МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри

_____Юрій ІСКРЕНКО
"_____"_____2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ "МАГІСТР"
ЗА СПЕЦІАЛЬНІСТЮ 123 "КОМП'ЮТЕРНА ІНЖЕНЕРІЯ"

Тема: Відновлення інформаційного забезпечення технічного обслуговування та ремонту літаків авіакомпанії у післявоєнний час

Виконавець: Максим ВОСВИЛО

Керівник: Вадим СУРАЄВ

Нормоконтролер: Наталія ФОМІНА

Київ - 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ЗАТВЕРДЖУЮ
Завідувач кафедри КСМ
_____Юрій ІСКРЕНКО.
"____" _____2025 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Восвила Максима Олеговича
(прізвище, ім'я, по батькові здобувача вищої освіти в родовому відмінку)

1. Тема кваліфікаційної роботи: «Відновлення інформаційного забезпечення технічного обслуговування та ремонту літаків авіакомпанії у післявоєнний час» Затверджена наказом ректора від " 24 " жовтня 2025 р.№2344/ст
2. Термін виконання роботи (проєкту): з 29.09.2025 по 31.12.2025
3. Вихідні дані до роботи (проєкту): мова програмуванн *Python*, фреймворк *FastAPI*, СУБД *PostgreSQL*, фреймворк *React*, бібліотека *Recharts*, інструмент збірки *Vite*.
4. Зміст пояснювальної записки: аналіз предметної області технічного обслуговування повітряних суден та існуючих *MRO-с*истем, обґрунтування вибору технологій розробки, проєктування архітектури системи та бази даних, розробка алгоритму *gap*-аналізу, реалізація веб-інтерфейсу
5. Перелік обов'язкового графічного (ілюстративного) матеріалу: результати роботи представленні у вигляді презентації.

6. Календарний план

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Затвердження теми роботи	29.09.2025	
2.	Ознайомлення з постановкою задачі та вивчення літератури	01.10.2025	
3.	Дослідження і аналіз існуючих *MRO*-систем	15.10.2025	
4.	Аналіз та вибір технологій для розробки системи	21.10.2025	
5.	Проектування архітектури системи та бази даних	03.11.2025	
6.	Розробка серверної частини	17.11.2025	
7.	Розробка клієнтської частинри	24.11.2025	
8.	Тестування та перевірка роботи системи	28.11.2025	
9.	Написання пояснювальної записки	12.12.2025	
10.	Отримання відгуку та рецензії	15.12.2025	
11.	Підготовка до захисту кваліфікаційної роботи	19.12.2025	
12.	Захист кваліфікаційної роботи	23.12.2025 – 31.12.2025	

7. Консультація з окремого(мих) розділу(ів):

Назва розділу	Консультант (посада, П.І.Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв

8. Дата видачі завдання: «29» вересня 2025 р.

Керівник кваліфікаційної роботи (проєкту): Вадим СУРАЄВ_____
(підпис керівника)

Завдання прийняв до виконання: Максим ВОСВИЛО_____
(підпис виконавця)

РЕФЕРАТ

Пояснювальна записка до кваліфіккаційної роботи “Відновлення інформаційного забезпечення технічного обслуговування та ремонту літаків авіакомпанії у післявоєнний час”: 100 сторінок, 39 рисунків, 6 таблиць, 38 інформаційних джерел.

Об’єкт дослідження – технічне обслуговування повітряних суден авіакомпанії.

Предмет дослідження – система відновлення інформаційного забезпечення технічного обслуговування та ремонту літаків.

Мета дослідження – розробка системи відновлення інформаційного забезпечення технічного обслуговування та ремонту літаків авіакомпанії.

Технічні та програмні засоби – мова програмування *Python* та фреймворк *FastAPI*, бібліотеки *Pandas* та *Openpyxl* для обробки даних, *ORM SQLAlchemy* та система міграцій *Alembic*, база даних *PostgresSQL*, бібліотека *React* для побудови веб-інтерфейсу, збірник *Vite*, бібліотека *Recharts* для візуалізації.

Основні характеристики та показники – система побудована за трирівневою клієнт-серверною архітектурою, база даних містить 6 взаємопов’язаних таблиць, *scoring*-оцінка враховує 5 факторів критичності з чотирирівневою шкалою пріоритетності прогалин, підтримується імпорт даних у форматах *CSV* та *Excel*.

Отримані результати та їх новизна – результатом виконання кваліфікаційної роботи є розроблена система відновлення інформаційного забезпечення технічного обслуговування літаків, призначена для авіакомпаній у післявоєнний час. При розробці була застосована методика *scoring*-оцінки критичності прогалин на основі п’яти факторів, що дозволяє автоматично визначити пріоритетність відновлення документації.

Рекомендації щодо використання результатів – розроблена система рекомендується для забезпечення консолідації фрагментованих даних

технічного обслуговування та виявлення прогалин у документації авіакомпанії у післявоєнний період.

ТЕХНІЧНЕ ОБСЛУГОВУВАННЯ ПОВІТРЯНИХ СУДЕН, *GAP*-АНАЛІЗУ, *SCORING*-ОЦІНКА, ФРЕЙМВОРК *FASTAPI*, МОВА ПРОГРАМУВАННЯ *PYTHON*, СУБД *POSTGRESQL*, ВІДНОВЛЕННЯ ДАНИХ, АВІАЦІЙНА ДОКУМЕНТАЦІЯ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ. ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СУБД – Система управління базами даних

БД – база даних

JSON – англ. *JavaScript Object Notation* – текстовий формат обміну даними, заснований на об’єктній нотації JavaScript.

MRO – англ. *Maintenance, Repair and Overhaul* – технічне обслуговування, ремонт і капітальний ремонт.

ТО – технічне обслуговування.

МАУ – міжнародні авіалінії України.

ER – англ. *Entity–Relationship* – “сутність–зв’язок”. Модель для проєктування структури бази даних і зв’язків між сутностями.

ВСТУП

Потреба у створенні ефективної системи відновлення даних технічного обслуговування повітряних суден для українських авіакомпаній зумовлена викликами, що виникли в процесі відновлення після повномасштабного вторгнення. За цей період авіаційна галузь зазнала значних втрат: було знищено або пошкоджено інфраструктуру, втрачено доступ до серверів і архівів, а також порушено взаємодію з *MRO*-провайдерами. У таких умовах особливої важливості набуває надійна інформаційна система, побудована на технологіях з відкритим кодом, яка дасть змогу консолідувати фрагментовані дані та відновити технічну документацію, необхідну для підтвердження льотної придатності авіаційного флоту.

Актуальність теми. Технічне обслуговування повітряних суден супроводжується обов'язковим документуванням усіх виконаних робіт. Відсутність запису про виконання роботи юридично еквівалентна її невиконанню, що унеможливлює отримання сертифіката льотної придатності. Війна призвела до фрагментації даних ТО українських авіакомпаній: сервери знищені або недоступні, паперові архіви втрачені, записи розкидані по резервних копіях, *Excel*-файлах та у сторонніх *MRO*-провайдерів.

Після завершення воєнних дій очікується відновлення авіаційного сполучення, тому авіакомпаніям необхідний інструмент для швидкої консолідації фрагментованих даних та виявлення прогалин у документації. Обмежений бюджет на відновлення спонукає до використання технологій з відкритим вихідним кодом, які дозволять зменшити витрати на розробку.

Мета і завдання виконання кваліфікаційної роботи. Метою кваліфікаційної роботи є розробка системи відновлення інформаційного забезпечення технічного обслуговування та ремонту літаків авіакомпанії. Для досягнення поставленої мети необхідно дослідити предметну область технічного

обслуговування повітряних суден, регуляторні вимоги до документації та існуючі *MRO*-системи.

Розробити архітектуру та алгоритми програмного забезпечення, які забезпечать консолідацію фрагментованих даних, автоматичне виявлення прогалин та пріоритезацію їх відновлення. Обрати оптимальне середовище розробки, мови програмування, фреймворки та інші засоби, що дозволять реалізувати поставлені завдання.

Об'єкт і предмет дослідження. Об'єктом дослідження даної роботи є технічне обслуговування повітряних суден авіакомпанії. Предметом даної роботи є система відновлення інформаційного забезпечення технічного обслуговування та ремонту літаків.

Методи та дослідження. Для розробки системи використовувалась мова *Python* з фреймворком *FastAPI*. Також додатково використовувалися бібліотеки *SQLAlchemy, Alembic, Pydantic, Pandas* та *Openpyxl*. Розробка відбувалась у інтегрованому середовищі розробки *PyCharm*. Для роботи з даними була використана система управління базами даних (СУБД) *PostgreSQL*. Клієнтська частина розроблена з використанням бібліотеки *React* та інструменту збірки *Vite*. Для візуалізації даних використано бібліотеку *Recharts*.

Під час розробки системи використовувалась трирівнева клієнт-серверна архітектура як основа веб-додатку. Взаємодія між клієнтом та сервером реалізована через *REST API* з обміном даними у форматі *JSON*. Було реалізовано модулі імпорту даних з форматів *CSV* та *Excel*, нормалізації даних, *gap*-аналізу та *scoring*-оцінки критичності прогалин.

Наукова новизна отриманих результатів. В ході виконання кваліфікаційної роботи було створено систему відновлення даних технічного обслуговування для українських авіакомпаній. Під час розробки була застосована методика *scoring*-оцінки критичності прогалин, що враховує п'ять факторів: критичність компонента, час з останнього ТО, тип роботи, наявність обмеженого ресурсу та надійність джерела даних. Це дозволяє автоматично

визначати пріоритетність відновлення документації та оптимізувати використання обмежених ресурсів.

Практичне значення отриманих результатів. Розроблену систему можуть використовувати українські авіаційні компанії, яким необхідний інструмент для консолідації фрагментованих даних технічного обслуговування та підготовки до відновлення сертифікації флоту. Система розроблена з використанням технологій з відкритим вихідним кодом, що дозволяє зменшити витрати на впровадження. З використанням даної системи авіакомпанії можуть швидше відновити документацію ТО та підготувати дані для міграції у промислові *MRO*-системи.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Проблематика втрати даних технічного обслуговування в кризових умовах

Воєнні конфлікти, політична нестабільність та техногенні катастрофи створюють унікальні виклики для авіаційної галузі, що виходять далеко за межі фізичних пошкоджень інфраструктури та повітряних суден. Одним із найбільш недооцінених, проте критичних наслідків кризових ситуацій є втрата або фрагментація даних технічного обслуговування. На відміну від пошкодженого літака, який можна відремонтувати або списати, втрачена документація ТО створює юридичну прогалину, що унеможливлює експлуатацію технічно справних повітряних суден.

Сучасна авіаційна система базується на принципі документального підтвердження льотної придатності. Повітряне судно вважається придатним до польотів не тоді, коли воно технічно справне, а тоді, коли існують документальні докази виконання всіх регламентних робіт, директив льотної придатності та відстеження історії компонентів. Відсутність запису юридично еквівалентна невиконанню роботи.

У штатних умовах авіакомпанії підтримують централізовані системи обліку ТО, регулярно створюють резервні копії, зберігають паперові архіви. Однак кризові ситуації руйнують цю інфраструктуру комплексно: сервери знищуються або стають недоступними, персонал евакуюється, паперові архіви втрачаються, зв'язок із закордонними провайдерами ТО переривається.

Після завершення активної фази кризи авіакомпанії стикаються з ситуацією, коли дані про ТО існують фрагментарно у різних джерелах: часткові резервні копії різних дат, *Excel*-файли з локальних комп'ютерів, паперові документи, що вціліли, записи від *MRO*-провайдерів за кордоном, дані лізингодавців. Жодне з цих джерел не є повним, формати різняться, а хронологічна безперервність порушена.

Досвід авіакомпаній, що пережили кризові ситуації, демонструє системність та довготривалість наслідків втрати даних ТО. Авіакомпанія *Iraqi Airways* після вторгнення 2003 року втратила доступ до централізованих систем обліку. Незважаючи на наявність технічно справних повітряних суден, авіакомпанія не могла відновити міжнародні рейси протягом 18 місяців через неможливість підтвердити льотну придатність флоту. Станом на 2025 рік *Iraqi Airways* досі перебуває під забороною польотів до Європейського Союзу, що частково пов'язано з питаннями щодо стандартів ТО та документації [1] .

Лівійські авіакомпанії *Afriqiyah Airways* та *Libyan Airlines* зазнали катастрофічних втрат під час конфлікту 2014 року, коли значна частина флоту була знищена в аеропорту Тріполі. Проте навіть для літаків, що вціліли, відновлення експлуатації виявилося надзвичайно складним через втрату документації. У 2025 році міжнародні правоохоронні органи провели арешти, пов'язані з використанням підроблених запасних частин та документів ТО лівійськими операторами – прямий наслідок зруйнованої системи обліку та верифікації [1].

Українські авіакомпанії з початку повномасштабного вторгнення 2022 року опинилися в безпрецедентній ситуації. Авіакомпанія "Міжнародні авіалінії України" (МАУ) [34] перебуває в процесі банкрутства, маючи близько 37 повітряних суден, заблокованих у різних юрисдикціях. Питання збереження та консолідації даних ТО для цього флоту залишається невирішеним і безпосередньо впливатиме на можливість повернення літаків до експлуатації або їх передачі іншим операторам. Авіакомпанія *SkyUp*, усвідомлюючи ризики, превентивно впровадила систему *AMOS* для збереження даних ТО літаків, що

перебувають за кордоном. Цей випадок демонструє як важливість проблеми, так і обмеженість рішення: промислова *MRO*-система ефективна для поточного обліку, але не призначена для роботи з уже фрагментованими даними.

Аналіз кризових ситуацій дозволяє виділити характерні сценарії, що призводять до фрагментації даних ТО:

- **Перший сценарій** – фізичне знищення інфраструктури. Сервери, що містять бази даних *MRO*-систем, знищуються внаслідок бойових дій, пожеж або затоплення. Паперові архіви, що традиційно слугували резервом, втрачаються разом із приміщеннями. Резервні копії, якщо вони зберігалися локально, також знищуються.
- **Другий сценарій** – втрата доступу без фізичного знищення. Сервери залишаються на окупованій або недоступній території. Хмарні сервіси блокуються через санкції або припинення оплати. Персонал із правами доступу евакуюється або втрачає зв'язок з організацією.
- **Третій сценарій** – організаційний розпад. Авіакомпанія припиняє діяльність, персонал звільняється, знання про структуру даних та паролі втрачаються. Ліквідатори або нові власники не мають компетенцій для роботи з авіаційною документацією.
- **Четвертий сценарій** – розподіл флоту між юрисдикціями. Літаки, що перебували за кордоном на момент кризи, обслуговуються різними *MRO*-провайдерами. Кожен провайдер має лише фрагмент історії ТО. Центральна система, що мала б консолідувати дані, недоступна.

Фрагментація даних ТО має каскадний ефект на всі аспекти діяльності авіакомпанії. Неможливість підтвердити льотну придатність означає заземлення флоту незалежно від технічного стану. Кожен день простою генерує збитки: лізингові платежі продовжують нараховуватися, вартість зберігання літаків у закордонних аеропортах є значною, втрачаються слоти та комерційні контракти. Повторне виконання робіт, які вже були виконані, але не можуть бути підтверджені документально, вимагає величезних витрат. *C*-перевірка одного середньомагістрального літака коштує від 500 тисяч до 2 мільйонів доларів.

Повторна перевірка компонентів з обмеженим ресурсом може вимагати їх передчасної заміни. Регуляторна невизначеність ускладнює будь-які транзакції з повітряними суднами. Продаж або передача в сублізинг літака без повної документації ТО практично неможливі. Страхові компанії відмовляються покривати повітряні судна з прогалинами в історії ТО.

Аналіз ринку програмного забезпечення для управління ТО показує відсутність спеціалізованих рішень, призначених для консолідації та відновлення фрагментованих даних після кризових ситуацій. Існуючі системи розроблені для інших сценаріїв використання та не покривають цю нішу. Промислові *MRO*-системи – *AMOS, Ramco, TRAX, OASES* – призначені для штатної експлуатації авіакомпаній та провайдерів ТО [2]. Вони забезпечують поточний облік робіт, планування регламентів, контроль ресурсів компонентів, інтеграцію з іншими системами підприємства. Однак їхня архітектура передбачає безперервний потік даних від початку експлуатації або від моменту впровадження системи. Вони не мають вбудованих механізмів для імпорту даних з різнорідних джерел у довільних форматах, автоматичного виявлення прогалин у документації, пріоритезації зусиль з відновлення даних та роботи в умовах неповноти інформації.

Системи управління документацією, зокрема *flydocs*, надають розвинені функції *gap*-аналізу та роботи з технічними записами [3]. Проте ці інструменти орієнтовані на процеси передачі літаків між операторами та лізингодавцями, де передбачається наявність базового масиву даних, який потрібно перевірити та доповнити. Робота з фрагментованими даними після кризових ситуацій вимагає залучення дорогих професійних послуг з міграції та не масштабується на рівень цілого флоту в умовах обмежених ресурсів.

Таким чином, існує незаповнена ніша інструментів первинної консолідації та аналізу фрагментованих даних ТО – проміжної ланки між хаотичними джерелами різних форматів та повноцінними *MRO*-системами, яка дозволила б авіакомпаніям швидко оцінити стан документації флоту та пріоритезувати зусилля з відновлення.

1.2 Оргназіація технічного обслуговування повітряних суден та вимоги до документації

Технічне обслуговування повітряних суден є системою регламентованих процедур, спрямованих на підтримання льотної придатності авіаційної техніки протягом усього життєвого циклу. Розуміння структури ТО та ролі документації є необхідною передумовою для проєктування системи відновлення даних, оскільки саме ця структура визначає, які записи є критичними та як вони пов'язані між собою.

Система ТО цивільних повітряних суден поділяється на два основні напрями: лінійне технічне обслуговування та базове технічне обслуговування.

Лінійне ТО виконується безпосередньо на стоянці літака між рейсами або під час короткочасних зупинок. До нього належать передпольотний огляд, післяпольотний огляд, транзитне обслуговування, усунення дефектів, виявлених екіпажем, та позапланова заміна компонентів. Лінійне ТО забезпечує щоденну експлуатаційну придатність повітряного судна без виведення його з розкладу польотів. Характерною особливістю лінійного ТО є висока частота виконання робіт та відносно невеликий обсяг кожної окремої операції.

Базове ТО передбачає виконання планових регламентних робіт, що вимагають виведення повітряного судна з експлуатації на тривалий період. Традиційно базове ТО структурується за системою літерних перевірок різної глибини та періодичності:

- ***A*-перевірка** виконується приблизно кожні 500-800 льотних годин або 200-300 циклів і включає детальний огляд критичних систем, перевірку рівнів експлуатаційних рідин, функціональні тести систем, змащування механізмів. Тривалість *A*-перевірки зазвичай становить 1-2 доби.
- ***B*-перевірка** є проміжною формою ТО між *A*- та *C*-перевірками, що традиційно виконувалась кожні 4-6 місяців. Вона включає більш детальні огляди та перевірки систем порівняно з *A*-перевіркою. Варто зазначити, що в сучасних програмах ТО багатьох виробників, зокрема для літаків *Boeing* та *Airbus*, *B*-

перевірка як окрема форма поступово виходить з ужитку – її завдання розподіляються між розширеними *A*-перевірками або включаються до пакетів регламентних робіт. Проте для деяких типів повітряних суден та у програмах окремих експлуатантів *B*-перевірка залишається самостійною формою ТО.

- ***C*-перевірка** є значно глибшою інспекцією, що виконується кожні 18-24 місяці або після напрацювання 4000-6000 льотних годин. Вона передбачає детальний огляд конструкції планера, перевірку систем з частковим розбиранням, заміну компонентів за напрацюванням, виконання модифікацій та директив льотної придатності. Тривалість *C*-перевірки становить 2-4 тижні, а вартість для середньомагістрального літака – від 500 тисяч до 1,5 мільйона доларів США.
- ***D*-перевірка**, або важка структурна форма ТО, виконується кожні 8-12 років і є найбільш комплексною інспекцією повітряного судна. Вона включає повне зняття інтер'єру салону, розбирання та дефектацію силових елементів конструкції, капітальний ремонт або заміну систем, повне перефарбування. Тривалість *D*-перевірки може досягати 2-3 місяців, а вартість – декількох мільйонів доларів.

Сучасні програми ТО часто використовують концепцію пакетування робіт, коли замість чітко визначених літерних перевірок виконуються пакети завдань, згруповані за зручністю виконання та оптимізацією простою повітряного судна.

Окремим напрямом є ТО знімних компонентів: двигунів, допоміжних силових установок, агрегатів шасі, блоків авіоніки, гідравлічних та пневматичних агрегатів. Компонентне ТО виконується спеціалізованими майстернями за окремими програмами, що визначаються виробниками компонентів.

Особливістю компонентного ТО є принцип модульності: несправний або такий, що виробив ресурс, компонент демонтується з повітряного судна та замінюється справним, а ремонт виконується окремо. Це дозволяє мінімізувати час простою літака, але створює додаткові вимоги до системи обліку: необхідно

відстежувати історію кожного компонента незалежно від того, на якому повітряному судні він встановлений.

Кожен тип повітряного судна експлуатується відповідно до затвердженої програми технічного обслуговування, яка визначає повний перелік регламентних робіт, їх періодичність та обсяг. Формування програми ТО є багаторівневим процесом, що починається від виробника повітряного судна та завершується затвердженням авіаційною владою держави реєстрації. На етапі сертифікації типу виробник разом з авіаційними властями розробляє базову програму ТО – звіт наглядової ради з технічного обслуговування [4].

Цей документ містить мінімальні вимоги до переліку робіт, їх періодичності та критерії прийнятності результатів. Базова програма формується на основі аналізу проєктної документації, результатів випробувань, оцінки надійності систем та конструкції, а також досвіду експлуатації аналогічних типів повітряних суден.

На основі базової програми та рекомендацій виробника експлуатант розробляє власну програму ТО, адаптовану до конкретних умов використання флоту. При цьому враховуються інтенсивність експлуатації – кількість льотних годин та циклів на рік; профіль польотів – середня тривалість рейсу, частка зльотів і посадок у загальному напрацюванні; кліматичні умови базування та маршрутів; вік повітряних суден та їх індивідуальна історія; особливості конфігурації та встановленого обладнання.

Структурно програма ТО складається з переліку карток завдань. Кожна картка описує окрему роботу та містить унікальний ідентифікатор завдання, опис змісту роботи та критерії прийнятності, періодичність виконання за льотними годинами, циклами або календарним часом, застосовність до конкретних серійних номерів та конфігурацій, посилання на документацію виробника та регуляторні вимоги. Сукупність завдань групується у пакети робіт, що виконуються разом під час планових форм ТО.

Програма ТО є динамічним документом, що переглядається протягом усього періоду експлуатації типу на основі нових директив льотної придатності,

рекомендацій виробника, результатів аналізу надійності та змін у регуляторних вимогах. Кожна зміна програми підлягає затвердженню авіаційною владою.

Документування є критичним елементом системи ТО. З юридичної та регуляторної точки зору, саме документи підтверджують виконання робіт та є основою для сертифікації льотної придатності. Відсутність запису юридично еквівалентна невиконанню роботи, незалежно від фактичного технічного стану повітряного судна.

Бортовий технічний журнал є первинним документом оперативного обліку. У ньому фіксуються всі польоти із зазначенням напрацювання, виявлені екіпажем дефекти, виконані роботи з лінійного ТО та їх результати. Записи засвідчуються підписами інженерно-технічного персоналу, який має відповідні допуски до виконання та підтвердження робіт. Записи про виконане ТО формують повну історію обслуговування кожного повітряного судна. Вони включають заповнені картки завдань з підписами виконавців, дані про дефекти та усунення несправностей, акти виконаних робіт, сертифікати повернення до експлуатації після базового ТО.

