розрядний (x86) або 64–розрядний (x64) процесор з тактовою частотою 1 ГГц або вище;

– 1 ГБ (для 32–розрядної процесора) або 2 ГБ (для 64–розрядної процесора) ОЗУ;

– 16 ГБ (для 32–розрядної системи) або 20 ГБ (для 64–розрядної системи) вільного місця на жорсткому диску;

– графічний пристрій DirectX 9 з драйвером WDDM 1.0 або більш пізньої версії;

– для роботи програми необхідна операційна система Windows 7 або новіше;

– встановлений набір для роботи із Javascript додатками;

– активне підключення до інтернету.

Інші характеристики пристрою не впливають на роботу інформаційної системи. Додаток не потребує великих обчислювальних можливостей ПК, тому може використовуватись на більшості комп’ютерів.

Таким чином, у другому розділі спроектовано систему розпізнавання реквізитів платіжних документів. Були розроблені функціональні та нефункціональні вимоги. Також були розроблені мокапи інтерфейсу майбутнього застосунку та діаграма станів екранів, що надає уявлення про роботу системи.

Розроблена діаграма прецедентів, що надає повну інформацію зв’язку між користувачами та їх взаємодію із інформаційною системою. Була розроблена діаграма компонентів, що надає інформацію про компоненти майбутньої системи та зв’язку між ними. Були розроблені діаграми активності і послідовності роботи головних процесів програми, що надають інформацію про динамічне представлення інформаційної системи.

Розроблені діаграми зв’язків у таблицях БД, спроектовані мінімальні необхідні вимоги до можливостей комп’ютеру для функціонування застосунку.

Після проведеного проектування інформаційної системи можна переходити до етапу реалізації.

3. РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Уявлення про структуру проекту ІС

Опис структури проекту виконано через розробку packages додатку. Загалом інформаційна система матиме 9 пакетів.

У пакеті CreditHelper реалізовані класи вікон (Window) етапів розпізнавання у програмі та класи контролери (Controller), що саме і оброблюють дані вікон.

У Database зберігаються 3 класи DAO сервісу, що розроблені для роботи із базою даних. DAO Service це сутність бази даних, у цих класах прописані методи роботи із базою даних.

Main – цей пакет, до складу якого входять інші. Він зберігає класи main та деякі моделі інших класів, у тому числі клас налаштування у програмі, клас вікна, що інформує про помилки.

Main.accounts – пакет, що зберігає класи контролерів вікон та класи вікон акаунтів. У цьому пакету зберігаються класи моделей акаунтів, моделей списку акаунтів.

Main.login – пакет, що зберігає класи контролерів вікон для авторизації у системі.

Main.rates – пакет, що відповідає за розпізнавання платіжних документів користувачами системи. У цих класах реалізована побудова таблиці із списку, що було отримано від бази даних за допомогою DAO Service та Hibernate налаштунку. Main–registration – пакет реєстрації у програмі. Зберігає класи вікна реєстрації та контролера реєстрації, що саме обробляє дані, введені у форми вікна.

Neural.network – пакет нейронної системи. Цей модуль потрібен для прийняття рішень щодо типу платіжного документу, Він складається із

класів, що зберігають та передають інформацію до нейронної системи з бази даних та навпаки. Класів, в яких відбувається розрахунки нейронної системи, клас нейрону самої нейронної системи, класів першого та максимального слоїв.

Profile.windows – пакет, що використовується для виводу інформації про користувача. У цьому модулі зберігаються класи вікна виводу, його контролеру, що керує інформацією. Для рис 3.1 зображено структуру пакетів розроблених у системі.

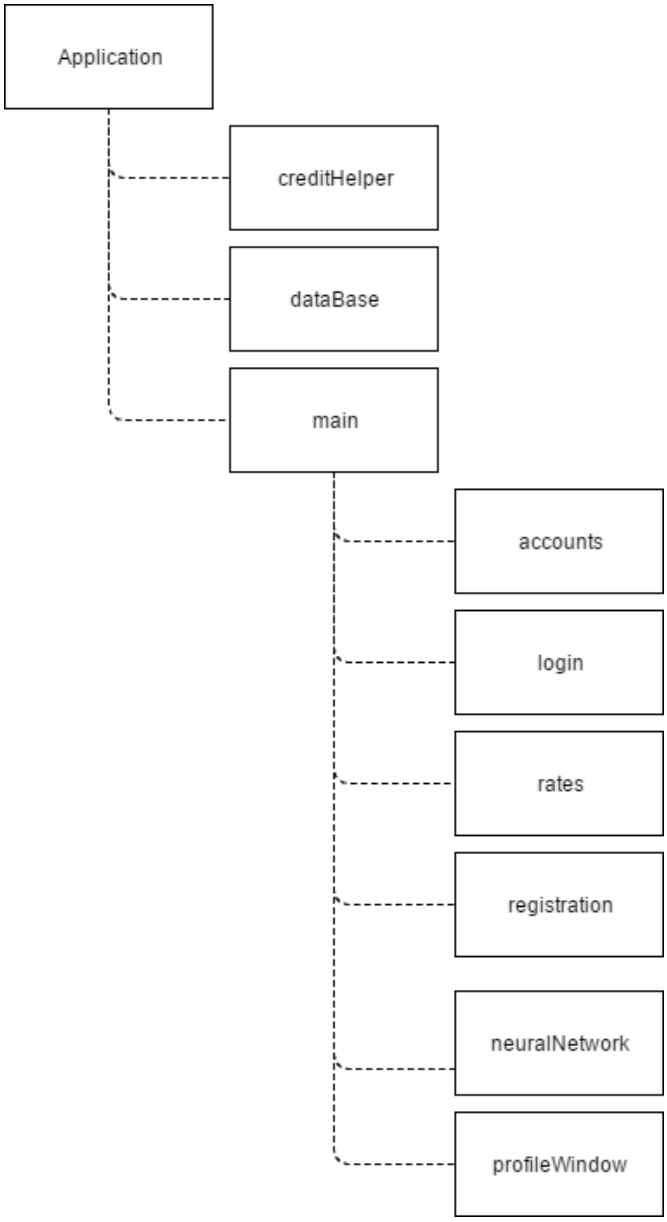

Рисунок 3.1 – Структура проекту ІС

3.2 Структура класів ІС

Для визначення типу платіжного документу та його атрибутів були розроблені такі класи: Step2IndividualWindow, Step2Individual, Step3IndividualWindow, Step3Individual, Step4IndividualWindow, Step4Individual, StepWindow, Step, AbstractStep, AbstractStepWindow. На рисунку 3.2 зображена діаграма класів та вікон системи при розпізнаванні платіжного документу.

Step2IndividualWindow – у цьому класі описується вікно кроку у програмі, коли користувач буде заносити свої дані: Прізвище, Ім’я, По–батькові. Головним методом цього класу є public Step2IndividualWindow(). У цьому методі описані усі налаштунки для кожної форми запису. Також у цьому класі описано налаштунки самого вікна.

Step2Individual – клас, у якому здійснюється логіка вікна передоднього класу. У цьому класі є метод Step2Individual(), що за допомогою конструктора створюють нову модель даних користувача та заноситься у модель інформація.

Наступний клас Step3IndividualWindow. У цьому класі реалізоване графічне уявлення вікна етапу. У методі Step3IndividualWindow() описані налаштунки вікна та форм запису у ньому. У цьому вікні за необхідністю збирається інформація про користувача: контактний телефон, ІНН, дата народження, тощо.

Клас Step3Individual. Метод Step3Individual (IndividualModel model). У цьому методі працює конструктор super(model), що є тимчасовим екзепляром класу IndividualModel. У методі init() задаються налаштунки роботи вікна.

Step4IndividualWindow – клас, у якому зберігається налаштунки вікна. Саме у цьому вікні заповнюється інформація про фінансові витрати користувача, за попередньо введеними платіжними документами. У методі Step4IndividualWindow() описані усі налаштунки цього вікна, форм запису, а також метод actionPerformed(ActionEvent e) для перевірки, чи усі поля були заповнені.

