НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет Інформаційних технологій

ПОГОДЖЕНО
Декан факультету (Директор ННІ)
_____
(назва факультету (ННІ))
_____ _____
(підпис) (ім'я ПРІЗВИЩЕ)
"___" _____ 2025 р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
_____
(назва кафедри)
_____ _____
(підпис) (ім'я ПРІЗВИЩЕ)
"___" _____ 2025 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему ___Система автоматизації документообігу у сфері виконавчих проваджень з аналітичним модулем_____
_____

Спеціальність _____121 Інженерія Програмного Забезпечення_____
(код і найменування

Освітня програма ___Програмне забезпечення інформаційних систем_____
(назва)

Орієнтація освітньої програми_____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми
кандидат фізико-математичних наук, доцент _____ Віктор КІРИЧЕНКО
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Керівник магістерської кваліфікаційної роботи
кандидат технічних наук, доцент _____ Олексій ТКАЧЕНКО
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Виконав _____ _____Артем СТОЛЯРЧУК_____
(підпис) ((ім'я ПРІЗВИЩЕ здобувача)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ)
_____

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
_____
___к.т.н., доцент_____ _____ _Голуб Б.Л._
(науковий ступінь, вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)
"____" _____2025 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

_____Столярчук Артем Олегович_____
(прізвище, ім'я, по батькові)

Спеціальність_____121 – Інженерія програмного забезпечення _____
(код і найменування)

Освітня програма _____Програмне забезпечення інформаційних систем_____
(назва)

Орієнтація освітньої програми_____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи_Система автоматизації документообігу у сфері виконавчих проваджень з аналітичним модулем_____
_____

затверджена наказом від "1" листопада 2024р. № 1963

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи нормативно-правова база України у сфері виконавчих проваджень, приклади систем електронного документообігу, база даних, у PostgreSQL, сучасні підходи до побудови web-застосунків (ASP.NET Core MVC, EF Core).

Перелік питань, що підлягають дослідженню:

1. Аналіз проблем ручного документообігу у сфері виконавчих проваджень та визначення вимог до автоматизованої системи
2. Проектування архітектури системи на основі ASP.NET Core MVC з використанням PostgreSQL і EF Core.
3. Розробка модуля управління документами та обліку виконавчих дій
4. Реалізація аналітичного модуля для оцінки ефективності виконання рішень, навантаження на виконавців та виявлення закономірностей у роботі.
5. Дослідження можливості інтеграції з іншими інформаційними системами.

Перелік графічного матеріалу (за потреби) _____
_____

Дата видачі завдання "_____"_____20___ р.

Керівник магістерської кваліфікаційної роботи _____ _____
(підпис) (ім'я ПРІЗВИЩЕ)

Завдання прийняв до виконання _____ _____
(ім'я ПРІЗВИЩЕ)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АСВП — Автоматизована система виконавчого провадження

ЄДР (ЕДР) — Єдиний державний реєстр юридичних осіб, фізичних осіб-підприємців та громадських формувань

ДРРП — Державний реєстр речових прав на нерухоме майно

API — Application Programming Interface (інтерфейс програмування застосунків)

MVC — Model–View–Controller (архітектурний шаблон)

BPMN — Business Process Model and Notation (стандарт моделювання бізнес-процесів)

IDEF0 — Integration Definition for Function Modeling (методологія функціонального моделювання)

KPI — Key Performance Indicator (ключовий показник ефективності)

OLAP — Online Analytical Processing (технологія оперативної аналітичної обробки даних)

ETL — Extract, Transform, Load (процеси завантаження даних)

BI — Business Intelligence (засоби бізнес-аналітики)

GDPR — General Data Protection Regulation (Загальний регламент захисту даних ЄС)

CRUD — Create, Read, Update, Delete (базові операції з даними)

SQL — Structured Query Language (мова структурованих запитів)

ВСТУП

Актуальність теми. Сучасний етап розвитку України нерозривно пов'язаний з процесами глобальної цифрової трансформації, що охоплює усі сфери державного управління та надання публічних послуг. Однією з найбільш консервативних та водночас критично важливих галузей є система юстиції, зокрема сфера примусового виконання судових рішень. Виконавче провадження в Україні традиційно супроводжується великим обсягом паперової документації, а процеси, що склалися історично, здебільшого орієнтовані на ручне опрацювання. Такий підхід неминуче призводить до значних затримок, накопичення операційних помилок та, як наслідок, суттєвого зниження загальної ефективності процесу.

Ключовий бізнес-процес, що охоплює весь життєвий цикл справи — від моменту подання заяви стягувачом до її фактичного завершення виконавцем — вимагає суворого та прозорого контролю статусів, дотримання чітко регламентованих процесуальних дедлайнів та коректної регіональної належності.

Аналіз поточного стану показує, що існуючі в Україні інформаційні системи, такі як Автоматизована система виконавчих проваджень (АСВП), хоча і вирішують важливі задачі реєстрації та часткової автоматизації (наприклад, автоматизований арешт банківських рахунків), проте вони є фрагментарними та не пропонують комплексного, інтегрованого інструменту для *управління робочим процесом* (Workflow) самого виконавця. Відсутність сучасних, гнучких IT-рішень у цій сфері значно утруднює об'єктивний моніторинг ефективності, поглиблений аналіз вузьких місць та прийняття обґрунтованих управлінських рішень. Таким чином, розробка спеціалізованої системи автоматизації документообігу класу Workflow з інтегрованим аналітичним модулем є надзвичайно актуальною науково-прикладною задачею.

Мета роботи полягає у науково-технічному обґрунтуванні та створенні архітектурних і програмних рішень для підвищення ефективності виконавчого провадження. Це досягається шляхом **розробки та впровадження програмного прототипу** веб-орієнтованої системи «Сокіл». Ця система покликана комплексно автоматизувати облік та рух документів на всіх етапах життєвого циклу виконавчої справи, забезпечити прозорий моніторинг статусів справ в реальному часі та надати керівництву аналітичні інструменти, включаючи підтримку OLAP-технологій, для оцінки ефективності роботи.

Для досягнення поставленої мети було визначено та послідовно вирішено такі **завдання**:

- Провести глибокий системний аналіз предметної області виконавчого провадження, детально вивчити його нормативно-правову структуру та декомпозувати ключові бізнес-процеси.
- Виконати всебічний порівняльний аналіз існуючих в Україні рішень (зокрема, АСВП) та релевантного міжнародного досвіду автоматизації, виявити їх переваги та недоліки у форматі порівняльної таблиці.
- Здійснити класифікацію та систематизацію ключових операційних, технічних та безпекових бар’єрів, що перешкоджають ефективній цифровізації даної галузі.
- Розробити деталізований набір функціональних та нефункціональних вимог до проектованої програмної системи.
- Спроектувати концептуальну та логічну архітектуру системи, включаючи розробку моделей даних (ER-діаграми), моделювання поведінки та структури за допомогою діаграм UML (прецедентів, класів, станів, послідовності) та проектування логічної схеми гібридного аналітичного модуля (OLTP+OLAP).
- Здійснити практичну програмну реалізацію прототипу системи «Сокіл», використовуючи сучасну архітектуру MVC (на базі фреймворку Laravel

12.x), та інтегрувати інтерактивні аналітичні дашборди (на базі бібліотеки Chart.js).

- Провести комплексну верифікацію та валідацію прототипу шляхом тестування на репрезентативному синтетичному наборі даних (1 500 справ) та оцінити його вплив на ключові показники ефективності процесу.

Об’єкт дослідження — комплексний процес управління виконавчими провадженнями та пов'язаним з ним документообігом в органах виконання рішень, в усій його сукупності від ініціації до архівування.

Предмет дослідження — моделі, методи та інформаційні технології для створення веб-орієнтованої автоматизованої системи документообігу «Сокіл», з акцентом на інтегрованому аналітичному модулі для підтримки прийняття управлінських рішень.

Методи дослідження. Методологічну основу дослідження склала сукупність загальнонаукових та спеціальних методів. Було використано методи системного аналізу для декомпозиції бізнес-процесів виконавчої служби; об'єктно-орієнтоване проектування та мова UML для моделювання архітектури; методи моделювання баз даних (ER-діаграми) для проектування сховища; а також технології сучасної веб-розробки (PHP, Laravel, SQL) для практичної реалізації прототипу. Для аналізу даних використано методи SQL-агрегації та фундаментальні принципи OLAP-технологій.

Наукова новизна. Наукова новизна даної роботи полягає у вирішенні актуальної науково-прикладної задачі шляхом **розробки програмного прототипу «Сокіл»** — комплексної системи, що поєднує:

1. Практичну реалізацію повного життєвого циклу управління справою (від applicant до executor) в рамках єдиної сучасної MVC-архітектури, адаптованої до специфіки українського виконавчого провадження.
2. Впровадження гнучкого гібридного аналітичного модуля, що забезпечує як оперативний моніторинг ключових показників ефективності (KPI) в

реальному часі (на базі Chart.js та SQL), так і підтримку поглибленого багатовимірного аналізу через інтеграцію з зовнішнім OLAP-джерелом.

Практичне значення одержаних результатів. Розроблений прототип системи «Сокіл» є готовим програмним рішенням, що доводить життєздатність обраної архітектури та може бути впроваджене в роботу державних або приватних виконавців для автоматизації обліку справ. Тестування на 1500 провадженнях підтвердило, що система забезпечує прозоре та централізоване ведення справ, кардинально підвищує ефективність роботи виконавців завдяки оперативному доступу до інформації та надає керівництву потужні інструменти для виявлення "вузьких місць" і об'єктивної оцінки навантаження. Система також генерує готові звітні документи у форматах PDF та Excel.

Апробація результатів. Основні положення та практичні результати дослідження пройшли апробацію, були представлені на студентській науково-практичній конференції та опубліковані у тезах «Система автоматизації документообігу у виконавчому провадженні з аналітичним модулем “Сокіл”».

Структура роботи. Магістерська робота має логічну та послідовну структуру і складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. **Перший розділ** присвячено глибокому системному аналізу предметної області, огляду існуючих рішень та постановці задачі. У **другому розділі** проведено детальне моделювання майбутньої системи з використанням діаграм UML та проектуванням архітектури баз даних (OLTP та OLAP). У **третьому розділі** описано програмну реалізацію, обрані технології та архітектурні рішення. **Четвертий розділ** демонструє результати тестування прототипу та оцінку його ефективності. У **висновках** підсумовано виконання поставлених завдань. У **додатках** наведено допоміжні матеріали, зокрема ключові фрагменти програмного коду.

1. АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Опис предметної області та правові засади

Виконавче провадження в Україні посідає особливе місце в національній системі права, виступаючи завершальною та іманентно обов'язковою стадією судового процесу. Його фундаментальна мета полягає у забезпеченні фактичної реалізації рішень судів та інших уповноважених органів, що чітко регламентовано положеннями Закону України «Про виконавче провадження». Необхідно підкреслити, що саме на цьому, по суті фінальному, етапі реалізується конституційне право особи на справедливий та ефективний суд. Адже навіть найбільш обґрунтоване, виважене та ретельно винесене судове рішення, яке набрало законної сили, не матиме жодної практичної цінності та перетвориться на декларацію, якщо механізм його реального виконання відсутній або функціонує неналежним чином. [1]

Виходячи з цього, ефективність функціонування системи виконавчого провадження є одним із ключових індикаторів не лише стану правосуддя, але й загального рівня верховенства права в державі, демонструючи її інституційну спроможність та рівень суспільної довіри до судової гілки влади.

Предметна область виконавчого провадження охоплює комплекс процесуальних та практичних дій, що здійснюються уповноваженими суб'єктами — державними та приватними виконавцями — і спрямовані на примусове виконання судових та інших рішень. Чинне законодавство детально визначає етапність цього складного процесу, суворо регламентує усі процедури, процесуальні строки, вичерпні повноваження виконавців, а також гарантії прав усіх сторін провадження (стягувача, боржника та інших учасників).

Структурно, виконавче провадження складається з низки чітко окреслених та послідовних стадій:

- **Відкриття провадження:** ініціація процесу на підставі отримання виконавчого документа, ретельна перевірка його відповідності формальним вимогам (реквізитам) та винесення процесуального рішення — постанови про відкриття.
- **Збір інформації (виявлення активів):** активна фаза, що включає направлення формалізованих запитів до ключових державних реєстрів (зокрема, ЄДР, ДРРП, автоматизованого реєстру боржників, банківських реєстрів) з метою встановлення актуального майнового стану боржника та джерел його доходів.
- **Застосування заходів примусового виконання:** накладення обтяжень з метою унеможливлення відчуження активів, зокрема арешт грошових коштів на рахунках, арешт рухомого та нерухомого майна, а також встановлення обмежень у користуванні таким майном.
- **Примусова реалізація майна:** у випадку відсутності достатніх коштів, здійснюється процедура продажу виявлених та заарештованих активів боржника через прозорі електронні аукціони (наприклад, СЕТАМ).
- **Розподіл стягнутих коштів:** пропорційне або пріоритетне перерахування виручених сум на користь стягувача (стягувачів) відповідно до встановленої черговості.
- **Завершення провадження:** фіналізація процесу шляхом винесення відповідної постанови про завершення (у разі повного виконання) або закінчення провадження (за наявності інших законних підстав).

Варто зауважити, що кожна із перелічених стадій включає в себе як інтелектуальні (аналіз документів, прийняття процесуальних рішень), так і фізичні дії (вихід на об'єкт, опис та арешт майна). Це вимагає від виконавця неухильного дотримання процесуальних строків, ведення деталізованого документообігу та обов’язкової фіксації всіх етапів у відповідних журналах або, що більш сучасно, в інформаційних системах.

Проте, незважаючи на наявність централізованої державної Автоматизованої системи виконавчого провадження (АСВП), значна частина операційних процесів досі залишається неавтоматизованою, спираючись на паперовий документообіг та ручну працю.[6] Такий стан речей неминуче спричиняє бюрократичні затримки, підвищує ризики людських помилок, призводить до дублювання інформації в різних джерелах та суттєво знижує загальну прозорість і підконтрольність процесів.

Додатковим каталізатором змін стали глобальні виклики — пандемія COVID-19 та особливо повномасштабне військове вторгнення Російської Федерації. Ці безпрецедентні обставини екстремально актуалізували гостру потребу у впровадженні інструментів віддаленої роботи, тотальній цифровізації процесів та мінімізації фізичного людського контакту для забезпечення безперервності роботи системи правосуддя. Попри помітний прогрес у впровадженні нових цифрових можливостей у сфері електронного судочинства (в рамках ЄСІТС), виконавче провадження залишається інформаційно фрагментованим і недостатньо інтегрованим із суміжними державними реєстрами та системами. Ця системна проблема формує нагальну, об'єктивну потребу в розробці та впровадженні сучасної, комплексної, гнучкої та аналітично потужної інформаційної системи нового покоління.

1.2 Огляд існуючих рішень та технологічних підходів

Ефективне та обґрунтоване проектування нової системи автоматизації для сфери виконавчого провадження вимагає проведення глибокого, всебічного аналізу поточних рішень, релевантних міжнародних практик та доступних на ринку технологічних підходів. Такий аналіз повинен охоплювати як державні, так і комерційні та міжнародні системи, а також ті архітектурні й аналітичні підходи, які вже довели свою практичну ефективність у подібних за складністю та масштабом процесах[1].

- **1.2.1 Міжнародні практики автоматизації виконавчих проваджень**

Критичний аналіз світових моделей цифровізації виконавчих процесів, зокрема у країнах Європейського Союзу, демонструє широкий спектр ефективних підходів, що можуть слугувати орієнтирами:

- **Естонія:** Ця країна традиційно демонструє високий рівень цифровізації. Їхня національна платформа e-Justice уможливлює практично стовідсотково електронний документообіг між судовими інстанціями, виконавцями та громадянами. Функціональні електронні кабінети сторін надають вичерпні можливості для перегляду статусів проваджень у реальному часі та здійснення необхідних платежів онлайн.
- **Німеччина та Польща:** У цих юрисдикціях основний акцент цифрової трансформації зроблено на забезпеченні максимальної прозорості процесу реалізації майна боржників через обов'язкові електронні аукціони.[18] Процедури цифрових торгів є чітко та детально врегульованими на законодавчому рівні, що забезпечує не лише прозорість, але й об'єктивне зростання середньої вартості реалізації активів.
- **Франція та Нідерланди:** Ці країни активно експериментують із впровадженням інструментів штучного інтелекту (AI) у виконавчі процеси — від побудови предикативних моделей для прогнозування поведінки боржників до використання алгоритмів для пошуку прихованих або неявних активів. При цьому особливий наголос робиться на відповідальному використанні AI відповідно до етичних принципів, розроблених, зокрема, CEPEJ (Європейська комісія з питань ефективності правосуддя).
- **Фінляндія:** Демонструє фокус на мобільності та клієнт орієнтованості, розвиваючи мобільні додатки. Вони дають змогу як боржникам, так і стягувачам отримувати оперативні консультації, відстежувати хід провадження та, що важливо, здійснювати добровільні платежі у зручний спосіб.

Узагальнюючи, ці передові моделі демонструють ключові стратегічні орієнтири для будь-якої реформи у цій сфері: максимальна прозорість, глибока автоматизація рутинних операцій, мобільність доступу, повна інтегрованість (Інтер операбельність) систем та потужна аналітична складова.

- **1.2.2 Аналіз аналогічних комерційних систем**

Для розуміння принципів побудови процесних систем, доцільно розглянути системи, що мають подібну бізнес-логіку, а саме CRM-платформи (Customer Relationship Management) та системи класу Service Desk / Helpdesk.