Формуляри компонентів з обмеженим ресурсом містять повну історію кожного критичного агрегату: дати виробництва та введення в експлуатацію, напрацювання з початку експлуатації та після останнього ремонту, перелік виконаних ремонтів та модифікацій, дані про переміщення між повітряними суднами. До компонентів з обмеженим ресурсом належать диски та вали двигунів, силові елементи шасі, деталі системи керування польотом та інші агрегати, руйнування яких створює пряму загрозу безпеці польоту.

Фундаментальною вимогою до документації ТО є безперервність записів – можливість простежити повний ланцюг від виробництва повітряного судна або компонента через усі виконані роботи до поточного стану. Для компонентів безперервність означає можливість відстежити так звану історію від народження: хто виробив компонент, коли він був введений в експлуатацію, на яких повітряних суднах встановлювався, які ремонти виконувались, яке поточне напрацювання та залишковий ресурс. Розрив цього ланцюга ставить під сумнів

придатність компонента до подальшої експлуатації. Для повітряного судна в цілому безперервність означає наявність записів про виконання всіх регламентних робіт згідно з програмою ТО, всіх застосовних директив льотної придатності, усіх модифікацій, що впливають на конфігурацію. Прогалина в цій історії унеможливлює підтвердження льотної придатності.

Таким чином, інформаційне забезпечення ТО є не допоміжною адміністративною функцією, а критичною складовою системи підтримання льотної придатності, юридично рівнозначною фізичному стану повітряного судна.

1.3 Регуляторні вимоги щодо льотної придатності

Льотна придатність повітряного судна є юридичним статусом, що підтверджує відповідність його конструкції, технічного стану та документації встановленим вимогам безпеки. Регулювання льотної придатності здійснюється на міжнародному рівні через стандарти Міжнародної організації цивільної авіації та на національному або регіональному рівні через вимоги відповідних авіаційних властей. Для українських авіакомпаній, що орієнтуються на європейський ринок, ключовими є вимоги Агентства авіаційної безпеки Європейського Союзу.

Міжнародна організація цивільної авіації встановлює базові стандарти льотної придатності через систему Додатків до Конвенції про міжнародну цивільну авіацію. Додаток 8 визначає загальні вимоги до льотної придатності повітряних суден, а Додаток 6 встановлює вимоги до експлуатації, включаючи технічне обслуговування [5]. Ключовим принципом, закріпленим у стандартах *ICAO*, є відповідальність держави реєстрації за підтримання льотної придатності повітряних суден, внесених до її реєстру.

Держава реєстрації зобов'язана забезпечити наявність системи підтримання льотної придатності, яка включає затверджену програму ТО для кожного повітряного судна; виконання ТО організаціями з відповідними

допусками; ведення записів про виконане ТО; контроль виконання директив льотної придатності.

Стандарти *ICAO* визначають мінімальні вимоги до записів ТО [6], які повинні зберігатися: загальний час експлуатації планера, двигунів та повітряних гвинтів у льотних годинах та циклах; поточний статус відповідності директивам льотної придатності; статус виконання модифікацій та ремонтів згідно з рекомендаціями виробника; час експлуатації та статус компонентів з обмеженим ресурсом; дані про масу та центрування; записи про виконане ТО до наступної планової форми.

Регулювання льотної придатності в Європейському Союзі здійснюється через систему Регламентів Комісії ЄС, імплементація яких є обов'язковою для держав-членів та асоційованих країн. Основним документом є Регламент (ЄС) № 1321/2014, що встановлює вимоги до підтримання льотної придатності повітряних суден [7]. Частина *M* цього Регламенту визначає загальні вимоги до підтримання льотної придатності: обов'язки власника та експлуатанта щодо забезпечення льотної придатності; вимоги до програм ТО та їх затвердження; вимоги до організацій з управління підтриманням льотної придатності; порядок видачі та поновлення сертифікатів льотної придатності. Частина 145 встановлює вимоги до організацій з технічного обслуговування: умови отримання та підтримання схвалення на виконання ТО; вимоги до персоналу, приміщень, обладнання та інструменту; процедури виконання та документування робіт; систему управління якістю. Частина 66 визначає вимоги до сертифікаційного персоналу з ТО – інженерів та техніків, які мають право підтверджувати виконання робіт та повернення повітряного судна до експлуатації.

Сертифікат льотної придатності є офіційним документом, що підтверджує відповідність повітряного судна встановленим вимогам. Без чинного сертифіката експлуатація повітряного судна заборонена. Первинний сертифікат видається після виробництва повітряного судна та підтвердження його відповідності схваленій типовій конструкції. Надалі чинність сертифіката підтримується через систему періодичних перевірок льотної придатності.

Перевірка льотної придатності виконується щорічно уповноваженим персоналом організації з управління підтриманням льотної придатності або авіаційною владою. За результатами перевірки видається *с*відоцтво перевірки льотної придатності, яке підтверджує, що на дату перевірки повітряне судно відповідало встановленим вимогам.

Критичним елементом перевірки є аналіз документації ТО. Перевіряється наявність та повнота записів про виконане ТО згідно з програмою; статус виконання всіх застосовних директив льотної придатності; статус компонентів з обмеженим ресурсом; документація на встановлені компоненти та їх походження; записи про модифікації та ремонти.

Якщо під час перевірки льотної придатності виявлено відсутність записів про виконання обов'язкових робіт, можливі наслідки включають призупинення або анулювання сертифіката льотної придатності; вимогу повторного виконання робіт з невідомим статусом; обмеження експлуатації до усунення невідповідностей; у крайніх випадках – вимогу повної ресертифікації повітряного судна. Для компонентів з обмеженим ресурсом відсутність повної історії від виробництва може означати неможливість подальшого використання компонента, незалежно від його фактичного стану та залишкового ресурсу за оцінкою.

Державна авіаційна служба України як національна авіаційна влада регулює льотну придатність повітряних суден, зареєстрованих в Україні. Україна є договірною державою *ICAO* та імплементує міжнародні стандарти у національне законодавство. У контексті європейської інтеграції Україна поступово приводить власні авіаційні правила у відповідність з вимогами Європейського Союзу. Угода про спільний авіаційний простір між Україною та ЄС передбачає повну імплементацію європейських авіаційних стандартів, включаючи вимоги до підтримання льотної придатності.

Для українських авіакомпаній, що планують відновлення діяльності після завершення воєнних дій, відповідність європейським стандартам документації ТО є критичною передумовою доступу до європейського ринку авіаперевезень.

1.4 Огляд та порівняльний аналіз існуючих *MRO*-систем

Сучасний ринок програмного забезпечення для управління технічним обслуговуванням повітряних суден представлений низкою промислових рішень, що забезпечують комплексну автоматизацію процесів ТО. Аналіз провідних систем дозволяє визначити їхні функціональні можливості, архітектурні особливості та обмеження в контексті завдання відновлення фрагментованих даних після кризових ситуацій.

Система ***AMOS***, розроблена компанією *Swiss AviationSoftware*, є одним з лідерів ринку *MRO*-програмного забезпечення (див. рисунок 1.1). Станом на 2025 рік систему використовують понад 180 клієнтів по всьому світу [8], включаючи авіакомпанії різного масштабу – від стартапів до найбільших перевізників та авіаційних груп.

Рисунок 1.1 – Інтерфейс інформаційної системи *AMOS* для управління технічним обслуговуванням і ремонтом літаків

AMOS побудована за модульним принципом та охоплює повний спектр функцій управління ТО (див. рисунок 1.2): планування та прогнозування регламентних робіт; управління матеріально-технічним забезпеченням та складськими запасами; контроль конфігурації флоту; управління виробництвом та ресурсами; забезпечення регуляторної відповідності через модуль управління якістю; інтеграцію з фінансовими системами підприємства.

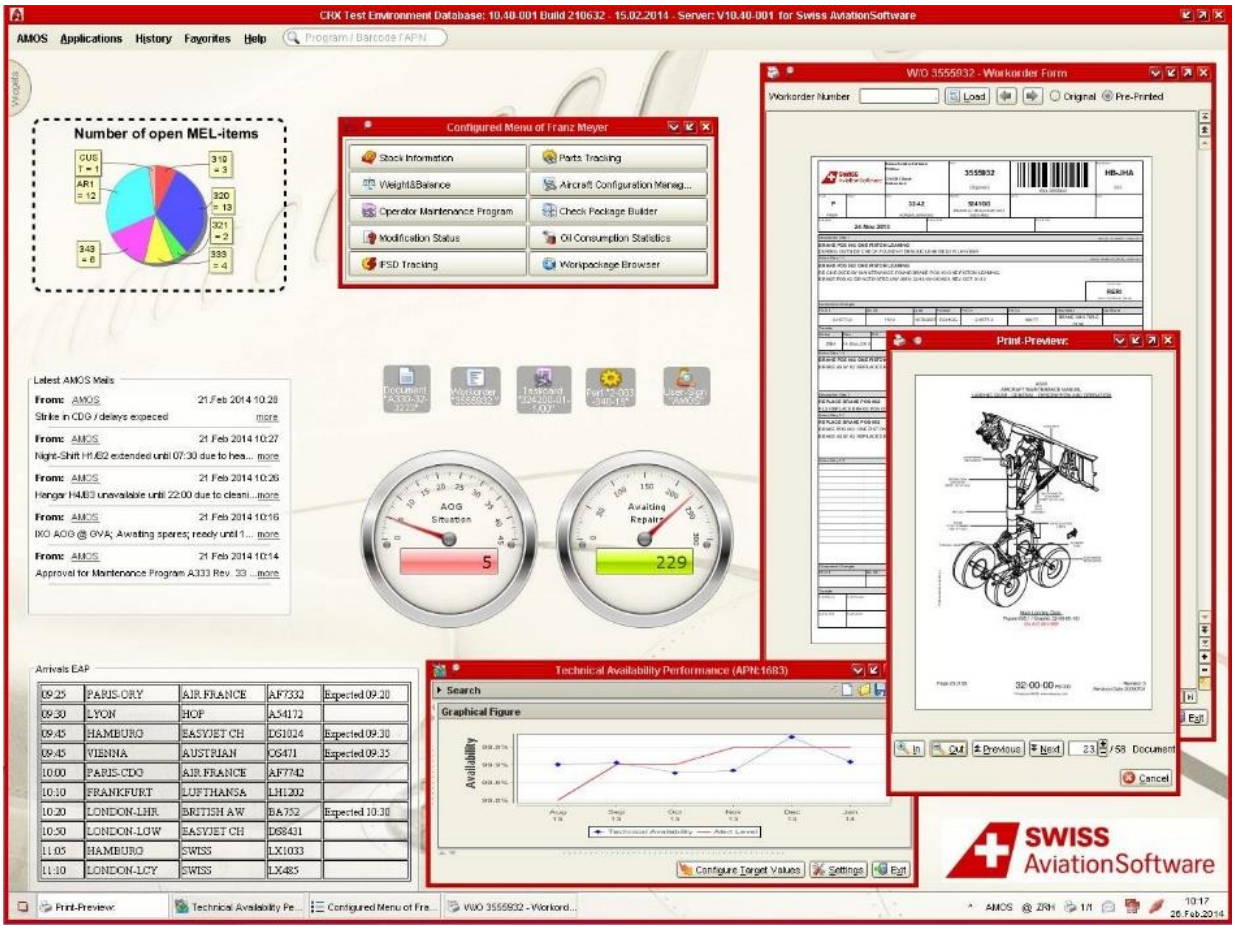


Рисунок 1.2 – Інтерфейс інформаційної системи *AMOS* із відображенням панелі керування, показників запасів, управління конфігурацією та типів повітряних суден

Архітектурно *AMOS* є комплексно інтегрованою системою, де всі модулі працюють з єдиною базою даних, що забезпечує цілісність інформації та відсутність дублювання. Система підтримує хмарне розгортання через сервіс

AMOS Cloud Service, а також мобільні розширення *AMOSmobile* для роботи інженерного персоналу безпосередньо біля повітряного судна.

Серед переваг *AMOS* – висока функціональна глибина, перевірена надійність, широка клієнтська база та активна підтримка розробника. Проте система орієнтована на штатну експлуатацію з безперервним потоком даних. Впровадження *AMOS* передбачає тривалий проєкт міграції даних: наприклад, впровадження у *Vietnam Airlines Engineering Company* тривало 16 місяців та включало міграцію 36 мільйонів рядків даних з попередніх систем.

Такий підхід передбачає наявність структурованих вихідних даних та не адаптований до роботи з фрагментованими джерелами.

Ramco Aviation є комплексним рішенням для управління ТО, що розвивається понад 20 років [9] та позиціонується як система нового покоління для заміни застарілих *ERP*-рішень.

Система пропонує спеціалізовані модулі для різних типів *MRO*-діяльності (див. рисунок 1.3): компонентне ТО – від приймання компонента до видачі сертифіката повернення до експлуатації; ТО двигунів – управління слотами, оцінка обсягу робіт, відстеження ремонту.

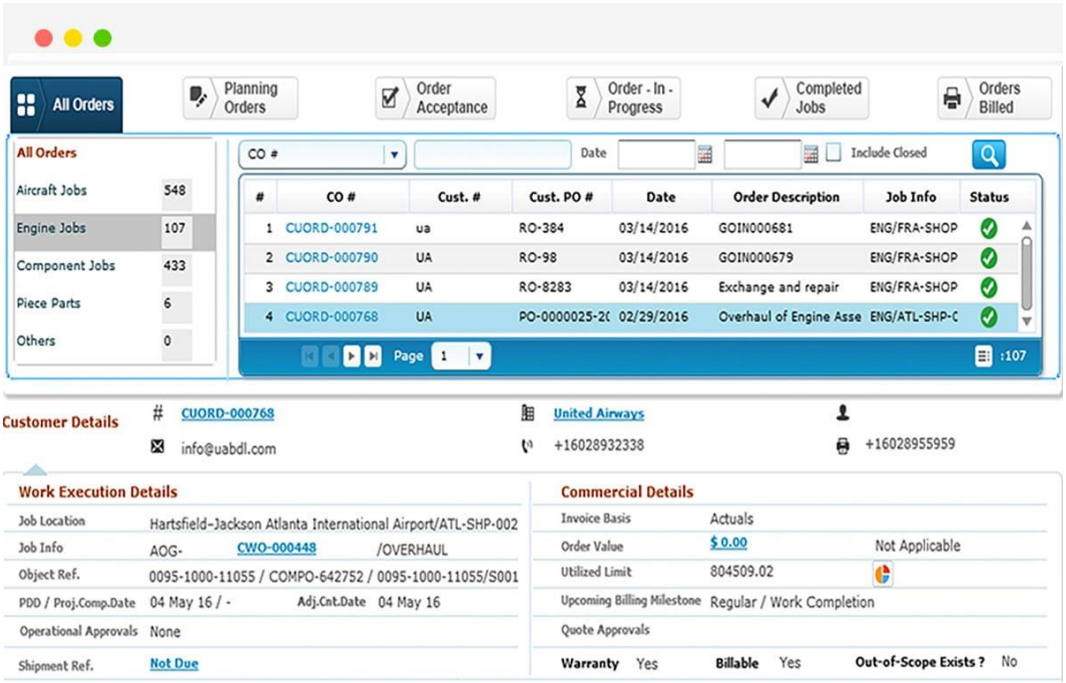


Рисунок 1.3 – Інтерфейс інформаційної системи *Ramco Aviation*

Ramco активно впроваджує технології штучного інтелекту, машинного навчання та роботизованої автоматизації процесів. Система підтримує мобільні технології та конвергентний інтерфейс для різних типів користувачів. Аналогічно до *AMOS*, *Ramco* є системою для поточного управління ТО та не містить спеціалізованих інструментів для консолідації фрагментованих даних з різнорідних джерел.

TRAX eMRO, що належить корпорації *AAR*, позиціонується як глобальний лідер у сегменті програмного забезпечення для технічного обслуговування [10] та інженерії в авіації. Система *eMRO* використовується понад 190 авіакомпаніями та *MRO*-провайдерами по всьому світу, включаючи як комерційних операторів, так і державні та військові організації. *eMRO* є веб-орієнтованим рішенням, що забезпечує доступ з будь-якого браузера та пристрою (див. рисунок 1.4). Система охоплює управління парком повітряних суден та компонентами; планування регламентних робіт на короткий, середній та довгий термін; управління матеріалами з підтримкою стандарту *Spec*2000 для електронного обміну даними; управління виробництвом для лінійного, базового та компонентного ТО; забезпечення регуляторної відповідності та аудиту.

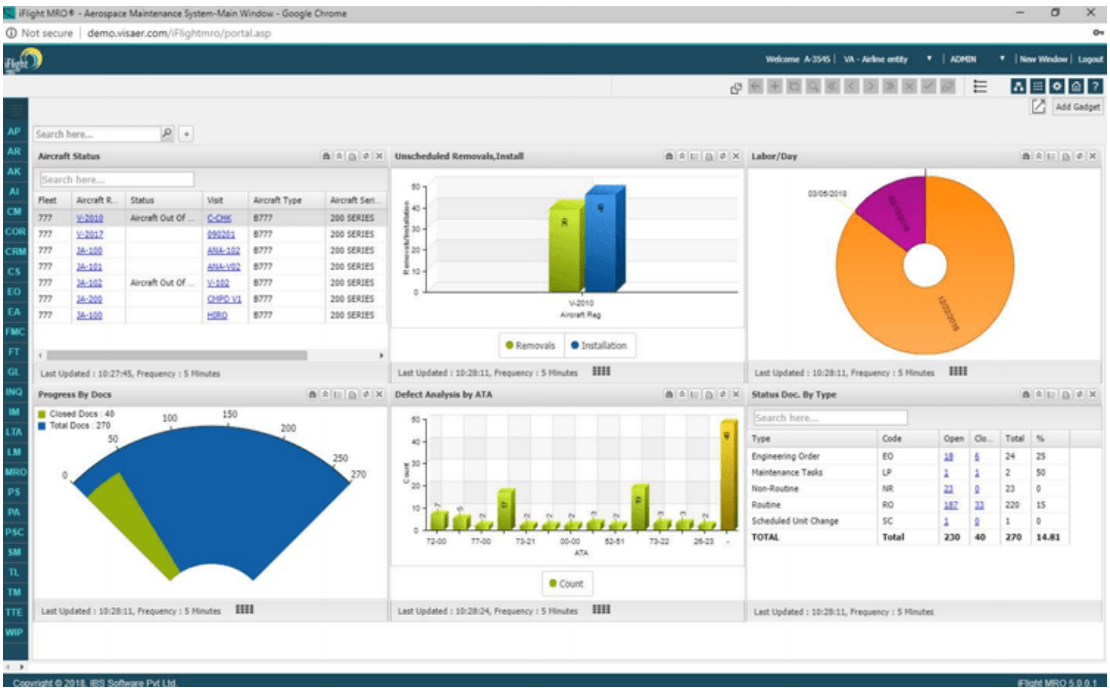


Рисунок 1.4 – Інтерфейс інформаційної системи *TRAX*

Окремою перевагою *TRAX* є розвинена екосистема мобільних додатків *eMobility*, що дозволяють технікам працювати безпосередньо біля повітряного судна, пілотам – реєструвати дефекти з кабіни, а наземному персоналу – вводити дані обслуговування на стоянці.

OASES, розроблена компанією *Commsoft*, є хмарною *MRO*-системою, що обслуговує понад 130 авіаційних організацій [11] по всьому світу. Система повністю розгорнута в хмарній інфраструктурі *AWS* та пропонує модульну архітектуру з дев'яти основних модулів.

Функціональність *OASES* охоплює управління підтриманням льотної придатності для організацій *CAMO*; планування та контроль базового ТО; управління лінійним обслуговуванням; контроль матеріалів та складських запасів; управління міжнародною логістикою та митним оформленням; комерційне управління та виставлення рахунків; гарантійне управління; аналітичний модуль *OASES Insights* (див. рисунок 1.5). Система підтримує інтеграцію з іншими рішеннями через *API*-фреймворк *OASES Gateway* та інструмент автоматизації *OASES Workflow*. Мобільний додаток *OASES Mobile* забезпечує роботу персоналу в режимі реального часу.

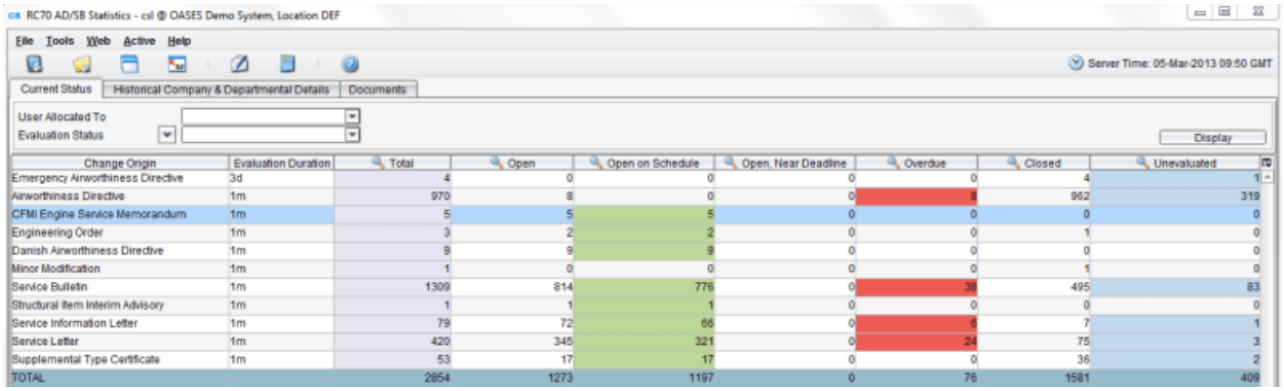


Рисунок 1.5 – Інтерфейс дашборду системи *OASES*

OASES позиціонується як рішення з оптимальним співвідношенням функціональності та вартості для середніх операторів, проте також не містить інструментів для відновлення даних після кризових ситуацій.

flydocs є спеціалізованою системою управління авіаційними записами та документацією [12], що доповнює функціональність *MRO*-систем (див. рисунок 1.6). На відміну від попередніх рішень, *flydocs* зосереджена саме на роботі з технічними записами, а не на оперативному управлінні ТО.

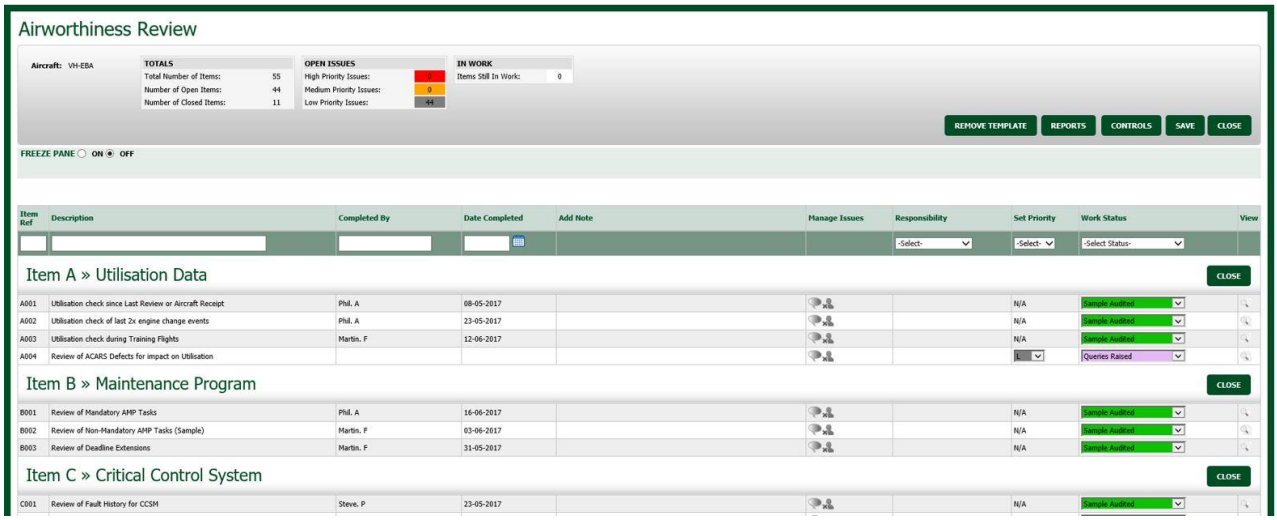


Рисунок 1.6 – Інтерфейс системи управління записами *flydocs*

Платформа *Digital Records Management* забезпечує централізоване цифрове сховище всіх технічних записів повітряного судна з історією від виробництва; автоматизований пошук документів з використанням технології оптичного розпізнавання символів; аудит документації з виявленням невідповідностей; підготовку цифрових портфоліо для передачі повітряних суден.