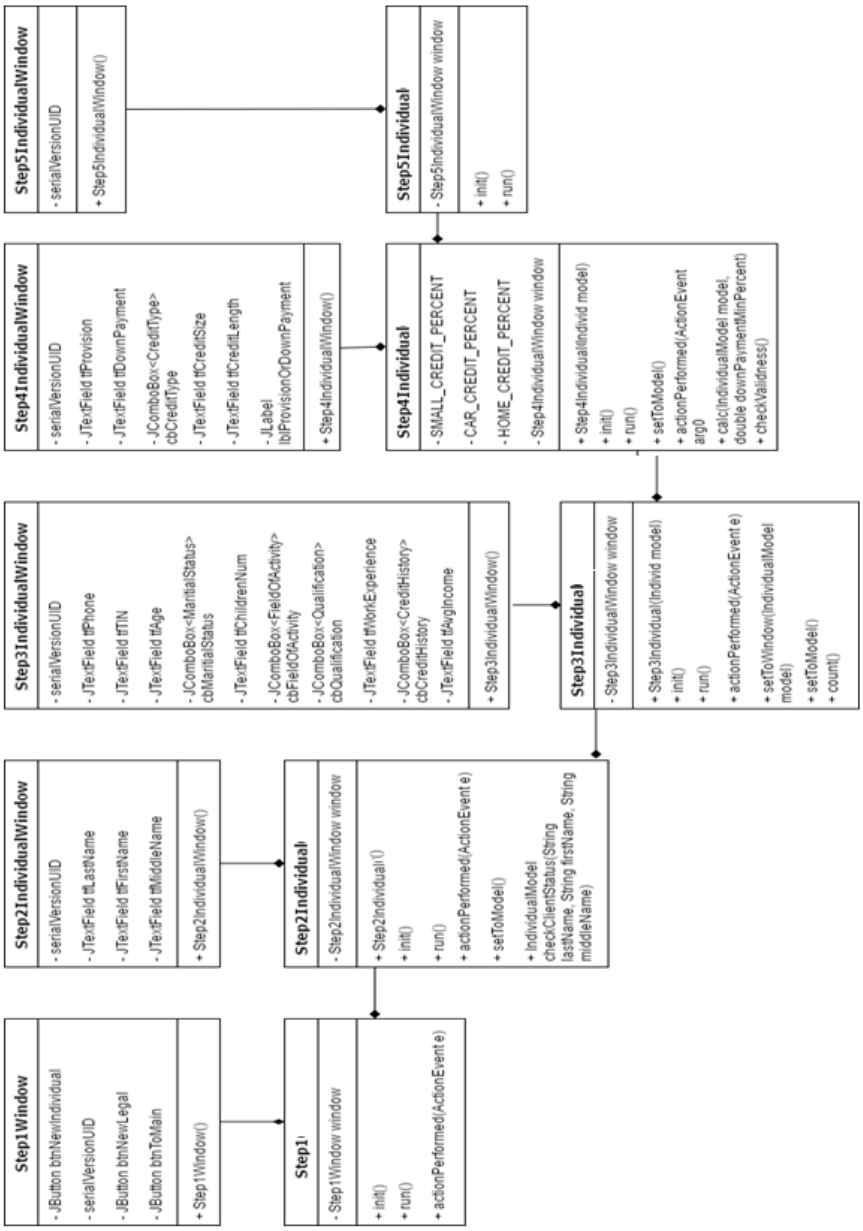

Рисунок 3.2 – Діаграма класів зв’язку та вікон розпізнавання платіжних документів

Step4Individual – клас перевірки вікна. Звичайно перший метод Step4Individual(IndividualModel model), що містить конструктор Model. Далі метод init(), у якому відбувається перевірка роботи вікна програми.

StepWindow та Step це інтерфейси, що мають у собі лише один метод пустий init().

AbstractStep – абстрактний клас, що реалізує інтерфейс Step. Має 2 методи у собі: пустий init() та AbstractStep(IndividualModel model), що реалізує конструктор Model.

У класі AbstractStepWindow було реалізовано 3 панелі вікна та одна кнопка. У методі AbstractStepWindow() реалізовано налаштунки для елементів вікна, описане їх місцезнаходження.

Ці класи керують вікнами, а саме формами, у які заносять інформацію про користувача. Для розпізнаванні платіжного документу використовується нейронна система. Одним із етапів аналізу платіжного документу є робота нейронної системи. Пакет з нейронною системою включає у себе наступні класі : NetworkHelper, NeuralNetwork, NeuralNetworkIpml, Neuron, NeuronFirstLayer, NeuronHiddenLayer, NeuronLayer, NeuronImpl, NeuronNoSigmoidImpl.

На рисунку 3.3 зображена діаграма класів роботи нейронної мережі.

AccountDAO – це клас, що реалізує інтерфейс AutoCloseable. У цьому класі реалізовано декілька публічних методів для роботи з даними акаунтів користувачів. getSession() та setSession() для передачі до БД інформації та навпаки. Метод close() завершує сесію. Метод AccountModel getAccount(String login, char[] password) реалізовано для отримання акаунтів від моделі AccountModel, а саме пароль та логін користувача. Метод ArrayList<AccountModel> getAllAccounts() розроблений для отримання списку користувачів із моделі AccountListModel. Методи saveOrUpdate(AccountModel model) та delete(AccountModel model) реалізують функції збереження та видалення. На діаграмі класів DAO акаунтів зображені

класи DAO та sessionFactory, session, що реалізують логіку обміну даних із БД та моделі акаунту та списку акаунтів. Моделі викликають методи із AccountDAO.