- **CRM-системи (напр., 1С, MS Dynamics):** Ці платформи є сильними у веденні фінансового обліку, управлінні комунікаціями та відстеженні історії взаємодії, але вони абсолютно не придатні для ведення суворого юридичного процесуального документообігу, який обмежений жорсткими нормативними вимогами.
- **Helpdesk / Service Management (напр., Jira Service Management):** Такі системи чудово працюють з управлінням життєвим циклом завдань (тікетів), їх статусами, призначенням виконавців та моніторингом дотримання SLA (Service Level Agreement). Однак вони так само не мають необхідного юридичного функціонала та не інтегровані з державними реєстрами.
- **АСВП та ЄСІТС:** Як вже зазначалося, це основні державні системи, проте вони не забезпечують повноцінного управління внутрішнім робочим процесом приватного виконавця (як суб'єкта господарювання) та страждають від фрагментованої, неповної інтеграції між собою.

1.2.3 Принципові технологічні підходи

Проведений аналіз проблем предметної області та функціональних обмежень існуючих рішень дозволив окреслити ключові технологічні основи, що мають бути закладені в архітектуру проектованої системи «Сокіл»:

- **MVC-архітектура (на базі фреймворку Laravel):** Використання патерну Model-View-Controller забезпечує необхідну структурованість

коду, його високу розширюваність, модульність та полегшує подальшу підтримку системи.

- **API Gateway (Шлюз додатків):** Цей підхід є критично важливим для уніфікації та стандартизації процесів інтеграції з численними зовнішніми державними реєстрами та іншими сторонніми системами, виступаючи єдиною точкою входу.
- **Гібридна аналітика (OLTP + OLAP):** Поєднання транзакційної обробки даних (OLTP) для оперативної роботи та аналітичної обробки (OLAP) дозволяє одночасно отримувати миттєві показники ефективності та проводити глибокий багатовимірний аналіз накопичених великих даних для прийняття стратегічних рішень.
- **1.2.4 Порівняльний аналіз рішень**

Проведене порівняння наявних на ринку державних та комерційних систем із концепцією проектованої системи «Сокіл» виявляє суттєві відмінності. «Сокіл» орієнтований на досягнення вищого рівня масштабованості, архітектурної гнучкості, значно більшої глибини аналітики та надання спеціалізованих інструментів управління саме юридичними процесами, що наразі відсутнє у конкурентних рішень в єдиному комплексі.

1.3 Постановка завдання

Системно аналізуючи предметну область виконавчого провадження та існуючі цифрові рішення (як державні, так і комерційні аналоги), можна чітко ідентифікувати та виділити низку ключових, взаємопов'язаних проблем, що потребують вирішення:

- відсутність єдиної, цілісної стратегії цифровізації саме для приватних виконавців;
- домінування ручних, паперових операцій та, як наслідок, часте дублювання даних у різних системах обліку;
- висока трудомісткість та складність технічної взаємодії з численними державними реєстрами;

- хронічна неузгодженість та відсутність синхронізації інформаційних довідників;
- високі ризики інформаційної безпеки та потенційного витоку чутливих персональних даних через децентралізоване зберігання;
- критична недостатність вбудованих аналітичних інструментів для оцінки ефективності та планування.

На основі цього комплексу виявлених проблем було чітко сформульовано головну мету даної роботи — створити функціональний прототип інформаційної системи «Сокіл». Ця система має на меті автоматизувати наскрізний документообіг, глибоко інтегрувати аналітичні інструменти в повсякденну роботу та забезпечити комплексне, прозоре управління повним життєвим циклом виконавчої справи в єдиному цифровому середовищі.

1.4 Формулювання вимог до системи

Усі вимоги до проектованої системи, відповідно до стандартів інженерії програмного забезпечення, поділяються на дві основні категорії: функціональні та нефункціональні.

Функціональні вимоги детально описують, *що* саме система повинна робити. Вони охоплюють модулі управління справами (повний цикл), управління користувачами та розмежуванням доступу, ведення електронного документообігу, розширений модуль аналітики та звітності, механізми інтеграції з OLAP-кубами та комплексну систему сповіщень для користувачів.

Нефункціональні вимоги визначають, *як* система повинна функціонувати, встановлюючи критерії якості. До них належать вимоги до продуктивності (швидкодія), інформаційної безпеки (захист даних), масштабованості (здатність витримувати зростання навантаження), супроводжуваності (легкість внесення змін) та тестованості.

Важливо зазначити, що усі вимоги сформульовані відповідно до сучасних стандартів розробки веб-систем корпоративного рівня та

максимально повно відображають актуальні потреби та практичні виклики, що стоять перед учасниками реального виконавчого процесу в сучасних умовах України.

2. МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Загальні відомості

Уніфікована мова моделювання (UML) — це стандартна мова для візуалізації, специфікації, проектування та документування програмних систем. У межах цієї магістерської роботи UML використовується для формального опису предметної області виконавчого провадження та програмної реалізації системи супроводу справ. За допомогою структурних діаграм (класів, компонентів, розгортання) описуються основні сутності (справа, виконавець, заявник, документ, дія) та взаємозв’язки між ними, а також структура програмних модулів і фізична інфраструктура (сервер додатка, база даних, файлове сховище). Діаграми поведінки (варіантів використання, послідовності, діяльності, станів) відображають сценарії роботи користувачів, послідовність операцій та життєвий цикл виконавчої справи. Таким чином UML забезпечує єдине формалізоване уявлення про систему для розробників, аналітиків і представників виконавчої служби, полегшуючи розробку, тестування, супровід та слугуючи основою технічної документації.

Інтелектуальний аналіз даних у роботі застосовується для виявлення закономірностей у масивах даних виконавчих проваджень і підтримки управлінських рішень. На основі оперативної бази (реєстр справ, журнал дій, довідники) формується вибірка, яка проходить очищення, узгодження форматів та побудову похідних показників (тривалість провадження, кількість дій, частка стягнення тощо). Далі використовуються методи класифікації, кластеризації, регресії та правила асоціацій для прогнозування ймовірності своєчасного завершення справи, групування схожих проваджень і оцінки зв’язку між застосованими заходами та результатом. Якість моделей

перевіряється за допомогою стандартних метрик і перехресної перевірки, а результати візуалізують у вигляді діаграм та інформаційних панелей для подальшого використання фахівцями виконавчої служби. Поєднання UML-моделювання та інтелектуального аналізу даних забезпечує комплексний підхід: UML описує структуру й поведінку системи, а Data Mining перетворює накопичені дані на практичні рекомендації щодо підвищення ефективності виконавчого провадження.

2.2 Об’єктне та функціональне моделювання

2.2.1 Діаграма прецедентів

Діаграма варіантів використання для системи «Сокіл» є інструментом Уніфікованої мови моделювання (UML), який наочно демонструє, як різні категорії користувачів і зовнішні підсистеми взаємодіють із розробленою системою. Вона відображає основні функції, доступні користувачам, і забезпечує високо рівневі уявлення про те, які задачі можуть виконувати учасники процесу: від створення та супроводу виконавчих справ до формування звітності й перегляду аналітики KPI/SLA. При цьому діаграма фокусується на поведінці системи з точки зору користувача, не деталізуючи внутрішні алгоритми й технічну реалізацію.

Ключовими елементами діаграми є актори, варіанти використання та зв’язки між ними. У ролі акторів виступають «Адміністратор», «Виконавець», «Керівник» та зовнішній «OLAP-модуль». Адміністратор має розширені повноваження: здійснює авторизацію користувачів, керування обліковими записами, створення нових справ, призначення виконавців, перегляд і пошук справ, а також ініціює генерацію звітів у форматах PDF/Excel і перегляд аналітики. Окремий актор «Виконавець» взаємодіє з варіантами використання «Додавання документів» та «Фіксація дій у справі», відображаючи операційну роботу з конкретним провадженням. Актор «Керівник» разом з «OLAP-модулем» пов’язаний із варіантом використання «Перегляд аналітики

(KPI/SLA)», що демонструє доступ керівництва до агрегованих показників ефективності.

Зв’язки між елементами діаграми уточнюють логіку використання системи. Лінії асоціації з’єднують акторів з відповідними варіантами використання, фіксуючи пряму взаємодію. Для варіантів «Створення справи» та «Перегляд/пошук справ» введено зв’язок «включає» з варіантом «Валідація даних/файлів», що відображає обов’язковий виклик механізмів перевірки коректності введених реквізитів і прикріплених документів у межах цих сценаріїв. Між ролями «Виконавець» і «Адміністратор» показано відношення узагальнення («розширює»), яке вказує, що адміністратор успадковує типові функції виконавця та доповнює їх адміністративними повноваженнями. У сукупності така діаграма структурує функціональні можливості системи «Сокіл» і слугує зрозумілим засобом комунікації між розробниками, замовником та майбутніми користувачами.

Спроектована діаграма прецедентів представлена на рис.1.

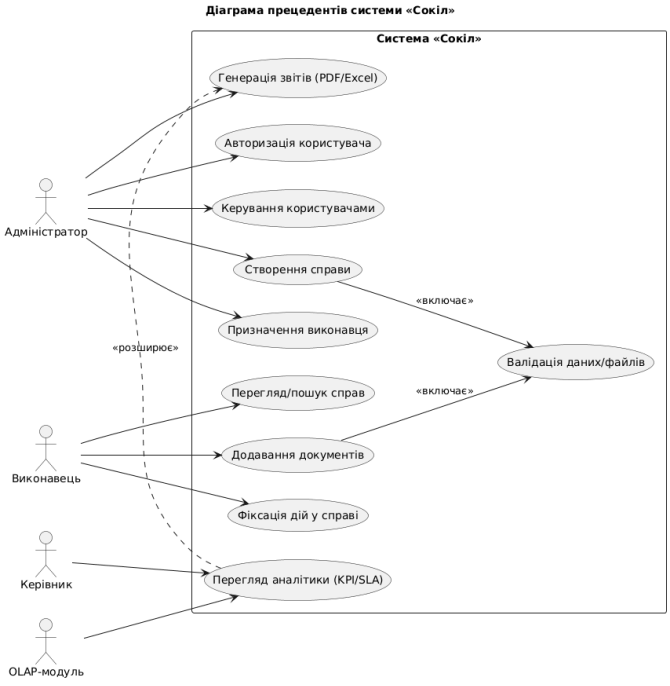

Рис. 1 Діаграма прецедентів

Створена діаграма прецедентів системи «Сокіл» містить акторів

- «Адміністратор»;
- «Виконавець»;
- «Керівник»;
- «OLAP-модуль».

Актор «Адміністратор» включає такі прецеденти:

- авторизація користувача;
- керування користувачами;
- створення справи;
- призначення виконавця;
- перегляд/пошук справ;
- генерація звітів (PDF/Excel);
- перегляд аналітики (KPI/SLA).

Актор «Виконавець» включає такі прецеденти:

- додавання документів;
- фіксація дій у справі.
- Актор «Керівник» включає прецедент:
- перегляд аналітики (KPI/SLA).
- Актор «OLAP-модуль» також пов’язаний з прецедентом:
- перегляд аналітики (KPI/SLA).

Окремий прецедент «Валідація даних/файлів» використовується як підпрецедент: він «включається» під час створення нової справи та під час перегляду/пошуку справ, забезпечуючи перевірку коректності введених реквізитів та завантажених документів. Роль «Виконавець» задається як розширення ролі «Адміністратор», що відображає успадкування базових прав

доступу з додатковою функціональністю, пов’язаною з безпосереднім веденням виконавчих справ. Прецеденти певним чином залежать один від одного: операції з документами та фіксацією дій спираються на попереднє створення й призначення справ, а звітність та аналітика використовують накопичені дані про виконані дії.

Розглянемо детальніше вищеописані прецеденти.

Назва прецеденту: Створення справи

Актор: Адміністратор

Опис:

Адміністратор має на меті зареєструвати в системі нову справу виконавчого провадження, щоб надалі забезпечити її супровід виконавцем, відстеження стану та формування звітності. У межах прецеденту формується картка справи з основними реквізитами, виконується перевірка даних і підготовка справи до призначення конкретному виконавцю.

Передумови:

Адміністратор має дійсний обліковий запис у системі «Сокіл» і необхідні права доступу. Наявні вхідні дані щодо нової справи (реквізити документів, сторін провадження, суми стягнення тощо). Система доступна та функціонує в штатному режимі.

Основний потік:

1. Адміністратор входить у систему «Сокіл», використовуючи свої облікові дані.
2. Після успішної авторизації переходить до розділу «Справи» на головній панелі.
3. Адміністратор обирає опцію «Створити справу», у результаті чого відкривається форма реєстрації нової справи.
4. У формі адміністратор заповнює основні поля: номер документа-підстави, сторони провадження, тип справи, суму стягнення, дату відкриття тощо.

5. Система ініціює прецедент «Валідація даних/файлів», перевіряючи коректність заповнення обов’язкових полів, формат дат і числових значень, а також цілісність прикріплених файлів.
6. У разі успішної валідації система формує попередній запис справи та відображає зведену інформацію для перегляду.
7. Адміністратор за потреби коригує окремі поля (наприклад, уточнює категорію справи або додаткові атрибути).
8. Після підтвердження адміністратор зберігає нову справу в системі.
9. Система присвоює справі унікальний ідентифікатор, встановлює початковий статус (наприклад, «нова») і відображає її в загальному переліку справ.
10. За необхідності адміністратор одразу переходить до прецеденту «Призначення виконавця» для подальшого супроводу справи.

Альтернативні потоки:

– Якщо під час валідації виявлено помилки (незаповнені обов’язкові поля, некоректний формат даних, відсутній або пошкоджений файл), система відображає повідомлення з переліком проблем і пропонує адміністратору повернутися до форми для виправлення даних.

– Якщо під час збереження справи виникають технічні збої (наприклад, тимчасова недоступність бази даних), система інформує адміністратора та пропонує повторити операцію пізніше, не втрачаючи вже введені дані.

Назва прецеденту: Перегляд аналітики (KPI/SLA)

Актори: Керівник, OLAP-модуль, Адміністратор

Опис:

Керівник та інші уповноважені користувачі (зокрема адміністратор, який має доступ до звітного модуля) використовують прецедент «Перегляд аналітики (KPI/SLA)» для оцінювання ефективності роботи виконавців і стану виконання справ. OLAP-модуль забезпечує агрегування даних, побудову зрізів

та візуалізацію ключових показників, таких як кількість завершених справ, середня тривалість провадження, виконання цільових SLA тощо.

Передумови:

У системі накопичено достатній обсяг даних про зареєстровані справи, дії у справах та їх поточний статус. OLAP-модуль налаштований і має доступ до відповідних таблиць сховища даних. Керівник і адміністратор мають права доступу до аналітичного модуля.

Основний потік:

1. Керівник входить у систему «Сокіл» та переходить до розділу аналітики.
2. Система звертається до OLAP-модуля для формування початкового аналітичного подання (наприклад, загальний огляд KPI за поточний період).
3. Керівник обирає потрібний тип аналітичного звіту: за виконавцями, за категоріями справ, за регіонами або за періодами часу.
4. Система застосовує вибрані фільтри та формує відповідні агреговані показники (кількість відкритих і закритих справ, середній час опрацювання, відсоток дотримання SLA тощо).
5. Результати відображаються у вигляді таблиць, діаграм або інформаційних панелей, що дозволяє швидко оцінити поточний стан виконання.
6. Керівник за потреби деталізує дані, провалюючись від зведених показників до окремих груп справ або конкретних проваджень.
7. На основі отриманої інформації керівник може ухвалювати управлінські рішення: перерозподіл навантаження між виконавцями, коригування пріоритетів, ініціювання додаткових перевірок тощо.
8. За необхідності аналітичний звіт експортується у формат PDF/Excel або зберігається в системі для подальшого використання.

Альтернативні потоки:

– Якщо OLAP-модуль тимчасово недоступний, система повідомляє користувача про неможливість сформувати динамічну аналітику та пропонує скористатися попередньо збереженими звітами.

– Якщо користувач задає некоректні комбінації фільтрів (наприклад, пустий період або несумісні параметри), система відображає попередження та пропонує скоригувати налаштування запиту.

2.2.2 Діаграма класів.

Діаграма класів є важливою частиною уніфікованої мови моделювання (UML), яка візуально відображає статичну структуру спроєктованої в магістерській роботі системи «Сокіл» для підтримки процесів виконавчого провадження. На ній показано основні класи предметної області та елементи архітектурного шаблону MVC: сутності домену (моделі Eloquent) й контролери веб-рівня. Діаграма фіксує їхні атрибути, операції (методи), а також зв’язки між ними. Це є основою для розуміння структури програмної системи, розподілу відповідальності між компонентами та подальшого документування реалізованого програмного забезпечення.

Усі доменні класи згруповано в пакет Models (Eloquent) та позначено стереотипом <<entity>>, що підкреслює їхню роль як об’єктів предметної області, які безпосередньо відображаються на таблиці транзакційної бази даних. Клас «Посада» (Position) містить атрибути id та назва і використовується як довідник посад, а метод користувачі() визначає зв’язок «один до багатьох» із класом «Користувач» (User). Клас «Користувач» описує обліковий запис виконавця чи керівника: атрибути id, ім’я, email, роль, посада_id та навігаційні методи справи(), дії(), документи(), які повертають пов’язані об’єкти відповідних сутностей.

Клас «Справа» (Case) є центральною сутністю системи та містить атрибути id, власник_id, виконавець_id, статус, регіон, дедлайн. Методи дії() і документи()

реалізують зв’язки з класами «Дія» (Action) і «Документ» (Document). Клас «Документ» зберігає метадані файлів, що приєднані до справи (id, справа_id, назва, mime_type, шлях). Клас «Дія» фіксує історію виконання у справі (атрибути id, справа_id, користувач_id, тип, створено), що дає змогу відстежувати всі значимі операції виконавця. Таким чином, на рівні моделей формується повна об’єктна модель виконавчого провадження з прив’язкою до користувача, посади, справи, дій та документів.