Платформа *Lifecycle Asset Management* надає інструменти для порівняння умов лізингу з фактичним станом повітряного судна; прогнозування витрат на ТО та резервів; оптимізації використання активів протягом життєвого циклу.

flydocs інтегрується з провідними *MRO*-системами, зокрема з *AMOS*, що дозволяє автоматично пов'язувати робочі пакети з відповідною документацією. Компанія також надає професійні послуги з управління переходами повітряних суден між операторами та лізингодавцями. Серед усіх розглянутих систем *flydocs* найближча до завдання роботи з документацією ТО та має певні функції *gap*-аналізу. Проте ці функції орієнтовані на процеси передачі повітряних суден, де

передбачається наявність базового масиву структурованих даних. Робота з фрагментованими даними після кризових ситуацій вимагає залучення дорогих професійних послуг з міграції та індивідуального підходу до кожного випадку. Для систематизації результатів аналізу існуючих рішень складено порівняльну таблицю 1.1, за ключовими критеріями, що є релевантними для завдання відновлення даних ТО після кризових ситуацій.

Таблиця 1.1 – Порівняльний аналіз існуючих *MRO*-систем

Критерій	*AMOS*	*Ramco Aviation*	*TRAX eMRO*	*OASES*	*flydocs*
Тип системи	Комплексна *MRO*	Комплексна *MRO*	Комплексна *MRO*	Комплексна *MRO*	Управління записами
Модель розгортання	Хмарна/локальна	Хмарна/локальна	Хмарна/веб	Хмарна (*AWS*)	Хмарна
Кількість клієнтів	180+	100+	190+	130+	75+
Імпорт з різнорідних джерел	Ні	Ні	Ні	Ні	Частково
Автоматичний *gap*-аналіз	Ні	Ні	Ні	Ні	Частково
Scoring-пріоритезація прогалин	Ні	Ні	Ні	Ні	Ні
Робота з неповними даними	Ні	Ні	Ні	Ні	Ні
Візуалізація стану флоту	Так	Так	Так	Так	Так
Термін впровадження	12-18 міс.	12-18 міс.	12-18 міс.	6-12 міс.	3-6 міс.
Орієнтація на кризові сценарії	Ні	Ні	Ні	Ні	Ні

Аналіз таблиці дозволяє зробити такі висновки. Усі розглянуті системи є зрілими промисловими рішеннями з широкою функціональністю для штатного управління ТО. Вони забезпечують повний цикл планування, виконання та обліку регламентних робіт, управління матеріалами та ресурсами, підтримку регуляторної відповідності. Проте жодна з систем не надає комплексного рішення для роботи з фрагментованими даними після кризових ситуацій. Критерії, що є ключовими для такого сценарію – імпорт з різнорідних джерел, автоматичний *gap*-аналіз, *scoring*-пріоритезація прогалин та робота з неповними даними – або відсутні, або реалізовані лише частково.

Система *flydocs* найближча до потреб післякризового відновлення завдяки спеціалізації на управлінні записами та наявності базових функцій *gap*-аналізу. Проте ці функції орієнтовані на процеси передачі повітряних суден між операторами, де передбачається наявність структурованого базового масиву даних. Міграція фрагментованих даних у *flydocs* вимагає залучення професійних послуг та індивідуального підходу, що суттєво збільшує вартість та терміни.

Тривалі терміни впровадження промислових *MRO*-систем – від 6 до 18 місяців – також є критичним обмеженням для кризових сценаріїв, де авіакомпанії потребують швидкої оцінки стану документації флоту для прийняття управлінських рішень.

Таким чином, порівняльний аналіз підтверджує наявність незаповненої ніші спеціалізованого інструменту для первинної консолідації та аналізу фрагментованих даних ТО. Такий інструмент має забезпечувати імпорт даних з різнорідних джерел у довільних форматах; автоматичне виявлення прогалин на основі порівняння наявних записів з вимогами програми ТО; пріоритезацію прогалин за критичністю для фокусування зусиль з відновлення;

1.5 Постановка задачі та формування вимог до системи

Проведений аналіз предметної області дозволив виявити актуальну проблему відновлення даних технічного обслуговування повітряних суден після

кризових ситуацій та обґрунтувати необхідність розробки спеціалізованого програмного інструменту для її вирішення.

Метою роботи є розробка інформаційної системи відновлення даних технічного обслуговування повітряних суден, призначеної для консолідації фрагментованих записів ТО з різнорідних джерел, автоматичного виявлення прогалин у документації та пріоритезації зусиль з відновлення даних для забезпечення підтвердження льотної придатності флоту авіакомпанії у післякризовий період. Для досягнення мети необхідно вирішити такі задачі:

- розробити архітектуру системи, що забезпечує імпорт даних з різнорідних джерел, їх нормалізацію та зберігання;
- спроєктувати модель бази даних для зберігання інформації про флот, компоненти, вимоги до ТО, записи про виконані роботи та виявлені прогалини;
- розробити алгоритм *gap*-аналізу для автоматичного виявлення відсутніх записів ТО на основі порівняння наявних даних з вимогами програми обслуговування;
- розробити методику *scoring*-оцінки критичності прогалин з урахуванням типу компонента, виду робіт, часу з останнього відомого ТО та регуляторних вимог;
- реалізувати веб-інтерфейс користувача з візуалізацією стану документації флоту та інструментами підтримки прийняття рішень;
- виконати тестування системи та верифікацію коректності роботи алгоритмів.

Функціональні вимоги. На основі аналізу потреб цільових користувачів та особливостей предметної області сформульовано функціональні вимоги до системи.

Модуль імпорту даних повинен забезпечувати завантаження даних з файлів форматів *CSV*, *Excel* (*XLSX*) та *JSON*; автоматичне розпізнавання структури вхідних даних та маппінг полів на внутрішню модель; валідацію імпортованих даних з виявленням помилок та невідповідностей; збереження

інформації про джерело кожного імпортованого запису для забезпечення простежуваності.

Модуль нормалізації даних повинен забезпечувати приведення даних з різних джерел до єдиної структури та формату; стандартизацію найменувань, дат, одиниць вимірювання напрацювання; виявлення та обробку дублікатів записів; формування консолідованої бази даних ТО.

Модуль *gap*-аналізу повинен забезпечувати порівняння наявних записів ТО з вимогами програми обслуговування для кожного повітряного судна; автоматичне виявлення відсутніх записів про регламентні роботи; виявлення прогалин у документації компонентів з обмеженим ресурсом; перевірку статусу виконання директив льотної придатності; формування переліку виявлених прогалин із зазначенням типу та контексту.

Модуль *scoring*-оцінки повинен забезпечувати розрахунок числового показника критичності для кожної виявленої прогалини; врахування множинних факторів: категорії компонента, типу роботи, часу з останнього відомого ТО, регуляторного статусу; ранжування прогалин за пріоритетом відновлення; можливість налаштування вагових коефіцієнтів формули оцінки.

Модуль візуалізації повинен забезпечувати відображення загального стану документації флоту у вигляді дашборду; деталізацію стану кожного повітряного судна з переліком виявлених прогалин; візуалізацію розподілу прогалин за категоріями критичності; формування звітів для підтримки прийняття управлінських рішень.

Нефункціональні вимоги. Вимоги до продуктивності: система повинна забезпечувати імпорт файлу обсягом до 100 000 записів за час не більше 5 хвилин; виконання *gap*-аналізу для флоту до 50 повітряних суден за час не більше 2 хвилин; відображення дашборду з актуальними даними за час не більше 3 секунд.

Вимоги до масштабованості: система повинна підтримувати роботу з флотом до 100 повітряних суден; зберігання до 10 мільйонів записів ТО; одночасну роботу до 20 користувачів.

Вимоги до надійності: система повинна забезпечувати збереження цілісності даних при збоях; резервне копіювання бази даних; журналювання операцій імпорту та модифікації даних.

Вимоги до зручності використання: система повинна мати інтуїтивно зрозумілий веб-інтерфейс; не вимагати спеціальної підготовки користувачів понад базові навички роботи з веб-додатками.

Висновки до розділу

У першому розділі проведено аналіз предметної області та обґрунтовано актуальність розробки системи відновлення даних технічного обслуговування повітряних суден.

Досліджено проблематику втрати даних ТО в кризових умовах на прикладі реальних випадків авіакомпаній *Iraqi Airways*, *Libyan Airlines* та українських операторів. Встановлено, що фрагментація документації унеможливлює підтвердження льотної придатності технічно справних повітряних суден та призводить до значних фінансових втрат.

Проаналізовано організацію технічного обслуговування повітряних суден, структуру програм ТО та вимоги до документації. Показано, що безперервність записів є фундаментальною вимогою для підтримання льотної придатності, а відсутність документального підтвердження виконання роботи юридично еквівалентна її невиконанню.

Розглянуто регуляторні вимоги щодо льотної придатності на рівні стандартів *ICAO* та вимог Європейського Союзу. Визначено критичну роль документації ТО у процесах сертифікації та наслідки прогалин у записах.

Виконано огляд та порівняльний аналіз існуючих *MRO*-систем: *AMOS, Ramco Aviation, TRAX eMRO, OASES* та *flydocs*. Встановлено, що жодна з систем не надає комплексного рішення для консолідації фрагментованих даних. Сформульовано постановку задачі, визначено функціональні та нефункціональні вимоги до системи..

РОЗДІЛ 2
ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Обґрунтування архітектури системи

Вибір архітектури інформаційної системи є ключовим етапом проєктування, що визначає технічні можливості, масштабованість, зручність розгортання та супроводу розробленого рішення. Для системи відновлення даних технічного обслуговування повітряних суден обрано клієнт-серверну архітектуру з чітким розділенням на серверну частину, що реалізує бізнес-логіку та взаємодію з базою даних, та клієнтську частину у вигляді веб-інтерфейсу користувача (див. рисунок 2.1).

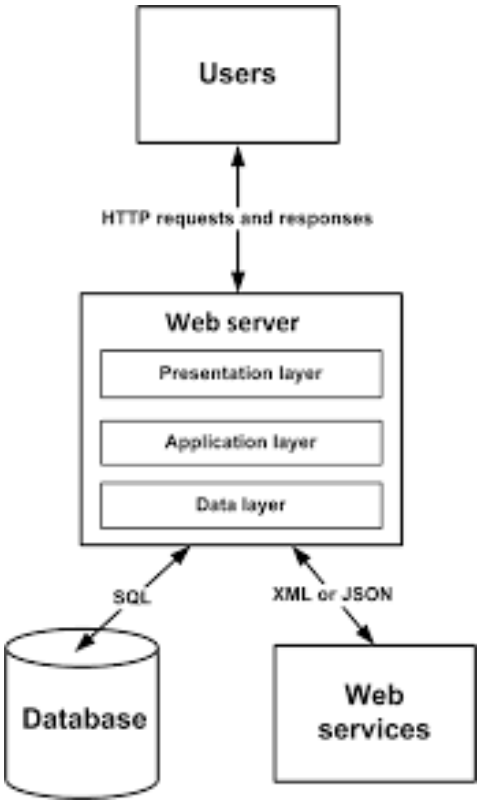


Рисунок 2.1 – Клієнт-серверна архітектура

Клієнт-серверна архітектура є оптимальним вибором для даної системи з огляду на такі фактори (див. рисунок 2.2). По-перше, система передбачає роботу кількох користувачів з різними ролями – адміністраторів, інженерів з ТО, аналітиків – які потребують одночасного доступу до єдиної бази даних. Централізоване зберігання даних на сервері забезпечує узгодженість інформації та унеможливлює конфлікти версій.

По-друге, обробка великих обсягів даних ТО, виконання *gap*-аналізу та розрахунок *scoring*-оцінок вимагають значних обчислювальних ресурсів. Винесення цих операцій на серверну частину дозволяє використовувати потужності сервера незалежно від характеристик клієнтських пристроїв.

По-третє, веб-інтерфейс забезпечує доступ до системи з будь-якого пристрою через браузер без необхідності встановлення спеціалізованого програмного забезпечення, що є важливим для роботи в умовах обмеженої IT-інфраструктури післякризового періоду.

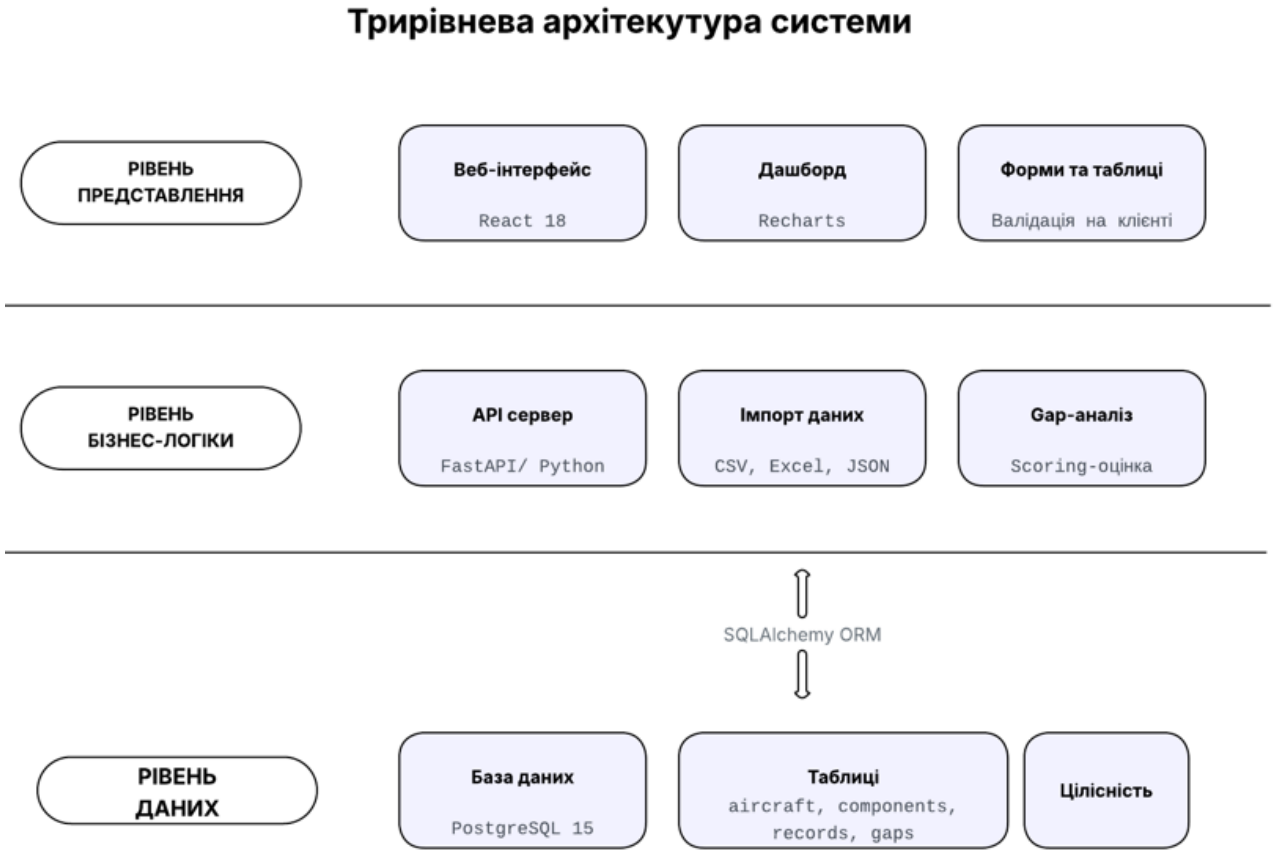


Рисунок 2.2 – Архітектура системи

Ресурсо-орієнтований підхід *REST* забезпечує інтуїтивно зрозумілу структуру *API*, де кожен тип сутності доступний за відповідним шляхом [13]: /*api*/*aircraft* для повітряних суден, /*api*/*components* для компонентів, /*api*/*maintenance-records* для записів ТО, /*api*/*gaps* для виявлених прогалин.

Формат обміну даними – *JSON*, що є стандартом для сучасних веб-застосунків та забезпечує легку інтеграцію з клієнтськими фреймворками [14] та можливість подальшого розширення системи.

Серверну частину системи організовано за модульним принципом, де кожен модуль відповідає за окрему функціональну область.

- **Модуль імпорту** даних забезпечує завантаження файлів різних форматів, парсинг їх вмісту, валідацію структури та збереження імпортованих даних у базі. Модуль підтримує формати *CSV, Excel* та *JSON*, а також забезпечує логування процесу імпорту для можливості аудиту.
- **Модуль нормалізації** виконує приведення імпортованих даних до єдиної структури, стандартизацію форматів дат, найменувань компонентів та одиниць вимірювання, виявлення та обробку дублікатів.
- **Модуль *gap*-аналізу** реалізує алгоритм порівняння наявних записів ТО з вимогами програми обслуговування та формує перелік виявлених прогалин.
- **Модуль *scoring*-оцінки** виконує розрахунок критичності прогалин за розробленою методикою та забезпечує ранжування результатів.
- **Модуль звітності** формує агреговані дані для відображення на дашборді та генерує звіти у різних форматах.

Така модульна організація забезпечує низьку зв'язність між компонентами системи, що спрощує тестування окремих модулів та їх подальшу модифікацію без впливу на інші частини застосунку. Кожен модуль має чітко визначений інтерфейс взаємодії, що дозволяє за необхідності замінити реалізацію окремого модуля без зміни загальної архітектури (див. рисунок 2.3). Крім того, модульний підхід сприяє паралельній розробці різних функціональних блоків та полегшує масштабування системи при зростанні навантаження.

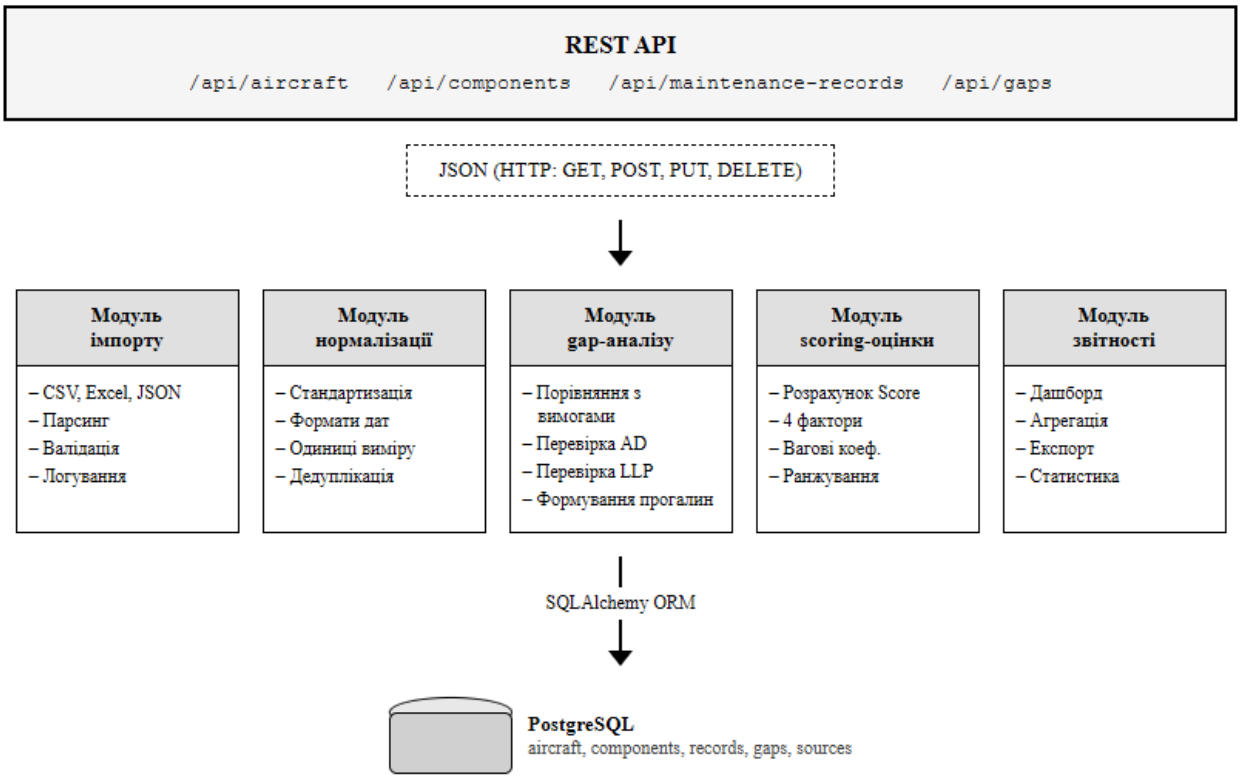


Рисунок 2.3 – Модульна структура серверної частини системи

2.2 Проєктування бази даних

База даних є центральним елементом системи, що забезпечує структуроване зберігання інформації про флот повітряних суден, компоненти, вимоги до технічного обслуговування, записи про виконані роботи та виявлені прогалини. Проєктування бази даних виконано з урахуванням специфіки предметної області авіаційного ТО та вимог до системи, сформульованих у попередньому розділі.

На етапі концептуального проєктування визначено основні сутності предметної області та зв'язки між ними:

- **Сутність “Повітряне судно”** представляє літаки флоту авіакомпанії та містить ідентифікаційні дані, тип, напрацювання та статус.

- **Сутність “Компонент”** представляє агрегати та системи повітряного судна, що підлягають окремому обліку: двигуни, допоміжні силові установки, агрегати шасі, блоки авіоніки тощо.
- **Сутність “Вимога до ТО”** описує регламентні роботи, що повинні виконуватися згідно з програмою технічного обслуговування: періодичні перевірки, заміни компонентів, виконання директив льотної придатності.
- **Сутність “Запис ТО”** представляє документально підтверджені факти виконання робіт з технічного обслуговування.
- **Сутність “Прогалина”** представляє виявлену відсутність запису про виконання обов'язкової роботи ТО з відповідною оцінкою критичності.
- **Сутність “Джерело даних”** містить інформацію про походження імпортованих записів для забезпечення простежуваності.

Зв'язки між сутностями відображають логіку предметної області [15]: повітряне судно має множину встановлених компонентів; для типу повітряного судна визначено множину вимог до ТО; записи ТО пов'язані з конкретним повітряним судном або компонентом; прогалини виявляються на основі порівняння записів ТО з вимогами.

На етапі логічного проєктування концептуальну модель перетворено на реляційну схему з визначенням таблиць, атрибутів та зв'язків.

Таблиця aircraft зберігає інформацію про повітряні судна флоту. Основні поля включаютьреєстраційний номер, тип та серійний номер повітряного судна, дату виробництва, загальне напрацювання у льотних годинах та циклах зльоту-посадки, поточний статус експлуатації та дату останнього відомого технічного обслуговування.