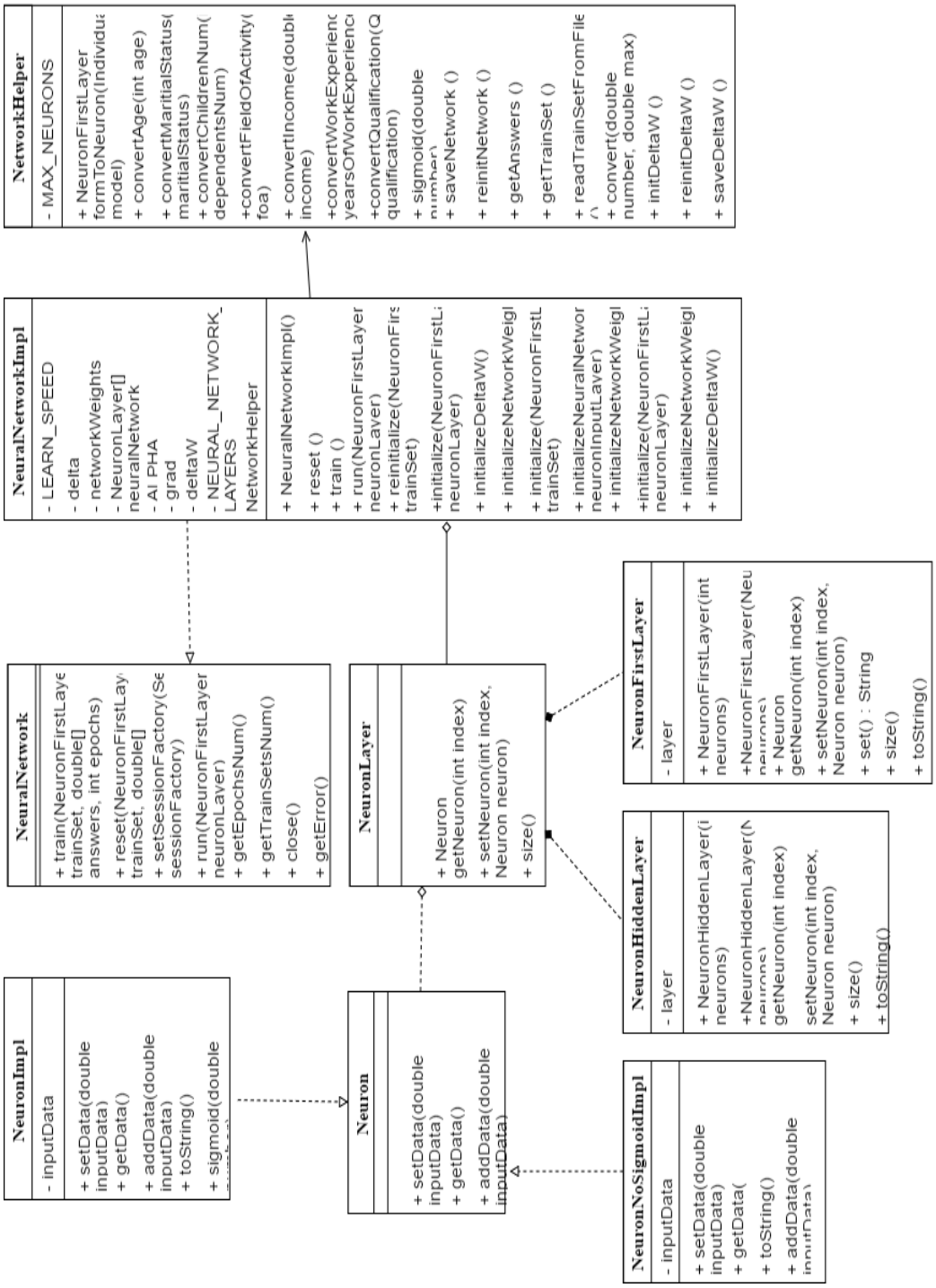

Рисунок 3.3 – Діаграма класів нейронної мережі

На рисунку 3.4 зображена діаграма класів роботи DAO сервісу з моделями акаунтів. AccountModel – це клас, що реалізує інтерфейс Serializable та використовується як модель акаунтів. Зберігає інформацію про користувачів, їх статус. Має дуже багато однакових методів таких як get() та set(), що саме реалізують логіку обміну інформації (рис.3.5).

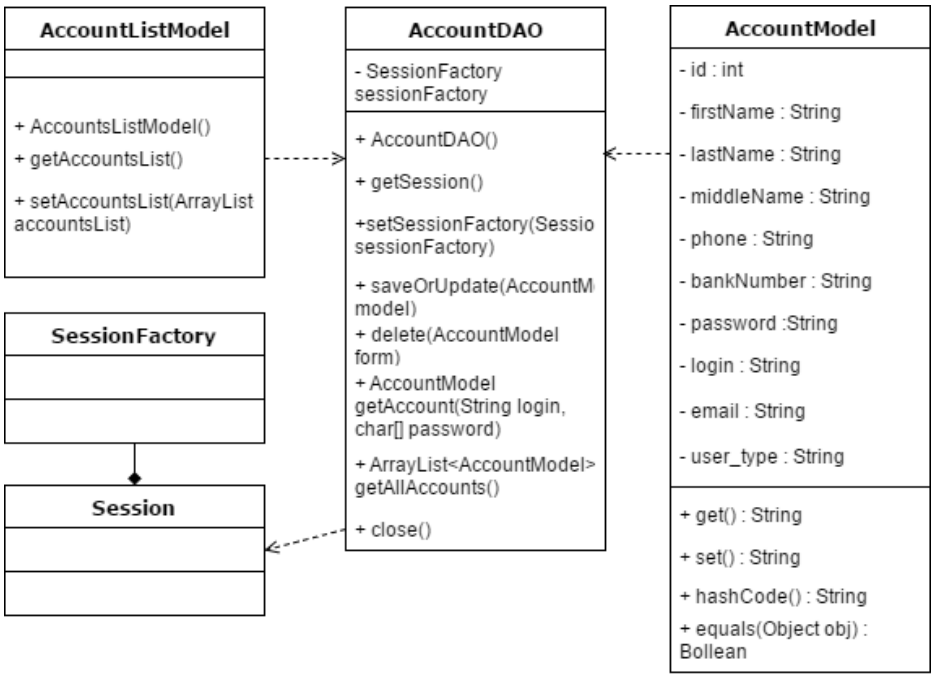

Рисунок 3.4 – Діаграма класів DAO акаунтів

Було реалізовано класи для виводу рейтингового списку фізичних осіб та юридичних : IndividualRates та LegalRates.

IndividualRates – у цьому класі створюється таблиця із рейтинговим списком клієнтів фізичних осіб. Метод IndividualRates() описує логіку отримання моделей даних клієнтів та створення таблиці з їх даними.

За авторизацію у системі відповідає вікно логіну. Для реалізації вікна логіну було розроблено 2 класи: LoginWindow та Login.

LoginWindow – це клас, що наслідує JFrame, у якому розроблене графічне уявлення вікна логіну. А саме поля вводу, кнопки та гіпертекст для переходу до реєстрації. Саме у методі LoginWindow() реалізоване графічне

уявлення вікна, його формат, формат текстових полів, полів вводу: логіну та паролю.

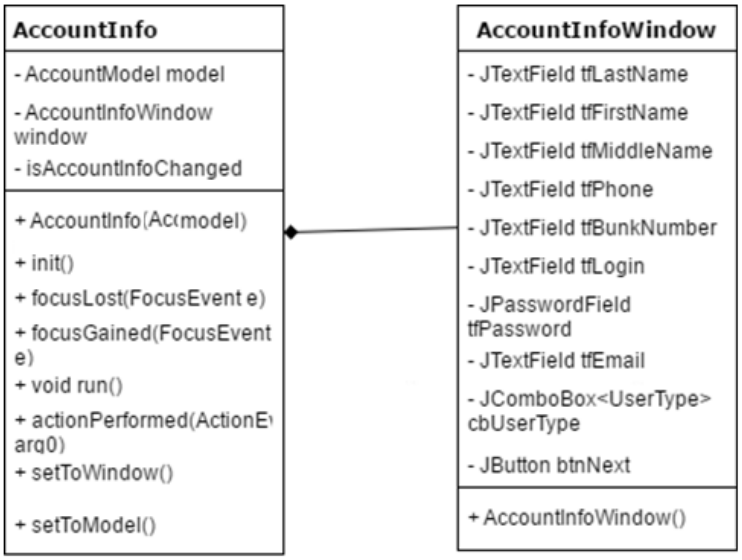

Рисунок 3.5 – Діаграма класів уявлення вікна акаунтів

Login – це контролер до класу LoginWindow. Для реалізації використовується модель акаунтів AccountModel. У методах init() і run() описується логіка виконання операцій у вікні. А саме відгук від натискання кнопок, вводу тексту у поля вводу та натискання гіпертексту. На рисунку 3.6 зображена діаграма класів контролера та вікна логіну.

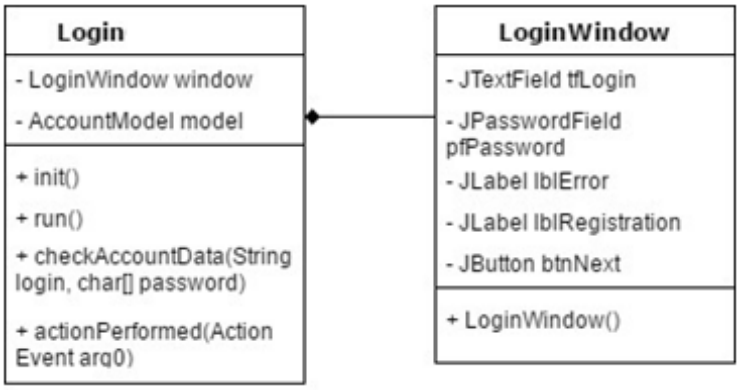

Рисунок 3.6 – Діаграма класів уявлення вікна логіну

Для реєстрації було розроблено 2 класи (рис. 3.7). Клас контролера та клас вікна RegistrationWindow – це клас, що наслідує JFrame, у якому розроблене графічне уявлення вікна реєстрації. А саме поля вводу та кнопки. У методі RegistrationWindow() саме реалізоване графічне уявлення вікна. Формат кнопок та кнопок вводу.