Група Controllers об’єднує класи, позначені стереотипом <<controller>>, які відповідають за керування сценаріями роботи користувача та координацію доступу до моделей. «Контролер_Справ» реалізує сценарії CRUD для сутності «Справа» та пов’язаних об’єктів: методи store(), show(), addAction(), uploadDocument() відповідають за створення та перегляд справи, реєстрацію дій і завантаження документів. «Контролер_Аналітики» забезпечує отримання агрегованих показників KPI/SLA та аналітичних вибірок через методи index(), records(), loadOlapSummary(), використовуючи моделі «Справа», «Дія» та інші джерела даних.

Зв’язки на діаграмі класів відображають як структурні відношення між сутностями, так і залежності між контролерами та моделями. Асоціація між класами «Посада» та «Користувач» з кратністю 1…N демонструє, що одна посада може бути призначена багатьом користувачам. Між класом «Користувач» та «Справа» зображено асоціації «власник/виконавець», які фіксують відповідальність користувача за конкретні провадження. Клас «Справа» пов’язаний відношенням «один до багатьох» з класами «Дія» та «Документ», що інтерпретується як агрегація: справа містить набір дій і документів, але окремі дії й файли можуть бути змінені або видалені без знищення самої справи.

Пунктирні стрілки від контролерів до моделей позначають відношення залежності: контролери викликають методи моделей для реалізації бізнес-

функцій. Наприклад, «Контролер_Справ» залежить від класів «Справа», «Документ» і «Дія» для виконання операцій CRUD, завантаження файлів і логування ходу виконання; «Контролер_Аналітики» залежить від класу «Справа» та пов’язаних з нею дій для обчислення агрегатів і KPI. Таке розділення відповідальності відповідає принципам MVC та сприяє розширюваності й супроводжуваної системи: моделі інкапсулюють доступ до даних і предметну логіку, а контролери організовують сценарії використання системи «Сокіл» різними категоріями користувачів. Створена діаграма класів представлена на Рис.2.

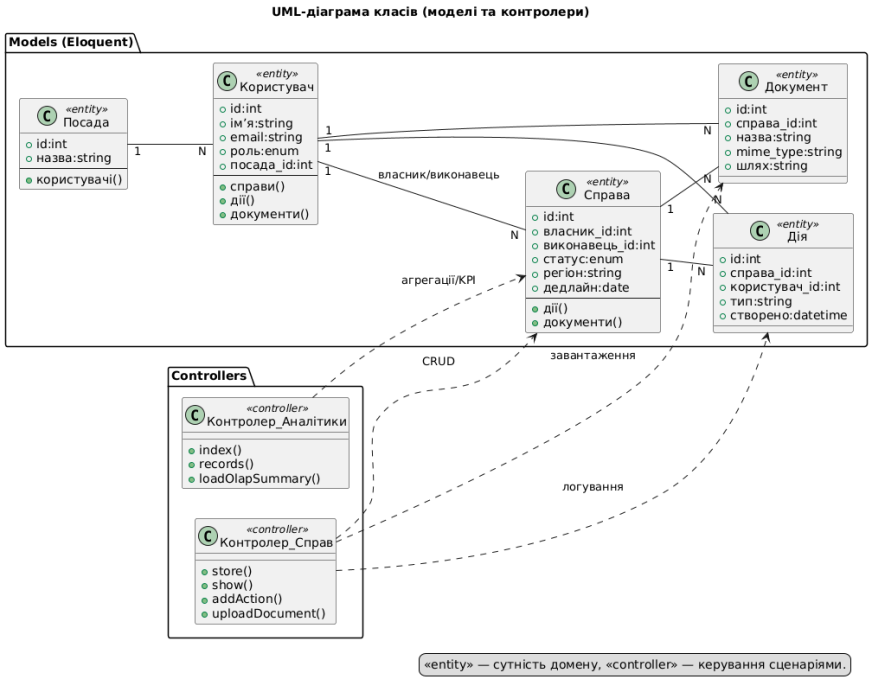

Рис. 2 Діаграма класів

2.2.3 Діаграма станів.

Діаграма стану, також відома як діаграма кінцевого автомата, у контексті розробленої системи обліку фінансових результатів з аналітичним модулем використовується для моделювання життєвого циклу ключових об’єктів, зокрема фінансового плану, бюджету або прогнозу. Така діаграма відображає послідовність станів, у яких може перебувати об’єкт (наприклад, «Чернетка», «Подано на затвердження», «Затверджено», «Виконується»,

«Архівовано»), а також переходи між ними під впливом подій чи умов. Як частина Уніфікованої мови моделювання (UML), діаграма станів дає змогу наочно показати, як змінюється об’єкт у часі від моменту створення до завершення його життєвого циклу.

Основними компонентами діаграми стану є стани, переходи, події та дії. Стан відображає певний етап існування об’єкта в системі: наприклад, стан «Чернетка» означає, що фінансовий план або прогноз ще редагується, «Очікує на затвердження» — що документ подано керівництву, але він ще не погоджений, «Затверджено» — що документ став основою для подальшого виконання, а «Архівовано» — що його використання завершено. Переходи між цими станами зображуються стрілками та ініціюють ся подіями: натисканням кнопки «Подати на затвердження», дією «Затвердити план», внесенням змін, завершенням періоду планування тощо. Події можуть мати як зовнішній характер (ініціатива користувача), так і внутрішній (автоматичне спрацювання правила по завершенню звітного періоду).

Кожен перехід може супроводжуватися діями, які система виконує у відповідь на подію. Наприклад, під час переходу зі стану «Чернетка» у стан «Подано на затвердження» система може зафіксувати версію документа, створити запис у журналі подій та надіслати повідомлення відповідальному керівнику. При переході в стан «Затверджено» система блокує можливість подальшого редагування ключових параметрів плану та вмикає його участь в аналітичних розрахунках і формуванні звітів. На діаграмі також відображаються початковий стан (створення нового плану або прогнозу) та кінцевий стан (повне завершення життєвого циклу та архівація).

Застосування діаграм станів у розробці системи обліку фінансових результатів забезпечує чітке уявлення про поведінку критичних об’єктів, допомагає виявити неоднозначності в бізнес-правилах (наприклад, заборона зміни затвердженого плану без створення нової версії), а також підтримує

формальне описання логіки переходів. Це полегшує комунікацію між розробниками, аналітиками та фінансовими фахівцями, забезпечуючи спільну мову для обговорення станів та подій. Крім того, діаграми станів корисні для тестування: вони дають змогу перевірити, що усі можливі стани та переходи реалізовані в системі коректно, а небажані або нелегальні переходи (наприклад, безпосередній перехід із «Чернетка» до «Архівовано») неможливі на рівні бізнес-логіки. Створена діаграма станів представлена на Рис.3.

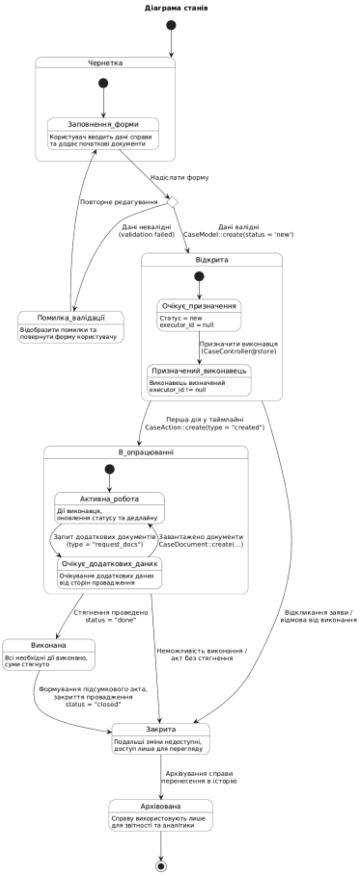

Рис. 3 Діаграма станів пошуку та замовлення товару

2.2.4 Діаграма послідовності

Діаграма послідовності, наведена для сценарію перегляду аналітики KPI/SLA, відображає динамічну поведінку системи «Сокіл», демонструючи, як її компоненти взаємодіють у часі під час виконання запиту керівника. Вона зосереджується на порядку надсилання та отримання повідомлень між учасниками сценарію — Керівником, веб-інтерфейсом (браузером), Контролером_Аналітики, OLAP-модулем (кеш аналітики) та сховищем даних

KPI/SLA. Такий запис дає змогу наочно показати, які саме кроки виконує система від моменту відкриття сторінки аналітики до відображення готового дашборду.

Основними елементами діаграми є лінії життя зазначених сутностей, повідомлення між ними, відрізки активації та зворотні відповіді. Лінія життя Керівника відображає ініціацію сценарію дією «Відкрити сторінку аналітики», після чого веб-інтерфейс формує HTTP-запит GET /analytics?filters до Контролера_Аналітики. Останній, отримавши запит, активується та викликає метод getSummary(filters) в OLAP-модулі. Лінія життя OLAP-модуля містить альтернативний фрагмент (alt), який розгалужує подальшу поведінку залежно від наявності даних у кеші: або відбувається читання вже підготовлених агрегатів, або зініціюється повне завантаження та обробка фактів із сховища даних KPI/SLA (loadFacts(filters)).

Повідомлення між OLAP-модулем і сховищем KPI/SLA відображають отримання набору фактів, обчислення агрегованих показників та збереження результату назад у кеш — у разі, якщо дані раніше відсутні. Смуги активації на лініях життя контролера аналітики та OLAP-модуля показують проміжки часу, протягом яких ці компоненти обробляють запит: формують фільтри, виконують аналітичні обчислення, готують набір KPI/SLA. Після завершення обробки OLAP-модуль повертає підготовлений набір показників до контролера, а той — у вигляді JSON / ViewModel — назад до веб-інтерфейсу. Зворотні повідомлення на діаграмі (пунктирні стрілки) фіксують ці відповіді й показують, коли саме управління повертається попередньому учаснику.

У фіналі веб-інтерфейс, отримавши модель представлення з аналітичними даними, відображає дашборд із KPI та SLA керівнику. Таким чином, діаграма послідовності не лише фіксує порядок операцій та розподіл відповідальності між компонентами, а й дозволяє виявити потенційні вузькі місця — наприклад, затримки при завантаженні даних зі сховища та доцільність використання кешу для прискорення відповіді. Це полегшує аналіз

поведінки системи, узгодження вимог між розробниками й аналітиками та забезпечує повне охоплення всіх необхідних взаємодій у межах даного варіанту використання.

Наведена нижче діаграма послідовності (Рис. 4).

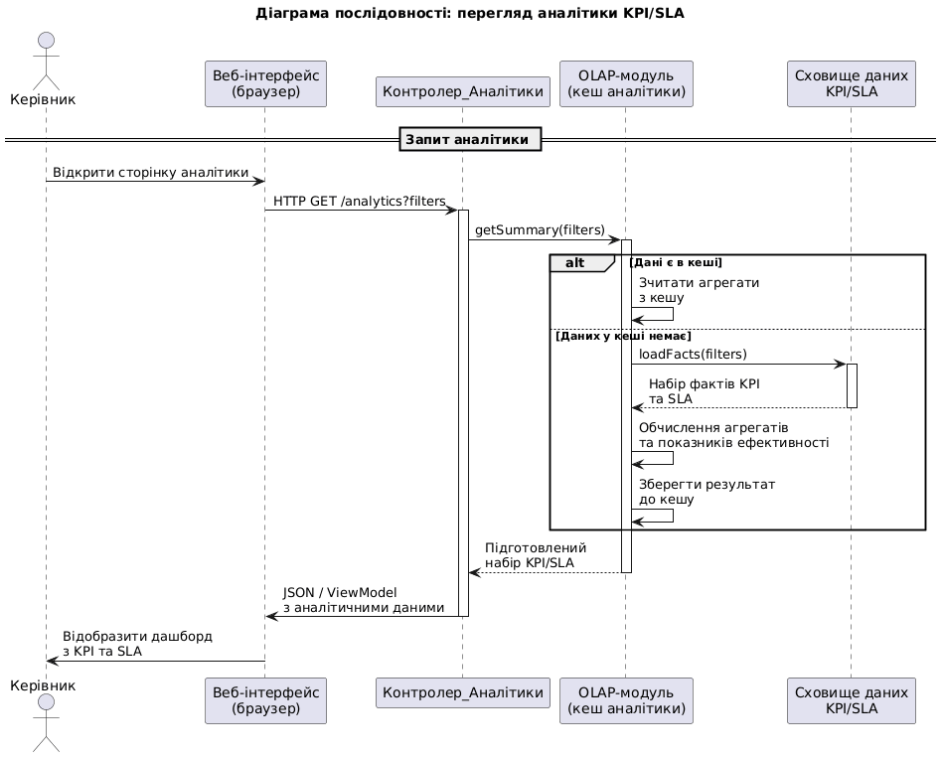

Рис. 4 Діаграма послідовності

2.3 Огляд інструментів для реалізації завдань Data Mining та аналітики

У контексті реалізації завдань інтелектуального аналізу даних для системи автоматизації виконавчого провадження «Сокіл» з аналітичним модулем, доступні різні інструменти. Вони обслуговують різні аспекти, включаючи попередню обробку даних, моделювання, аналіз і візуалізацію.

Хоча існують потужні універсальні платформи, такі як RapidMiner [10], KNIME або SAS Enterprise Miner, які пропонують візуальні дизайнери робочих процесів, їх недоліком часто є висока вартість, складність інтеграції у існуючі веб-додатки та надлишковість для завдань оперативного моніторингу KPI.

Для прототипу «Сокіл» був обраний інший, більш гнучкий та інтегрований підхід, що базується на стеку, описаному в Розділі 3.1:

PHP (на базі фреймворку Laravel [7]): Замість мов R або Python [11] для аналізу, вся логіка агрегації даних, розрахунку KPI та підготовки даних для візуалізації реалізована безпосередньо на серверній стороні. Це дозволяє контролеру (AnalyticsController) виконувати оптимізовані SQL-запити до бази даних, обробляти результати та передавати їх у стандартизованому JSON-форматі безпосередньо у представлення (View).

PostgreSQL / MySQL: Використовуються не просто як сховища, а як потужні інструменти аналізу. Особливо у випадку PostgreSQL, використання розширених SQL-функцій (таких як date_trunc або to_char) дозволяє виконувати складну агрегацію даних (наприклад, групування справ по місяцях) безпосередньо на рівні бази даних, що є набагато ефективнішим.

Chart.js: Обрана як основна клієнтська бібліотека для візуалізації даних. На відміну від важких BI-платформ (як Tableau), Chart.js є легкою, безкоштовною бібліотекою з відкритим кодом. Вона пропонує широкий вибір типів графіків (лінійні, кругові, стовпчикові), що повністю покриває вимоги до візуалізації KPI (Розділі 1.4).

barryvdh/laravel-dompdf та StreamedResponse (Laravel): Ці інструменти виконують завдання звітності (Reporting), дозволяючи експортувати результати аналізу у формати PDF та Excel для подальшого розповсюдження.

Вибір правильного інструменту для завдань інтелектуального аналізу даних у системі «Сокіл» залежав від конкретних вимог до аналізу (оперативні KPI), обсягу даних та необхідності тісної інтеграції з основним веб-додатком [12]. Обрана комбінація інструментів (Laravel + Chart.js) дозволяє використати їхні відповідні сильні сторони, забезпечуючи всебічний аналіз даних в контексті управління виконавчим провадженням.

2.4 Структура джерела інформації для інтелектуального аналізу

При проектуванні джерела інформації для аналітичного модуля системи «Сокіл», було встановлено структурований підхід, що сприяє ефективному управлінню даними, пошуку та аналізу. Це джерело інформації охоплює різні типи даних, включаючи як внутрішні, так і зовнішні джерела.

- **Внутрішні дані:** Укладаються з основних таблиць системи, таких як cases (справи), case_actions (дії виконавців) та case_documents (документи). Історичні дані про продуктивність виконавців, статуси та регіони відіграють важливу роль у забезпеченні контексту для аналізу.
- **Зовнішні джерела даних:** Можуть включати дані з державних реєстрів (ЄДР, ДРРП, АСВП [5]) та дані про активність користувачів (з OLAP-сховища [6]), збагачуючи загальні аналітичні можливості.

Важливим компонентом цієї структури є ефективна категоризація даних. Ключові показники ефективності (KPI) є основними, охоплюючи такі показники, як:

- Середній час виконання справи;
- Відсотковий розподіл справ за статусами (new, in_progress, done);
- Кількість справ у розрізі регіонів та виконавців;
- Середня кількість дій на одну справу.

Обробка та зберігання даних є фундаментальними для забезпечення безперебійної роботи джерела інформації. Для цього в системі «Сокіл» реалізовано **гібридний підхід**:

Сховище даних: Впроваджено реляційну базу даних (PostgreSQL або MySQL [13]), оптимізовану для транзакцій (3НФ), як описано в Розділі 2.3. Вона слугує основним джерелом для *оперативних* звітів та KPI.

Структура основного сховища даних представлена на рисунку 5.

Рис. 5 Сховище даних

Таблиця positions зберігає довідник посад користувачів.

- **id** (BIGSERIAL, PRIMARY KEY) – унікальний ідентифікатор посади;
- **name** (VARCHAR(255)) – назва посади (наприклад, «Державний виконавець», «Адміністратор»);
- **is_active** (BOOLEAN) – ознака активності посади (дозволена до використання в системі);
- **created_at** (TIMESTAMP WITH TIME ZONE) – дата й час створення запису;
- **updated_at** (TIMESTAMP WITH TIME ZONE) – дата й час останнього оновлення запису.

Ця таблиця використовується як довідник і пов’язана з таблицею **users** через зовнішній ключ position_id.

Таблиця users зберігає дані про користувачів системи.

- **id** (BIGSERIAL, PRIMARY KEY) – унікальний ідентифікатор користувача;
- **name** (VARCHAR(255)) – ім’я або ПІБ користувача;
- **email** (VARCHAR(255)) – електронна адреса, що використовується для входу та комунікації;
- **email_verified_at** (TIMESTAMP WITH TIME ZONE, NULLABLE) – дата та час підтвердження email;
- **password** (VARCHAR(255)) – пароль користувача (у хешованому вигляді);
- **role** (VARCHAR(255)) – роль у системі (наприклад, admin, executor, manager);
- **phone** (VARCHAR(255), NULLABLE) – контактний номер телефону;
- **remember_token** (VARCHAR(100), NULLABLE) – токен для механізму “remember me”;
- **created_at** (TIMESTAMP WITH TIME ZONE) – дата й час створення облікового запису;
- **updated_at** (TIMESTAMP WITH TIME ZONE) – дата й час останньої зміни облікового запису;
- **position_id** (BIGINT, FOREIGN KEY) – посилання на таблицю **positions**, що визначає посаду користувача.