Таблиця components містить інформацію про компоненти, встановлені на повітряних суднах. Для кожного компонента зберігається номер частини за каталогом, серійний номер, тип (двигун, шасі, APU, авіоніка тощо), позиція встановлення та напрацювання. Окремо фіксуються обмеження ресурсу за годинами та циклами для компонентів з обмеженим терміном служби.

Таблиця maintenance_requirements зберігає вимоги програми технічного обслуговування. Кожна вимога містить номер завдання, опис роботи, тип (директива льотної придатності, сервісний бюлетень, періодична перевірка), інтервали виконання за льотними годинами, циклами або календарним часом, а також посилання на регуляторний документ.

Таблиця maintenance_records містить записи про фактично виконані роботи технічного обслуговування. Кожен запис пов'язаний з конкретним повітряним судном та містить дату виконання, напрацювання на момент виконання, інформацію про виконавця, посилання на сертифікат та джерело даних. Поле рівня достовірності дозволяє враховувати якість імпортованих записів.

Таблиця data_gaps зберігає виявлені системою прогалини у документації. Для кожної прогалини фіксується тип (відсутній запис, неповні дані, невідповідність), опис, дата виявлення та розрахований показник критичності з окремими складовими оцінки. Також зберігається статус обробки прогалини та примітки щодо вирішення.

Таблиця **data_sources** забезпечує простежуваність імпортованих даних. Для кожного джерела зберігається назва, тип (резервна копія, Excel, паперовий документ, MRO-провайдер), дата імпорту, ім'я файлу, кількість імпортованих записів та оцінка надійності джерела.

Зв'язок між таблицями *aircraft* та *components* є зв'язком «один-до-багатьох»: одне повітряне судно може мати множину встановлених компонентів. Зв'язок реалізовано через зовнішній ключ *aircraft_id* у таблиці *components*.

Зв'язок між таблицями *aircraft* та *maintenance_records* також є зв'язком «один-до-багатьох»: для одного повітряного судна може існувати множина записів ТО.

Зв'язок між таблицями *maintenance_requirements* та *maintenance_records* є зв'язком «один-до-багатьох»: одна вимога може бути виконана багаторазово протягом експлуатації.

Зв'язок між таблицями *data_sources* та *maintenance_records* є зв'язком «один-до-багатьох»: з одного джерела може бути імпортовано множину записів [15].

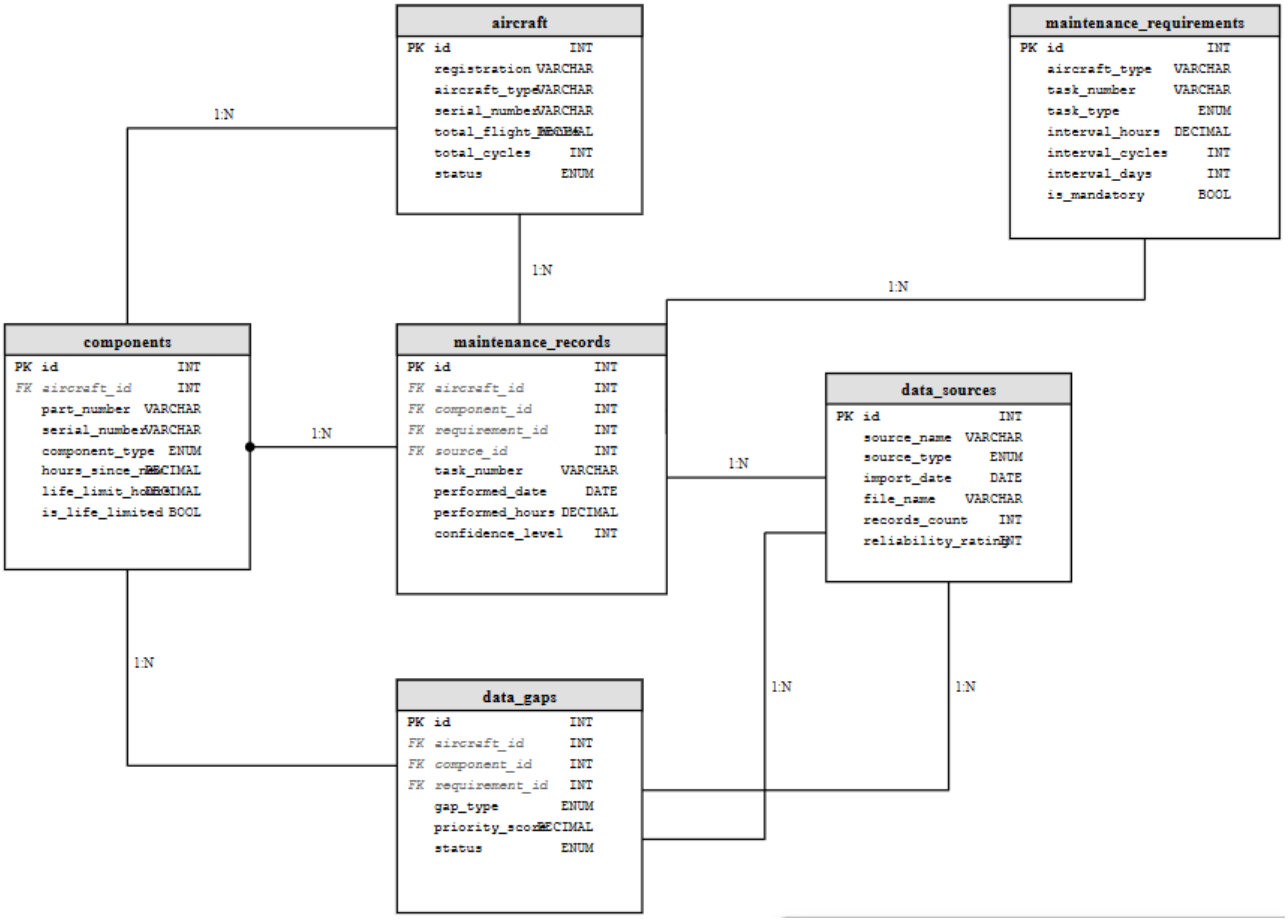


Рис 2.4. *ER*-діаграма бази даних системи

Для забезпечення продуктивності запитів передбачено створення індексів на полях, що часто використовуються у критеріях пошуку та з'єднання таблиць: зовнішні ключі *aircraft_id*, *component_id*, *requirement_id*, *source_id*; поля реєстраційного номера повітряного судна та серійних номерів компонентів; поле *priority_score* у таблиці *data_gaps* для ефективного сортування за критичністю; поля дат для фільтрації за періодами.

2.3 Проєктування серверної частини

Серверна частина системи реалізує бізнес-логіку, забезпечує взаємодію з базою даних та надає *REST API* для клієнтського застосунку [13]. Проєктування серверної частини виконано з урахуванням принципів модульності, розділення відповідальності та можливості тестування.

На рисунку 2.5 представлена структура каталогів серверної частини:

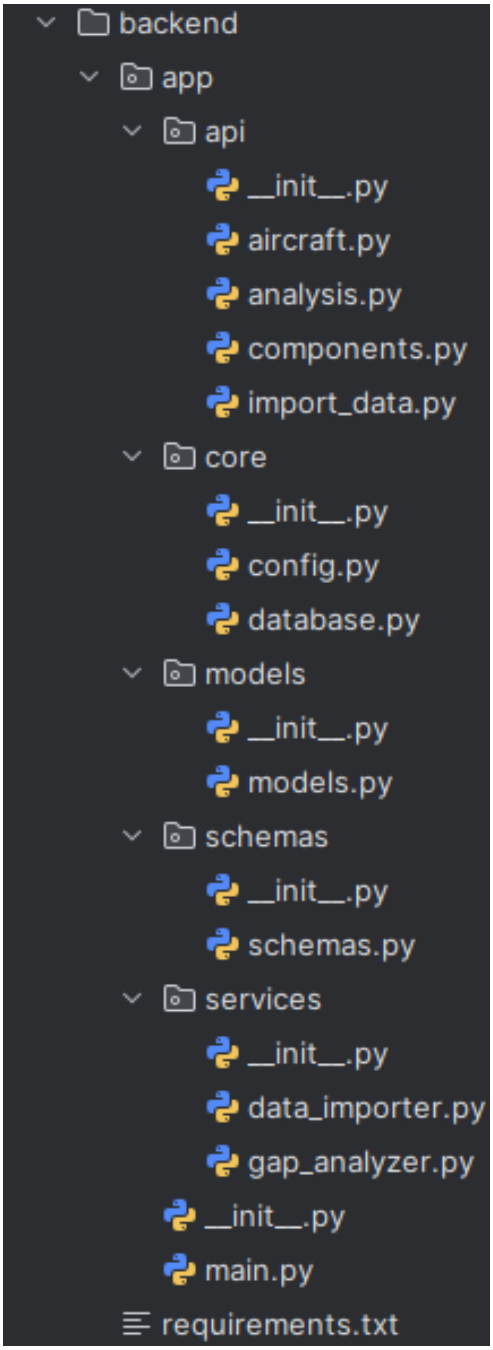


Рисунок 2.5 – Структура серверної частини проєкту

Каталог *api* містить модулі обробників *HTTP*-запитів, згруповані за функціональними областями:

- *aircraft.py* – операції з повітряними суднами;
- *components.py* – операції з компонентами;
- *import_data.py* – імпорт даних з файлів;
- *analysis.py* – виконання *gap*-аналізу та *scoring*-оцінки.

Каталог *core* містить базову конфігурацію системи: *config.py* – налаштування застосунку, параметри підключення до бази даних та інші конфігураційні змінні; *database.py* – ініціалізація підключення до бази даних та створення сесій *SQLAlchemy*.

Каталог *models* містить *ORM*-моделі, що описують структуру таблиць бази даних. Усі моделі зібрано у файлі *models.py*, що забезпечує зручність імпорту та підтримки.

Каталог *schemas* містить *Pydantic*-схеми для валідації вхідних даних та серіалізації відповідей *API*. Схеми визначено у файлі *schemas.py* та забезпечують типізацію даних на рівні *API*.

Каталог *services* містить сервіси бізнес-логіки:

- *data_importer.py* – сервіс імпорту та нормалізації даних з файлів різних форматів;
- *gap_analyzer.py* – сервіс;
- *gap*-аналізу та розрахунку *scoring*-оцінок критичності прогалин.

Для забезпечення взаємодії між клієнтською та серверною частинами системи розроблено REST API, специфікацію якого наведено у таблиці 2.1.

Таблиця 2.1 – Специфікація *REST API* системи

Метод	Ендпоінт	Опис
1	2	3
GET	/*api*/*aircraft*	Отримати список повітряних суден

Закінчення таблиці 2.1

1	2	3
GET	/*api*/*aircraft*/{*id*}	Отримати деталі повітряного судна
PUT	/*api*/*aircraft*/{*id*}	Оновити дані повітряного судна
DELETE	/*api*/*aircraft*/{*id*}	Видалити повітряне судно
POST	/*api*/*aircraft*	Створити запис повітряного судна
GET	/*api*/*aircraft*/{*id*}/*components*	Отримати компоненти повітряного судна
GET	/*api*/*aircraft*/{*id*}/*gaps*	Отримати прогалини повітряного судна
GET	/*api*/*components*	Отримати список компонентів
GET	/*api*/*components*/{*id*}	Отримати деталі компонента
POST	/*api*/*components*	Створити запис компонента
GET	/*api*/*maintenance-records*	Отримати записи ТО
POST	/*api*/*maintenance-records*	Створити запис ТО
GET	/*api*/*requirements*	Отримати вимоги програми ТО
POST	/*api*/*requirements*	Створити вимогу ТО
POST	/*api*/*import*/*csv*	Імпортувати дані з *CSV*
POST	/*api*/*import*/*excel*	Імпортувати дані з *Excel*
POST	/*api*/*import*/*json*	Імпортувати дані з *JSON*
POST	/*api*/*analysis*/*gap-analysis*	Виконати *gap*-аналіз
POST	/*api*/*analysis*/*scoring*	Розрахувати *scoring*-оцінки
GET	/*api*/*gaps*	Отримати список прогалин
GET	/*api*/*gaps*/{*id*}	Отримати деталі прогалини
PATCH	/*api*/*gaps*/{*id*}/*status*	Оновити статус прогалини
GET	/*api*/*dashboard*/*summary*	Отримати зведену статистику
GET	/*api*/*dashboard*/*fleet-status*	Отримати статус флоту
GET	/*api*/*dashboard*/*gaps-distribution*	Отримати розподіл прогалин

2.4 Розробка алгоритму *gap*-аналізу

Алгоритм *gap*-аналізу є ключовим компонентом системи, що забезпечує автоматичне виявлення відсутніх записів про виконання обов'язкових робіт технічного обслуговування. Алгоритм базується на порівнянні наявних записів ТО з вимогами програми обслуговування для кожного повітряного судна.

Вхідні дані алгоритму. Для виконання *gap*-аналізу алгоритм використовує такі вхідні дані: перелік повітряних суден флоту з їхнім поточним напрацюванням та датою останнього відомого ТО; перелік компонентів кожного повітряного судна з напрацюванням та обмеженнями ресурсу; вимоги програми ТО для відповідних типів повітряних суден; наявні записи про виконане ТО.

Алгоритм виконує послідовну перевірку для кожного повітряного судна флоту:

- **Крок 1**. Для обраного повітряного судна отримати перелік застосовних вимог ТО на основі типу повітряного судна та його конфігурації.
- **Крок 2**. Для кожної вимоги визначити очікувані точки виконання на основі інтервалів (за годинами, циклами або календарним часом) від початку експлуатації до поточного напрацювання.
- **Крок 3**. Для кожної очікуваної точки виконання перевірити наявність відповідного запису ТО у базі даних. Запис вважається відповідним, якщо він стосується тієї самої вимоги та виконаний у межах допустимого відхилення від очікуваної точки.
- **Крок 4**. Якщо відповідний запис не знайдено, створити запис прогалини із зазначенням повітряного судна, вимоги, очікуваної дати або напрацювання виконання та типу прогалини.
- **Крок 5**. Додатково перевірити компоненти з обмеженим ресурсом: порівняти поточне напрацювання з встановленими лімітами та перевірити наявність записів про капітальний ремонт або заміну для компонентів, що наближаються до вичерпання ресурсу.

- **Крок 6**. Перевірити статус виконання директив льотної придатності: для кожної застосовної директиви перевірити наявність запису про виконання у встановлений термін.

Нехай A – множина повітряних суден флоту, $R(a)$ – множина застосовних вимог для повітряного судна $a \in A$, $M(a)$ – множина наявних записів ТО для повітряного судна a. Для кожної вимоги $r \in R(a)$ визначимо множину очікуваних точок виконання $E(a, r)$ на основі інтервалу вимоги та напрацювання повітряного судна (див. таблицю 2.2). Прогалина фіксується, якщо для очікуваної точки $e \in E(a, r)$ не існує запису $m \in M(a)$, що задовольняє умови відповідності:

- запис m стосується вимоги r;
- дата або напрацювання виконання запису m знаходиться у допустимому інтервалі відносно точки e.

Множина виявлених прогалин G формується за формулою:

$$G = \{(a, r, e) \mid a \in A, r \in R(a), e \in E(a, r), \neg\exists m \in M(a): match(m, r, e)\}$$

де $match(m, r, e)$ – предикат відповідності запису вимозі та очікуваній точці.

Таблиця 2.2– Позначення множин та предикатів алгоритму

Символ	Значення	Приклад
A	Всі літаки	*UR-PSA*, *UR-PSB*, *UR-PSC*
a	Один літак	*UR-PSA*
$R(a)$	Вимоги для літака	*A-check*, *C-check*, *AD*-2024-001
r	Одна вимога	*A-check* кожні 500 год
$E(a,r)$	Коли мало бути виконано	500, 1000, 1500... годин
e	Одна точка	1500 годин
$M(a)$	Наявні записи ТО	Список з бази даних
m	Один запис	"*A-check* виконано 12.03.2023"
G	Прогалини	Те, що не знайдено

В умовах фрагментованих даних алгоритм повинен коректно обробляти ситуації невизначеності. Якщо дата останнього відомого ТО передує даті імпортованих записів, алгоритм припускає можливу наявність невідомих прогалин у цьому періоді та маркує відповідний інтервал як «період невизначеності». Якщо запис ТО не містить повних даних про напрацювання на момент виконання, алгоритм використовує оцінку на основі дати запису та середньої інтенсивності експлуатації повітряного судна. Для записів з низьким рівнем достовірності (імпортованих з ненадійних джерел) алгоритм може генерувати попередження про необхідність верифікації.

2.5 Розробка методики *scoring*-оцінки критичності прогалин

Методика *scoring*-оцінки забезпечує кількісну оцінку критичності кожної виявленої прогалини для пріоритезації зусиль з відновлення даних. На відміну від бінарного підходу, де всі прогалини вважаються рівнозначними, *scoring*-методика дозволяє диференціювати прогалини за ступенем важливості та терміновості вирішення.

Методика враховує чотири групи факторів, що впливають на критичність прогалини:

1. **Критичність компонента** відображає важливість компонента для безпеки польоту та можливі наслідки його відмови. Двигуни та допоміжні силові установки мають найвищу критичність, оскільки їх відмова безпосередньо загрожує безпеці. Шасі та системи керування польотом також є критичними. Системи авіоніки, гідравліки, пневматики мають середню критичність. Елементи салону та некритичне обладнання мають найнижчу критичність.

2. **Тип роботи** відображає регуляторний статус та обов'язковість виконання. Директиви льотної придатності мають найвищий пріоритет, оскільки їх невиконання автоматично призводить до втрати льотної придатності. *C*-перевірки та інші форми базового ТО мають високий пріоритет. *A*-перевірки та

компонентне ТО мають середній пріоритет. Рекомендовані роботи за сервісними бюлетенями мають нижчий пріоритет.

3. **Регуляторний статус** враховує, чи стосується прогалина компонента з обмеженим ресурсом, обов'язкової регламентної роботи або рекомендованої операції. Прогалини у документації компонентів з обмеженим ресурсом є найбільш критичними, оскільки можуть унеможливити подальшу експлуатацію компонента.

4. **Часовий фактор** відображає час, що минув з останнього відомого виконання роботи або з очікуваної дати виконання. Чим більший цей час, тим вища ймовірність порушення інтервалів ТО та тим критичнішою є прогалина.

Інтегральний показник критичності прогалини розраховується за формулою:

$$S = W_c \times C_c + W_t \times C_t + W_r \times C_r + W_d \times f(Td)$$

де:

- S – інтегральний показник критичності (*score*);
- Cc – коефіцієнт критичності компонента;
- Ct – коефіцієнт типу роботи;
- Cr – коефіцієнт регуляторного статусу;
- $f(Td)$ – функція часового фактора від часу Td з останнього відомого ТО;
- Wc, Wt, Wr, Wd – вагові коефіцієнти факторів.

За замовчуванням система використовує такі вагові коефіцієнти:

- $Wc = 0.30$ – критичність компонента;
- $Wt = 0.30$ – тип роботи;
- $Wr = 0.25$ – регуляторний статус;
- $Wd = 0.15$ – часовий фактор.

Сума вагових коефіцієнтів дорівнює 1, що забезпечує нормалізацію інтегрального показника. При такому розподілі максимальне значення *score* становить 5 балів.

Вагові коефіцієнти можуть бути налаштовані користувачем відповідно до пріоритетів конкретної авіакомпанії. Наприклад, якщо основним пріоритетом є виконання директив льотної придатності, вагу *Wt* можна збільшити за рахунок інших факторів.

Часовий фактор відображає залежність критичності прогалини від часу, що минув з останнього відомого технічного обслуговування. Для розрахунку цього фактора використовується логарифмічна функція:

$$f(Td) = \min(5, 1 + \ln(1 + Td \div T_0))$$

де *Td* – кількість днів з останнього відомого ТО, T_0 – нормуючий параметр, що за замовчуванням дорівнює 365 днів (типовий інтервал між регламентними роботами), *ln* – натуральний логарифм, *min* – функція мінімуму для обмеження максимального значення.

Вибір логарифмічної залежності обумовлено кількома факторами. По-перше, логарифмічна функція забезпечує швидке зростання показника на початковому етапі – навіть незначний період без підтвердженого ТО одразу впливає на оцінку критичності.

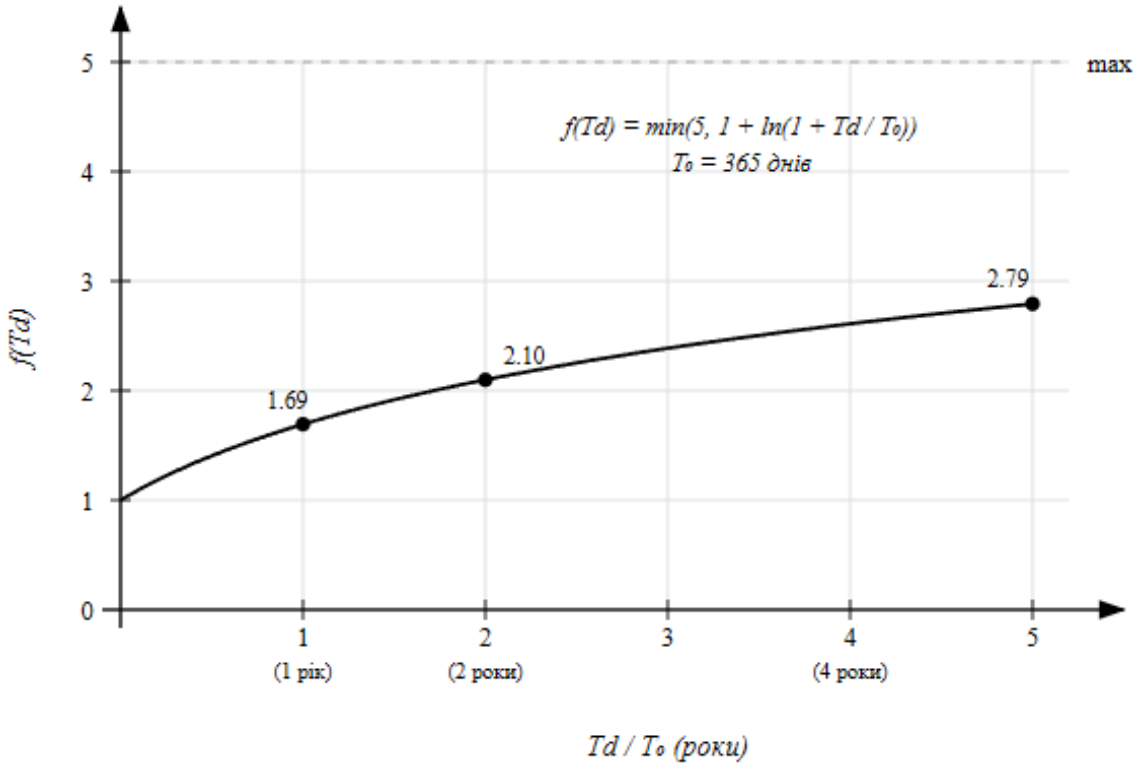
По-друге, функція характеризується поступовим насиченням при збільшенні часу – різниця в критичності між прогалинами давністю 5 та 10 років є меншою, ніж між прогалинами давністю 1 та 2 роки, оскільки обидві давні прогалини вже є критичними.

По-третє, обмеження через функцію мінімуму гарантує, що значення часового фактора не перевищить 5 балів незалежно від тривалості періоду без ТО. Значення функції часового фактора залежно від тривалості періоду без технічного обслуговування наведено у таблиці 2.3.