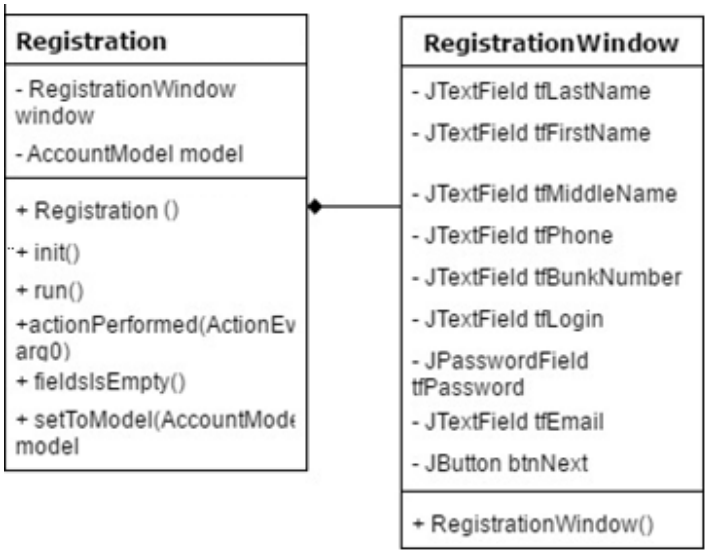

Рисунок 3.7 – Діаграма класів уявлення вікна реєстрації

Registration – це контролер до класу RegistrationWindow. Для реалізації використовується модель акаунтів AccountModel. Метод public Registration() створює новий акаунт у моделі AccountModel.У методах init() і run() описується логіка виконання операцій у вікні. А саме відгук від натискання кнопок, вводу тексту у поля вводу. Діаграма класів вікна та контролеру реєстрації у системі.

3.3 Розрахунок метрик програмного коду ІС

Розрахунок метрик коду представлено у таблиці 3.1. Для розрахунку показників метрики ми скористалися плагіном Metric.

Таблиця 3.1 – Метрики програмного коду

Метрика	Значення
Загальна кількість рядків у проекті	4331
Середня кількість рядків у класі	75
Максимальна кількість рядків у класі	350
Середня кількість рядків у методі	1.4
Максимальна кількість рядків у методі	50
Максимальна цикломатична складність	1.5
Максимальна глибина дереву наслідування	1
Середня цикломатична складність	1
Коментування коду, %	10
Покриття коду модульними тестами	61

3.5 Документація ІС

Для використання розробленої ІС було написано керівництво користувача (Додаток Б), за яким нові користувачі зможуть легко використовувати усі функції додатку.

3.6 Уявлення безпеки ІС

Безпека даних не є ключовою у ІС, але вона була здійснена на відповідному рівні.

Було реалізовано можливість реєстрації у системі. У ІС є два типи користувачів: користувач та адміністратор, який може редагувати їх данні, видаляти чи додавати до системи.

Для доступу до БД використовується захищене підключення, завдяки якому ІС виходить на новий рівень безпеки.

Безпека бази даних реалізовано завдяки hibernate. На програмному рівні реалізований рівень безпеки із підстановкою параметрів.

У ньому реалізована своя мова запитів (HQL), що захищає від можливих SQL–ін’єкцій.

Також для реалізації безпеки було вирішено використовувати паролі для акаунтів не коротше за 8 символів.

Це обміркована мінімальна довжина паролю, яку дуже складно розкрити. Було реалізовано хешування паролів MD5. Усі ці алгоритми гарантують певну безпеку даних ІС.

Таким чином, у третім розділі було реалізовано та протестовано інформаційну систему для розпізнавання реквізитів платіжних документів. Реалізовані усі компоненти та їх класи. Було створено діаграми головних класів для показу основних їх зв’язків та показу статичної моделі програми.

Були розглянуті методи головних класів, їх призначення та логіку у програмі. Було побудовано діаграму зв’язку класів між різними компонентами програми. Ця діаграма надає повну інформацію між взаємодією компонентів програми.

Було розраховано метрики коду програмного застосунку, наведені розрахунки метрик роботи із GiHub та метрики з використання популярних технологій.

Також було проведено тестування програми. Було проведено функціональне тестування, у ході якого були протестовані функціональні можливості ІС та перевірені усі фунціональні вимоги, спроектовані у розділі проектування інформаційної системи.

Було проведено модульне тестування, у ході якого було перевірено та протестовано виконавчі методи та класи програми. Була перевірена

правильність роботи методів програми та розраховано покриття модульними тестами програми. Усі спроектовані функціональні вимоги були реалізовані та протестовані, з чого можна стверджувати, що реалізація і тестування програми пройшло успішно.

Після проведення реалізації та тестування була створена документація до продукту з інструкціями користувача та інсталяцією програми як з програмного коду так і для повсякденного використання.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра було розроблено інформаційну систему розпізнавання платіжних документів. Запропонована інформаційна система стане у нагоді для фізичних осіб та громадських установ, які бажатимуть контролювати власні витрали, через аналіз платіжних документів паперового вигляду. Інформаційна система надає можливості автоматизувати облік витрат через перетворення паперових документів у електронні з коректним розпізнаванням їх типу та атрибутів.

В роботі проведено аналіз аналогів розробленої інформаційної системи, були виділені переваги та недоліки для проектування та реалізації інформаційної системи. Були оглянуті та обрані технології для проектування та розробки інформаційної системи. Розроблена система вібрала у себе переваги аналогів та була покращена новими можливостями.

Після етапу огляду технологій та аналогів було спроектовано систему розпізнавання платіжних документів. Розроблені діаграми прецедентів, що надають повну інформацію зв’язку між користувачами та їх взаємодію із інформаційною системою. Спроектовані діаграми активності і послідовності роботи головних процесів програми, що надають інформацію про динамічне представлення інформаційної системи. Також були розроблені діаграми зв’язків у таблицях БД, спроектовані мінімальні необхідні вимоги до можливостей комп’ютеру для функціонування застосунку. А також було спроектовано два алгоритми що пов’язані з розпізнавання платіжних документів для фізичних осіб.

Було реалізовано та протестовано інформаційну систему. Реалізовані та описані усі компоненти та класи застосунку, створено діаграми головних класів для показу основних їх зв’язків та показу статичної моделі програми.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Додаток ABBYY Hot Folder. URL:https://help.abbyy.com/uk-ua/finereader/12/hotfolder (дата звернення 14.04.2023)
2. Властивості проекту. URL: https://help.abbyy.com/ua-ua/flexicapture/12/distributed_administrator/project_properties (дата звернення 14.04.2023)
3. Программное обеспечение «Regula Forensic Studio». URL: https://regulaforensics.com/ru/products/software/regula-forensic-studio/ (дата звернення 14.04.2023)
4. Smart ReTypeDoc. URL: https://smartengines.ru/smart-retypedoc/ (дата звернення 14.04.2023)
5. Мова JavaScript та її можливості. URL: https://sites.google.com/site/webtehnologiietawebdizajn/mova-javascript-ta-ieie-mozlivosti (дата звернення 14.04.2023)
6. Douglas Crockford. JavaScript:The World's Most Misunderstood Programming Language. URL: www.crockford.comhttp://www.crockford.com/javascript/javascript.html (дата звернення 14.04.2023)
7. Standard ECMA-262. ECMAScript® 2019 Language Specification. URL: https://www.ecma-international.org/publications/standards/Ecma-262.htm (дата звернення 14.04.2023)
8. JavaScript. URL: https://uk.wikipedia.org/wiki/JavaScript (дата звернення 14.04.2023)
9. Офіційний сайт ReactJS. URL: https://reactjs.org/docs/ (дата звернення 14.04.2023)
10. Node.js. URL: https://uk.wikipedia.org/wiki/Node.js (дата звернення 14.04.2023)

11. Цеслів О.В. Основи програмування та веб-дизайн: Навч. посіб. – К.,2020. 149с. URL: https://ela.kpi.ua/bitstream/123456789/40499/1/OP_veb-dyzain.pdf (дата звернення 19.05.2023)
12. Авраменко В.С., Авраменко А.С. Проектування інформаційних систем: навчальний посібник. Черкаси: Черкаський національний університет ім. Б. Хмельницького, 2017. – 434 с.: іл.
13. Коваленко А.А., Кучук Г.А. Сучасний стан та тенденції розвитку комп'ютерних систем об'єктів критичного застосування// Системи управління, навігації та зв’язку. Полтава . ПНТУ, 2018. Вип. 1(47). С. 110-113.
14. Нікітіна Т.С., Морозова О.І. Порівняльний аналіз продуктивності баз даних SQL та NOSQL. URL: http://journals.nupp.edu.ua/sunz/article/download/1386/1179 (дата звернення 14.04.2023)
15. Apache HBase. URL: https://hbase.apache.org/ (дата звернення 14.04.2023)
16. Amazon DynamoDB. URL: https://aws.amazon.com/documentation/dynamodb/ (дата звернення 14.04.2023)
17. Apache Cassandra. URL: http://cassandra.apache.org/ (дата звернення 14.04.2023)
18. MongoDB Atlas. URL: https://www.mongodb.com/ (дата звернення 14.04.2023)