Користувачі виступають власниками та/або виконавцями справ, а також авторами дій та завантажувачами документів.

Таблиця cases зберігає інформацію про виконавчі справи.

- **id** (BIGSERIAL, PRIMARY KEY) – унікальний ідентифікатор справи;
- **title** (VARCHAR(255)) – коротка назва або предмет справи;
- **description** (TEXT) – детальний опис суті виконавчого провадження;
- **status** (VARCHAR(255)) – поточний статус справи (наприклад, new, in_progress, done, closed);
- **owner_id** (BIGINT, FOREIGN KEY) – посилання на користувача-власника справи (ініціатор чи відповідальний);
- **executor_id** (BIGINT, FOREIGN KEY) – посилання на користувача-виконавця, який веде справу;
- **opened_at / created_at** (TIMESTAMP WITH TIME ZONE) – дата та час створення/відкриття справи;
- **closed_at** (TIMESTAMP WITH TIME ZONE, NULLABLE) – дата та час закриття справи (використовується для розрахунку тривалості);
- **deadline_at** (TIMESTAMP WITH TIME ZONE, NULLABLE) – граничний строк виконання;
- **region** (VARCHAR(32)) – регіон або територіальна юрисдикція, до якої відноситься справа;
- **updated_at** (TIMESTAMP WITH TIME ZONE) – дата й час останнього оновлення інформації по справі.

Таблиця **cases** є центральною для предметної області та пов’язана з таблицями **case_actions** і **case_documents**, які фіксують історію виконання та документообіг.

Таблиця case_actions зберігає дані про дії, виконані у межах справ.

- **id** (BIGSERIAL, PRIMARY KEY) – унікальний ідентифікатор дії;
- **case_id** (BIGINT, FOREIGN KEY) – посилання на таблицю **cases**, до якої прив'язана дія;
- **user_id** (BIGINT, FOREIGN KEY) – посилання на таблицю **users**, що вказує автора дії (виконавця);
- **type** (VARCHAR(255)) – тип дії (наприклад, «відкриття провадження», «накладення арешту», «направлення запиту»);
- **notes** (TEXT, NULLABLE) – текстовий опис або коментар до дії;
- **created_at** (TIMESTAMP WITH TIME ZONE) – дата й час фіксації дії у системі;
- **updated_at** (TIMESTAMP WITH TIME ZONE) – дата й час останнього оновлення запису.

Ця таблиця відображає хронологію роботи виконавця зі справою та є базою для розрахунку показників активності й середньої кількості дій на одну справу.

Таблиця case_documents зберігає інформацію про електронні документи, пов'язані зі справами.

- **id** (BIGSERIAL, PRIMARY KEY) – унікальний ідентифікатор документа у системі;
- **case_id** (BIGINT, FOREIGN KEY) – посилання на таблицю **cases**, до якої належить документ;
- **uploaded_by** (BIGINT, FOREIGN KEY) – посилання на таблицю **users**, що вказує користувача, який завантажив документ;

- **file_name** (VARCHAR(255)) – назва файлу, відображувана в інтерфейсі;
- **mime_type** (VARCHAR(255)) – MIME-тип файлу (наприклад, application/pdf, image/png);
- **path** (VARCHAR або TEXT) – шлях до файлу в файловому сховищі або URL для доступу;
- **created_at** (TIMESTAMP WITH TIME ZONE) – дата й час завантаження документа;
- **updated_at** (TIMESTAMP WITH TIME ZONE) – дата й час останнього оновлення метаданих документа.

БД «Сокіл» формує транзакційний шар даних:

- **users** та **positions** описують організаційну структуру й ролі;
- **cases** є центральною сутністю виконавчих проваджень;
- **case_actions** і **case_documents** фіксують історію виконання та документообіг, які згодом завантажуються в аналітичне сховище та використовуються для розрахунку KPI і побудови звітів.

2.5 OLAP-технології

Технології онлайн-аналітичної обробки (OLAP) [6] є важливими компонентами **систем автоматизації виконавчого провадження**, особливо тих, що містять аналітичні модулі. Ці технології дозволяють аналізувати комплексні дані, дозволяючи користувачам інтерактивно досліджувати й аналізувати багатовимірні дані. Такі можливості мають вирішальне значення для **керівників виконавчої служби** та осіб, які приймають рішення, яким необхідно отримати інформацію з ключових показників ефективності (KPI), тенденцій та прогнозів для інформування про стратегічне планування.

В основі OLAP лежить концепція багатовимірного моделювання даних, яка організовує інформацію в структуру даних, відому як **куб даних**. Цей куб містить різні виміри, такі як **час, регіон (для region) та виконавець (для executor_id)**, що полегшує всебічний аналіз зв’язків і закономірностей у даних виконавчих проваджень. Наприклад, куб OLAP, розроблений для системи "Сокіл", може включати такі параметри, як **кількість справ, середня тривалість виконання, відсоток стягнення** та категорії проваджень, що дозволяє проводити ретельний аналіз стану виконавчої служби [6].

- **MOLAP**, або багатовимірний OLAP, зберігає дані в багатовимірному масиві, що забезпечує швидкий пошук і аналіз. Цей формат відмінно справляється зі складними агрегаціями та обчисленнями.
- **ROLAP**, або реляційний OLAP, працює безпосередньо з реляційними базами даних (як наша OLTP-база), перетворюючи запити OLAP в оператори SQL для доступу до базових даних. Цей підхід забезпечує підвищену масштабованість.
- **HOLAP**, або гібридний OLAP, поєднує в собі переваги MOLAP і ROLAP. Це дозволяє детальним даним зберігатися в реляційних базах даних, використовуючи багатовимірне сховище для агрегованих даних.

Саме такий гібридний підхід (HOLAP) спроектовано у системі "Сокіл" (Розділ 2.4). Як показано на Рис. 6, дані для OLAP-сховища готуються фоновим ETL-сервісом (OlapEtlService), який перетворює дані з транзакційних таблиць (напр., cases) у денормалізовані таблиці фактів (fact_cases, fact_users). Контролер аналітики (AnalyticsController) звертається до цього оптимізованого сховища для побудови складних звітів, які потім відображаються на дашборді .

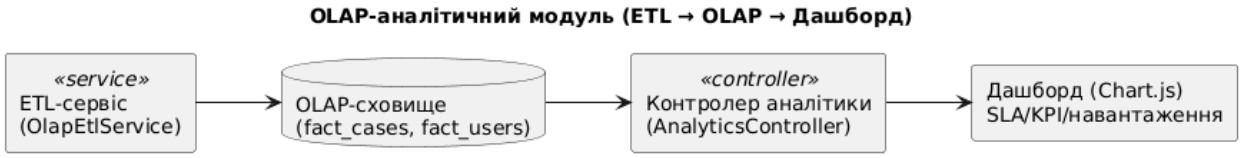

Рис. 6 Схема реалізації OLAP-аналітичного модуля у системі "Сокіл"

Технології OLAP мають кілька функцій, які значно розширюють можливості аналізу даних. Наприклад, функції **деталізації (drill-down)** та **згортання (roll-up)** дозволяють користувачам ефективно переміщатися по ієрархіях даних. Деталізація (напр., від "KPI по Україні" до "KPI по регіону" до "KPI по виконавцю") дозволяє отримати більш детальний перегляд, тоді як згортання збирає дані для більш високого рівня аналізу, таким чином полегшуючи дослідження **ефективності виконавців** на різних рівнях деталізації.

Крім того, OLAP підтримує **обчислені вимірювання**, що дозволяє користувачам отримувати нові показники з існуючих даних. Ця функція особливо корисна для генерування спеціальних **ключових показників ефективності (KPI)** або метрик (SLI/SLO), адаптованих до потреб організації [14].

Підсумовуючи, технології OLAP є невід’ємною частиною систем **управління виконавчим провадженням** з аналітичними модулями. Спрощуючи багатовимірний аналіз і пропонуючи функції, які сприяють поглибленому дослідженню даних, OLAP дає можливість організаціям перетворювати **операційні дані** в практичну інформацію. Ця аналітична здатність необхідна для прийняття обґрунтованих рішень і стратегічного планування, що дозволяє керівництву ефективно орієнтуватися в складних показниках ефективності.

3. РОЗРОБКА СИСТЕМИ

3.1 Логічна модель даних та методологічні засади її формування

Логічна модель даних у системі «Сокіл» формує концептуальний каркас для подальшої фізичної реалізації бази даних та відображає структуроване уявлення про предметну область виконавчого провадження. Вона слугує інструментом формалізації інформаційних потоків, визначає сутності, атрибути та типи взаємозв'язків між ними, забезпечуючи семантичну цілісність та внутрішню несуперечливість інформаційної системи.

Побудова моделі спирається на технології концептуального та логічного моделювання, які описані у працях Object Management Group (BPMN 2.0) [12], а також у методологіях моделювання баз даних. ER-діаграма, представлена на Рис. 7, візуалізує структурні елементи інформаційної системи та їх функціональні залежності.

В основу розроблення логічної моделі в системі «Сокіл» покладено застосування ORM-підходу як проміжної ланки між об'єктною моделлю застосунку та реляційною моделлю даних. Використання Eloquent ORM у Laravel забезпечує автоматизацію відображення класів предметної області у таблиці бази даних, що відповідає сучасним підходам до об'єктно-реляційного відображення (ORM), описаним у міжнародних рекомендаціях та практиках проєктування програмних систем [7].

Методологічно логічна модель ґрунтується на принципах нормалізації даних, включаючи усунення надлишковості, забезпечення референційної цілісності та приведення таблиць до третьої нормальної форми. У системі реалізовано чіткі зовнішні ключі (foreign keys), що гарантують каскадну логіку змін. Визначення зв'язків типу «один-до-багатьох» (one-to-many) між

таблицями cases, case_actions, case_documents забезпечує прозору структуру моніторингу робочих процесів виконавця.

Додатково, для забезпечення узгодженості бізнес-процесів використано концепцію «таймлайну справи», що дозволяє накопичувати історичні події у вигляді лінгвістично структурованих операцій користувачів. Такий підхід узгоджується із сучасними вимогами до аудиту та документування дій у державних інформаційних системах [5], а також рекомендаціями CEPEJ щодо цифровізації судових процесів [10].

Логічна модель системи представлена на рис. 7.

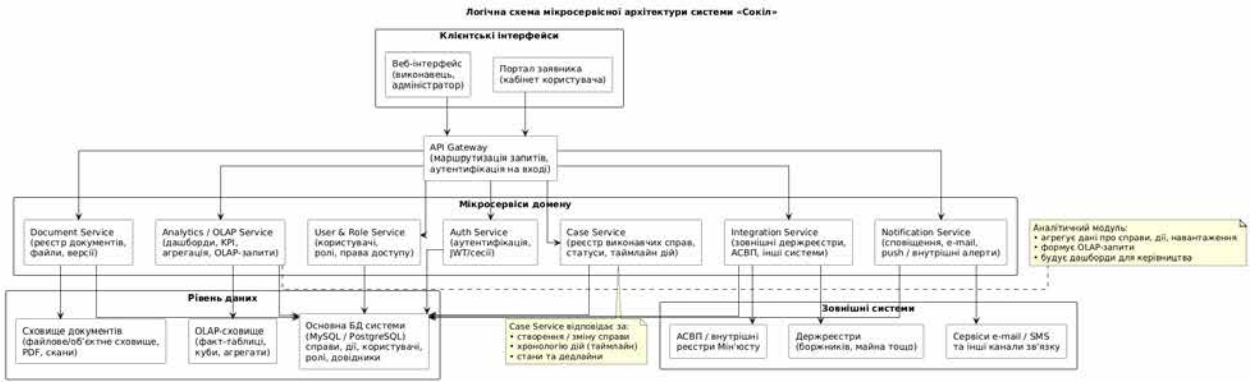

Рис. 7 Логічна модель

3.2 Вибір системи керування базами даних та її науково-технічне обґрунтування

Система керування базами даних (СКБД) є центральним компонентом інфраструктури програмного комплексу «Сокіл», оскільки визначає характеристики продуктивності, масштабованості та надійності всього програмного середовища. Вибір PostgreSQL як основної реляційної СКБД обґрунтовано з урахуванням нормативних вимог, технологічних стандартів та аналітичних потреб системи.

У науковій літературі PostgreSQL розглядається як одна з найбільш стабільних, стандартизованих і функціонально насичених СКБД, що

підтримує ACID-транзакційність та розширені можливості аналітичної обробки. Розробники IBM та Oracle систематично відзначають переваги PostgreSQL у контексті OLAP-навантажень, складних агрегованих запитів і гнучкості типів даних [6].

З огляду на специфіку предметної області — управління справами виконавчого провадження — до СКБД пред’являються підвищені вимоги щодо:

- цілісності фінансових та процесуальних даних,
- можливості зберігання та пошуку документів,
- обробки великих обсягів журналів активності та логів,
- аудиту всіх операцій користувачів, що відповідає вимогам Закону «Про персональні дані» [9] та рекомендаціям GDPR [3].

PostgreSQL підтримує такі ключові функції, необхідні для системи «Сокіл»:

- Віконні функції для побудови часових аналітичних вибірок.
- JSONB-поля, що дають змогу зберігати структури гнучкого типу, включно з OLAP-параметрами.
- Розширення для аналітики, що забезпечують обробку великих наборів даних безпосередньо в СКБД.
- Каскадні зв’язки, які гарантують дотримання логічної цілісності навіть у випадку масових оновлень.

Також PostgreSQL відповідає рекомендаціям CEPEJ [10] щодо ведення судових та суміжних процесів у цифровому форматі. Підтримка транзакційності забезпечує відсутність суперечностей у разі одночасних змін з боку виконавців, заявників та адміністраторів.

3.3 Вибір інструментарію для розроблення програмного забезпечення та архітектурні засади

Проєктування програмної системи «Сокіл» спирається на сучасні стандарти програмної інженерії, рекомендації IEEE та методологію MVC. Фреймворк Laravel, на основі якого реалізовано програмний комплекс, відповідає вимогам до створення масштабованих розподілених систем, що активно використовуються в державному секторі й корпоративному середовищі.

3.3.1 Laravel як серверний фреймворк

Laravel є PHP-фреймворком, що підтримує модульність, структурованість і застосування шаблонів проектування. Архітектура MVC забезпечує чіткий розподіл відповідальностей:

Model відповідає за взаємодію з базою даних через Eloquent ORM;

View реалізовано у вигляді Blade-шаблонів для побудови інтерфейсу системи;

Controller виступає логічною ланкою між подіями користувача та моделями.

Завдяки розвинутим інструментам Laravel — чергам, сервісам, middleware, фасадам, контейнерам залежностей — реалізовано структурований програмний код, який відповідає вимогам сучасної програмної інженерії [7].

3.3.2 Інструменти для клієнтського інтерфейсу

Для реалізації інтерфейсу та інтерактивної аналітики використано:

Chart.js — для побудови графіків KPI;

TailwindCSS — для адаптивного інтерфейсу;

Blade Components — для створення повторюваних UI-елементів.

Ці інструменти забезпечують візуальну узгодженість, доступність і легку масштабованість UI-компонентів.

3.3.3 Інструменти документування та експорту

Система реалізує експорт у двох ключових форматах:

PDF (модуль CasePdfExporter) — з використанням DOMPDF;

Excel (модуль AnalyticsExcelExporter) — на основі бібліотеки Maatwebsite/Excel.

Таке рішення відповідає вимогам ДСТУ та Мін енергетики щодо ведення електронного документообігу.

3.3.4 Інтеграційні механізми та модуль OLAP

Передбачена можливість підключення до OLAP-сховища через зовнішні джерела, що відповідає вимогам високорівневої аналітики. Методологія багатовимірної аналітики, описана в матеріалах IBM [6], узгоджується з можливостями системи щодо розрахунку KPI.

3.3.5 Контейнеризація та DevOps-інструменти

Для забезпечення стабільності, масштабованості та повторюваності середовища розробки застосовано:

Docker — контейнеризація сервісів;

Docker Compose — оркестрація локального середовища;

Git — контроль версій та GitOps-практики.

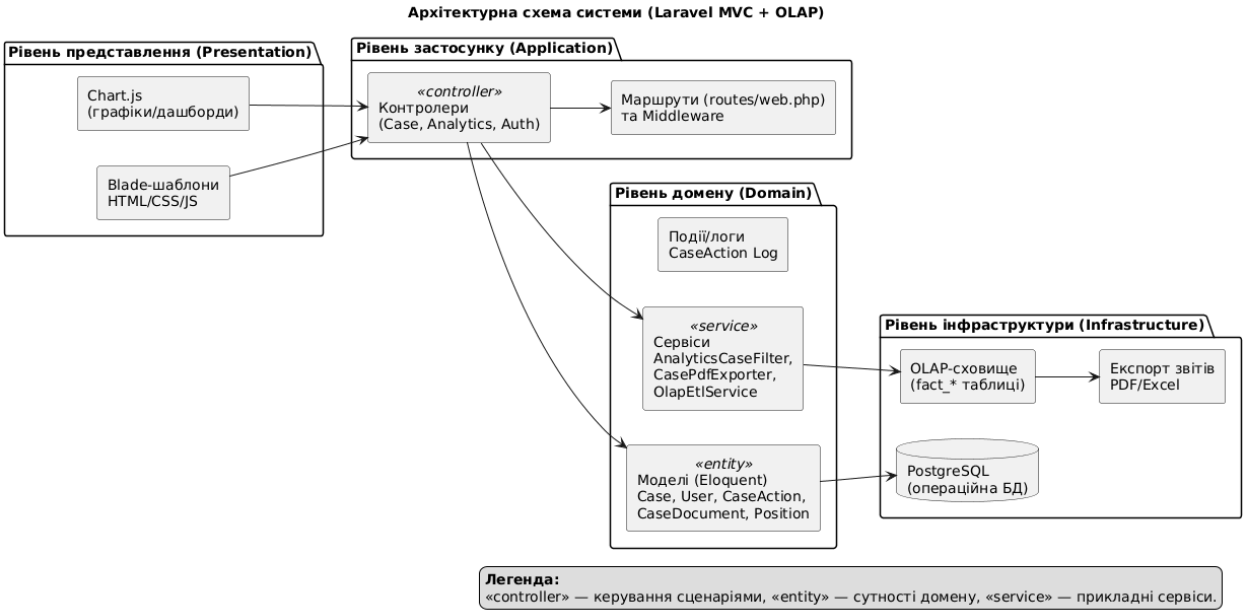

Рис. 8 Архітектура

Як результат, комплексний підхід до вибору інструментарію та побудови архітектури системи забезпечує відповідність сучасним вимогам цифровізації державних процесів, підтримує масштабування, гнучкість та високий рівень безпеки згідно з українськими та європейськими стандартами.