Таблиця 2.3. Значення функції часового фактора

Період без ТО	*Td*, днів	Розрахунок	*f*(*Td*)
0 днів	0	1 + *ln*(1 + 0/365)	1.00
1 місяць	30	1 + *ln*(1 + 30/365)	1.08
6 місяців	180	1 + *ln*(1 + 180/365)	1.40
1 рік	365	1 + *ln*(1 + 365/365)	1.69
2 роки	730	1 + *ln*(1 + 730/365)	2.10
5 років	1825	1 + *ln*(1 + 1825/365)	2.79
10 років	3650	1 + *ln*(1 + 3650/365)	3.40

Як видно з таблиці, функція швидко зростає протягом першого року (від 1.00 до 1.69), потім темп зростання уповільнюється (див. рисунок 2.6). Це відповідає практичній логіці: нещодавня прогалина потребує оперативної уваги, тоді як давні прогалини, хоч і залишаються критичними, вже потребують комплексного підходу до відновлення незалежно від точної давності.



Рисиунок 2.6 – Графік функції часового фактора *f*(*Td*)

Розглянемо приклад розрахунку *score* для прогалини: відсутній запис про виконання директиви льотної придатності щодо перевірки кріплення двигуна, останнє відоме ТО виконано 400 днів тому. Визначаємо значення коефіцієнтів: $Cc = 5$ (двигун – компонент найвищої критичності); $Ct = 5$ (директива льотної придатності – найважливіший тип роботи); $Cr = 3$ (обов'язкова робота за програмою ТО).

Обчислюємо інтегральний показник: $S = 0.30 \times 5 + 0.30 \times 5 + 0.25 \times 3 + 0.15 \times 1.74 = 1.50 + 1.50 + 0.75 + 0.26 = 4.01$.

Отримане значення 4.01 відповідає категорії «критична прогалина» (*score* ≥ 4.0), що вимагає негайної уваги та може блокувати підтвердження льотної придатності повітряного судна.

2.6 Проєктування користувацького інтерфейсу

Користувацький інтерфейс системи проєктується з урахуванням потреб цільових користувачів – інженерів з технічного обслуговування, аналітиків та керівників авіакомпаній, які потребують швидкого доступу до інформації про стан документації флоту та інструментів для прийняття рішень.

Інтерфейс організовано за принципом односторінкового застосунку з навігацією через бічне меню [16]. Основні розділи інтерфейсу включають дашборд – головну сторінку зі зведеною інформацією про стан флоту; розділ флоту – перегляд та управління повітряними суднами та компонентами; розділ записів ТО – перегляд імпортованих записів про виконане обслуговування; розділ прогалин – перегляд та управління виявленими прогалинами; розділ імпорту – завантаження даних з файлів; розділ аналізу – запуск *gap*-аналізу та налаштування *scoring*-параметрів; розділ звітів – формування та експорт звітів.

Головна сторінка дашборду надає миттєвий огляд стану документації флоту та виділяє критичні питання, що потребують уваги (див. рисунок 2.7). У верхній частині розміщено блоки ключових показників: загальна кількість повітряних суден у системі; кількість виявлених прогалин за категоріями

критичності; відсоток повітряних суден з критичними прогалинами; дата останнього виконання *gap*-аналізу.

Центральну частину займає візуалізація стану флоту у вигляді карткового представлення, де кожне повітряне судно відображається окремою карткою з колірною індикацією загального стану: червоний – є критичні прогалини, жовтий – є прогалини високого рівня, зелений – лише незначні прогалини або їх відсутність.

Нижня частина містить графіки розподілу прогалин за категоріями критичності, типами робіт та компонентами, а також список останніх імпортованих джерел даних.

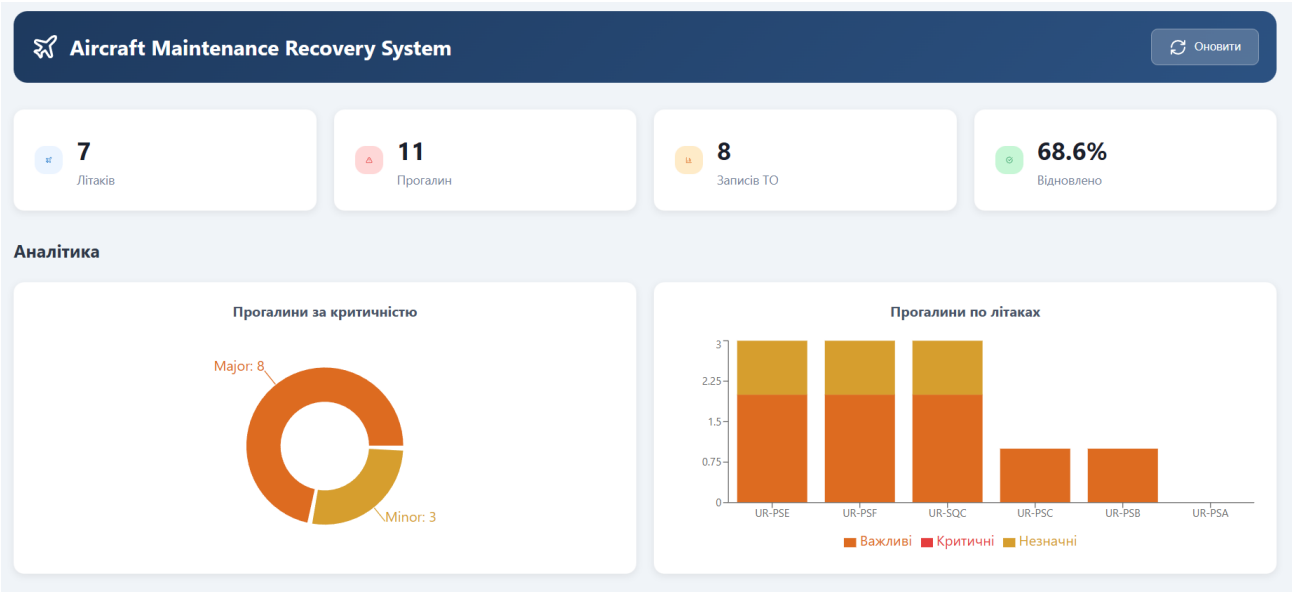


Рисунок 2.7 – Скріншот макету головного дашборду системи

Сторінка деталей повітряного судна містить повну інформацію про конкретний літак та його стан документації (див. рисунок 2.8). Заголовок відображає реєстраційний номер, тип повітряного судна та загальний статус з колірною індикацією. Блок загальної інформації містить серійний номер, дату виробництва, поточне напрацювання у годинах та циклах, дату останнього відомого ТО.

Блок компонентів відображає перелік встановлених компонентів з їхнім напрацюванням та статусом. Компоненти з обмеженим ресурсом виділяються окремо із зазначенням залишкового ресурсу.

Блок прогалин містить таблицю виявлених прогалин для даного повітряного судна з можливістю сортування за *score* та фільтрації за типом. Кожен рядок відображає опис прогалини, категорію критичності, *score* та поточний статус.

Блок записів ТО відображає хронологічний перелік імпортованих записів з можливістю пошуку та фільтрації.

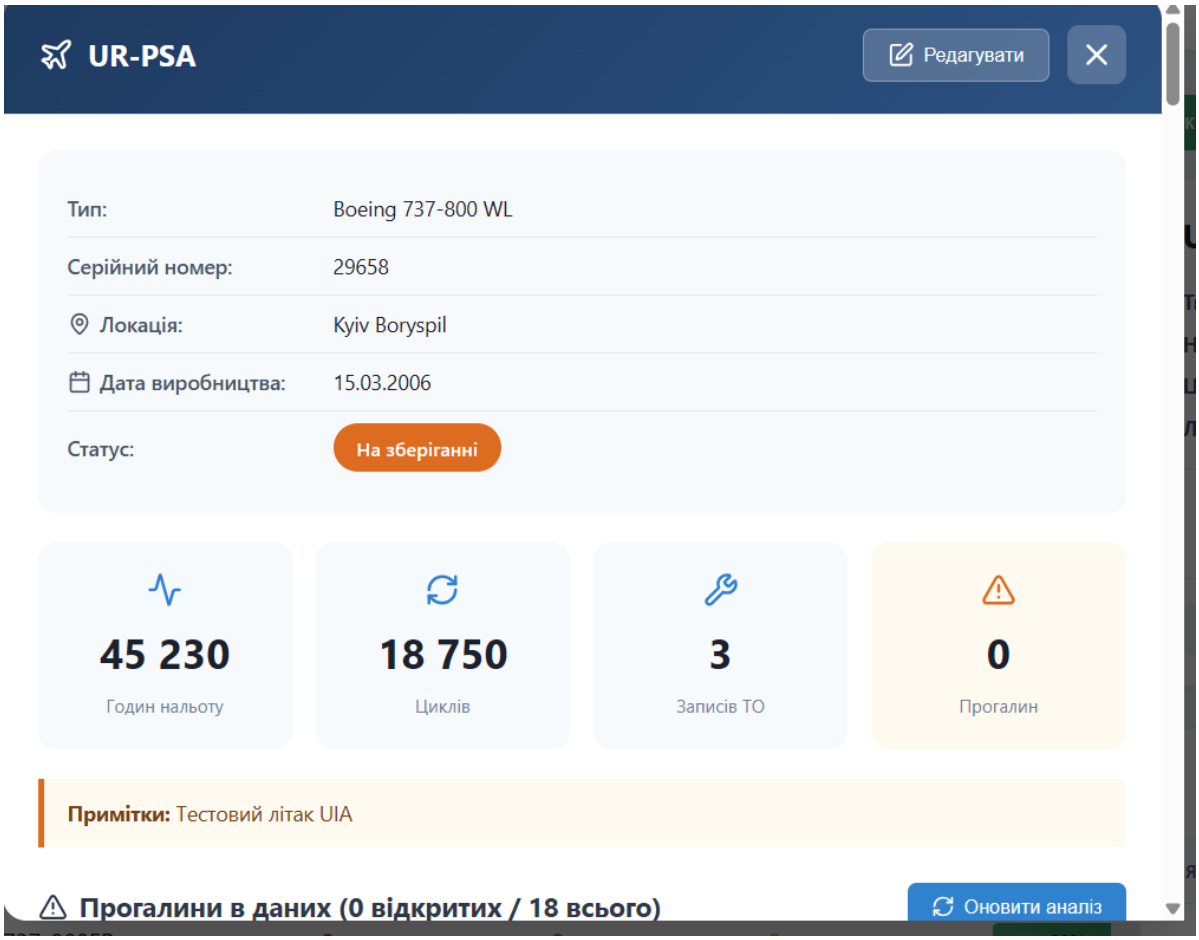


Рисунок 2.8 – Скріншот макету сторінки деталей повітряного судна

Сторінка прогалин надає централізований перегляд усіх виявлених прогалин флоту з інструментами для управління процесом відновлення. Фільтри

дозволяють обмежити відображення за повітряним судном, категорією критичності, типом роботи, статусом прогалини.

Таблиця прогалин відображає повний перелік з колонками: повітряне судно, компонент, опис прогалини, тип, *score*, категорія, статус. Таблиця підтримує сортування за будь-якою колонкою та пагінацію. Детальний перегляд прогалини відкривається при виборі рядка та містить повну інформацію: розбивку *score* за факторами, історію змін статусу, поле для приміток щодо вирішення. Управління статусом дозволяє змінювати статус прогалини: відкрита, в роботі, закрита з відповідними примітками.

Кольорова індикація рядків таблиці відповідає рівню критичності: червоний для критичних прогалин, жовтий для важливих, синій для незначних та зелений для косметичних. Це забезпечує швидку візуальну оцінку стану документації без необхідності аналізувати числові показники (див. рисунок 2.9).

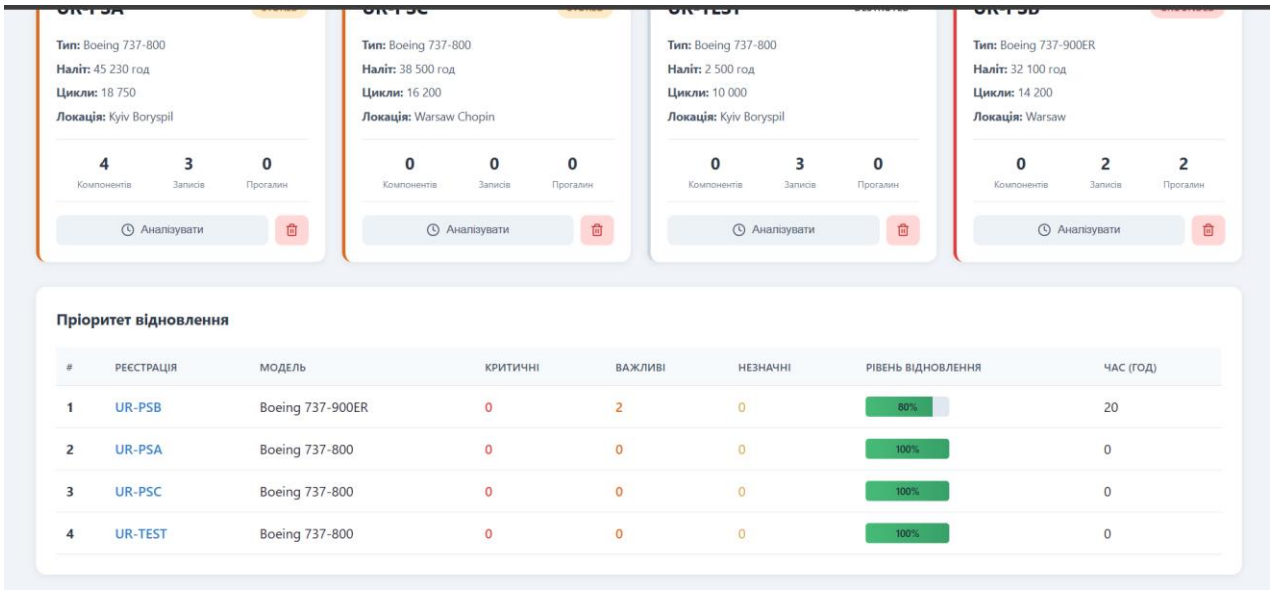


Рисунок 2.9 – Скріншот макету сторінки управління прогалинами

Сторінка імпорту забезпечує завантаження даних з файлів та контроль результатів. Зона завантаження підтримує перетягування файлів або вибір через діалог. Відображається список підтримуваних форматів.

Налаштування імпорту дозволяють вказати тип даних, що імпортуються (записи ТО, компоненти, вимоги), та опис джерела. Попередній перегляд відображає перші рядки розпізнаних даних з можливістю корекції маппінгу полів. Результати імпорту показують кількість успішно імпортованих записів, кількість помилок та їх деталі.

Висновки до розділу

У другому розділі виконано проєктування системи відновлення даних технічного обслуговування повітряних суден. Обґрунтовано вибір клієнт-серверної трирівневої архітектури з *REST API* для взаємодії між компонентами. Така архітектура забезпечує розділення відповідальності, масштабованість та можливість доступу через веб-браузер без встановлення спеціалізованого програмного забезпечення.

Спроєктовано реляційну базу даних, що включає таблиці для зберігання інформації про повітряні судна, компоненти, вимоги до ТО, записи про виконані роботи, виявлені прогалини та джерела даних. База даних нормалізована до третьої нормальної форми [15] та забезпечена необхідними індексами для продуктивності. Розроблено структуру серверної частини за модульним принципом з виділенням сервісів імпорту, нормалізації, *gap*-аналізу та *scoring*-оцінки. Специфіковано *REST API* з визначенням ендпоінтів для всіх операцій системи.

Розроблено алгоритм *gap*-аналізу, що виявляє відсутні записи ТО шляхом порівняння наявних даних з вимогами програми обслуговування. Алгоритм враховує особливості роботи з неповними даними та періодами невизначеності. Розроблено методику *scoring*-оцінки критичності прогалин, що враховує чотири групи факторів: критичність компонента, тип роботи, регуляторний статус та часовий фактор. Формула з налаштовуваними ваговими коефіцієнтами забезпечує кількісну оцінку для пріоритезації зусиль з відновлення.

Спроєктовано користувацький інтерфейс з дашбордом для огляду стану флоту, сторінками деталей повітряних суден, централізованим управлінням прогалинами та інструментами імпорту даних.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Вибір та обґрунтування технологій розробки

Вибір технологій розробки є важливим етапом реалізації програмного продукту, що впливає на продуктивність, масштабованість, зручність супроводу та швидкість розробки. Для реалізації системи відновлення даних технічного обслуговування повітряних суден обрано сучасний стек технологій, що забезпечує ефективне вирішення поставлених задач.

Для реалізації серверної частини обрано мову програмування *Python* версії 3.11. *Python* є однією з найпопулярніших мов для розробки веб-застосунків та систем обробки даних, що підтверджується щорічними рейтингами *TIOBE* та *Stack Overflow Developer Survey* [17], де *Python* стабільно займає провідні позиції. Вибір *Python* обґрунтовано кількома ключовими факторами:

- По-перше, мова має чистий та зрозумілий синтаксис, що сприяє швидкій розробці та легкому супроводу коду. Філософія *Python*, викладена у документі *PEP* 20 "*The Zen of Python*" [18], наголошує на читабельності коду та простоті рішень, що є важливим для проєкту з обмеженими часовими ресурсами.
- По-друге, *Python* має розвинену екосистему бібліотек для роботи з даними. Бібліотека *pandas* забезпечує ефективні структури даних для обробки табличної інформації та підтримує читання файлів різних форматів [19]. Бібліотека *openpyxl* дозволяє працювати з файлами *Excel*, включаючи читання даних з кількох аркушів та обробку форматування [20].

Вбудовані модулі *csv* та *json* забезпечують роботу з відповідними форматами без додаткових залежностей. Ця функціональність є критично важливою для системи, що повинна імпортувати дані з різнорідних джерел.

- По-третє, *Python* має потужні веб-фреймворки для створення *REST API*. Наявність типізації через модуль *typing* та інтеграція з бібліотекою *Pydantic* забезпечують валідацію даних на рівні мови та покращують надійність коду.

Для побудови *REST API* обрано фреймворк *FastAPI*, що є відносно новим, але вже зрілим рішенням для створення веб-сервісів на *Python* [21]. *FastAPI* було представлено у 2018 році і за короткий час набув значної популярності завдяки поєднанню продуктивності та зручності розробки. Головною перевагою *FastAPI* є висока продуктивність. Фреймворк побудований на основі *Starlette* для веб-частини та *Pydantic* для валідації даних [22], що забезпечує швидкість обробки запитів, порівнянну з *Node.js* [31] та *Go*. Згідно з незалежними бенчмарками *TechEmpower*, *FastAPI* демонструє одні з найкращих показників серед *Python*-фреймворків. *FastAPI* забезпечує автоматичну валідацію вхідних даних на основі анотацій типів та *Pydantic*-схем. Це означає, що розробнику не потрібно писати окремий код для перевірки коректності даних у запитах – фреймворк автоматично валідує вхідні дані та повертає інформативні повідомлення про помилки у разі невідповідності.

Важливою особливістю *FastAPI* є автоматична генерація інтерактивної документації *API*. На основі анотацій типів та *docstrings* фреймворк генерує документацію у форматах *Swagger UI* та *ReDoc*, доступну за стандартними адресами /*docs* та /*redoc*. Це значно спрощує тестування *API* під час розробки та інтеграцію з клієнтськими застосунками. *FastAPI* має вбудовану підтримку асинхронних операцій через синтаксис *async/await*, що дозволяє ефективно обробляти одночасні запити без блокування. Це особливо важливо для операцій імпорту великих файлів та виконання тривалих аналітичних запитів.

Для взаємодії з базою даних обрано бібліотеку *SQLAlchemy* версії 2.0.36, що є найпопулярнішим рішенням для об'єктно-реляційного відображення у *Python* [23]. *SQLAlchemy* розвивається з 2006 року та має репутацію надійного та

гнучкого інструменту. *SQLAlchemy* реалізує патерн *ORM*, що дозволяє працювати з базою даних через *Python*-об'єкти замість написання *SQL*-запитів вручну. Кожна таблиця бази даних представляється класом, а рядки таблиці – екземплярами цього класу. Це забезпечує типізацію даних, автодоповнення в *IDE* та зменшує ймовірність помилок.

Важливою перевагою *SQLAlchemy* є абстрагування від конкретної СУБД. Код, написаний з використанням *SQLAlchemy*, може працювати з різними базами даних – *PostgreSQL, MySQL, SQLite* – без змін у бізнес-логіці. Це забезпечує гнучкість при розгортанні системи та спрощує тестування з використанням легковагих баз даних. *SQLAlchemy* надає потужний конструктор запитів *Query API* для формування складних вибірок з фільтрацією, сортуванням, групуванням та з'єднанням таблиць.

Це дозволяє писати запити мовою *Python* з повним контролем над генерованим *SQL*. Для управління міграціями схеми бази даних *SQLAlchemy* інтегрується з бібліотекою *Alembic* [24], що забезпечує версіонування структури бази даних та автоматичну генерацію міграційних скриптів на основі змін у моделях.

Для зберігання даних обрано реляційну систему управління базами даних *PostgreSQL* версії 16 [25]. *PostgreSQL* є однією з найпотужніших відкритих СУБД, що активно розвивається з 1996 року та широко використовується у виробничих середовищах. *PostgreSQL* забезпечує повну підтримку *ACID*-транзакцій, що гарантує цілісність даних навіть у випадку збоїв системи.

Для системи обліку технічного обслуговування, де втрата або пошкодження даних може мати серйозні наслідки, ця властивість є критично важливою. База даних підтримує розширені можливості індексації, включаючи *B-tree, Hash, GiST, GIN* та *BRIN* індекси. Правильне використання індексів дозволяє оптимізувати продуктивність запитів на великих обсягах даних, що є важливим для системи, яка може зберігати мільйони записів ТО.

PostgreSQL має вбудовану підтримку *JSON* та *JSONB* типів даних, що дозволяє зберігати напівструктуровані дані у реляційній базі. Це може бути

корисним для зберігання додаткових атрибутів записів ТО, формат яких може відрізнятися залежно від джерела імпорту. Важливою перевагою *PostgreSQL* є активна спільнота та регулярні оновлення. Нові версії виходять щороку та містять покращення продуктивності, нові функції та виправлення безпеки.

Для реалізації клієнтської частини обрано бібліотеку *React* версії 18 [26], що є найпопулярнішим інструментом для побудови користувацьких інтерфейсів веб-застосунків. *React* розроблено компанією *Meta* та випущено як відкрите програмне забезпечення у 2013 році. *React* реалізує компонентний підхід до побудови інтерфейсу, де кожен елемент сторінки представляється окремим компонентом з власним станом та логікою. Компоненти можуть бути повторно використані та комбінуватися для створення складних інтерфейсів [27].

Ключовою особливістю *React* є віртуальний *DOM* – внутрішнє представлення структури сторінки, що дозволяє оптимізувати оновлення реального *DOM* браузера. При зміні стану компонента *React* обчислює мінімальний набір змін та застосовує лише їх, що забезпечує високу продуктивність інтерфейсу. *React* має велику екосистему готових компонентів та бібліотек.

Для даного проєкту використано бібліотеку *Recharts* для візуалізації даних у вигляді графіків та діаграм [28], бібліотеку *Axios* для виконання *HTTP*-запитів до *API* [29], бібліотеку *React Router* для організації навігації між сторінками.

Для візуалізації даних на дашборді обрано бібліотеку *Recharts*, що є *React*-обгорткою над потужною бібліотекою візуалізації *D3.js*. *Recharts* забезпечує декларативний підхід до створення графіків, де візуалізація описується як набір *React*-компонентів. Бібліотека підтримує широкий набір типів діаграм: стовпчикові та гістограми для порівняння значень, кругові та кільцеві для відображення часток, лінійні та площинні для трендів у часі, комбіновані діаграми для складних візуалізацій.