Д О Д А Т К И

ДОДАТОК А
ЛІСТИНГ ПРОГРАМНОГО КОДУ

docman-api-master
Db/cruds/Document

```
const DocumentModel = require('../models/document');

module.exports = {
  addDocument: data => require('./templates/add')
(DocumentModel, data),
  getDocument: (params, sort, selectedFields) =>
    require('./templates/get')(DocumentModel, params, sort,
selectedFields),
  updateDocument: (findField, setField) =>
    require('./templates/update')(DocumentModel, findField,
setField),
  deleteDocument: findField =>
    require('./templates/delete')(DocumentModel, findField),
}
```

Db/cruds/Groups

```
const GroupModel = require('../models/group');

module.exports = {
  addGroup: data => require('./templates/add')(GroupModel, data),
  getGroup: (params, sort, selectedFields) =>
    require('./templates/get')(GroupModel,        params,        sort,
selectedFields),
  updateGroup: (findField, setField) =>
    require('./templates/update')(GroupModel, findField, setField),
  deleteGroup: findField =>
    require('./templates/delete')(GroupModel, findField),
}
```

Db/cruds/User

```
const UserModel = require('../models/user');

module.exports = {
  addUser: data => require('./templates/add')(UserModel, data),
  getUser: (params, sort, selectedFields) =>
    require('./templates/get')(UserModel,        params,        sort,
selectedFields),
  updateUser: (findField, setField) =>
    require('./templates/update')(UserModel, findField, setField),
  deleteUser: findField =>
    require('./templates/delete')(UserModel, findField),
}
```

Db/cruds/templates/add

```
module.exports = (mongooseModel, data) => {
    return new Promise((resolve, reject) => {
        if (Array.isArray(data) && data.length) {
            mongooseModel.insertMany(data, (err, res) => {
                if     (err)     reject(`Error     inserting     into
${mongooseModel}: ${err}`)
```

```
                resolve(res)
            })
        } else {
            mongooseModel.create(data, (err, res) => {
                if (err) reject(`Error inserting into
${mongooseModel}: ${err}`)

                resolve(res)
            })
        }
    })
}
```

Db/cruds/templates/delete

```
module.exports = (mongooseModel, findField) => {
    return new Promise((resolve, reject) => {
        mongooseModel.deleteMany(findField, (err, res) => {
            if (err) reject(`Error updating a user field ${err}`)

            resolve(res)
        })
    })
}
```

Db/cruds/templates/get

```
module.exports = (
    mongooseModel,
    params,
    sort = null,
    selectedFields = null
) => {
    return new Promise((resolve, reject) => {
        mongooseModel.find(params, selectedFields, sort, (err, texts)
=> {
            if (err) reject(`Error while finding text ${err}`)

            resolve(texts)
        })
    })
}
```

Db/cruds/templates/update

```
module.exports = (mongooseModel, findField, setField) => {
    return new Promise((resolve, reject) => {
        const [setFieldVals] = Object.keys(setField);
        let setter = {};

        if (setFieldVals.includes('$')) {
            setter = setField;
        } else {
            setter = { $set: setField }
        }
```

```
        mongooseModel.updateMany(findField, setter, (err, res) => {
            if (err) reject(`Error updating a Text field ${err}`)

            resolve(res)
        })
    })
}
```

docman-api-master\db\models
document

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema
const timestamp = require('mongoose-timestamp')

const DocumentSchema = new Schema({
  name: String,
  type: String,
  url: String,
  cost: String,
  userId: Schema.Types.ObjectId
});

DocumentSchema.plugin(timestamp)

module.exports = mongoose.model('Document', DocumentSchema)
```

group

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema
const timestamp = require('mongoose-timestamp')

const GroupSchema = new Schema({
  name: String,
  documents: Array,
  userId: Schema.Types.ObjectId
});

GroupSchema.plugin(timestamp)

module.exports = mongoose.model('Group', GroupSchema)
```

user

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema
const timestamp = require('mongoose-timestamp')

const UserSchema = new Schema({
  name: String,
  email: String,
  password: String,
  avatarUrl: String,
});

UserSchema.plugin(timestamp)
```

```
module.exports = mongoose.model('User', UserSchema)


connect

const mongoose = require('mongoose')
const { MONGO_URI } = require('../helpers/dbConfig')

module.exports = async () => {
    try {
        mongoose.connect(MONGO_URI, {
            useNewUrlParser: true,
            useUnifiedTopology: true,
        })
    } catch (error) {
        console.error(
            `${__filename}\n${__line}\nDATABASE     CONNECTION     ER-
ROR\n${error.stack.toString()}`
        )
    }

    mongoose.Promise = global.Promise
    const db = mongoose.connection

    db.on('error', error => {
        console.error(error.stack.toString())
        console.error(`${__filename} ${__line} DATABASE CONNECTION ER-
ROR`)
    })

    db.once('open', () => {
        console.log('DATABASE HAS SUCCESSFULLY BEING OPENED')
    })

    return db
}

docman-api-master\helpers
additionalInit.js
// Add special global keyword for some flexible debug
Object.defineProperty(global, '__stack', {
    get: function () {
        const orig = Error.prepareStackTrace;
        Error.prepareStackTrace = (_, stack) => stack;
        const err = new Error;
        Error.captureStackTrace(err, arguments.callee);
        const stack = err.stack;
        Error.prepareStackTrace = orig;
        return stack;
    }
});

Object.defineProperty(global, '__line', {
    get: () => __stack[1].getLineNumber()
});

Object.defineProperty(global, '__function', {
```

```
    get: () => __stack[1].getFunctionName()
});


addMockData.js
const { addOrder, deleteOrder } = require('../db/cruds/Order')
const { addRepair, deleteRepair } = require('../db/cruds/Repair')
const     {     addExpensis,     deleteExpensis     }     =     re-
quire('../db/cruds/Expensis')

module.exports = async () => {
  await deleteOrder({ _id: { $exists: true }});
  await deleteRepair({ _id: { $exists: true }});
  await deleteExpensis({ _id: { $exists: true }});

  const orders = [
    {
      date: new Date('05.12.2017'),
      startPoint: 'РћРґРµСЃСЃР°',
      destination: 'РљРёРµРІ',
      deliveryCost: '920',
      cargo: '50',
      weight: '30',
      distance: '284',
      price: '20000',
      fuelConsumption: '2439.25'
    },
    {
      date: new Date('05.13.2017'),
      startPoint: 'РЃР°СЂСЊРєРѕРІ',
      destination: 'Р›СЊРІРѕРІ',
      deliveryCost: '920',
      cargo: '50',
      weight: '30',
      distance: '296',
      price: '1200',
      fuelConsumption: '2601.59'
    },
    {
      date: new Date('05.14.2017'),
      startPoint: 'РЈР¶РіРѕСЂРѕРґ',
      destination: 'РџРѕР»С‚Р°РІР°',
      deliveryCost: '920',
      cargo: '50',
      weight: '30',
      distance: '393',
      price: '4500',
      fuelConsumption: '2891.49'
    },
    {
      date: new Date('05.15.2017'),
      startPoint: 'РЇРџСЂРѕРЅРѕ',
      destination: 'РЎС‚РјС‹',
      deliveryCost: '920',
      cargo: '50',
      weight: '30',
      distance: '119',
```