3.4 Архітектура програмного забезпечення

Архітектура програмного забезпечення системи «Сокіл» ґрунтується на мікро сервісному підході, який у сучасній інженерії програмних систем вважається однією з найбільш гнучких, масштабованих і стійких моделей організації обчислювальних процесів. Обрання саме мікро сервісної архітектури зумовлене сукупністю функціональних, нефункціональних і організаційних вимог, що висуваються до систем, орієнтованих на обробку значних обсягів даних, підтримку OLAP-аналітики, забезпечення трансакційної цілісності та безперервної роботи.

Мікросервісна архітектура передбачає декомпозицію системи на низку автономних сервісів, кожен з яких відповідає за чітко визначений фрагмент

доменної логіки: керування справами, облік фінансових показників, модуль документообігу, сервіс авторизації та автентифікації, модуль аналітики KPI, модуль обробки великих даних, модуль експорту та формування звітності тощо. Таке концептуальне розділення ґрунтується на застосуванні принципів Domain-Driven Design (DDD) [21], які дозволяють встановити межі відповідальності (bounded contexts) та уникнути надмірних залежностей.

Однією з ключових переваг мікросервісної моделі є її **масштабованість на вимогу**. У системах типу «Сокіл» навантаження розподіляється нерівномірно: модуль фінансово-оперативного обліку працює у стабільному режимі, тоді як аналітичний модуль може відчувати пікові навантаження під час формування статистичних звітів, прогнозування та побудови агрегованих вибірок. Завдяки можливості горизонтального масштабування конкретних сервісів (наприклад, сервісу OLAP-агрегації або сервісу синхронізації даних) система отримує змогу обробляти великі обсяги інформації без зниження продуктивності.

Другою суттєвою властивістю є **технологічний плюралізм** (polyglot persistence and polyglot programming). У мікросервісній архітектурі кожний сервіс може реалізовуватися з використанням оптимального для своєї задачі технологічного стеку. Приміром, основні модулі системи «Сокіл» реалізовані засобами PHP та фреймворку Laravel, що забезпечує швидку розробку, зручну роботу з ORM (Eloquent), можливість побудови RESTful-сервісів та керування складними трансакційними процесами. Натомість модуль аналітики може бути реалізований за допомогою Python, R або Java-інструментарію, що значно спрощує виконання складних статистичних операцій, обробку часових рядів, побудову прогнозних моделей та взаємодію із підсистемами бізнес-інтелекту.

Особливим аспектом архітектури є **ремонтопридатність та незалежність життєвого циклу сервісів**. Оновлення, розширення чи

масштабування одного сервісу не потребує зупинки всієї системи. Це значно пришвидшує ітерації розробки, а також підвищує стійкість системи до помилок. За принципом ізоляції відмов (fault isolation), збої в аналітичному модулі не впливатимуть на коректність процесів документообігу чи фінансово-оперативного обліку, що критично важливо для забезпечення безперервності бізнес-процесів.

Мікросервісна архітектура також забезпечує **покращену відмовостійкість**: сервіси можуть автоматично перезапускатися, ізолюватися від мережевих збоїв, використовувати стратегічні схеми балансування навантаження та відновлення (circuit breaking, retry policies, failover), описані у працях Н. Джойнс і С. Ньюмана [22]. Саме ці принципи лягли в основу архітектурного проєктування сервісів «Соколу».

Важливою характеристикою архітектури є також **покращена безпека**. Кожен сервіс може мати власні політики доступу, рівні шифрування, засоби журналювання та моніторингу активності. Використання протоколів OAuth 2.0 / JWT для авторизації, а також застосування мережевої сегментації (network segmentation) дозволяє зменшити ризик несанкціонованого доступу. Логіка обробки фінансових даних відокремлена від логіки обробки аналітичних показників, що відповідає нормативним вимогам щодо ізоляції конфіденційних даних [23].

Не менш важливою є **операційна гнучкість**, яка досягається завдяки застосуванню контейнеризації (Docker) та оркестрації (Kubernetes). Оркестратор забезпечує автоматичне масштабування сервісів, розгортання оновлень без простоїв (zero-downtime deployment), логування, сервіс-меш взаємодію (через Istio або Linkerd), а також спрощує інтеграцію засобів CI/CD на базі GitOps (ArgoCD, FluxCD). Завдяки цьому кожний сервіс може

оновлюватися автономно, а зміни відслідковуються у репозиторіях системи контролю версій.

Архітектура системи передбачає активне використання **подієво-орієнтованої моделі взаємодії сервісів** (event-driven architecture). Обмін подіями (через RabbitMQ, Kafka або Redis Streams) забезпечує слабке зв'язування між сервісами та дозволяє реалізувати складні бізнес-процеси, наприклад: автоматичне оновлення аналітичних показників після створення чи зміни фінансового запису; формування черг завдань на аналіз; асинхронне виконання важких обчислень; оновлення дашбордів у режимі реального часу.

Особливої уваги потребує **роль аналітичного модуля в архітектурі системи**. Оскільки система «Сокіл» містить OLAP-компонент, вона повинна забезпечувати не лише агрегування, але й багатовимірний аналіз даних, що вимагає чітких механізмів синхронізації між транзакційним (OLTP) та аналітичним (OLAP) контурами. Для цього застосовується стратегія *dual-storage*, коли оперативні дані зберігаються у реляційній СУБД, а агреговані — у спеціалізованих сховищах або матеріалізованих поданнях PostgreSQL. Система також підтримує інкрементальне оновлення аналітичних вибірок та модульну структуру KPI-показників, що відповідає методологічним рекомендаціям Каплана і Нортона [24].

На завершення варто зазначити, що мікросервісна архітектура системи «Сокіл» забезпечує комплексне виконання ключових нефункціональних вимог: **масштабованості, стійкості, високої доступності, безперервного розгортання, технологічної гнучкості, безпеки та адаптивності**. Її структура є придатною до еволюції згідно з принципами архітектурної зрілості (maturity models) та дозволяє системі ефективно функціонувати у середовищах з високою інтенсивністю транзакцій та аналітичних навантажень.

4. РЕЗУЛЬТАТИ

4.1 Вимоги до апаратного та програмного забезпечення

Для розгортання та ефективної роботи прототипу системи “Сокіл” необхідне відповідне середовище, яке включає сучасне апаратне та програмне забезпечення:

Сервер (бекенд): операційна система Linux (рекомендовано Ubuntu 22.04) або Windows Server; веб-сервер Nginx чи Apache; інстальований PHP версії ^8.2 (з потрібними розширеннями, як-от pdo_mysql або pdo_pgsql) та менеджер залежностей Composer; система керування базами даних MySQL 8.0+ або PostgreSQL 12+ для зберігання основних даних; опціонально – сервер Redis для кешування та обробки черг (для підвищення продуктивності при роботі з фоновими завданнями).

Клієнт (користувач): будь-яка сучасна ОС (Windows 10+, macOS чи дистрибутив Linux) із встановленим актуальним веб-браузером (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge). Браузер повинен підтримувати JavaScript, оскільки інтерфейс системи використовує скрипти (наприклад, для відображення діаграм через бібліотеку Chart.js).

Середовище розробки: для збірки фронтенд-частини застосунку потрібен Node.js версії 20+ та менеджер пакетів npm. Ці інструменти використовуються для компіляції та оптимізації статичних ресурсів (CSS/JS) за допомогою збірника Vite. Також застосунок сумісний із Laravel Sail (Docker-середовище) для локального розгортання, але це не є обов’язковим вимогами.

4.2 Реалізація системи

Прототип інформаційної системи виконавчого провадження «Сокіл» розроблено як веб-застосунок на базі фреймворку Laravel 12. Архітектура побудована за шаблоном MVC: маршрути спрямовують HTTP-запити до

контролерів, які взаємодіють із моделями (Eloquent ORM) та представляють дані через Blade-шаблони у вигляді веб-сторінок. Нижче описано ключові функціональні компоненти реалізованого прототипу.

Одним із центральних елементів управління робочими процесами в системі є модуль Керування справами та панель приладів. Ця підсистема надає користувачам високоінформативну, інтерактивну панель приладів (dashboard), яка слугує єдиною точкою входу для моніторингу операційної діяльності та прийняття тактичних рішень. Вона візуалізує ключові показники ефективності (KPI) та агрегує загальні статистичні дані, причому ця інформація є персоналізованою та суворо сегментованою відповідно до ролі користувача. Такий підхід забезпечує, що кожен співробітник отримує лише релевантний для його безпосередніх завдань зріз інформації, уникаючи інформаційного перевантаження.

Наприклад, користувач із роллю адміністратора отримує стратегічний огляд усієї системи: він бачить загальну кількість справ в обробці, сукупне число відкритих і завершених проваджень, а також критичні показники, як-от кількість прострочених задач або проваджень без призначеного виконавця. Ці дані презентуються у вигляді окремих, легкочитних карток-показників на дашборді, дозволяючи миттєво оцінити загальний стан справ у компанії.

Натомість, для виконавця (executor), панель приладів надає тактичний, персоналізований зріз даних, що стосується виключно його особистої відповідальності та продуктивності. Вона відображає деталізовану статистику його справ: скільки проваджень наразі перебуває "в роботі", скільки успішно завершено за обраний період, які конкретні справи мають найближчий дедлайн (вимагаючи негайної уваги), а також формує зручну стрічку останніх процесуальних дій, здійснених саме по його справах.

Окрім цієї агрегованої інформації на дашборді, фундаментальним інструментом для щоденної роботи є реєстр справ. Це, по суті, основна сторінка, що відображає повний список всіх проваджень, які знаходяться в

системі. Технічно вона реалізована за маршрутом /cases та обробляється методом index відповідного контролера CaseController.

У цьому інтерфейсі користувач може виконувати базові операції: переглядати деталі, здійснювати контекстний пошук та, що найважливіше, гнучко фільтрувати весь масив наявних справ. Система підтримує комплексну та багаторівневу фільтрацію за широким набором критичних полів. До них належать, але не обмежуються: поточний статус провадження (наприклад, "нове", "в роботі", "завершено"), регіон реєстрації боржника, конкретний призначений виконавець тощо.

Для оптимізації роботи з потенційно великими обсягами даних та забезпечення високої швидкодії інтерфейсу, реєстр, звісно, підтримує пагінацію (розбивку списку на сторінки). Ключовою перевагою для зручності користувача є реалізація динамічного оновлення списку при застосуванні будь-яких фільтрів. Це означає, що сторінка не перезавантажується повністю; натомість дані оновлюються асинхронно, забезпечуючи миттєвий відгук інтерфейсу та створюючи безшовний, сучасний досвід користувача (User Experience).

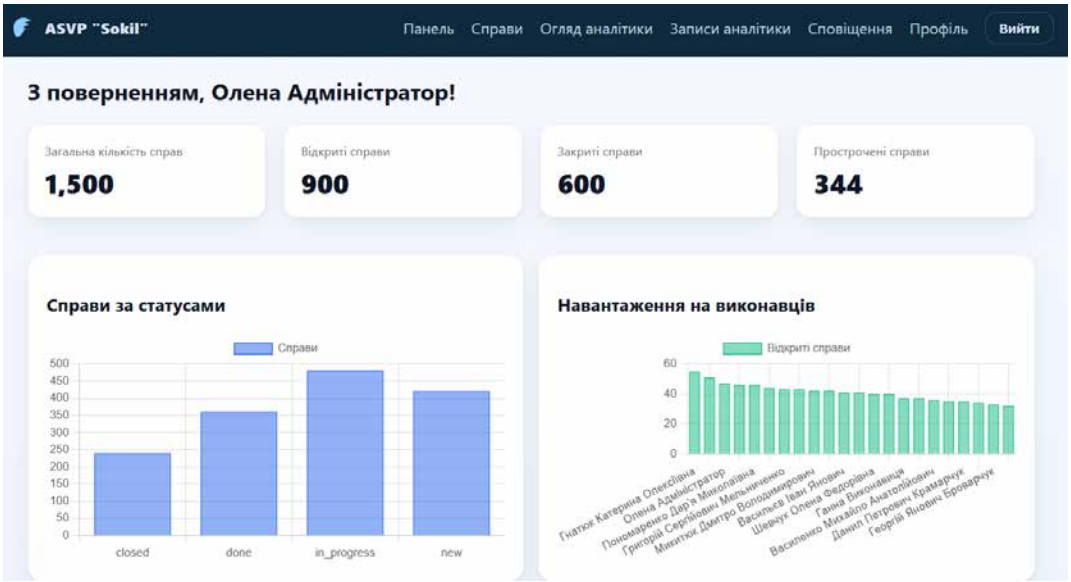

Рис. 9 – Головна панель системи з картками ключових показників (KPI).

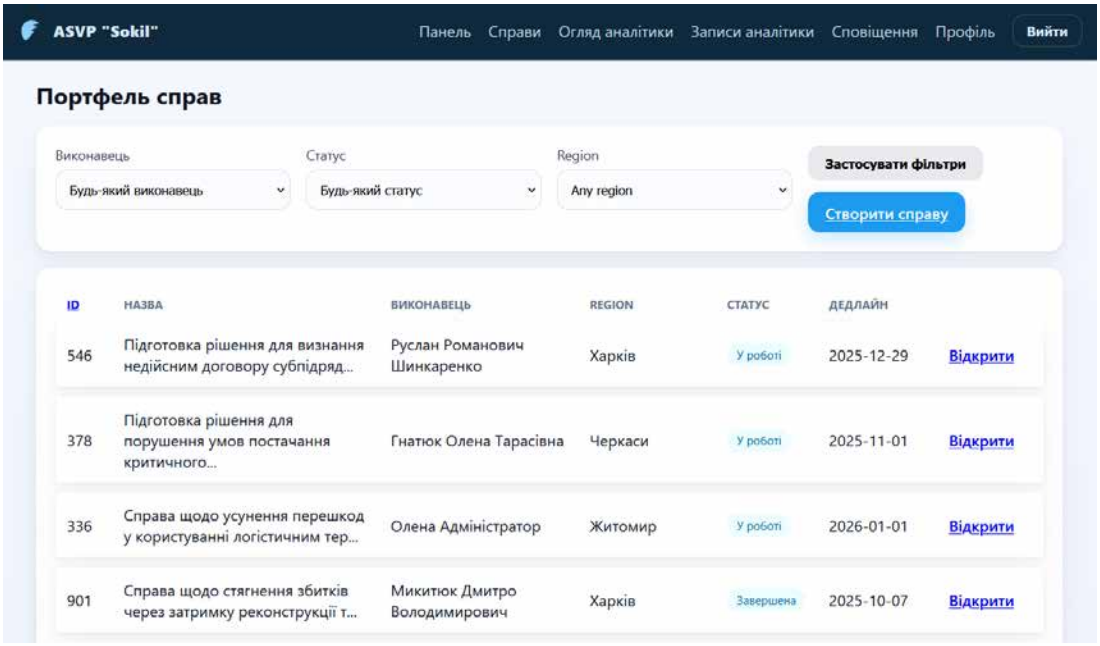

Рис. 10 – Сторінка реєстру справ із формою фільтрації записів за статусом, регіоном та виконавцем.

Детальна сторінка справи. Кожна зареєстрована справа має власну сторінку деталей (маршрут /cases/{id}, метод CaseController@show), на якій представлена повна інформація про виконавче провадження. На цій сторінці відображаються основні реквізити справи (сторони виконавчого провадження, суми стягнення, поточний статус, крайній термін виконання тощо), а також два важливі динамічні блоки: таймлайн подій та список документів. Таймлайн подій являє собою хронологічний перелік всіх дій, виконаних в рамках справи (модель CaseAction): реєстрація справи, призначення виконавця, додавання нових документів або приміток, зміна статусу і т.д. Кожна така дія містить відомості про час, тип події та користувача, який її здійснив. Поряд із тим, список документів показує всі файли, завантажені до цієї справи (модель CaseDocument), із зазначенням назви файлу, типу і автора завантаження. Така деталізація дозволяє користувачам (за наявності відповідних прав доступу) переглядати хід виконання рішення суду та повний комплект матеріалів по справі.