Recharts забезпечує адаптивність візуалізацій до розміру контейнера через компонент *ResponsiveContainer*, інтерактивність через вбудовані *tooltip* та *legend*,

анімацію переходів при зміні даних. Ці можливості дозволяють створити інформативний та зручний дашборд.

Обрані технології утворюють узгоджений стек для повноцінної веб-розробки. Серверна частина на *Python* з *FastAPI* забезпечує швидкий та типізований *API*. *SQLAlchemy* та *PostgreSQL* забезпечують надійне зберігання даних з потужними можливостями запитів. *Pandas* забезпечує ефективну обробку імпортованих файлів. *React* з *Recharts* забезпечує інтерактивний та інформативний інтерфейс користувача.

Усі обрані технології є відкритим програмним забезпеченням, що виключає ліцензійні витрати та забезпечує доступ до вихідного коду. Технології мають активні спільноти, якісну документацію та регулярні оновлення, що гарантує довгострокову підтримку системи. Обрані технології, використані під час розробки системи представленні в таблиці 3.1.

Таблиця 3.1 – Технології розробки системи

Компонент	Технологія	Версія	Призначення
Мова програмування	*Python*	3.11	Серверна частина
Веб-фреймворк	*FastAPI*	0.104	*REST API*
ORM	*SQLAlchemy*	2.0	Взаємодія з БД
База даних	*PostgreSQL*	15	Зберігання даних
Обробка даних	*Pandas*	2.1	Імпорт та аналіз
Міграції БД	*Alembic*	1.12	Версіонування схеми
Фронтенд	*React*	18	Користувацький інтерфейс
Візуалізація	*Recharts*	2.8	Графіки та діаграми
HTTP-клієнт	*Axios*	1.6	Запити до *API*

3.2 Реалізація бази даних та *ORM*-моделей

Реалізація бази даних виконана відповідно до спроєктованої у розділі 2 логічної моделі даних. Фізична реалізація включає створення таблиць у *PostgreSQL*, визначення *ORM*-моделей у *SQLAlchemy* та налаштування зв'язків між сутностями.

Підключення до бази даних реалізовано у модулі *database.py* каталогу *core*. Модуль використовує механізм *engine SQLAlchemy* для створення пулу з'єднань з базою даних. Пул з'єднань дозволяє повторно використовувати існуючі з'єднання замість створення нових для кожного запиту, що суттєво покращує продуктивність.

Для інтеграції з фреймворком *FastAPI* реалізовано функцію-генератор *get_db*, яка створює сесію бази даних на початку обробки *HTTP*-запиту та автоматично закриває її після завершення обробки.

Цей підхід реалізує патерн *dependency injection*, рекомендований документацією *FastAPI*, та забезпечує коректне управління ресурсами. Параметри підключення до бази даних винесено у модуль конфігурації *config.py* та читаються зі змінних оточення. Це дозволяє легко змінювати налаштування при розгортанні системи у різних середовищах – розробки, тестування та виробництва – без зміни коду.

Перелічуваний тип *AircraftStatus* визначає можливі стани повітряного судна: активний (літак в експлуатації), на зберіганні (тимчасово виведений з експлуатації), заземлений (заборона на польоти), на технічному обслуговуванні, знищений та невідомий. Такий набір статусів відображає реальну ситуацію з українським авіаційним флотом в умовах воєнного часу.

Перелічуваний тип *ComponentCategory* класифікує компоненти повітряного судна за дванадцятьма категоріями відповідно до стандартної класифікації систем літака: двигун, допоміжна силова установка, шасі, авіоніка, системи керування польотом, гідравліка, паливна система, електрична система, пневматика, планер, прилади та салон. Ця класифікація є ключовою для *scoring*-алгоритму при визначенні критичності компонента.

Перелічуваний тип *MaintenanceType* визначає типи робіт з технічного обслуговування: періодичні перевірки, директиви льотної придатності, сервісні бюлетені, капітальний ремонт, заміна компонента, інспекція та ремонт. Перелічуваний тип *CriticalityLevel* визначає чотири рівні критичності прогалин:

критичний, важливий, незначний та косметичний. Перелічуваний тип *GapStatus* відстежує стан прогалини: відкрита, в роботі, вирішена та прийнята.

Модель *Aircraft* є центальною сутністю системи та представляє повітряні судна флоту авіакомпанії. Таблиця *aircraft* містить повний набір полів для зберігання ідентифікаційних даних літака, інформації про напрацювання та поточний статус (див. рисунок 3.1).

	id [PK] integer	registration character varying (20)	serial_number character varying (50)	manufacturer character varying (50)	model character varying (50)	variant character varying (50)	total_flight_hours double precision	total_flight_cycles integer
1	1	UR-PSA	29658	Boeing	737-800	WL	45230	18750
2	2	UR-PSB	37542	Boeing	737-900ER		32100	14200
3	4	UR-PSC	34175	Boeing	737-800	NG	38500	16200
4	10	UR-TEST	99999	Boeing	737-800	NG	2500	10000
5	11	UR-PSE	35421	Boeing	737-800	[null]	41200	17100
6	12	UR-PSF	38754	Boeing	737-900ER	[null]	29800	12400
7	13	UR-SQC	3845	Airbus	A319-100	[null]	52100	24300

Рисунок 3.1 – Скріншот таблиці *aircraft*

Реєстраційний номер літака є унікальним ідентифікатором у межах системи та індексується для забезпечення швидкого пошуку. Обмеження *unique* гарантує неможливість створення двох записів з однаковою реєстрацією. Серійний номер виробника унікально ідентифікує кожен екземпляр літака у світі незалежно від зміни реєстрації при продажу або перереєстрації.

Поля *total_flight_hours* та *total_flight_cycles* зберігають загальне напрацювання повітряного судна у льотних годинах та циклах зльоту-посадки відповідно. Ці дані є ключовими для системи, оскільки більшість вимог технічного обслуговування прив'язані саме до напрацювання літака. Поле *manufacture_date* зберігає дату виробництва, яка використовується для розрахунку календарного віку літака та перевірки календарних інтервалів ТО.

Статус повітряного судна реалізовано через перелічуваний тип *AircraftStatus* із значенням за замовчуванням "невідомий", що відповідає ситуації при первинному внесенні літака до системи коли його поточний стан може бути не встановлений. Поле *last_known_location* зберігає останнє відоме

місцезнаходження літака, що особливо важливо в контексті евакуйованих повітряних суден, розкиданих по аеропортах різних країн.

Часові мітки *created_at ma updated_at* автоматично підтримуються базою даних: перша фіксує момент створення запису, друга оновлюється при кожній зміні завдяки параметру *onupdate*. Поле *notes* дозволяє зберігати довільні текстові примітки про літак.

Модель *AircraftComponent* представляє агрегати та вузли повітряного судна, що підлягають окремому обліку технічного обслуговування (див. рисунок 3.2). Необхідність окремого обліку компонентів обумовлена тим, що багато агрегатів мають власний ресурс, можуть переміщуватись між літаками та потребують індивідуального відстеження історії обслуговування.

	id [PK] integer	aircraft_id integer	part_number character varying (50)	serial_number character varying (50)	name character varying (200)	description text	category componentcategory	position character varying (50)	criticality criticalitylevel	is_life_limited boolean	life_limit_hours double precision	life_limit_cycles integer
1	2	1	CFM56-7B26	895422	CFM56-7B26 Engine		ENGINE	Engine 2 (Right)	CRITICAL	true	50000	250
2	3	1	131-9A	P-12458	Honeywell 131-9A APU		APU	Tail Section	MAJOR	false	[null]	[n
3	4	1	65C26101-41	NLG-2847	Nose Landing Gear Assembly		LANDING_GEAR	Nose	CRITICAL	true	60000	200
4	5	1	WXR-2100	WR-78541	Collins WXR-2100 Weather Ra...		AVIONICS	Nose Radome	MAJOR	false	[null]	[n
5	6	4	q		q		ENGINE		MAJOR	false	[null]	[n

Рисунок 3.2 – Скріншот таблиці *aircraft_components*

Зв'язок з повітряним судном реалізовано через зовнішній ключ, який посилається на первинний ключ таблиці *aircraft*. Цей зв'язок є обов'язковим – параметр *nullable=False* забороняє створення компонента без прив'язки до конкретного літака.

Поле *part_number* зберігає каталожний номер компонента згідно з документацією виробника, а *serial_number* – унікальний серійний номер конкретного екземпляра. Комбінація цих двох номерів дозволяє однозначно ідентифікувати будь-який компонент у світовому авіаційному парку.

Категорія компонента визначається через перелічуваний тип *ComponentCategory* та є обов'язковим полем. Рівень критичності встановлюється через *CriticalityLevel* із значенням за замовчуванням "важливий" та використовується при розрахунку пріоритету прогалин – компоненти вищої критичності отримують більший ваговий коефіцієнт.

Особливу увагу приділено підтримці компонентів з обмеженим ресурсом. Булеве поле *is_life_limited* позначає такі компоненти, а поля *life_limit_hours* та *life_limit_cycles* зберігають встановлені виробником обмеження ресурсу за напрацюванням у годинах та циклах відповідно. Поточне напрацювання компонента відстежується окремо від напрацювання літака через поля *hours_since_new* та *cycles_since_new*. Це принципово важливо, оскільки компонент може бути переміщений між різними літаками протягом свого життєвого циклу, і його ресурс накопичується незалежно від того, на якому борті він встановлений.

Модель *MaintenanceRecord* призначяена для зберігання записів про виконане технічне обслуговування та є ключовою для функціоналу *gap*-аналізу (див. рисунок 3.3). Саме на основі цих записів система визначає, які роботи були виконані та чи не прострочені наступні планові перевірки.

	id [PK] integer	aircraft_id integer	component_id integer	maintenance_type maintenancetype	work_order_number character varying (50)	description text	performed_date date	aircraft_hours double precision	aircraft_cycles integer	performed_by character varying (200)	certificate_number character varying (50)	requirement_id integer
1	1	1	[null]	A_CHECK	[null]	Routine A-Check inspecti...	2024-06-15	44800	18500	Turkish Technic	[null]	[nu
2	2	1	[null]	C_CHECK	[null]	Heavy maintenance C-Ch...	2023-03-20	42000	17200	Lufthansa Technik	[null]	[nu
3	3	1	[null]	D_CHECK	[null]	Complete overhaul D-Ch...	2020-01-10	35000	14000	SRT Technic	[null]	[nu
4	4	2	[null]	A_CHECK	[null]	A-Check completed	2024-08-10	31800	14000	LOT Aircraft Maintenan...	[null]	[nu
5	5	2	[null]	C_CHECK	[null]	C-Check inspection	2023-05-15	29500	12800	LOT Aircraft Maintenan...	[null]	[nu
6	7	10	[null]	A_CHECK	[null]	A-Check completed	2025-10-15	24800	9900	Ukraine MRO	[null]	[nu
7	8	10	[null]	C_CHECK	[null]	Heavy C-Check maintena...	2024-06-20	23500	9400	Lufthansa Technik	[null]	[nu
8	9	10	[null]	D_CHECK	[null]	Full D-Check overhaul	2021-03-10	18000	7200	Turkish Technic	[null]	[nu

Рисунок 3.3 – Скріншот таблиці *maintenance_records*

Кожен запис обов'язково пов'язаний з повітряним судном через зовнішній ключ. Опціональний зв'язок з компонентом через *component_id* дозволяє деталізувати, до якого саме агрегату відноситься запис – це важливо для компонентного ТО та відстеження історії окремих вузлів.

Тип виконаної роботи визначається через перелічуваний тип *MaintenanceType* та є обов'язковим полем. Дата виконання роботи та напрацювання літака на момент виконання є критично важливими для *gap*-аналізу – порівнюючи ці значення з поточним напрацюванням та встановленими інтервалами, система визначає, чи не прострочена наступна перевірка.

Поле *performed_by* зберігає інформацію про виконавця робіт – зазвичай це *MRO*-організація або авторизований сервісний центр. Для забезпечення простежуваності походження даних кожен запис пов'язаний з джерелом через зовнішній ключ *data_source_id*. Поле *confidence_score* містить оцінку достовірності запису у діапазоні від 0 до 1, яка встановлюється при імпорті на основі надійності джерела та може коригуватися вручну. Ця оцінка враховується при *scoring*-аналізі прогалин.

Модель *DataGap* є центральною для функціоналу аналізу прогалин та зберігає повну інформацію про кожну виявлену відсутність даних у документації технічного обслуговування (див. рисунок 3.4).

	id [PK] integer	aircraft_id integer	gap_type character varying (50)	requirement_id integer	component_category componentcategory	maintenance_type maintenancetype	period_start date	period_end date	priority_score double precision	criticality criticalitylevel	description text
1	1	1	expired_check	[null]	[null]	A_CHECK	[null]	2025-12-02	44.51	MINOR	Прострочений A-Check
2	2	1	expired_check	[null]	[null]	C_CHECK	[null]	2025-12-02	66.19	MAJOR	Прострочений C-Check
3	3	1	expired_check	[null]	[null]	D_CHECK	[null]	2025-12-02	72	MAJOR	Прострочений D-Check
4	4	2	expired_check	[null]	[null]	A_CHECK	[null]	2025-12-02	54.51	MAJOR	Прострочений A-Check
5	5	2	expired_check	[null]	[null]	C_CHECK	[null]	2025-12-02	69	MAJOR	Прострочений C-Check
6	6	2	expired_check	[null]	[null]	D_CHECK	[null]	2025-12-02	72	MAJOR	Прострочений D-Check
7	10	4	expired_check	[null]	[null]	A_CHECK	[null]	2025-12-02	44.51	MINOR	Прострочений A-Check
8	11	4	expired_check	[null]	[null]	C_CHECK	[null]	2025-12-02	66.19	MAJOR	Прострочений C-Check
9	12	4	expired_check	[null]	[null]	D_CHECK	[null]	2025-12-02	72	MAJOR	Прострочений D-Check
10	28	1	expired_check	[null]	[null]	A_CHECK	2024-06-15	2025-12-03	56.36	MAJOR	Прострочений A-Check
11	29	1	expired_check	[null]	[null]	C_CHECK	2023-03-20	2025-12-03	69	MAJOR	Прострочений C-Check
12	30	1	expired_check	[null]	[null]	A_CHECK	2024-06-15	2025-12-04	56.39	MAJOR	Прострочений A-Check
13	31	1	expired_check	[null]	[null]	A_CHECK	2024-06-15	2025-12-04	56.39	MAJOR	Прострочений A-Check
14	32	1	expired_check	[null]	[null]	C_CHECK	2023-03-20	2025-12-04	69	MAJOR	Прострочений C-Check
15	33	4	expired_check	[null]	[null]	D_CHECK	[null]	2025-12-04	72	MAJOR	Прострочений D-Check
16	34	1	expired_check	[null]	[null]	A_CHECK	2024-06-15	2025-12-04	56.39	MAJOR	Прострочений A-Check
17	35	1	expired_check	[null]	[null]	C_CHECK	2023-03-20	2025-12-04	69	MAJOR	Прострочений C-Check
18	36	1	life_limit_warning	[null]	ENGINE	OVERHAUL	[null]	[null]	80.75	MAJOR	Увага: CFM56-7B26 Engine - залишок ресурсу 15.0%
19	37	1	missing_component_hist...	[null]	ENGINE	INSPECTION	[null]	[null]	63.66	MAJOR	Відсутня історія ТО: CFM56-7B26 Engine (P/N: CFM56
20	38	1	life_limit_warning	[null]	ENGINE	OVERHAUL	[null]	[null]	80.75	MAJOR	Увага: CFM56-7B26 Engine - залишок ресурсу 15.0%
21	39	1	missing_component_hist...	[null]	ENGINE	INSPECTION	[null]	[null]	63.66	MAJOR	Відсутня історія ТО: CFM56-7B26 Engine (P/N: CFM56
22	40	1	missing_component_hist...	[null]	APU	INSPECTION	[null]	[null]	47.41	MINOR	Відсутня історія ТО: Honeywell 131-9A APU (P/N: 131-
23	41	1	life_limit_critical	[null]	LANDING_GEAR	OVERHAUL	[null]	[null]	91.06	CRITICAL	КРИТИЧНО: Nose Landing Gear Assembly - залишок р
24	42	1	missing_component_hist...	[null]	LANDING_GEAR	INSPECTION	[null]	[null]	60.16	MAJOR	Відсутня історія ТО: Nose Landing Gear Assembly (P/I
25	43	1	missing_component_hist...	[null]	AVIONICS	INSPECTION	[null]	[null]	45.66	MINOR	Відсутня історія ТО: Collins WXR-2100 Weather Radar
26	44	4	missing_component_hist...	[null]	ENGINE	INSPECTION	[null]	[null]	56.16	MAJOR	Відсутня історія ТО: q (P/N: q)
27	45	11	expired_check	[null]	[null]	A_CHECK	[null]	2025-12-05	44.51	MINOR	Прострочений A-Check
28	46	11	expired_check	[null]	[null]	C_CHECK	[null]	2025-12-05	66.19	MAJOR	Прострочений C-Check
29	47	11	expired_check	[null]	[null]	D_CHECK	[null]	2025-12-05	72	MAJOR	Прострочений D-Check

Рисунок 3.4 – Скріншот таблиці *data_gaps*

Кожна прогалина обов'язково пов'язана з повітряним судном. Типи прогалин:

- *expired_check* – для прострочених періодичних перевірок;
- *missing_component_history* – для компонентів без історії ТО;
- *life_limit_warning* та *life_limit_critical* – для компонентів з низькими залишком ресурсу.

Розрахований пріоритетний бал зберігається у полі *priority_score* як число з плаваючою точкою у діапазоні від 0 до 100. Рівень критичності визначається через перелічуваний тип *CriticalityLevel*. Текстові поля *description* та *recommended_action* містять опис прогалини та рекомендовані дії щодо її усунення відповідно. Поле *estimated_recovery_hours* зберігає оцінку часу в годинах, необхідного для відновлення відсутніх даних.

Модель *DataSource* призначена для зберігання інформації про джерела імпортованих даних та забезпечує простежуваність походження кожного запису в системі.

Для кожного джерела даних зберігається його найменування, тип (*CSV, Excel, JSON*, дані від *MRO*), оригінальне ім'я завантаженого файлу та назва організації, від якої отримано дані. Окреме поле містить оцінку надійності джерела за шкалою від 0 до 1, яка встановлюється користувачем під час імпорту. Дані від офіційних *MRO*-організацій, таких як *Lufthansa Technik* або *Turkish Technic*, зазвичай отримують високу оцінку надійності (0.9-1.0), оскільки ці організації ведуть сертифіковану документацію згідно з вимогами авіаційних регуляторів.

Натомість дані з неперевірених або неофіційних джерел, наприклад сканів паперових документів невідомого походження, отримують нижчу оцінку (0.3-0.5). Ця оцінка надійності автоматично присвоюється всім записам технічного обслуговування, що імпортуються з даного джерела, і враховується при розрахунку пріоритету прогалин.

Модель також містить статистичні поля, що фіксують результати імпорту: кількість успішно імпортованих записів та кількість записів, які не вдалося імпортувати через помилки формату або відсутність обов'язкових даних. Окрема часова мітка фіксує момент виконання імпорту. Зв'язок між таблицею джерел даних та таблицею записів технічного обслуговування дозволяє для будь-якого запису визначити його походження – з якого файлу, від якої організації та коли були отримані ці дані. Така простежуваність є критично важливою для авіаційної

галузі, де регулятори можуть вимагати підтвердження автентичності кожного запису в документації.

Зв'язки між усіма моделями системи реалізовано з використанням механізму *relationship* бібліотеки *SQLAlchemy*, який забезпечує автоматичне завантаження пов'язаних об'єктів та двосторонню навігацію між сутностями. Каскадне видалення налаштовано таким чином, що при видаленні повітряного судна автоматично видаляються всі пов'язані з ним компоненти, записи технічного обслуговування та прогалини. Це забезпечує цілісніть даних на рівні застосунку та запобігає появі записів у базі даних, що містять некоректні посилання на неіснуючі пов’язані елементи.

3.3 Реалізація серверної частини та бізнес-логіки

Серверна частина системи побудована за принципами *REST*-архітектури [13], де кожен ресурс має унікальну адресу, а операції над ресурсами виконуються через стандартні *HTTP*-методи: *GET* для отримання даних, *POST* для створення нових записів, *PUT* та *PATCH* для оновлення існуючих, *DELETE* для видалення. Такий підхід забезпечує передбачуваність інтерфейсу програмування та спрощує інтеграцію з клієнтськими застосунками.

Проєкт серверної частини організовано у вигляді модульної структури з чітким розділенням відповідальності між компонентами (див. рисунок 3.5).

Каталог *core* містить базову конфігурацію застосунку та модуль підключення до бази даних. Каталог *models* містить визначення всіх *ORM*-моделей, що відповідають таблицям бази даних.

Каталог *schemas* містить *Pydantic*-схеми, які використовуються для валідації вхідних даних та формування структурованих відповідей *API*.

Каталог *api* містить роутери – модулі з визначенням *HTTP-endpoints*, згруповані за функціональним призначенням.

Каталог *services* містить реалізацію бізнес-логіки системи, включаючи алгоритми аналізу прогалин та імпорту даних.

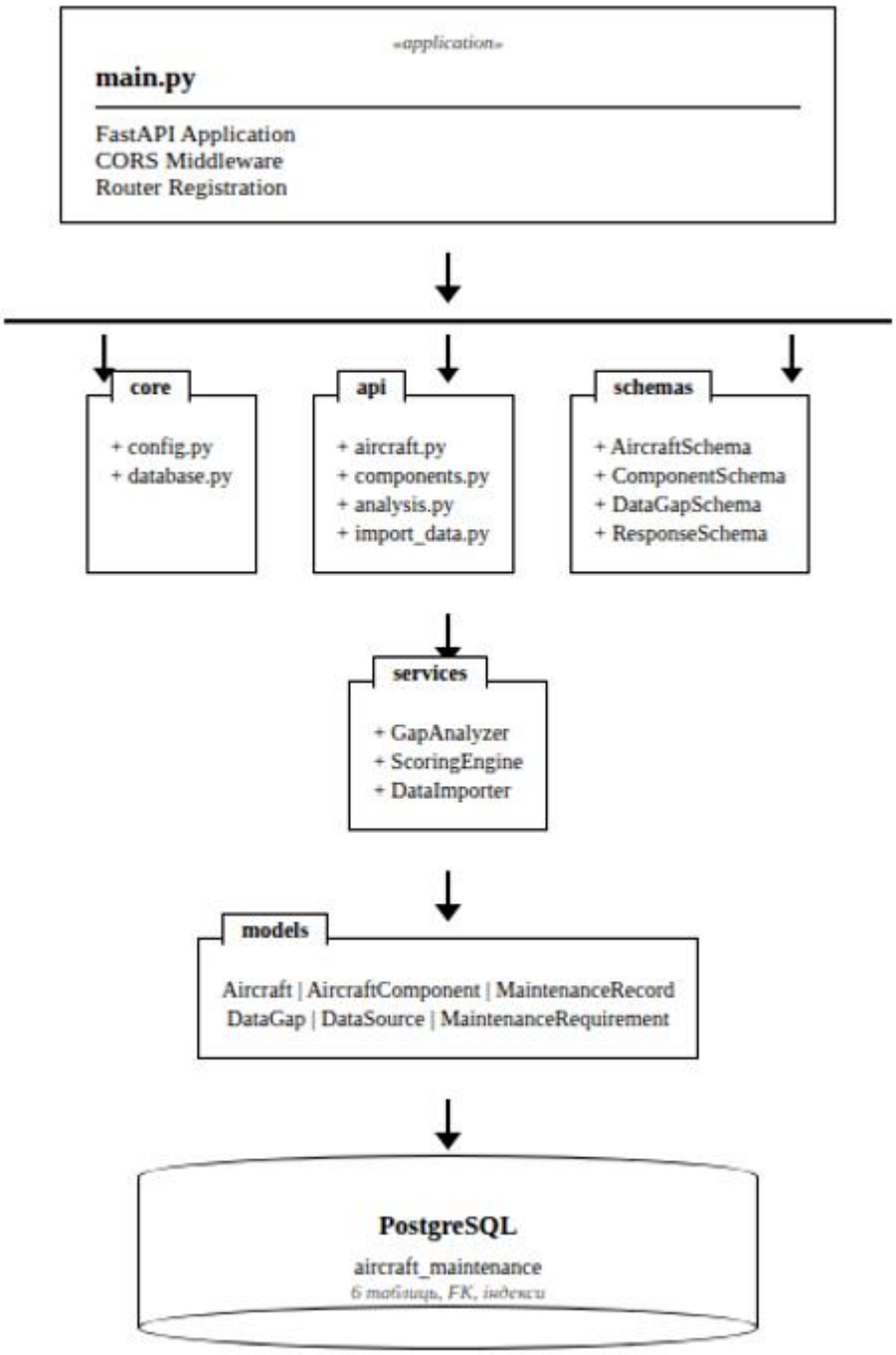


Рисунок 3.5 – Архітектура серверної частини системи

Така організація коду дозволяє легко орієнтуватися у проєкті та швидко знаходити потрібні компоненти. Розділення на шари забезпечує незалежність бізнес-логіки від способу її виклику — ті самі сервіси можуть використовуватися як через REST API, так і через командний рядок або фонові задачі. Крім того, модульна структура спрощує написання модульних тестів, оскільки кожен компонент можна тестувати ізольовано від інших

Головний файл застосунку виконує ініціалізацію *FastAPI*, налаштування проміжного програмного забезпечення та підключення всіх роутерів. При запуску сервера автоматично створюються всі таблиці бази даних, якщо вони ще не існують, що спрощує початкове розгортання системи. Для забезпечення взаємодії з клієнтським застосунком, який працює на іншому порті, налаштовано *CORS* (*Cross-Origin Resource Sharing*) – механізм, що дозволяє браузеру виконувати запити між різними доменами.