```
      price: '5000',
      fuelConsumption: '1044'
    },
  ];

  await addOrder(orders);

  const repairs = [
    {
      startedAt: new Date('05.12.2017'),
      endedAt: new Date('05.13.2017'),
      repairPrice: 5489.67,
      equipementPrice: 2489.67,
    },
    {
      startedAt: new Date('05.13.2017'),
      endedAt: new Date('05.14.2017'),
      repairPrice: 4601.44,
      equipementPrice: 2501.44,
    },
    {
      startedAt: new Date('05.14.2017'),
      endedAt: new Date('05.15.2017'),
      repairPrice: 3000,
      equipementPrice: 1521,
    },
  ];
  addRepair(repairs);

  const expensis = [
    {
      date: new Date('05.12.2017'),
      price: 2489.67,
      comment: 'РўРѕРїР»РёРІРѕ'
    },
    {
      date: new Date('05.13.2017'),
      price: 2601.44,
      comment: 'РўРѕРїР»РёРІРѕ'
    },
    {
      date: new Date('05.15.2017'),
      price: 2521,
      comment: 'Р”РµС‚Р°Р»Рь'
    },
  ];

  addExpensis(expensis);
}

button.constants.js

const USER_MENU_KEYBOARD = [
    [{ text: 'РџРѕРІС‚РёС‚Рь РїРѕСЃР»РµРґРЊ Р·Р°РєР°Р·' }],
    [{ text: 'Р—Р°РєР°Р·Р°С‚Рь РІРѕСЃСѓ' }, { text: 'Р□СЃС‚РѕСЏ
Р·Р°РєР°Р·РѕРІ' }],
    [{ text: 'Рђ РЅР°СЃ' }, { text: 'РљРѕРЅС‚Р°РєС‹' }],
```

```
]

module.exports = {
    KEYBOARD_USER_MENU: {
        parse_mode: 'markdown',
        reply_markup: {
            keyboard: USER_MENU_KEYBOARD,
        },
    },
    REMOVE_KEYBOARD: {
        reply_markup: {
            remove_keyboard: true,
        },
    },
    SEND_PHONE_NUMBER: {
        reply_markup: {
            keyboard: [
                [
                    {
                        text: 'РћС‚РїСЂР°РІРёС‚СЊ РјРѕР№ РЅРѕРјРµСЂ',
                        request_contact: true,
                    },
                ],
            ],
        },
    }
}

constants.js

module.exports = {
    ENVIRONMENT: {
        PRODUCTION: 'production',
        DEVELOPMENT: 'development',
        STAGING: 'staging',
        LOCAL: 'local',
    },
}


dbConfig.js
const ENV = process.env;

module.exports = {
    MONGO_URI:
`${ENV.DB_PROVIDER}://${ENV.DB_USER}:${encodeURIComponent(ENV.DB_PWORD
)}@${ENV.DB_HOST}:${ENV.DB_PORT}/${ENV.DB_NAME}`
}

getDefaultGroups.js


module.exports = userId => [
  {
    name: 'Shopping',
    documents: [],
    userId
```

```
  },
  {
    name: 'Bills',
    documents: [],
    userId
  },
  {
    name: 'Warrants',
    documents: [],
    userId
  },
  {
    name: 'Checks',
    documents: [],
    userId
  },
  {
    name: 'Receipts',
    documents: [],
    userId
  },
  {
    name: 'Others',
    documents: [],
    userId
  },
];
```

index.js

```
const { PORT = 9000 } = process.env
require('dotenv').config();
require('./helpers/additionalInit')

const mongoose = require('mongoose');
const { ObjectId } = mongoose.Types;

const express = require('express')
const cors = require('cors')
const bodyParser = require('body-parser')
const connectToDb = require('./db/connect')

const { getUser, addUser, updateUser } = require('./db/cruds/User')
const    {    getGroup,    addGroup,    updateGroup    }    =    re-
quire('./db/cruds/Group')
const { getDocument, addDocument } = require('./db/cruds/Document')
const getDefaultGroups = require('./helpers/getDefaultGroups')

const len = val => val.length;

const app = express()

// MONGODB ATLAS CONNECTION
connectToDb()

app.use(cors())
app.options('*', cors())
app.use(bodyParser.json())
```

```
/**
 * @route GET check
 * @route Check if server is running
 * @access Public
 */
app.get('/', (req, res) => {
  const date = new Date()

  res.send(`<h1>&copy; ${date.getFullYear()} API </h1>`)
})

/**
 * @route POST login
 * @route Login
 * @access Public
 */
app.post('/login', async (req, res) => {
  const { email = '', password = '' } = req.body;

  if (!len(email) || !len(password)) {
    return res.status(400).json({ success: false, message: 'Username
and password required' });
  }

  const [user] = await getUser({ email, password });

  if (!user) {
    return res.status(400).json({ success: false, message: 'Username
or Password is wrong' });
  }

  delete user.password;

  res.json({
    success: true,
    user,
  });
});

/**
 * @route POST register
 * @route Register
 * @access Public
 */
app.post('/register', async (req, res) => {
  const { name = '', email = '', password = '', avatarUrl = '' } =
req.body;

  if (!len(name) || !len(email) || !len(password)) {
    return res.status(400).json({ success: false, message: 'Username
and password required' });
  }

  const user = await getUser({ email });

  if (len(user)) {
```

```
    return res.status(400).json({ success: false, message: 'Nickname
already exists' });
  }

  const newUser = await addUser({
    name,
    email,
    password,
    avatarUrl
  });
  delete newUser.password;

  await addGroup(getDefaultGroups(newUser._id));

  res.json({
    success: true,
    user: newUser,
  });
});

// Add new document to the data store
app.post('/document', async (req, res) => {
  const data = req.body;

  console.log('DOCUMENT GOTTEN FROM REQUEST', data);
  data.userId = ObjectId(data.userId);
  console.log('data', data)
  await addDocument(data);

  res.json({
    success: true,
  });
});

// Get all groups in the data store
app.get('/documents', async (req, res) => {
  const { userId } = req.query;

  const documents = await getDocument({ userId }, { sort: { createdAt:
-1 } });

  res.json({ documents });
});


// Get all groups in the data store
app.get('/group', async (req, res) => {
  const userId = ObjectId(req.query.userId);

  const groups = await getGroup({ userId }, { sort: { createdAt: -1 }
});

  res.json({ groups });
});

// Get a group in the data store
app.get('/group/:groupId', async (req, res) => {
```

```
  const { groupId } = req.params;
  const userId = ObjectId(req.query.userId);

  const [group = {}] = await getGroup({ userId, _id: groupId }, {
    sort: {
      createdAt: -1
    }
  });

  const documents = [];
  for (const _id of group._doc.documents) {
    const [doc] = await getDocument({ _id: ObjectId(_id) })

    documents.push(doc);
  }
  group.documents = documents;

  res.json({ group });
});

// Get a group in the data store
app.put('/group/:groupId', async (req, res) => {
  const groupId = ObjectId(req.params.groupId);
  const documents = req.body.documents.map(d => ObjectId(d));

  await updateGroup({ _id: groupId }, {
    documents
      // $addToSet: {
      //   documents
      // }
  });

  const [group = {}] = await getGroup({  _id: groupId }, {
    sort: {
      createdAt: -1
    }
  });

  if (len(Object.values(group))) {
    const documents = [];
    for (const _id of group._doc.documents) {
      const [doc] = await getDocument({ _id: ObjectId(_id) })

      documents.push(doc);
    }
    group.documents = documents;
  }

  res.json({ group });
});

app.put('/profile', async (req, res) => {
  const { userId, name, avatarUrl } = req.body;

  if (!ObjectId.isValid(userId)) {
    return  res.status(400).json({  success:  false,  message:  'Invalid
userId' });
```

```
  }

  await updateUser({ _id: userId }, {
    name,
    avatarUrl
  });
  const [user] = await getUser({ _id: userId  });

  if (!user) {
    return res.status(400).json({ success: false, message: 'Username
or Password is wrong' });
  }

  delete user.password;

  res.json({
    success: true,
    user
  });
});

app.listen(PORT, () => {
  console.log(` Listening on port ${PORT}`);
});


Tests

const request = require('supertest');
const app = require('./index');

describe('GET /order', function() {
  it('responds with json', function(done) {
    request(app)
      .get('/order')
      .set('Accept', 'application/json')
      .expect('Content-Type', /json/)
      .expect(200, done);
  });
});

describe('GET /repair', function() {
  it('responds with json', function(done) {
    request(app)
      .get('/repair')
      .set('Accept', 'application/json')
      .expect('Content-Type', /json/)
      .expect(200, done);
  });
});

describe('GET /expensis', function() {
  it('responds with json', function(done) {
    request(app)
      .get('/expensis')
      .set('Accept', 'application/json')
      .expect('Content-Type', /json/)
```