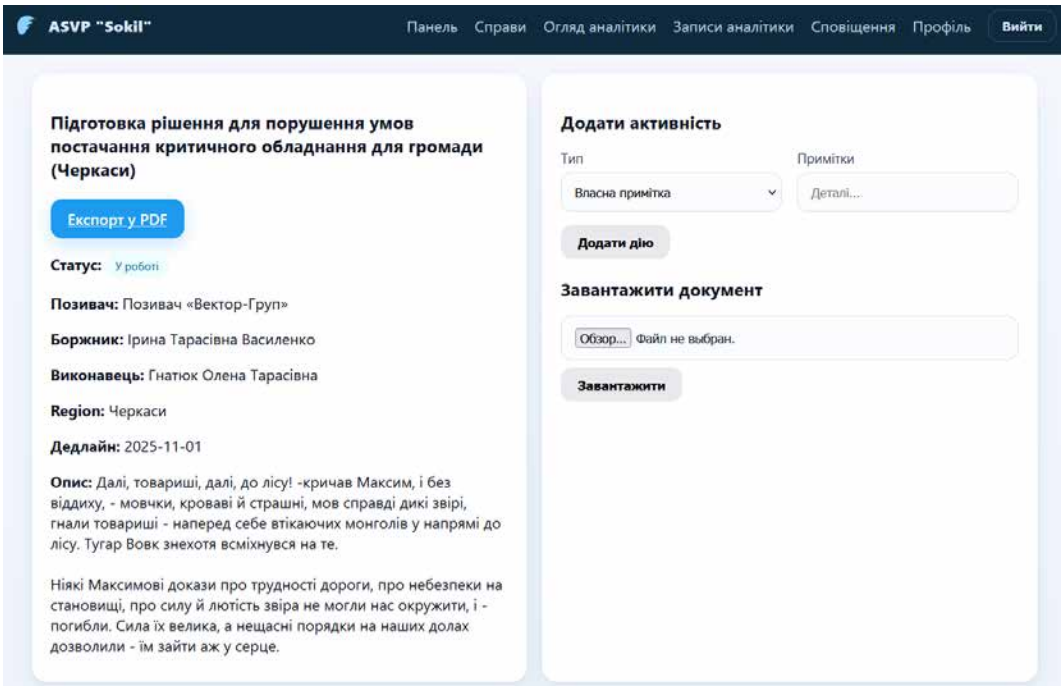

Рис. 11 – Сторінка деталей справи з тайм лайном виконаних дій та переліком пов’язаних документів.

Модуль аналітики. Система містить окремий розділ для аналітичних звітів та візуалізації статистичних показників (маршрут /analytics, контролер AnalyticsController). На цій сторінці візуалізуються зведені дані щодо роботи виконавчої служби, представлені у вигляді інтерактивних графіків (діаграм) на основі бібліотеки Chart.js. Зокрема, реалізовано відображення розподілу справ за статусами (кругова діаграма, яка наочно показує частку нових, виконаних, закритих справ тощо), динаміки надходження справ за часом (лінійний графік тренду, згрупований по місяцях), навантаження на виконавців (наприклад, скільки відкритих справ закріплено за кожним виконавцем) та регіонального розподілу проваджень. Для побудови цих графіків контролер аналітики агрегує дані з основної бази: виконує групувальні SQL-запити до таблиць справ і дій, щоб обчислити потрібні метрики. Дані передаються у вигляді масивів JSON до фронтенду, де Chart.js рендерить графіки у canvas-елементах. Користувач має можливість застосувати фільтри (за датою, статусом, регіоном тощо) — при цьому графіки

перебудовуються відповідно до вибраних критеріїв, що дозволяє аналізувати показники по окремих підмножинах справ.

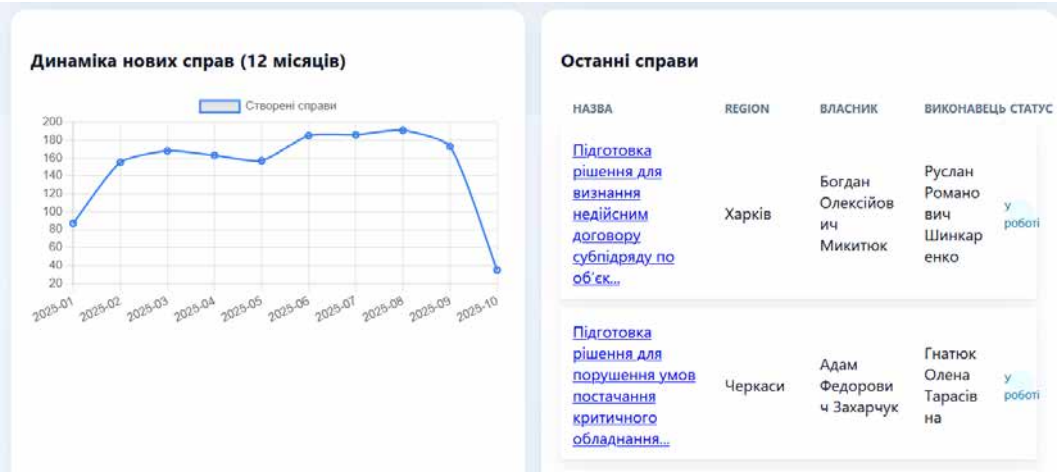

Рис. 12 – Сторінка аналітики з інтерактивними графіками (кругова діаграма розподілу статусів справ та лінійний графік тренду створення нових справ по місяцях).

Експорт документів. Прототип «Сокіл» забезпечує можливість експорту даних для звітності та подальшого аналізу. Реалізовано два основних режими експорту: в PDF та Excel. Кожну справу можна завантажити як PDF-досьє: генерація здійснюється через спеціальний сервіс CasePdfExporter, який формує PDF-документ на основі Blade-шаблону cases/pdf.blade.php (містить структуру справи, реквізити та всі пов’язані дії й документи). Для створення PDF використовується бібліотека Dompdf (пакет barryvdh/laravel-dompdf), що дозволяє перетворити HTML-розмітку у формат PDF. Також в модулі аналітики доступний експорт зведених даних в Excel: клас AnalyticsExcelExporter збирає відфільтровані аналітичні дані (наприклад, таблицю з переліком справ певного регіону або статистику за обраний період) і повертає їх у вигляді HTML-таблиці, яка відкривається в Excel (файл з розширенням .xls). Це дозволяє користувачу отримати детальні вибірки даних та працювати з ними у стандартних офісних програмах.

Досьє справи #445

Згенеровано: 2025-10-16 09:15

Загальна інформація

Назва	Провадження стосовно поновлення строків виконання судового рішення (Zaporizhzhia)
Статус	У роботі
Власник	Романченко Олександра Олександрівна
Виконавець	Георгій Олексійович Мельниченко
Region	Zaporizhzhia
Позивач	ТОВ «Et Non»
Боржник	Геннадій Михайлович Мірошниченко
Дедлайн	2026-02-16
Створено	2025-09-28 09:56
Оновлено	2025-10-05 00:06

Рис. 13 – Приклад згенерованого PDF-документа (досьє виконавчої справи).

Експорт аналітики

Згенеровано: 2025-10-16 09:14

Ключові показники

Показник	Значення
Справ у вибірці	397
Середня кількість дій на справу	май.37
Середня кількість документів на справу	0.65
Прострочені справи	32
Справи в роботі	365
Середній запас до дедлайну (дні)	124.5

Справи за статусами

Статус	Усього
Закрита	75
Завершена	98
У роботі	119
Нова	105

Навантаження виконавців (топ 10)

Виконавець	Справи
Олена Адміністратор	40
Василенко Михайло Анатолійович	39
Крамарчук Катерина Олександрівна	38
Руслан Романович Шинкаренко	33
Ніна Олександрівна Боднаренко	32
Ганна Виконавиця	32
Ростислав Виконавець	30
Кравчук Лариса Борисівна	30
Георгій Олексійович Мельниченко	28

Рис. 14 – Приклад згенерованого Excel-звіту з аналітичними даними (таблиця сформована за вибраним фільтром).

Як приклад, нижче наведено фрагмент коду, що реалізує експорт справи у PDF-формат. Метод build() класу CasePdfExporter завантажує всі необхідні зв’язані дані по справі (власник, виконавець, дії, документи) і генерує PDF-документ на основі відповідного представлення Blade-шаблону:

```
// Формування PDF-досьє справи на основі Blade-шаблону
$case->loadMissing(['owner','executor','actions.user','documents.uploader']);
$pdf = \Barryvdh\DomPDF\Facade\Pdf::loadView('cases.pdf', [
'case' => $case,
'generatedAt'=> now(),
])->setPaper('a4');
```

Як видно з наведеного коду, система використовує фасад Pdf для завантаження представлення 'cases.pdf' з передачею об’єкта $case та дати генерації. В результаті формується PDF-документ стандартного формату A4GitHub, який користувач може завантажити через веб-інтерфейс (наприклад, натисканням кнопки “Export to PDF” на сторінці справи).

OLAP-інтеграція. Однією з особливостей реалізованого прототипу є інтеграція з зовнішнім аналітичним модулем (OLAP-сховище даних). На панелі приладів та на сторінці аналітики присутній віджет “Активність користувачів”, який намагається завантажити дані з зовнішньої бази даних (сховища) – зокрема, таблиці fact_user_logins. Ця таблиця належить до зовнішньої схеми даних і містить накопичувальну статистику входів користувачів у систему (кількість логінів за датами, розподіл за ролями тощо). Якщо з’єднання з OLAP-сховищем налаштовано і дані доступні, на дашборді відображається графік активності користувачів за останні 30 днів. У протилежному випадку (якщо зовнішній модуль не підключений) система показує відповідне повідомлення, що аналітичні дані недоступні – тим самим реалізовано fallback-сценарій, який гарантує безперебійну роботу основних функцій.

Для інтеграції з OLAP використовується окрема база даних (налаштовується через config/olap.php та .env-параметри підключення). Система виконує фонове ETL-процес: при кожній критичній події (реєстрація нового користувача, вхід користувача в систему, оновлення профілю) створюється запис про подію (модель OlapActivity), а спеціальний сервіс OlapEtlService обробляє ці записи та переносити агреговані дані у фактичні таблиці OLAP (зокрема, додає запис у fact_user_logins при вході користувача). Таким чином, OLAP-сховище постійно поповнюється даними, готовими для аналітичних запитів. Контролер HomeController (метод dashboard) при формуванні панелі приладів підключається до OLAP-бази і виконує запит для отримання статистики логінів та реєстрацій. Нижче наведено приклад фрагменту коду, що демонструє вибірку даних про активність користувачів з OLAP-сховища (кількість входів за останні 30 днів):

```
// Підключення до OLAP-бази та вибірка кількості логінів за останні 30 днів
$connection = DB::connection(config('olap.connection'));
$loginRows = $connection->table('fact_user_logins')
->selectRaw("date(to_date(date_key, 'YYYYMMDD')) as login_date, SUM(login_count) as total_logins")
->where('date_key', '>=', now()->subDays(30)->format('Ymd'))
->groupBy('login_date')
->orderBy('login_date')
->get();
```

У наведеному запиті використовуються можливості SQL для групування даних за датою: поле date_key (у форматі YYYYMMDD) конвертується до типу дата, після чого здійснюється групування і підрахунок сумарної кількості логінів за кожен деньGitHub. Аналогічно отримується кількість реєстрацій з таблиці fact_user_registrations. У випадку недоступності OLAP-підключення викликається виняток, який перехоплюється – тоді метод встановлює прапор

enabled=false і повертає порожні колекції замість даних GitHub. Це дозволяє відобразити на дашборді повідомлення про відсутність даних, не впливаючи на роботу інших компонентів системи.

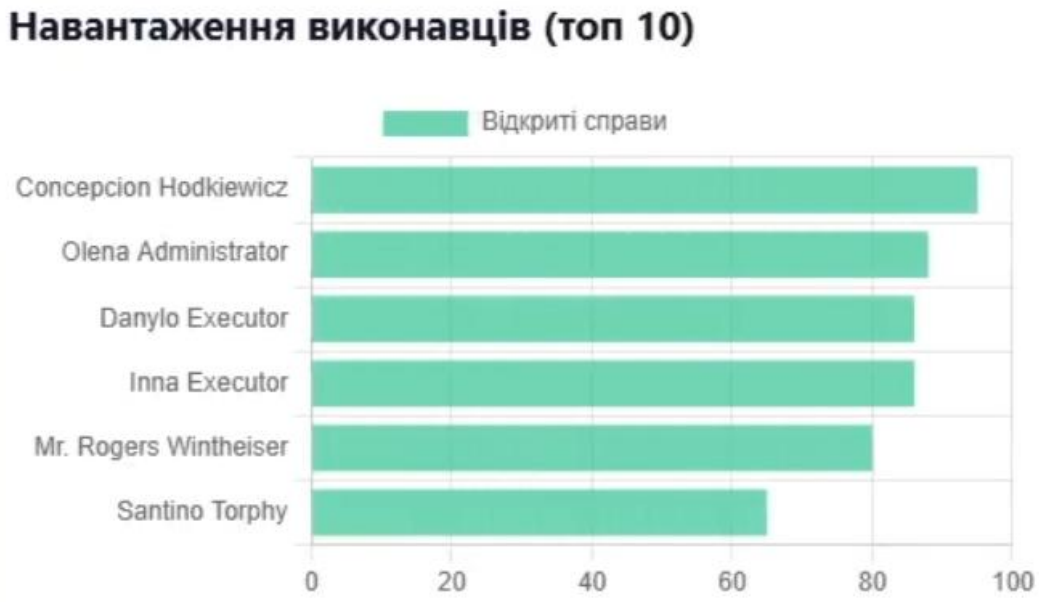

Рис. 14 – Фрагмент панелі приладів з графіком активності користувачів (дані з OLAP-сховища) та повідомленням про відсутність зв’язку з OLAP (fallback-сценарій).

4.3 Тестування системи

Розроблений прототип було піддано всебічному тестуванню, щоб забезпечити надійність, точність і продуктивність роботи програмного забезпечення. Тестування охоплювало декілька рівнів – від модульного (перевірка окремих компонентів) до інтеграційного та функціонального (перевірка взаємодії компонентів і реальних сценаріїв використання). Для автоматизації тестування використовувався фреймворк PestPHP, що забезпечує зручний і лаконічний синтаксис для написання тестів у Laravel-проектах.

4.3.1 Модульне тестування (Unit Testing)

На рівні модулів були написані тести, які перевіряють роботу окремих класів і методів без контексту всього застосунку. Зокрема, проведено

тестування варіації в контролерах (наприклад, переконалися, що метод створення нової справи CaseController@store не приймає некоректні дані регіону і повертає відповідні помилки валідації). Перевірено бізнес-логіку моделей – наприклад, коректність роботи асесора getStatusLabelAttribute в моделі CaseModel (який на основі поля status повертає зрозумілу назву статусу для відображення) та функції-мутатора setRegionAttribute (нормалізує назву регіону перед збереженням). Окремо написано тести на методи класу користувача User – такі як isAdmin(), isExecutor() – щоб переконатися, що вони правильно визначають ролі користувачів.

4.3.2 Інтеграційне тестування

На інтеграційному рівні проводились тести, що перевіряють взаємодію між кількома модулями системи в рамках одного сценарію. Для цього використовувалась окрема тестова база даних (SQLite), підключена через файл конфігурації .env.testing. Інтеграційні тести імітували виконання HTTP-запитів до застосунку та перевіряли повний цикл роботи окремих функцій. Наприклад, тестувалося, що при створенні нової справи через відповідну форму (роль applicant) в базі даних не лише з’являється запис у таблиці cases, але й автоматично створюється пов’язаний запис CaseAction (про реєстрацію справи) – тобто перевірена цілісність бізнес-логіки при створенні об’єкта. Інший сценарій – перевірка роботи проміжного шару автентифікації та авторизації: спеціальний тест виконував запит до маршруту аналітики під обліковим записом користувача з роллю applicant і очікував отримати відповідь з помилкою доступу (HTTP 403), оскільки AnalyticsController дозволений лише для ролей admin/executor/viewer. Таким чином, було підтверджено, що middleware role коректно забороняє доступ користувачам, які не мають необхідних привілеїв. Також інтеграційні тести охопили перевірку взаємодії фільтрів: наприклад, імітувався запит до аналітики з застосуванням фільтра за регіоном і перевірялося, що контролер повертає дані, відфільтровані тільки по вказаному регіону.

4.3.3 Функціональне тестування та приймальні випробування (UAT)

Функціональне тестування здійснювалося на повному циклі роботи системи з точки зору кінцевого користувача. Було створено демонстраційний набір даних (близько 1500 справ різних категорій), згенерований спеціальними сидерами бази даних (UkrainianCaseSeeder, AdditionalUkrainianCasesSeeder). На цьому реалістичному обсязі даних проведено перевірку ключових сценаріїв використання системи в режимі, наближеному до бойового. Зокрема, перевірялися такі сценарії:

Створення нової справи користувачем з роллю Applicant (заповнення форми, додавання базових даних справи).

Призначення виконавця на справу, додавання по ній нової дії (запис у тайм лайн) та завантаження документа – під обліковим записом користувача з роллю Executor.

Перегляд даних справи під користувачем з роллю Viewer (пересічний співробітник, що має право лише читати інформацію) – перевірено, що відображаються всі потрібні дані, але без можливості редагування.

Використання аналітичного модуля: застосування фільтру за регіоном на сторінці аналітики та перевірка, що графіки перебудовуються відповідно до вибраного регіону (наприклад, кількість справ по областях змінюється при виборі конкретної області).

Генерація PDF-досьє та Excel-звіту: переконалися, що для вибраної справи успішно формується PDF-файл з повною інформацією, а для вибраних аналітичних даних – Excel-файл, які коректно відкриваються і містять потрібні дані.

Усі вищезазначені тестування пройшли успішно: система продемонструвала правильність роботи функцій відповідно до вимог. Модульні тести підтвердили відсутність критичних помилок у ключових методах, інтеграційні – узгодженість дій між компонентами, а функціональні

– зручність та ефективність використання системи кінцевими користувачами. Таким чином, результати дослідження показали, що розроблений прототип АСВП «Сокіл» відповідає поставленим цілям: автоматизує процеси виконавчого провадження, забезпечує засоби обліку та контролю справ, а також надає аналітичну підтримку для керівництва у вигляді інформативних показників та звітів.