Роутер управління повітряними суднами реалізує повний набір операцій для роботи з літаками. Операція отримання списку всіх літаків повертає не лише базові дані кожного повітряного судна, але й обчислену статистику: кількість встановлених компонентів, кількість записів технічного обслуговування та кількість відкритих прогалин. Підрахунок прогалин виконується з фільтрацією за статусом, щоб закриті прогалини не відображались як активні проблеми на картках літаків у користувацькому інтерфейсі.

Операція створення нового літака включає перевірку унікальності реєстраційного номера перед збереженням у базу даних. Якщо літак з таким реєстраційним номером вже існує, система повертає помилку з відповідним повідомленням. Реєстраційний номер автоматично перетворюється у верхній регістр для уніфікації формату зберігання. Операція оновлення дозволяє змінювати окремі поля літака без необхідності передавати всі дані – система оновлює лише ті поля, які присутні у запиті. Операція видалення каскадно видаляє всі пов'язані записи – компоненти, записи технічного обслуговування та прогалини.

Роутер управління компонентами реалізує аналогічний набір операцій у контексті конкретного літака. Отримання списку компонентів виконується за ідентифікатором літака, що передається як параметр шляху. Створення компонента включає обов'язкову перевірку існування літака, до якого прив'язується компонент – система не дозволяє створювати компоненти для неіснуючих повітряних суден. Для компонентів з обмеженим ресурсом

зберігаються граничні значення напрацювання та поточний стан, що дозволяє системі розраховувати залишок ресурсу та генерувати попередження.

Роутер аналізу прогалин є ключовим для основної функціональності системи та надає кілька важливих операцій. Операція запуску аналізу для конкретного літака приймає ідентифікатор повітряного судна, викликає сервіс аналізу прогалин та повертає структурований результат, що включає кількість знайдених прогалин, їх розподіл за типами та рівнями критичності, загальний пріоритетний бал та перелік найбільш критичних прогалин.

Операція отримання статистики флоту агрегує дані по всіх повітряних суднах системи та повертає узагальнені показники: загальну кількість літаків, загальну кількість відкритих прогалин, загальну кількість записів технічного обслуговування, відсоток відновлення документації та розподіл прогалин за рівнями критичності. Ці дані використовуються для відображення статистичних карток та графіків на головній сторінці інтерфейсу.

Операція отримання рейтингу пріоритетів формує відсортований список літаків для таблиці пріоритетів відновлення. Для кожного літака обчислюється кількість прогалин за кожним рівнем критичності та показник готовності, який враховує як кількість, так і важливість відкритих прогалин. Літаки сортуються таким чином, що першими відображаються ті, які потребують найбільшої уваги – з найбільшою кількістю критичних прогалин та найнижчим показником готовності.

Операція оновлення статусу прогалини дозволяє закривати прогалини із зазначенням причини та додаткових приміток. При зміні статусу на "вирішена" система автоматично фіксує час закриття у київському часовому поясі, що важливо для коректного відображення історії роботи з прогалинами.

Модуль імпорту даних реалізує завантаження записів технічного обслуговування з файлів різних форматів. Система підтримує три основні формати: *CSV* як найпоширеніший формат обміну табличними даними, *Excel* для файлів, експортованих з корпоративних систем *MRO*-організацій, та *JSON* для структурованих даних з програмних інтерфейсів.

Операція імпорту приймає файл через механізм *multipart/form-data* разом з метаданими джерела: найменування джерела для ідентифікації в системі, назва організації-джерела та оцінка надійності за шкалою від 0 до 1. Перед обробкою записів створюється новий запис у таблиці джерел даних, який потім пов'язується з усіма імпортованими записами технічного обслуговування.

Клас імпортера даних інкапсулює логіку обробки файлів та нормалізації даних.

При читанні файлу система автоматично розпізнає колонки за різними варіантами іменування, що часто зустрічаються у реальних даних. Наприклад, колонка з реєстраційним номером може називатися "*registration*", "*reg*", "*aircraft_reg*" або "*tail_number*" – система коректно розпізнає всі ці варіанти.

Процес імпорту та нормалізації даних представлено на рисунку 3.6.

Нормалізація даних включає кілька етапів обробки. Дати перетворюються з різних форматів (*ISO*, європейський, американський) у єдиний внутрішній формат.

Текстові назви типів робіт нормалізуються до значень перелічуваного типу – наприклад, "*a-check*", "*A Check*", "*a_check*" та "*A-Check*" всі перетворюються на єдине внутрішнє значення. Реєстраційні номери приводяться до верхнього регістру. Числові значення напрацювання перевіряються на коректність діапазону.

При обробці кожного рядка система спочатку шукає літак за реєстраційним номером.

Якщо літак не знайдено у базі даних, рядок пропускається з відповідним попередженням – система не створює нові літаки автоматично при імпорті записів технічного обслуговування, оскільки це може призвести до появи неповних записів без необхідної інформації про повітряне судно. Для знайденого літака створюється новий запис технічного обслуговування з усіма наявними даними та присвоєною оцінкою достовірності.

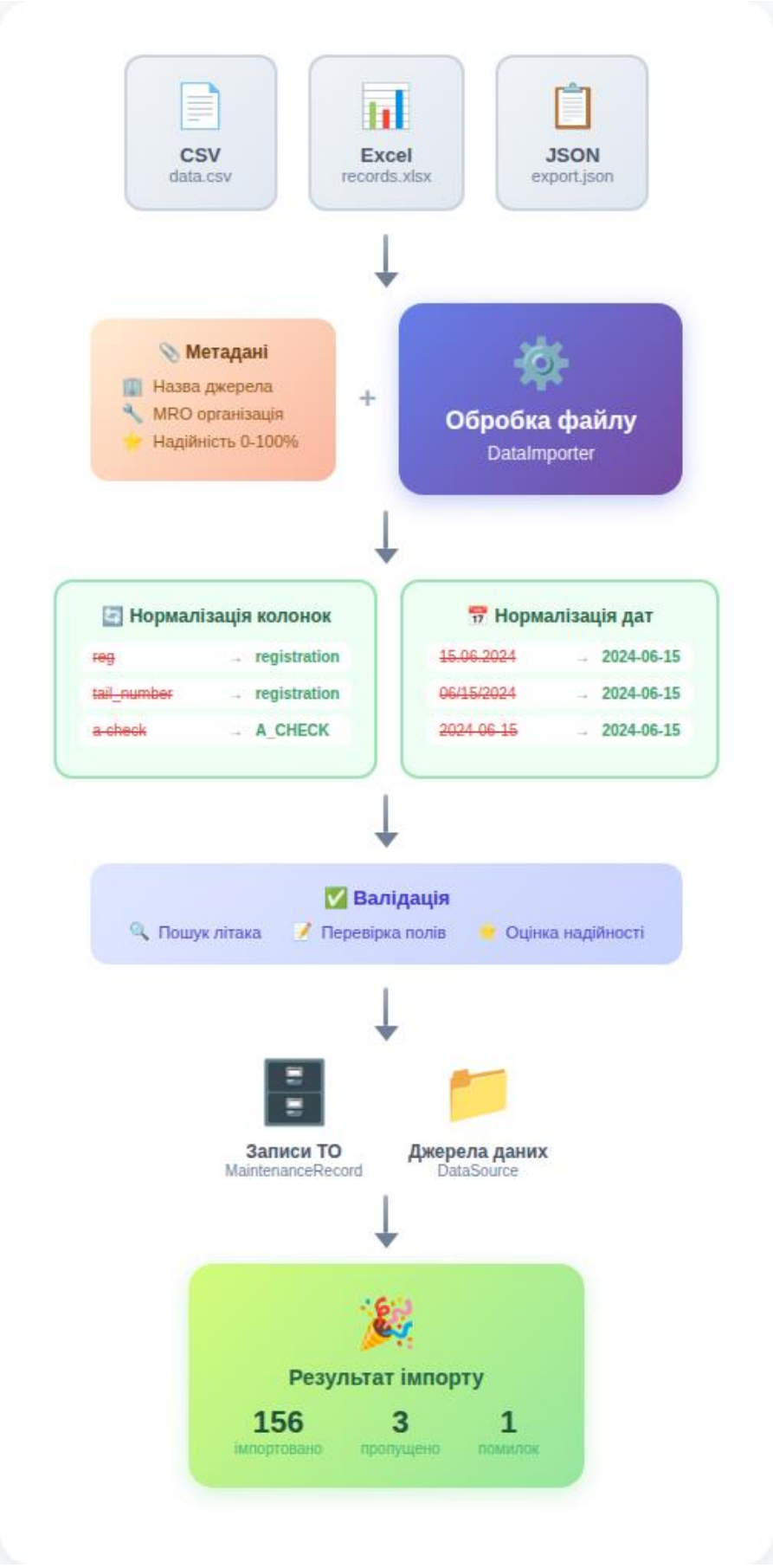


Рисунок 3.6 – Діаграма процесу імпорту даних в системі

Результат імпорту повертається як структурований об'єкт, що містить статистику обробки: кількість успішно імпортованих записів, кількість пропущених записів та перелік помилок з вказанням номера рядка та причини. Це дозволяє користувачу зрозуміти результат операції та виправити проблемні записи у вихідному файлі для повторного імпорту.

Метод формування статистики флоту агрегує дані по всіх повітряних суднах для відображення на головній сторінці інтерфейсу. Підраховується загальна кількість літаків у системі, загальна кількість відкритих прогалин (з фільтрацією за статусом), загальна кількість записів технічного обслуговування. Відсоток відновлення розраховується як відношення кількості закритих прогалин до загальної кількості всіх прогалин, що коли-небудь існували в системі. Додатково формується розподіл відкритих прогалин за рівнями критичності для візуалізації на круговій діаграмі.

Метод формування рейтингу пріоритетів створює відсортований список літаків для таблиці пріоритетів відновлення. Для кожного літака підраховується кількість відкритих прогалин за кожним рівнем критичності та розраховується показник готовності за формулою, що враховує зважену кількість прогалин: критичні прогалини мають найбільшу вагу, незначні – найменшу. Літаки сортуються за пріоритетом відновлення, щоб інженери могли швидко визначити, якому повітряному судну приділити увагу в першу чергу.

3.3.1 Алгоритм аналізу прогалин

Алгоритм аналізу прогалин є ключовим елементом бізнес-логіки системи та реалізує автоматичне виявлення відсутніх записів у документації технічного обслуговування з розрахунком пріоритету кожної знайденої прогалини (див. рисунок 3.7).

Реалізація алгоритму складається з двох основних класів. Клас механізму оцінювання інкапсулює логіку розрахунку пріоритетного балу та містить константи вагових коефіцієнтів для п'яти факторів оцінки, а також таблиці коефіцієнтів критичності для різних категорій компонентів та типів робіт. Клас

аналізатора прогалин координує процес аналізу, викликає різні типи перевірок та зберігає знайдені прогалини у базу даних.

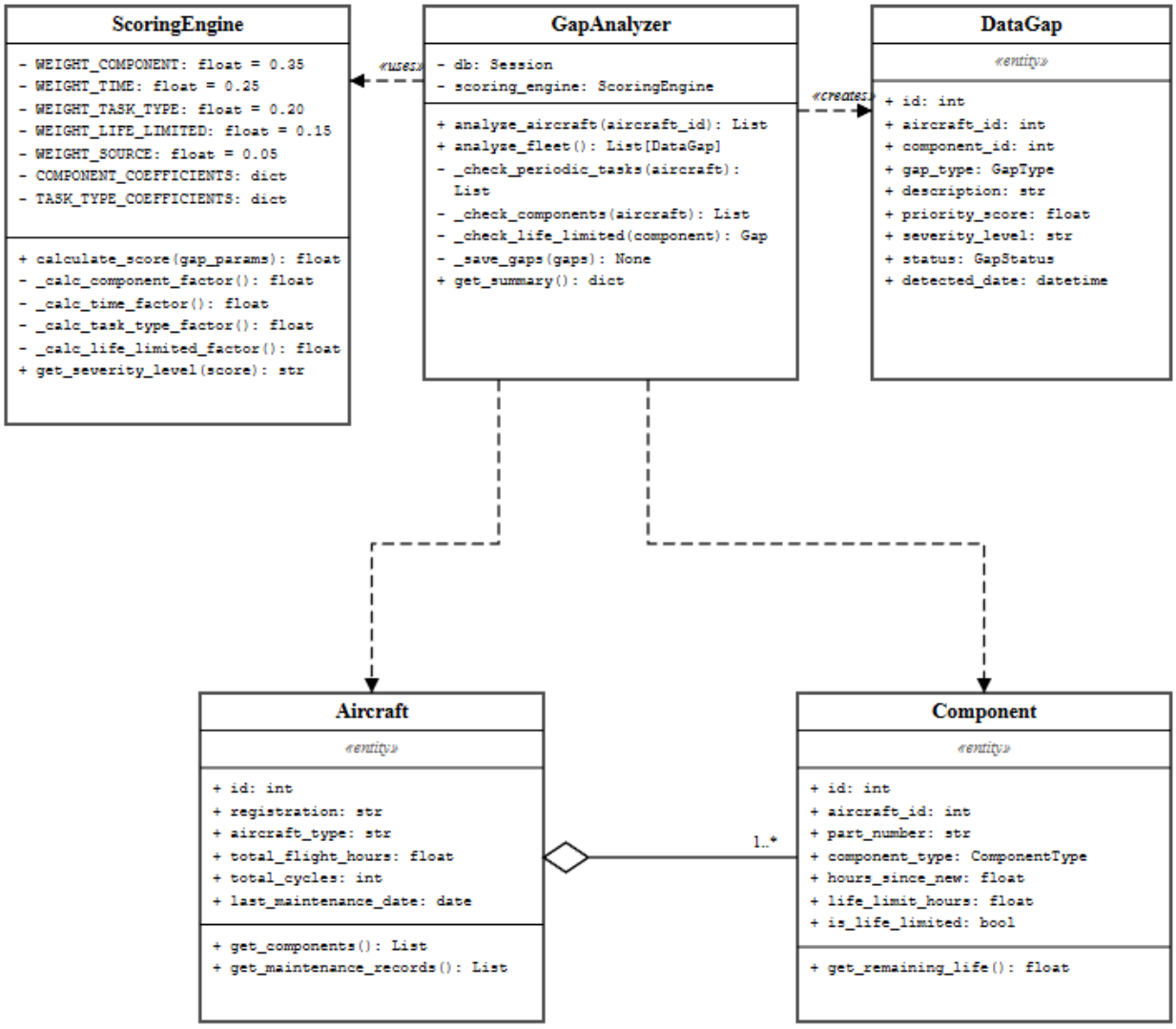


Рисунок 3.7 – Діаграма класів алгоритму аналізу прогалин

Вагові коефіцієнти факторів визначено на основі експертних оцінок важливості кожного аспекту для безпеки польотів та регуляторних вимог. Критичність компонента має найбільшу вагу (35%), оскільки відсутність документації на критичні системи, такі як двигуни або системи керування, становить найбільший ризик. Час з останнього технічного обслуговування має вагу 25% – чим довше літак експлуатується без підтвердженого ТО, тим вища ймовірність прихованих несправностей. Тип обов'язкової роботи (20%) враховує регуляторний статус – директиви льотної придатності мають найвищий

пріоритет як обов'язкові до виконання. Наявність обмеженого ресурсу (15%) важлива для компонентів, що наближаються до граничного напрацювання. Надійність джерела даних (5%) має найменшу вагу, але дозволяє враховувати якість наявної інформації.

Коефіцієнти критичності компонентів відображають їх важливість для безпеки польотів. Двигуни мають найвищий коефіцієнт (1.0), оскільки їх відмова безпосередньо загрожує безпеці польоту. Системи керування польотом (0.95) та шасі (0.90) також є критично важливими. Гідравлічна та паливна системи мають коефіцієнт 0.85. На іншому кінці шкали знаходиться салон (0.30), де несправності зазвичай не впливають на безпеку польоту.

Метод розрахунку пріоритетного балу приймає параметри конкретної прогалини та обчислює кожен фактор окремо. Фактор критичності компонента визначається за категорією з таблиці коефіцієнтів. Фактор часу розраховується як відношення кількості днів з останнього підтвердженого технічного обслуговування до максимального інтервалу (2190 днів для *D-check*), нормалізоване до діапазону від 0 до 1. Фактор типу роботи визначається за таблицею коефіцієнтів, де директиви льотної придатності мають значення 1.0, а звичайні інспекції – 0.4. Фактор обмеженого ресурсу розраховується як обернена величина залишку ресурсу: чим менше залишилось ресурсу, тим вище значення фактора. Загальний бал обчислюється як зважена сума всіх факторів, помножена на 100 для отримання значення у зручному діапазоні від 0 до 100.

На основі розрахованого балу визначається рівень критичності прогалини за чотирирівневою шкалою: критичний при балі вище 70, важливий при балі від 50 до 70, незначний при балі від 30 до 50, косметичний при балі нижче 30.

Клас аналізатора прогалин виконує послідовні перевірки для виявлення різних типів відсутніх даних. Метод аналізу літака є точкою входу та координує весь процес: отримує дані про літак з бази даних, викликає методи перевірки періодичних робіт та компонентів, збирає знайдені прогалини та зберігає їх у базу даних з урахуванням вже існуючих записів.

Перевірка періодичних робіт аналізує виконання трьох типів обов'язкових перевірок:

- *A-check* з інтервалом 90 днів;
- *C-check* з інтервалом 548 днів (приблизно 18 місяців) ;
- *D-check* з інтервалом 2190 днів (6 років).

Для кожного типу перевірки система шукає останній запис про виконання у таблиці записів технічного обслуговування.

Якщо такий запис відсутній, створюється прогалина типу "прострочена перевірка" з відповідним описом та рекомендованими діями. Якщо запис існує, система може додатково перевірити, чи не прострочена наступна перевірка, порівнюючи дату останнього виконання з поточною датою та встановленим інтервалом.

Перевірка компонентів виконує два типи аналізу для кожного компонента літака. Перший тип – перевірка залишку ресурсу для компонентів з обмеженим терміном служби.

Система розраховує залишок ресурсу у відсотках окремо за годинами та за циклами, якщо обидва обмеження встановлені, та бере менше значення. Якщо залишок ресурсу менше 10%, створюється критична прогалина з терміновою рекомендацією виконати капітальний ремонт або замінити компонент. Якщо залишок від 10% до 20%, створюється прогалина рівня "важливий" з рекомендацією запланувати ремонт найближчим часом.

Другий тип перевірки – наявність історії технічного обслуговування для кожного компонента. Система підраховує кількість записів технічного обслуговування, пов'язаних з конкретним компонентом. Якщо жодного запису не знайдено, створюється прогалина типу "відсутня історія компонента" з рекомендацією отримати дані від виробника або *MRO*-організації, яка обслуговувала компонент.

При збереженні знайдених прогалин система перевіряє наявність існуючих записів з таким самим типом для цього літака. Якщо прогалина вже існує зі статусом "вирішена" або "прийнята", нова прогалина не створюється – це

запобігає повторній появі вже закритих проблем при повторному запуску аналізу. Якщо існуюча прогалина має статус "відкрита", оновлюються її пріоритетний бал та рівень критичності, оскільки вони могли змінитися з часом.

3.4 Реалізація веб-інтерфейсу та візуалізації

Клієнтська частина системи реалізована як односторінковий застосунок (*Single Page Application*) з використанням бібліотеки *React* версії 18 [26]. Односторінковий підхід означає, що після початкового завантаження всі подальші переходи між розділами та оновлення контенту відбуваються без перезавантаження сторінки, що забезпечує швидкий і плавний користувацький досвід, аналогічний до настільних застосунків.

Проєкт клієнтської частини організовано у вигляді модульної компонентної архітектури. Каталог *components* містить *React*-компоненти – окремі багаторазово використовувані блоки інтерфейсу, кожен з яких відповідає за конкретну частину функціональності. Каталог *services* містить модулі для взаємодії з серверним *API*. Файл *App.jsx* є кореневим компонентом застосунку, а файли стилів *App.css* та *index.css* визначають зовнішній вигляд інтерфейсу.

Для збірки та запуску застосунку використовується *Vite* – сучасний інструмент розробки [31], який забезпечує миттєвий запуск *dev*-сервера завдяки використанню нативних *ES*-модулів браузера. На відміну від традиційних збірників, *Vite* не збирає весь код при кожній зміні, а використовує гарячу заміну модулів (*Hot Module Replacement*), що дозволяє бачити результати змін у коді практично миттєво.

Сервісний модуль *api.js* інкапсулює всю логіку взаємодії з серверним *API* та є єдиною точкою виконання *HTTP*-запитів. Модуль створює екземпляр бібліотеки *Axios* з налаштованим базовим *URL* та заголовками. Такий підхід централізує всю логіку мережевих запитів в одному місці, що спрощує зміну *URL* при розгортанні у різних середовищах та дозволяє додавати глобальну обробку помилок або автентифікацію без зміни кожного компонента окремо.

Головний компонент *Dashboard* є контейнером верхнього рівня та координує роботу всіх дочірніх компонентів (див. рисунок 3.8). При монтуванні компонента – моменті його першого відображення на екрані – автоматично виконується завантаження трьох наборів даних: списку літаків флоту, статистики флоту та рейтингу пріоритетів. Використання *Promise.all* дозволяє виконати всі три *HTTP*-запити паралельно, що значно скорочує час завантаження порівняно з послідовним виконанням.