```
      .expect(200, done);
  });
});

describe('POST /order', function() {
  it('responds with json', function(done) {
    request(app)
      .post('/order')
      .send({
        date: new Date('05.13.2017'),
        startPoint: 'РҐР°СЂСЊРєРѕРІ',
        destination: 'Р›СЊРІРѕРІ',
        deliveryCost: '920',
        cargo: '50',
        weight: '30',
        distance: '296',
        price: '1200',
        fuelConsumption: '2601.59'
      })
      .set('Accept', 'application/json')
      .expect('Content-Type', /json/)
      .expect(200)
      .end(function(err, res) {
        if (err) return done(err);
        done();
      });
  });
});

describe('POST /repair', function() {
  it('responds with json', function(done) {
    request(app)
      .post('/repair')
      .send({
        startedAt: new Date('05.12.2017'),
        endedAt: new Date('05.13.2017'),
        repairPrice: 5489.67,
        equipementPrice: 2489.67,
      })
      .set('Accept', 'application/json')
      .expect('Content-Type', /json/)
      .expect(200)
      .end(function(err, res) {
        if (err) return done(err);
        done();
      });
  });
});

describe('POST /expensis', function() {
  it('responds with json', function(done) {
    request(app)
      .post('/expensis')
      .send({
        date: new Date('05.12.2017'),
        price: 2489.67,
        comment: 'РўРѕРїР»РёРІРѕ'
```

```
      })
      .set('Accept', 'application/json')
      .expect('Content-Type', /json/)
      .expect(200)
      .end(function(err, res) {
        if (err) return done(err);
        done();
      });
  });
});
```

docman-client-master\src\components\AddDocToGroup
index

```
import React from 'react'
import { makeStyles } from '@material-ui/core/styles';
import Button from '@material-ui/core/Button';

import Table from './Table';
import config from '../../config';

const useStyles = makeStyles(() => ({
  root: {
    width: '100%',
    display: 'flex',
    alignItems: 'center',
    flexDirection: 'column'
  },
}));

export default function AddDocToGroup({ user, group, toggleAddNewDoc
}) {
  const classes = useStyles();
  const [documents, setDocuments] = React.useState([]);
  const [selectedDocs, setSelectedDocs] = React.useState([
    ...(!!group && group.documents
        ? group.documents.map(d => d._id)
        : []
    )
  ]);

  React.useEffect(() => {
    const fetchDocs = () => {
      fetch(`${config.apiUrl}/documents?userId=${user._id}`)
        .then(res => res.json())
        .then(result => {
          if (result && result.documents) {
            setDocuments(result.documents);
          }
        });
    }

    fetchDocs();
  }, []);

  const handleDoc = (documentId) => {
    if (selectedDocs.includes(documentId)) {
      setSelectedDocs(selectedDocs.filter(id => id !== documentId));
```

```
    } else {
      setSelectedDocs([
        ...selectedDocs,
        documentId
      ]);
    }
  }

  const handleFinish = () => {
    fetch(`${config.apiUrl}/group/${group._id}`, {
      method: 'PUT',
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        userId: user._id,
        documents: selectedDocs
      })
    }).then(res => res.json())
    .then(data => {
      toggleAddNewDoc();
    });
  }