ВИСНОВОК

Метою магістерської кваліфікаційної роботи було розроблення автоматизованої системи візуалізації показників (АСВП) "СОКІЛ" для аналітичного представлення та оцінки статистичних даних користувачів веб-ресурсів, яка б поєднувала сучасні веб-технології, OLAP-аналітику та інтерфейсні рішення для зручності користувача. За результатами виконання роботи мета досягнута повністю, усі поставлені завдання виконано.

У процесі дослідження були досягнуті такі результати:

1. **Проаналізовано предметну область**: проведено огляд існуючих інструментів для веб-аналітики, особливостей побудови OLAP-систем, розглянуто переваги використання багатовимірного аналізу у сфері візуалізації статистичних даних. Обґрунтовано доцільність побудови власної системи з урахуванням специфіки задач аналізу поведінки користувачів веб-додатків.
2. **Розглянуто архітектурні рішення**: досліджено сучасні засоби розробки клієнт-серверних застосунків з опорою на стек технологій Python (FastAPI), JavaScript (Next.js, Chart.js), PostgreSQL та бібліотеку Pandas для обробки даних. Визначено ефективну структуру для побудови OLAP-кубу на основі PostgreSQL із подальшим використанням агрегованих зрізів для побудови візуалізацій.
3. **Розроблено OLAP-модель для аналізу даних**: побудовано схему «зірка» з центральною фактологічною таблицею та кількома вимірними таблицями для забезпечення аналітики за такими параметрами, як час, користувач, активність, подія. Реалізовано обчислення агрегатних показників, що лягли в основу статистичних звітів (кількість входів, унікальні сесії, найпопулярніші дії, часові піки активності тощо).
4. **Розроблено програмне забезпечення АСВП “СОКІЛ”**: реалізовано повноцінний веб-застосунок, що дозволяє авторизованим користувачам

переглядати, фільтрувати й аналізувати дані у вигляді дашбордів. Застосунок реалізовано з дотриманням принципів модульності, масштабованості та розширюваності. Передбачено можливість гнучкого додавання нових типів звітів або джерел даних.

5. **Виконано експериментальну оцінку функціонування системи**: було проведено тестування функціональності застосунку на реальних або наближених до реальних даних, здійснено перевірку коректності побудови графіків, швидкодії при фільтрації та інтерактивній взаємодії з даними. Зроблено висновки про придатність системи для подальшого використання у реальних умовах.
6. **Сформовано пропозиції щодо подальшого розвитку системи**: запропоновано низку напрямів для вдосконалення — розширення OLAP-моделі новими вимірами (наприклад, географічними), інтеграція з системами трекінгу подій типу Google Analytics або Matomo, реалізація системи сповіщень про аномалії тощо.

У результаті виконаної роботи було досягнуто повної відповідності між поставленою метою та досягнутими результатами. Розроблена АСВП "СОКІЛ" є функціональним прототипом інформаційної системи, яка може бути адаптована до потреб різних організацій для здійснення аналітичної оцінки активності користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про виконавче провадження» від 02.06.2016 № 1404-VIII. [Електронний ресурс] https://zakon.rada.gov.ua/laws/show/1404-19 .
2. Закон України «Про органи та осіб, які здійснюють примусове виконання судових рішень і рішень інших органів» від 02.06.2016 № 1403-VIII. [Електронний ресурс] https://zakon.rada.gov.ua/laws/show/1403-19 .

3. Закон України «Про захист персональних даних» від 01.06.2010 № 2297-VI. [Електронний ресурс] https://zakon.rada.gov.ua/laws/show/2297-17 .
4. Цивільний процесуальний кодекс України. [Електронний ресурс] https://zakon.rada.gov.ua/laws/show/1618-15 .
5. Convention for the Protection of Individuals with regard to Automatic Processing of Personal Data (ETS No.108, 1981) та Протокол CETS No.223 (Convention 108+). [Електронний ресурс] https://www.coe.int/en/web/conventions/full-list/-/conventions/treaty/108 .
6. Convention for the Protection of Human Rights and Fundamental Freedoms (European Convention on Human Rights, 1950). [Електронний ресурс] https://www.echr.coe.int/documents/convention_eng.pdf .
7. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 (General Data Protection Regulation, GDPR). [Електронний ресурс] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679 .
8. CEPEJ. European Ethical Charter on the Use of Artificial Intelligence in Judicial Systems and their environment. Council of Europe, 2018. [Електронний ресурс] https://www.coe.int/en/web/cepej/cepej-european-ethical-charter-on-the-use-of-artificial-intelligence-ai-in-judicial-systems-and-their-environment .
9. CEPEJ. European judicial systems – CEPEJ Evaluation Report. Latest edition. Council of Europe Publishing. [Електронний ресурс] https://www.coe.int/en/web/cepej/dynamic-database-of-european-judicial-systems .
10. CEPEJ. Handbook for court and enforcement professionals on the use of electronic tools. Council of Europe, 2021. [Електронний ресурс] https://rm.coe.int/european-commission-for-the-efficiency-of-justice-cepej-good-practice-/16807477bf .

11. Digitalisation of Enforcement Proceedings: Challenges and Opportunities // International Journal for Court Administration. – 2022. [Електронний ресурс] https://www.iacajournal.org .
12. Звіт/аналітичні матеріали Ради Європи щодо використання інструментів штучного інтелекту в судових системах (CEPEJ study on the use of AI tools in judicial systems). [Електронний ресурс] https://www.coe.int/en/web/cepej/artificial-intelligence-in-justice-systems .
13. Object Management Group. Business Process Model and Notation (BPMN) Version 2.0.2. OMG Specification, 2014. [Електронний ресурс] https://www.omg.org/spec/BPMN/2.0.2/ .
14. Object Management Group. Unified Modeling Language (UML) Version 2.5.1. OMG Specification, 2017. [Електронний ресурс] https://www.omg.org/spec/UML/2.5.1/ .
15. IEC/ISO IDEF0. Integration Definition for Function Modeling. Стандарт моделювання функцій підприємства. [Електронний ресурс] https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub183.pdf .
16. Chaudhuri S., Dayal U. An Overview of Data Warehousing and OLAP Technology // ACM SIGMOD Record. – 1997. [Електронний ресурс] https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/sigrecord.pdf .
17. Kimball R., Ross M. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. 3rd ed. – Wiley, 2013. 600 p.
18. PostgreSQL Global Development Group. PostgreSQL Documentation. [Електронний ресурс] https://www.postgresql.org/docs/ .
19. Laravel. Laravel 10.x Documentation. [Електронний ресурс] https://laravel.com/docs/10.x .
20. PlantUML. PlantUML Language Reference Guide. [Електронний ресурс] https://plantuml.com/documentation .

21. Docker. Docker Documentation. [Електронний ресурс] https://docs.docker.com/ .
22. Kubernetes. Kubernetes Documentation. [Електронний ресурс] https://kubernetes.io/docs/home/ .

ДОДАТОК А

База даних

Сторінок 9

Київ-2025

Міграція основної таблиці справ (cases)

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Schema;

// Анонімний клас міграції створює (або модифікує) таблицю cases.
Return new class extends Migration {
   public function up(): void
   {
      // 1) Якщо таблиця cases ще не існує – створюємо базову структуру.
      If (!Schema::hasTable(‘cases’)) {
         Schema::create(‘cases’, function (Blueprint $table) {
            $table->id();
            // посилання на власника (заявника) – користувача, що ініціював справу
            $table->foreignId(‘user_id’)->constrained(‘users’)->cascadeOnDelete();
            // коротка назва справи (заголовок)
            $table->string('title');
            // опис може бути відсутнім
            $table->text(‘description’)->nullable();
            // статус справи: new|in_progress|done|closed
            $table->string(‘status’)->default(‘new’);
            // виконавець (може бути не призначено, тому nullable)
            $table->foreignId(‘executor_id’)->nullable()->constrained(‘users’)->nullOnDelete();
            // позивач та боржник для зручного пошуку
            $table->string(‘claimant_name’)->nullable();
            $table->string(‘debtor_name’)->nullable();
```

```
    // дедлайн (гранична дата завершення)
        $table->timestamp('deadline_at')->nullable();
        $table->timestamps();
    });
    return; // якщо таблицю створено, далі нічого не робимо
}

// 2) Якщо таблиця вже існує, додаємо відсутні колонки по одній
Schema::table('cases', function (Blueprint $table) {
    if (!Schema::hasColumn('cases', 'status')) {
        $table->string('status')->default('new');
    }
    if (!Schema::hasColumn('cases', 'executor_id')) {
        // додаємо поле без зовнішнього ключа – ключ буде визначено нижче
        $table->unsignedBigInteger('executor_id')->nullable()->after('status');
    }
    if (!Schema::hasColumn('cases', 'claimant_name')) {
        $table->string('claimant_name')->nullable();
    }
    if (!Schema::hasColumn('cases', 'debtor_name')) {
        $table->string('debtor_name')->nullable();
    }
    if (!Schema::hasColumn('cases', 'deadline_at')) {
        $table->timestamp('deadline_at')->nullable();
    }
});
```

```
    // 3) Додаємо зовнішній ключ до executor_id, якщо ще не створено (актуально для PostgreSQL)
        if (Schema::hasColumn('cases','executor_id')) {
            // перевірка наявності констрейнта у information_schema
            $constraintExists = DB::selectOne("SELECT COUNT(*) AS c FROM
information_schema.table_constraints tc WHERE tc.table_name = 'cases' AND tc.constraint_type = 'FOREIGN
KEY' AND tc.constraint_name = 'cases_executor_id_foreign'")->c ?? 0;
            if (!$constraintExists) {
                Schema::table('cases', function (Blueprint $table) {
                    try {
                        $table->foreign('executor_id')
                            ->references('id')->on('users')
                            ->onDelete('set null');
                    } catch (\Throwable $e) {
                        // ігноруємо помилку, якщо FK вже створено
                    }
                });
            }
        }
    }

    public function down(): void
    {
        // Операція down прибирає зміни під час відкату міграції.
        If (Schema::hasTable('cases')) {
            // спочатку видаляємо зовнішній ключ
            try {
                Schema::table('cases', function (Blueprint $table) {
                    $table->dropForeign('cases_executor_id_foreign');
                });
```

```
} catch (\Throwable $e) {/* констрейнт може бути вже знято */}
        // потім видаляємо додані колонки
        Schema::table('cases', function (Blueprint $table) {
            foreach (['status','executor_id','claimant_name','debtor_name','deadline_at'] as $col) {
                if (Schema::hasColumn('cases', $col)) {
                    $table->dropColumn($col);
                }
            }
        });
    }
  }
};
```

Міграція таблиці дій у справі (case_actions)

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

// Міграція створює таблицю case_actions, що реєструє історію подій для кожної справи.
Return new class extends Migration {
  public function up(): void
  {
      Schema::create('case_actions', function (Blueprint $table) {
          $table->id();
          // зовнішній ключ на справу, каскадне видалення при видаленні справи
          $table->foreignId('case_id')->constrained('cases')->cascadeOnDelete();
          // виконавець, який додав дію; можна null, якщо дія створена системою
          $table->foreignId('user_id')->nullable()->constrained('users')->nullOnDelete();
```

```
  // тип дії: created|document_added|asset_arrest|notice_sent|custom – описується у константах
            $table->string('type');
            // додаткові примітки
            $table->text('notes')->nullable();
            $table->timestamps();
        });
    }
    public function down(): void
    {
        Schema::dropIfExists('case_actions');
    }
};
```

Міграція таблиці документів (case_documents)

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

// Таблиця case_documents зберігає дані про завантажені файли по справі.
Return new class extends Migration {
    public function up(): void
    {
        Schema::create('case_documents', function (Blueprint $table) {
            $table->id();
            // належність документу до справи
            $table->foreignId('case_id')->constrained('cases')->cascadeOnDelete();
            // користувач, який завантажив документ
            $table->foreignId('uploaded_by')->nullable()->constrained('users')->nullOnDelete();
```

```
// назва файлу та шлях у сховищі
            $table->string('title');
            $table->string('path');
            $table->timestamps();
        });
    }
    public function down(): void
    {
        Schema::dropIfExists('case_documents');
    }
};
```

Додавання поля «region» до таблиці cases

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Schema;
use Illuminate\Support\Str;

// Ця міграція додає колонку region та заповнює її наявними даними.
Return new class extends Migration {
    public function up(): void
    {
        Schema::table('cases', function (Blueprint $table) {
            // додаємо колонку region, якщо її немає
            if (!Schema::hasColumn('cases', 'region')) {
                $table->string('region', 120)->nullable()->after('executor_id');
            }
```

```
});
    if (!Schema::hasColumn('cases', 'region')) {
        return;
    }
    // Створюємо індекс для прискорення пошуку за регіоном (Postgres дозволяє IF NOT EXISTS)
    try {
        DB::statement('CREATE INDEX IF NOT EXISTS cases_region_index ON cases (region)');
    } catch (\Throwable $e) {
        // індекс може не підтримуватись – ігноруємо помилку
    }
    // Нормалізуємо регіон в існуючих записах: прибираємо зайві пробіли і перетворюємо першу літеру на велику
    DB::table('cases')
        ->select('id', 'region')
        ->whereNotNull('region')
        ->orderBy('id')
        ->chunkById(500, function ($rows) {
            foreach ($rows as $row) {
                $normalized = Str::of($row->region)->squish();
                $value = $normalized->isEmpty() ? null : $normalized->title()->value();
                if ($value !== $row->region) {
                    DB::table('cases')->where('id', $row->id)->update(['region' => $value]);
                }
            }
        });
    // Якщо регіон не вказано, спробуємо витягнути його з назви (очікується формат «… (Регіон, …)»)
    DB::table('cases')
        ->select('id', 'title')
        ->whereNull('region')
        ->orderBy('id')
```

```
->chunkById(500, function ($rows) {
        foreach ($rows as $row) {
            if (empty($row->title)) {
                continue;
            }
            if (preg_match('/\(([^,]+),/u', $row->title, $matches)) {
                $region = Str::of($matches[1])->squish();
                $value = $region->isEmpty() ? null : $region->title()->value();
                if ($value !== null) {
                    DB::table('cases')->where('id', $row->id)->update(['region' => $value]);
                }
            }
        }
    });
}

public function down(): void
{
    // При відкаті видаляємо індекс та колонку region
    if (!Schema::hasColumn('cases', 'region')) {
        return;
    }
    try {
        DB::statement('DROP INDEX IF EXISTS cases_region_index');
    } catch (\Throwable $e) {
        // індекс може бути відсутній – ігноруємо
    }
    Schema::table('cases', function (Blueprint $table) {
```

```
  if (Schema::hasColumn('cases', 'region')) {
            $table->dropColumn('region');
        }
    });
  }
};
```

ДОДАТОК Б

Моделі Б.Д.

Сторінок 16

Київ-2025

Клас **CaseModel** представляє справу й оголошує зв’язки, доступні поля, а також визначає допоміжні атрибути.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Str;

class CaseModel extends Model
{
   use HasFactory;

   // Під час збереження автоматично витягуємо регіон із назви, якщо поле region порожнє.
   protected static function booted(): void
   {
      static::saving(function (CaseModel $case) {
         if ($case->region || empty($case->title)) {
            return; // регіон вже заданий або назва порожня
         }
         // очікуємо, що регіон вказано в дужках у форматі «(Регіон, …)»
         if (preg_match('/\(([^,]+),/u', $case->title, $matches)) {
            $case->region = $matches[1];
         }
      });
   }

   protected $table = 'cases';

   // Поля, які можна масово заповнювати через create/fill
   protected $fillable = [
      'title', 'description', 'user_id', 'status', 'executor_id', 'region',
      'claimant_name', 'debtor_name', 'deadline_at',
   ];

   // Кастування типів: deadline_at автоматично перетворюється на Carbon
```

```
protected $casts = [
        'deadline_at' => 'datetime',
    ];

    // Додаткові віртуальні атрибути, які повертаються при серіалізації моделі
    protected $appends = ['status_label', 'region_label'];

    // Доступні статуси
    public const STATUSES = ['new', 'in_progress', 'done', 'closed'];

    // Зв’язок з користувачем-власником
    public function owner() { return $this->belongsTo(User::class, 'user_id'); }
    // Зв’язок з виконавцем справи
    public function executor() { return $this->belongsTo(User::class, 'executor_id'); }
    // Дії у справі (останні спочатку)
    public function actions() { return $this->hasMany(CaseAction::class, 'case_id')->latest(); }
    // Документи у справі (останні спочатку)
    public function documents() { return $this->hasMany(CaseDocument::class, 'case_id')->latest(); }

    // Віртуальний атрибут: повертає локалізовану назву статусу (наприклад, «Нова», «У роботі»)
    public function getStatusLabelAttribute(): string
    {
        return __('statuses.' . $this->status);
    }
    // Віртуальний атрибут: нормалізоване відображення регіону або «Not specified»
    public function getRegionLabelAttribute(): string
    {
        return $this->region ?? __('Not specified');
    }

    // Список статусів з їхніми локалізованими назвами (для фільтрів)
    public static function statusOptions(): array
    {
        return collect(self::STATUSES)
            ->mapWithKeys(fn ($status) => [$status => __('statuses.' . $status)])
            ->toArray();
    }
    // Повертає унікальні регіони в базі, нормалізовані до стилю Title Case
    public static function regionOptions(): array
```

```
    {
        return self::query()
            ->whereNotNull('region')
            ->select('region')
            ->distinct()
            ->orderBy('region')
            ->pluck('region')
            ->map(fn ($value) => Str::of($value)->squish()->title()->value())
            ->filter()
            ->unique()
            ->values()
            ->toArray();
    }
    // Мутатор: при встановленні регіону нормалізуємо рядок (зайві пробіли, Title Case)
    public function setRegionAttribute(?string $value): void
    {
        if ($value === null) {
            $this->attributes['region'] = null;
            return;
        }
        $normalized = Str::of($value)->squish();
        $this->attributes['region'] = $normalized->isEmpty() ? null : $normalized->title()->value();
    }
}
```

Модель CaseAction

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class CaseAction extends Model
{
    use HasFactory;
    // Поля, які дозволено масово присвоювати
    protected $fillable = ['case_id','user_id','type','notes'];
    // Додаємо віртуальний атрибут type_label
```

```
    protected $appends = ['type_label'];
    // Зв’язок із справою
    public function case() { return $this->belongsTo(CaseModel::class, 'case_id'); }
    // Зв’язок із користувачем, який створив дію
    public function user() { return $this->belongsTo(User::class); }
    // Віртуальний атрибут: повертає локалізовану назву типу дії
    public function getTypeLabelAttribute(): string
    {
        return __('actions.' . $this->type);
    }
}
```

Модель CaseDocument

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class CaseDocument extends Model
{
    use HasFactory;
    // Поля, що дозволено масово призначати
    protected $fillable = ['case_id', 'uploaded_by', 'title', 'path', 'file_size', 'mime_type'];
    // Кастування поля file_size у ціле число
    protected $casts = ['file_size' => 'integer'];
    // Додаємо віртуальний атрибут human_size для зручного відображення розміру файлу
    protected $appends = ['human_size'];
    // Зв’язки
    public function case() { return $this->belongsTo(CaseModel::class, 'case_id'); }
    public function uploader() { return $this->belongsTo(User::class, 'uploaded_by'); }
    // Обчислення розміру файлу у зрозумілому форматі (КБ, МБ тощо)
    public function getHumanSizeAttribute(): ?string
    {
        if (!$this->file_size) {
            return null;
        }
```

```
            $units = ['B','KB','MB','GB','TB'];
            $bytes = max($this->file_size, 0);
            $pow = $bytes > 0 ? floor(min(count($units) - 1, log($bytes, 1024))) : 0;
            $bytes /= pow(1024, $pow);
            $decimals = $pow === 0 ? 0 : 1;
            return number_format($bytes, $decimals) . ' ' . $units[$pow];
        }
}
```

CaseController

```
<?php

namespace App\Http\Controllers;

use App\Models\CaseAction;
use App\Models\CaseDocument;
use App\Models\CaseModel;
use App\Models\User;
use App\Services\Reports\CasePdfExporter;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Gate;
use Illuminate\Support\Facades\Storage;
use Illuminate\Support\Str;

class CaseController extends Controller
{
    // Метод index відображає список справ з фільтрами та сортуванням
    public function index(Request $request)
    {
        Gate::authorize('list-cases'); // перевірка прав користувача
        $sort = $request->query('sort', 'created_at');
        $direction = $request->query('direction', 'desc');
        $query = CaseModel::with(['owner','executor']);
        // Якщо користувач виконавець – бачить лише свої справи
        if ($request->user()->isExecutor()) {
```

```
            $query->where('executor_id', $request->user()->id);
        }
        // Додаткові фільтри: за виконавцем, статусом, регіоном
        if ($executor = $request->query('executor')) {
            $query->where('executor_id', $executor);
        }
        if ($status = $request->query('status')) {
            $query->where('status', $status);
        }
        $selectedRegion = null;
        $regionInput = $request->query('region');
        if ($regionInput !== null && $regionInput !== '') {
            if ($regionInput === '__null__') {
                $query->whereNull('region');
                $selectedRegion = '__null__';
            } else {
                // нормалізуємо регіон і фільтруємо
                $normalizedRegion = Str::of($regionInput)->squish()->title()->value();
                $query->where('region', $normalizedRegion);
                $selectedRegion = $normalizedRegion;
            }
        }
        // пагінація та передача даних у подання
        $cases = $query->orderBy($sort, $direction)->paginate(12)->withQueryString();
        $executors = User::whereIn('role', ['executor','admin'])->orderBy('name')->get();
        $regions = CaseModel::regionOptions();
        $hasUnspecifiedRegion = CaseModel::whereNull('region')->exists();
        return view('cases.index',
compact('cases','sort','direction','executors','regions','hasUnspecifiedRegion','selectedRegion'));
    }

    // Метод create повертає форму створення справи
    public function create(Request $request)
    {
```

```
    Gate::authorize('create-case');
    $executors = User::whereIn('role', ['executor','admin'])->orderBy('name')->get();
    $regionOptions = CaseModel::regionOptions();
    return view('cases.create', compact('executors','regionOptions'));
}

// Метод store валідує форму та створює справу, а також записи в case_actions і case_documents
public function store(Request $request)
{
    Gate::authorize('create-case');
    $data = $request->validate([
        'title' => ['required','string','max:255'],
        'description' => ['nullable','string'],
        'executor_id' => ['nullable','exists:users,id'],
        'region' => ['nullable','string','max:120'],
        'claimant_name' => ['nullable','string','max:255'],
        'debtor_name' => ['nullable','string','max:255'],
        'deadline_at' => ['nullable','date'],
    ]);
    $data['user_id'] = $request->user()->id;
    $data['status'] = 'new';
    $case = CaseModel::create($data);
    // Реєструємо дію «створено»
    CaseAction::create([
        'case_id' => $case->id,
        'user_id' => $request->user()->id,
        'type' => 'created',
        'notes' => __('Case created'),
    ]);
    // Завантажуємо документи, якщо наявні
    if ($request->hasFile('documents')) {
        foreach ($request->file('documents') as $file) {
            $path = $file->store('cases/' . $case->id, 'public');
```

```
            $document = CaseDocument::create([
            'case_id' => $case->id,
            'uploaded_by' => $request->user()->id,
            'title' => $file->getClientOriginalName(),
            'path' => $path,
            'file_size' => $file->getSize(),
            'mime_type' => $file->getMimeType(),
        ]);
        // Реєструємо дію «додано документ»
        CaseAction::create([
            'case_id' => $case->id,
            'user_id' => $request->user()->id,
            'type' => 'document_added',
            'notes' => $document->title,
        ]);
      }
   }
   return redirect()->route('cases.show', $case)->with('ok', __('Case created successfully.'));
}

// Відображення конкретної справи
public function show(Request $request, CaseModel $case)
{
   Gate::authorize('view-case', $case);
   // Завантажуємо пов’язані дані (власника, виконавця, дії та документи)
   $case->load(['owner','executor','actions.user','documents.uploader']);
   $canUpdate = Gate::forUser($request->user())->check('update-case', $case);
   return view('cases.show', compact('case','canUpdate'));
}

// Додавання нової дії до справи
public function addAction(Request $request, CaseModel $case)
{
```

```
    Gate::authorize('update-case', $case);
    $data = $request->validate([
        'type' => ['required','string','max:50'],
        'notes' => ['nullable','string'],
    ]);
    $data['case_id'] = $case->id;
    $data['user_id'] = $request->user()->id;
    CaseAction::create($data);
    return back()->with('ok', __('Action added successfully.'));
}

// Експорт справи у PDF
public function exportPdf(Request $request, CaseModel $case, CasePdfExporter $exporter)
{
    Gate::authorize('view-case', $case);
    $pdf = $exporter->build($case);
    $filename = sprintf('case-%d-%s.pdf', $case->id, now()->format('Ymd-His'));
    return $pdf->download($filename);
}

// Завантаження окремого файлу у справу через AJAX
public function uploadDocument(Request $request, CaseModel $case)
{
    Gate::authorize('update-case', $case);
    $request->validate(['file' => ['required','file','max:20480']]);
    $uploadedFile = $request->file('file');
    $path = $uploadedFile->store('cases/' . $case->id, 'public');
    $document = CaseDocument::create([
        'case_id' => $case->id,
        'uploaded_by' => $request->user()->id,
        'title' => $uploadedFile->getClientOriginalName(),
        'path' => $path,
        'file_size' => $uploadedFile->getSize(),
```

```
            'mime_type' => $uploadedFile->getMimeType(),
        ]);
        CaseAction::create([
            'case_id' => $case->id,
            'user_id' => $request->user()->id,
            'type' => 'document_added',
            'notes' => $document->title,
        ]);
        return back()->with('ok', __('Document uploaded successfully.'));
    }
}
```

AnalyticsController

```
<?php

namespace App\Http\Controllers;

use App\Models\CaseAction;
use App\Models\CaseDocument;
use App\Models\CaseModel;
use App\Models\User;
use App\Services\Reports\AnalyticsExcelExporter;
use App\Support\AnalyticsCaseFilter;
use Illuminate\Http\Request;
use Illuminate\Support\Carbon;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Str;

class AnalyticsController extends Controller
{
    // Метод index агрегує дані для дашборду
    public function index()
```

```
{
    // Розподіл справ за статусами
    $statusTotals = CaseModel::select('status', DB::raw('COUNT(*) as total'))
        ->groupBy('status')
        ->orderBy('status')
        ->pluck('total','status')
        ->toArray();
    $statusLabels = CaseModel::statusOptions();
    $byStatus = collect($statusTotals)
        ->mapWithKeys(fn ($count,$status) => [$statusLabels[$status] ?? $status => (int) $count]);

    // Навантаження виконавців: топ-10 за кількістю справ
    $executorLoad = CaseModel::select('executor_id', DB::raw('COUNT(*) as total'))
        ->groupBy('executor_id')
        ->orderByDesc('total')
        ->take(10)
        ->get()
        ->map(function ($row) {
            $name = optional(User::find($row->executor_id))->name ?? __('Unassigned');
            return ['name' => $name, 'total' => (int) $row->total];
        });

    // Підрахунок прострочених та виконаних вчасно справ
    $overdue = CaseModel::whereIn('status', ['new','in_progress'])
        ->whereNotNull('deadline_at')
        ->where('deadline_at','<', now())
        ->count();
    $onTime = CaseModel::whereNotNull('deadline_at')
        ->where('deadline_at','>=', now())
        ->count();

    // Тренд реєстрацій справ за останні 12 місяців (формат YYYY-MM -> кількість)
    $trend = CaseModel::selectRaw("to_char(date_trunc('month', created_at), 'YYYY-MM') as label, COUNT(*) as total")
```

```
    ->where('created_at','>=', now()->startOfMonth()->subMonths(11))
    ->groupBy('label')
    ->orderBy('label')
    ->pluck('total','label');

// Топ-5 заявників за кількістю створених справ
$topApplicants = CaseModel::select('user_id', DB::raw('COUNT(*) as total'))
    ->groupBy('user_id')
    ->orderByDesc('total')
    ->take(5)
    ->get()
    ->map(function ($row) {
        $user = optional(User::find($row->user_id));
        return [ 'name' => $user?->name ?? __('Unknown'), 'total' => (int) $row->total ];
    });

// Сезонність: рахуємо кількість справ і середній термін виконання для кожного місяця
$casesForAnalysis = CaseModel::select('id','title','region','created_at','deadline_at')
    ->whereNotNull('created_at')
    ->get();
$seasonalitySeriesCollection = $casesForAnalysis
    ->groupBy(fn (CaseModel $case) => (int) $case->created_at->format('n'))
    ->sortKeys()
    ->map(function ($group,$monthNumber) {
        $monthDate = Carbon::create((int) now()->format('Y'), $monthNumber, 1);
        $monthName = $monthDate->locale(app()->getLocale())->isoFormat('MMMM');
        $normalizedName = Str::ucfirst(Str::lower($monthName));
        $total = $group->count();
        $leadValues = $group->map(function (CaseModel $case) {
            return $case->deadline_at ? $case->deadline_at->diffInDays($case->created_at) : null;
        })->filter();
        $avgLead = $leadValues->isNotEmpty() ? round(max($leadValues->avg(), 0), 1) : null;
        return [ 'month' => $normalizedName, 'total' => $total, 'avg_lead' => $avgLead ];
```

```
        })->values();
    // статистика за регіонами: кількість справ та середній термін виконання
    $regionStats = $casesForAnalysis
        ->groupBy(fn (CaseModel $case) => $this->resolveRegionLabel($case->region))
        ->map(function ($group,$regionLabel) {
            $leadValues = $group->map(function (CaseModel $case) {
                return $case->deadline_at ? $case->deadline_at->diffInDays($case->created_at) : null;
            })->filter();
            $avgLead = $leadValues->isNotEmpty() ? round(max($leadValues->avg(), 0), 1) : null;
            return [ 'region' => $regionLabel, 'total' => $group->count(), 'avg_lead' => $avgLead ];
        })->sortByDesc('total')->values();
    // Завантажуємо необов’язкову інформацію з OLAP-сховища (входи та реєстрації користувачів)
    $olap = $this->loadOlapSummary();
    // Повертаємо дані у подання 'analytics'
    return view('analytics', compact(
        'statusTotals','statusLabels','byStatus','executorLoad','onTime','overdue','trend',
        'topApplicants','olap','seasonalitySeriesCollection','regionStats'
    ));
}

// Метод records повертає детальний перелік справ із фільтрами та метриками
public function records(Request $request)
{
    // нормалізуємо параметри фільтрації (статус, виконавець, власник, регіон, діапазон дат)
    $filterData = $this->prepareFilters($request);
    $normalizedFilters = $filterData['normalized'];
    $filtersForView = $filterData['view'];
    $baseQuery = CaseModel::query()->with(['owner','executor']);
    // застосовуємо фільтри через окремий клас AnalyticsCaseFilter
    AnalyticsCaseFilter::apply($baseQuery, $normalizedFilters);
    // Копія запиту для підрахунку кількості дій та документів
    $casesQuery = (clone $baseQuery)
        ->withCount(['actions','documents'])
```

```
        ->withMax('actions','created_at');
    $cases = $casesQuery->orderByDesc('created_at')->paginate(25)->withQueryString();
    // Загальна кількість справ
    $totalCases = (clone $baseQuery)->count();
    // Підрахунок загальної кількості дій та документів для обчислення середніх значень
    $totalActions = CaseAction::whereHas('case', function ($query) use ($normalizedFilters) {
        AnalyticsCaseFilter::apply($query, $normalizedFilters);
    })->count();
    $totalDocuments = CaseDocument::whereHas('case', function ($query) use ($normalizedFilters) {
        AnalyticsCaseFilter::apply($query, $normalizedFilters);
    })->count();
    $avgActionsPerCase = $totalCases ? round($totalActions / $totalCases, 2) : 0;
    $avgDocumentsPerCase = $totalCases ? round($totalDocuments / $totalCases, 2) : 0;
    // Підготовка додаткових вибірок: розподіл за статусами, за виконавцями, динаміка за днями, типи дій
та типи документів
    // ...
    $filterOptions = [
        'statuses' => CaseModel::statusOptions(),
        'executors' => User::whereIn('role', ['executor','admin'])->orderBy('name')->get(),
        'owners' => User::orderBy('name')->get(),
        'regions' => CaseModel::regionOptions(),
        'hasUnspecifiedRegion' => CaseModel::whereNull('region')->exists(),
    ];
    $olap = $this->loadOlapSummary();
    return view('analytics.details', [
        'cases' => $cases,
        'totalCases' => $totalCases,
        'avgActionsPerCase' => $avgActionsPerCase,
        'avgDocumentsPerCase' => $avgDocumentsPerCase,
        // інші згенеровані масиви передаються у вигляді таблиць і графіків
        'filterOptions' => $filterOptions,
        'filters' => $filtersForView,
        'olap' => $olap,
    ]);
```

```
    }

    // Метод exportRecords дозволяє експортувати детальні записи у формат Excel
    public function exportRecords(Request $request, AnalyticsExcelExporter $exporter)
    {
        $filterData = $this->prepareFilters($request);
        return $exporter->download($filterData['normalized']);
    }

    // Допоміжний метод для нормалізації фільтрів
    protected function prepareFilters(Request $request): array
    {
        // валідуємо типи параметрів і перетворюємо пусті рядки на null
        $validated = $request->validate([
            'status' => ['nullable','string','max:50'],
            'executor' => ['nullable','integer','exists:users,id'],
            'owner' => ['nullable','integer','exists:users,id'],
            'region' => ['nullable','string','max:120'],
            'date_from' => ['nullable','date'],
            'date_to' => ['nullable','date'],
        ]);
        // ...
        // перетворюємо дати у об’єкти Carbon і застосовуємо типові діапазони
        $hasDateFilter = $request->filled('date_from') || $request->filled('date_to');
        $dateFrom = $request->filled('date_from') ? Carbon::parse($request->input('date_from'))->startOfDay() :
($hasDateFilter ? null : Carbon::now()->subDays(90)->startOfDay());
        $dateTo   = $request->filled('date_to')   ? Carbon::parse($request->input('date_to'))->endOfDay()   : null;
        // Повертаємо два варіанти: normalized (для застосування до запитів) і view (для відображення у формі)
        return [
            'normalized' => [ /* статус, executor, owner, region, date_from, date_to */ ],
            'view' => [ /* ті самі поля у форматі рядків */ ],
        ];
    }
```

```
    // Нормалізація відображення регіону (прибираємо зайві пробіли, приводимо до Title Case)
    protected function resolveRegionLabel(?string $region): string
    {
        if ($region === null) {
            return __('Not specified');
        }
        $normalized = Str::of($region)->squish();
        if ($normalized->isEmpty()) {
            return __('Not specified');
        }
        return $normalized->title()->value();
    }

    // Завантаження статистики входів та реєстрацій з OLAP (може повернути порожній набір при відсутності підключення)
    protected function loadOlapSummary(): array
    {
        try {
            $connection = DB::connection(config('olap.connection'));
            $logins = $connection->table('fact_user_logins')
                ->selectRaw("date(to_date(date_key, 'YYYYMMDD')) as metric_date, SUM(login_count) as total")
                ->groupBy('metric_date')
                ->orderBy('metric_date')
                ->take(30)
                ->get();
            $registrations = $connection->table('fact_user_registrations')
                ->selectRaw("date(to_date(date_key, 'YYYYMMDD')) as metric_date, COUNT(*) as total")
                ->groupBy('metric_date')
                ->orderBy('metric_date')
                ->take(30)
                ->get();
            return ['enabled' => true, 'logins' => $logins, 'registrations' => $registrations];
        } catch (\Throwable $e) {
            return ['enabled' => false, 'logins' => collect(), 'registrations' => collect()];
```