  console.log('selectedDocs', selectedDocs)
  return (
    <div className={classes.root}>
      <Table
        data={documents}
        selectedDocs={selectedDocs}
        handleDoc={handleDoc}
      />
      <Button
        variant="contained"
        color="primary"
        onClick={handleFinish}
      >
        Finish
      </Button>
    </div>
  )
}
```

ДОДАТОК Б
ІНСТРУКЦІЯ КОРИСТУВАЧА

Головна форма інформаційної системи наведена на рисунку Б.1.

Рисунок Б.1 – Головна форма ІС

У додатку реалізовано функцію «допомога», за якою користувач зможе отримати необхідну інформацію для зручної роботи у системі (рис Б.2).

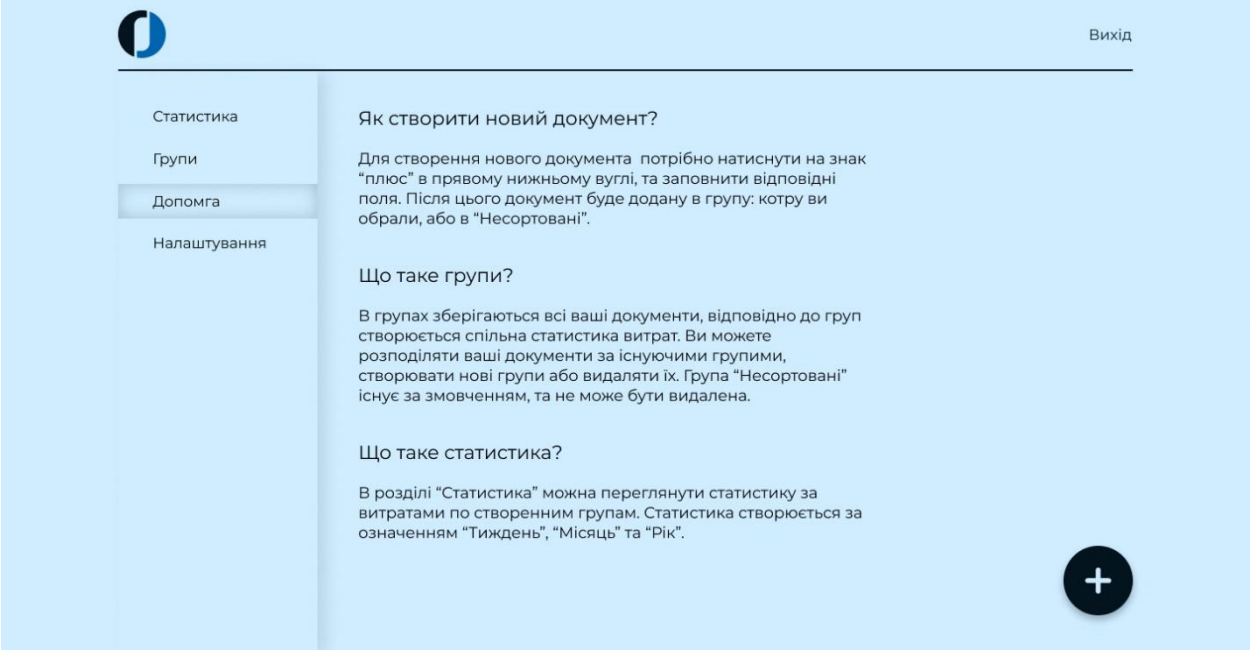

Рисунок Б.2 – Вкладка «Допомога» для зручної роботи у системі

Реєстрація користувачів проводиться за стандартизованою формою (рис Б.3). Необхідно ввести логін, пароль, повторити пароль, ім'я, прізвище, по батькові, та інші вимоги облікового запису.

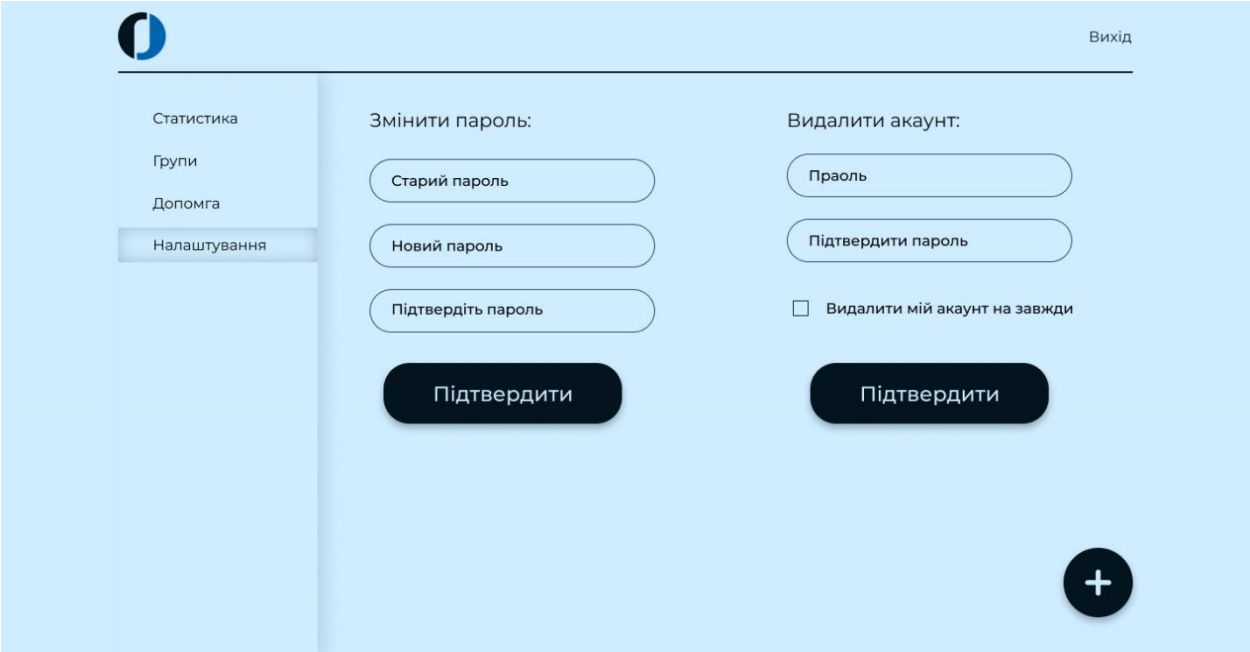

Рисунок Б.3 –Реєстрація користувачів

Після авторизації система пропоноватиме користувачу обрати групи платіжних документів, з якими він найчастіше буде мати спрау (рис.Б.4). Потрібно обрати, або додати групу документів.

Рисунок Б.4 – Групи платіжних документів користувача

Далі користувач може вводити до системи платіжний документ, який він воліє додати до власної обробки (рис. Б.5).

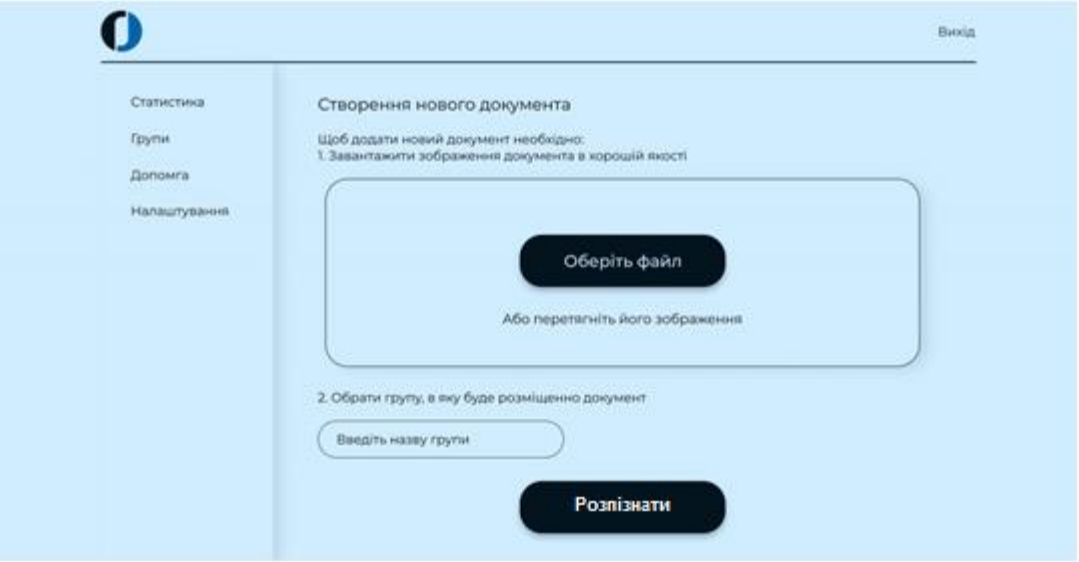

Рисунок Б.6 – Додавання та розпізнавання нового документу користувачем

За допомогою інтуітивно зрозумілого інтерфейсу, користувач завантажує новий платіжний документ, розпізнає його, та одразу може обрати групи, за якою він буде значитися.

Після обробки, система додасть документ, до необхідної групи. На рисунку Б.7 подано приклад оброблених документів, які користувач, ще не додав до жодної із груп.

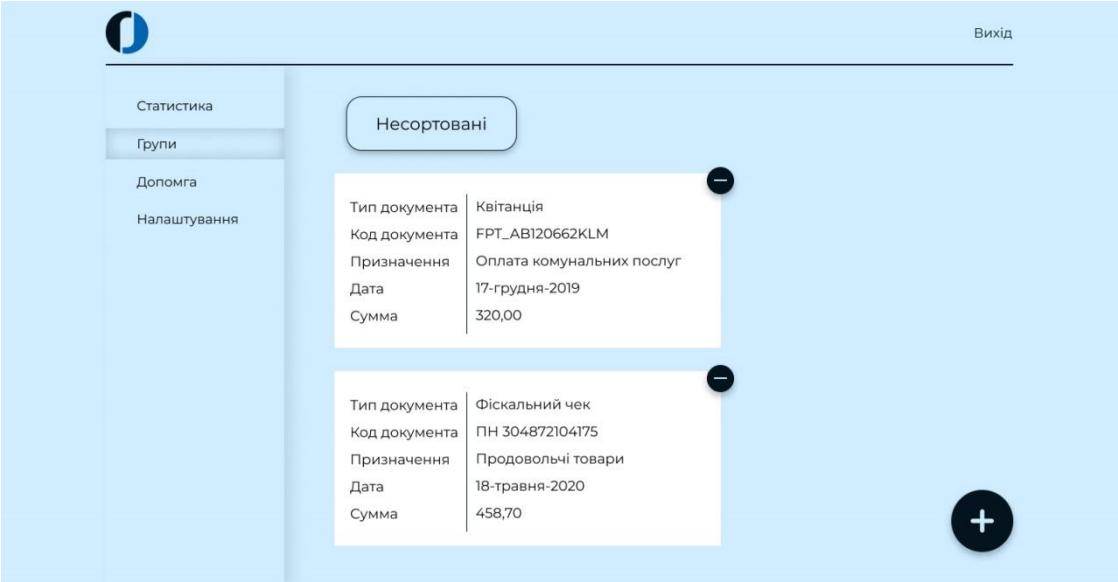

Рисунок Б.7 – Розміщення оброблених документів за групами

На рисунку Б.8 подано приклад роботи розрахункового алгоритму системи, а саме групування документів за типом, та визначення загальної суми витрат за обраний період.

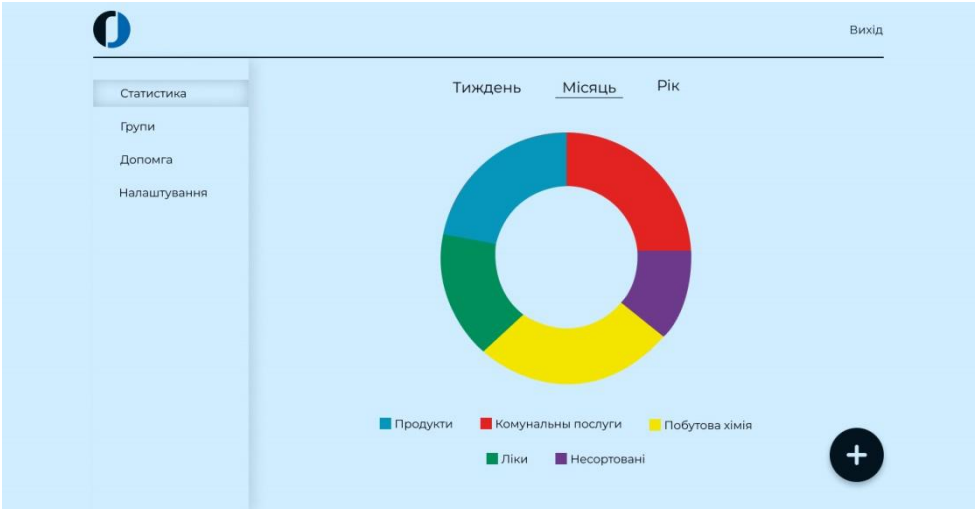

Рисунок Б.8 – Розрахунок витрат користувача за місяц