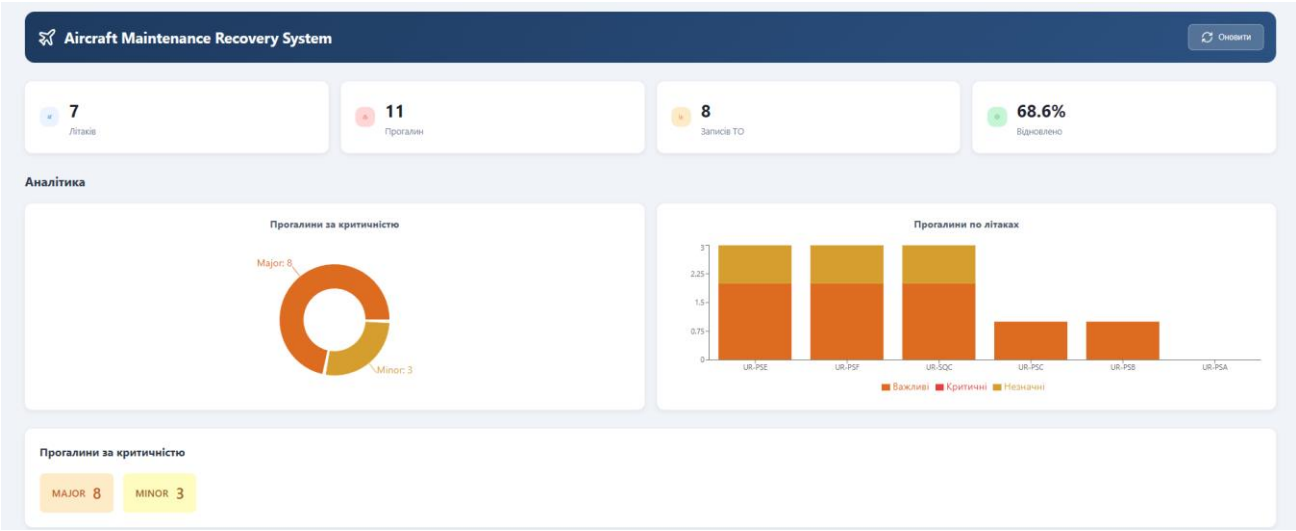


Рисунок 3.8 – Скріншот головної сторінки *Dashboard*

Компонент відображає декілька функціональних секцій. Заголовок містить назву системи "*Aircraft Maintenance Recovery System*" та кнопку оновлення даних з іконкою, яка повторно завантажує всю інформацію з сервера. Секція статистики представлена чотирма кольоровими картками з ключовими показниками: загальна кількість літаків у синій картці з іконкою літака, кількість відкритих прогалин у червоній картці з іконкою попередження (див. рисунок 3.9).

Кількість записів технічного обслуговування у помаранчевій картці з іконкою інструментів, відсоток відновлення документації у зеленій картці з іконкою галочки (див. рисунок 3.10). Кожна картка містить числове значення та текстовий підпис внизу.

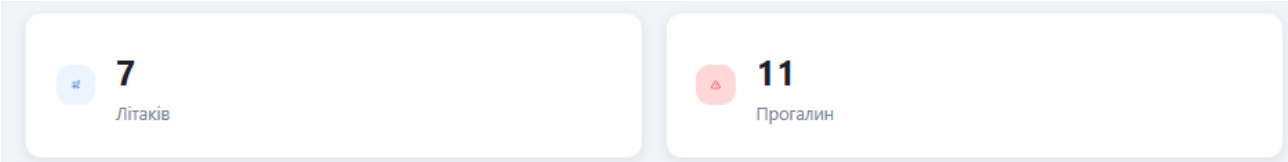


Рисунок 3.9 – Скріншот секції статистики про загальну кількість літаків та прогалин

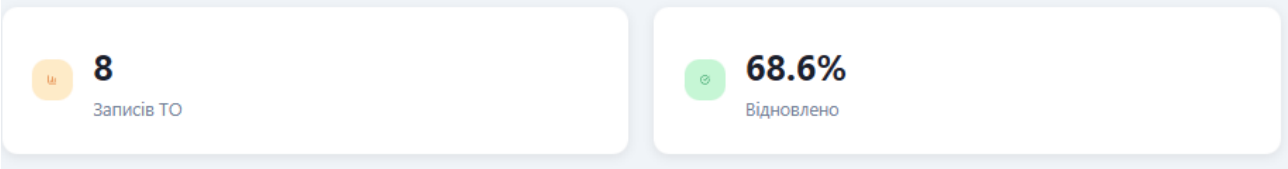


Рисунок 3.10 – Скріншот секції статистики про кількість записів ТО та відсотку відновлення

Секція розподілу прогалин за критичністю відображає горизонтальні смуги для кожного рівня критичності з кольоровим кодуванням. Критичні прогалини відображаються червоним кольором, важливі – помаранчевим, незначні – жовтим, косметичні – світло-сірим. Довжина кожної смуги пропорційна кількості прогалин відповідного рівня, а числове значення відображається всередині смуги (див. рисунок 3.11). Така візуалізація дозволяє швидко оцінити структуру проблем у документації флоту без необхідності читати числові дані.

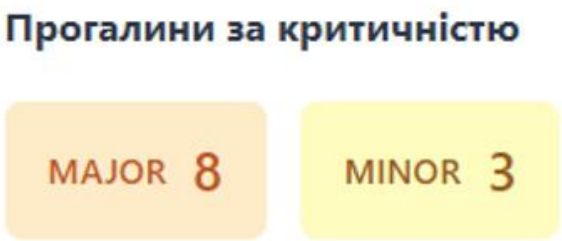


Рисунок 3.11 – Скріншот секції розподілу прогалин за критичніст

Секція графіків використовує компонент *Charts* для відображення двох типів діаграм. Кругова діаграма показує розподіл прогалин за критичністю у відсотках, де кожен сегмент пофарбований у відповідний колір та автоматично підписаний з кількістю (див. рисунок 3.12).

Рисунок 3.12 – Скріншот кругової діаграми

Стовпчикова діаграма відображає топ-6 літаків з найбільшою кількістю прогалин, де стовпчики розділені на кольорові сегменти за рівнями критичності (див. рисунок 3.13). Це дозволяє порівняти не тільки загальну кількість прогалин між літаками, але й їх структуру – наприклад, один літак може мати багато незначних прогалин, а інший менше загальної кількості, але всі критичні.

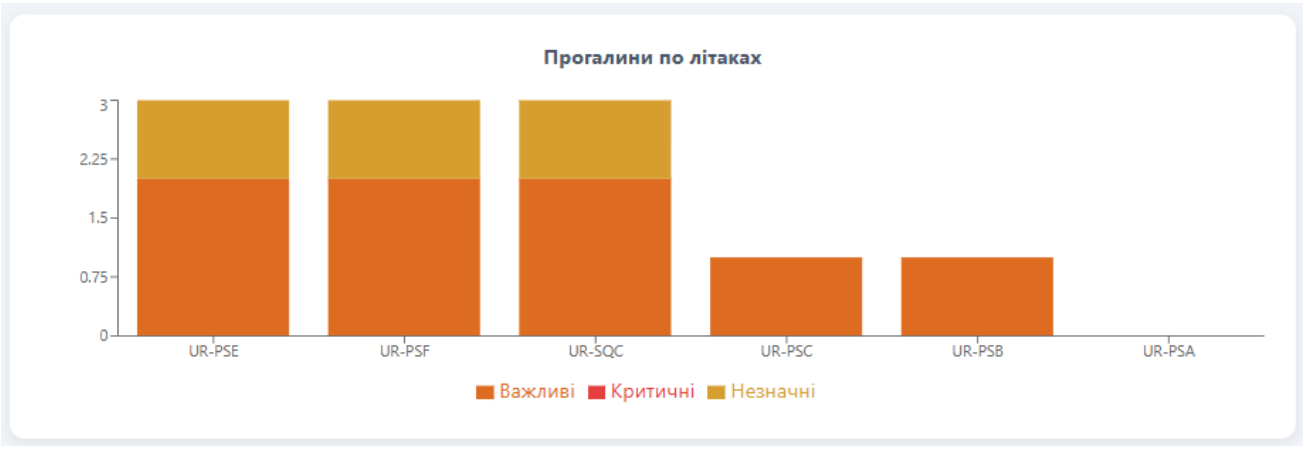


Рисунок 3.13 – Скріншот стовпчикової діаграми

Секція карток літаків є основною робочою областю інтерфейсу. Літаки відображаються у вигляді адаптивної сітки карток, де кількість колонок автоматично змінюється залежно від ширини екрану: чотири колонки на широких моніторах, три на середніх, дві на планшетах, одна на мобільних пристроях. Кожна картка є інтерактивним елементом з декількома можливостями взаємодії.

Картка літака містить реєстраційний номер як заголовок великими літерами, бейдж статусу з кольоровим кодуванням (зелений для активних, помаранчевий для тих що на зберіганні, червоний для заземлених, чорний для знищених), основну інформацію про виробника та модель в одному рядку, наліт у годинах та циклах з іконками, останнє відоме місцезнаходження з іконкою геолокації (див. рисунок 3.14).

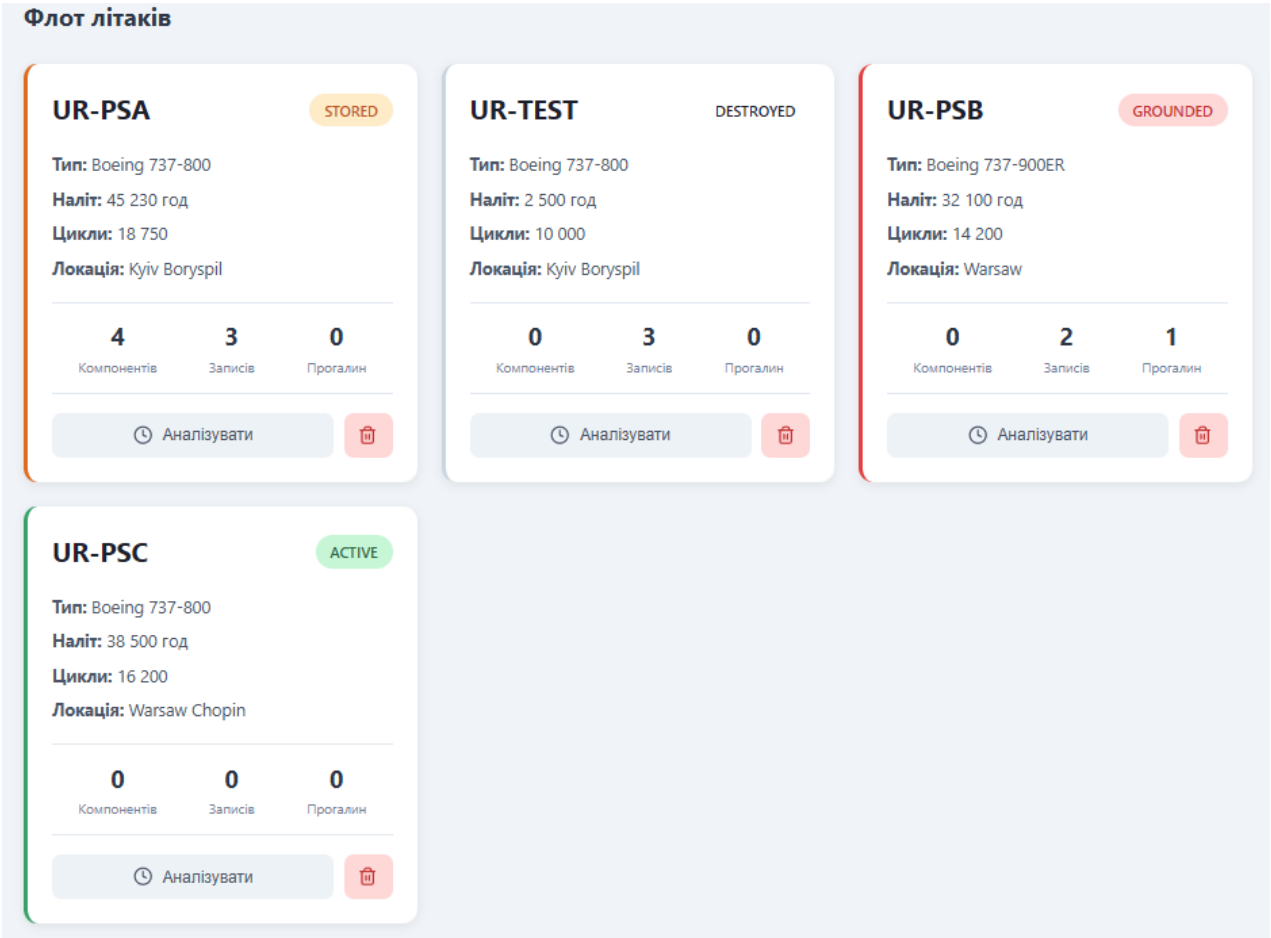


Рисунок 3.14 – Скріншот сітки карток літаків

Нижня частина картки містить три міні-показники з іконками та підписами: кількість встановлених компонентів з іконкою деталі, кількість записів технічного обслуговування з іконкою документа, кількість відкритих прогалин з іконкою попередження.

Панель дій внизу картки містить три кнопки: синю кнопку "Аналізувати" для запуску *gap*-аналізу, іконку ока для перегляду деталей, іконку кошика для видалення літака.

Клік на будь-яку частину картки відкриває детальний перегляд літака у модальному вікні.

Модальне вікно затемнює весь фон напівпрозорим чорним оверлеєм та відображає компонент *AircraftDetails* по центру екрану у білому прямокутнику з заокругленими кутами. Натискання на затемнену область або кнопку закриття з іконкою хрестика повертає користувача до головного екрану.

Компонент *AircraftDetails* реалізує повноекранний інтерфейс для роботи з конкретним літаком. При відкритті виконується паралельне завантаження трьох наборів даних: детальної інформації про літак, списку всіх прогалин незалежно від статусу та списку встановлених компонентів. Модальне вікно має фіксовану висоту з власною прокруткою всередині, що дозволяє переглядати великий обсяг інформації без впливу на фоновий інтерфейс.

Заголовок модального вікна містить реєстраційний номер літака великими жирними літерами зліва, кнопку редагування з іконкою олівця для зміни інформації про літак та кнопку закриття з іконкою хрестика справа. **Секція основної інформації** (див. рисунок 3.15) відображає всі характеристики літака у вигляді таблиці з двох колонок: назва поля сірим кольором зліва, значення чорним кольором справа. Поля включають тип повітряного судна, серійний номер виробника, виробника, модель, варіант конфігурації, дату виробництва, поточний статус з кольоровим бейджем, останнє відоме місцезнаходження та примітки якщо вони є.

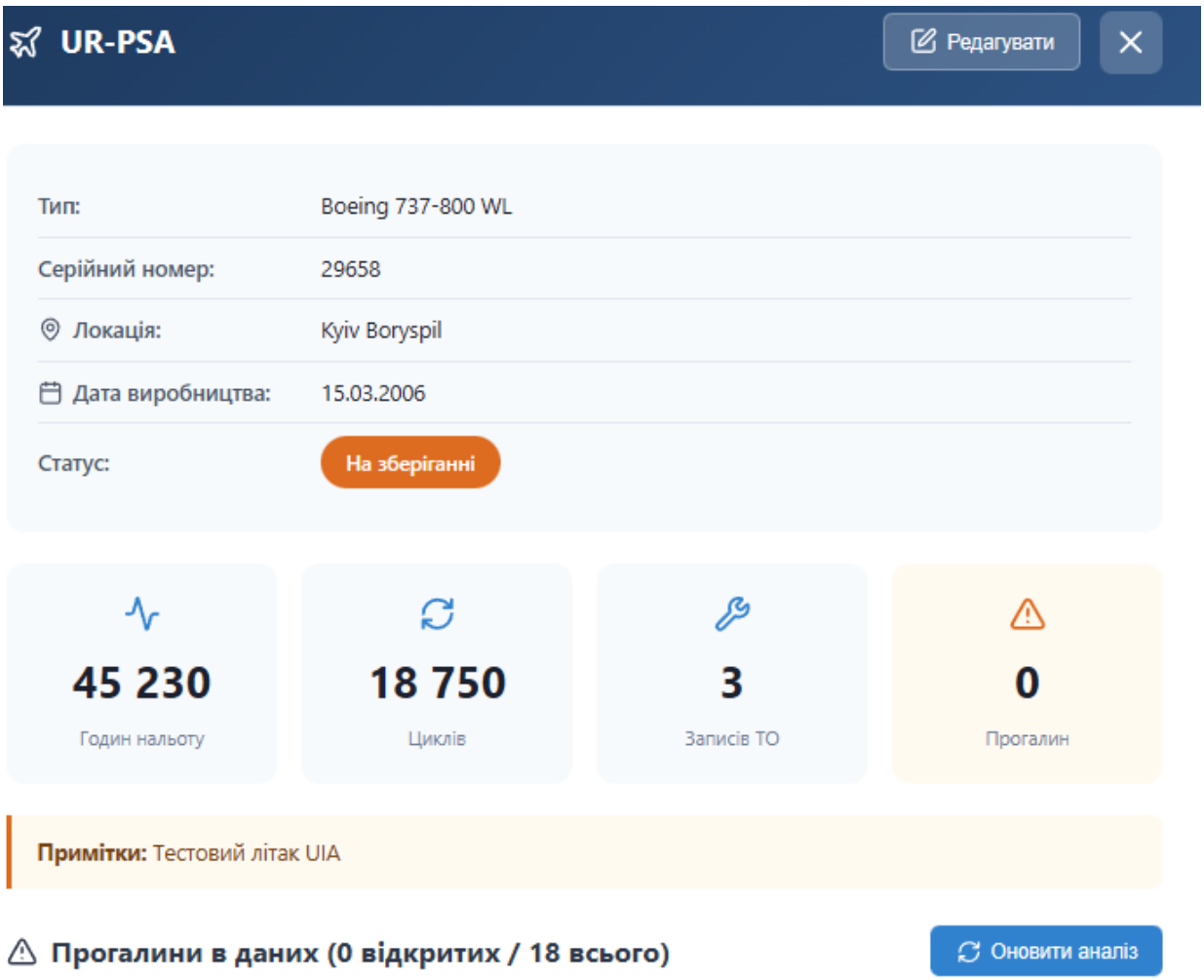


Рисунок 3.15 – Скріншот секції основної інформації характеристики літака

Сітка показників складається з чотирьох блоків розташованих у два рядки по два блоки: наліт у годинах з іконкою активності та великим числовим значенням, кількість циклів з іконкою оновлення, кількість записів ТО з іконкою інструментів, кількість відкритих прогалин з іконкою попередження на помаранчевому фоні для привернення уваги. Кожен блок має рамку та внутрішні відступи.

Секція прогалин починається з заголовка "Прогалини в даних" з кількістю відкритих прогалин відносно загальної кількості в форматі "(*X* відкритих / *Y* всього)" та синьою кнопкою "Повторити аналіз" справа. Якщо прогалин немає, відображається зелений блок з великою іконкою галочки та текстом "Прогалин не знайдено – документація повна". Якщо прогалини

присутні, вони відображаються як вертикальний список карток з відступами між ними.

Кожна **картка прогалини** містить повну інформацію про виявлену проблему у зручному для сприйняття форматі (див. рисунок 3.16). У верхній частині відображається рівень критичності та розрахований пріоритетний бал, що дозволяє інженеру швидко оцінити важливість прогалини. Основний текст картки описує суть проблеми та містить додаткову інформацію про тип необхідного технічного обслуговування та орієнтовний час, який знадобиться для відновлення відсутніх даних.

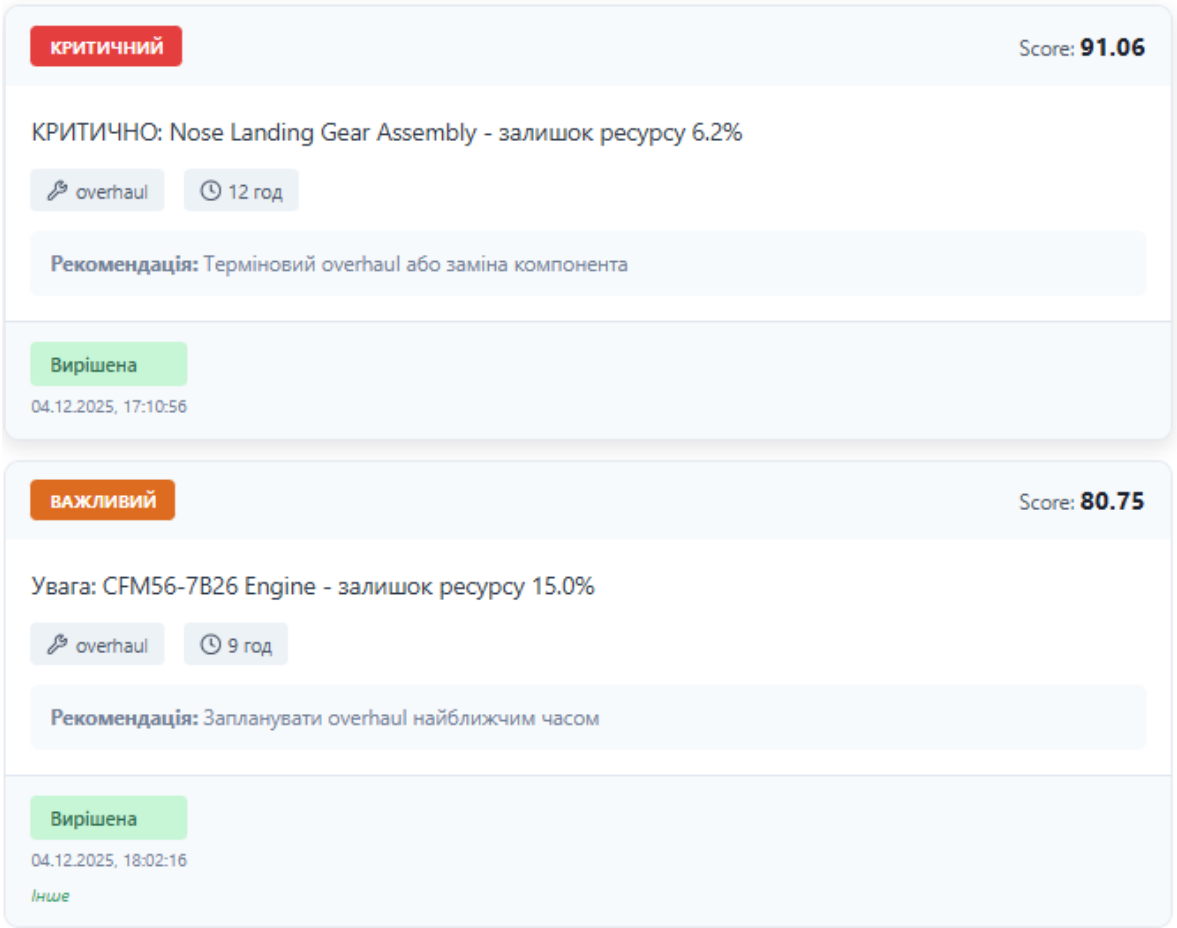


Рисунок 3.16 – Скріншот картки прогалин різних рівнів критичності

Система також пропонує рекомендовані дії для вирішення кожної конкретної прогалини – наприклад, знайти паперові документи, зв'язатися з *MRO*-організацією або провести фізичну інспекцію компонента. Нижня частина

картки показує поточний статус роботи з прогалиною. Для ще не вирішених прогалин доступна кнопка закриття, натискання якої відкриває форму вибору причини та додавання коментарів. Для вже закритих прогалин відображається повна історія вирішення: коли саме було закрито прогалину, хто це зробив і з якої причини, що забезпечує повну простежуваність процесу відновлення документації.

Натискання на кнопку "Вирішено" відкриває спеціальне модальне вікно для **закриття прогалини** (див. рисунок 3.18).

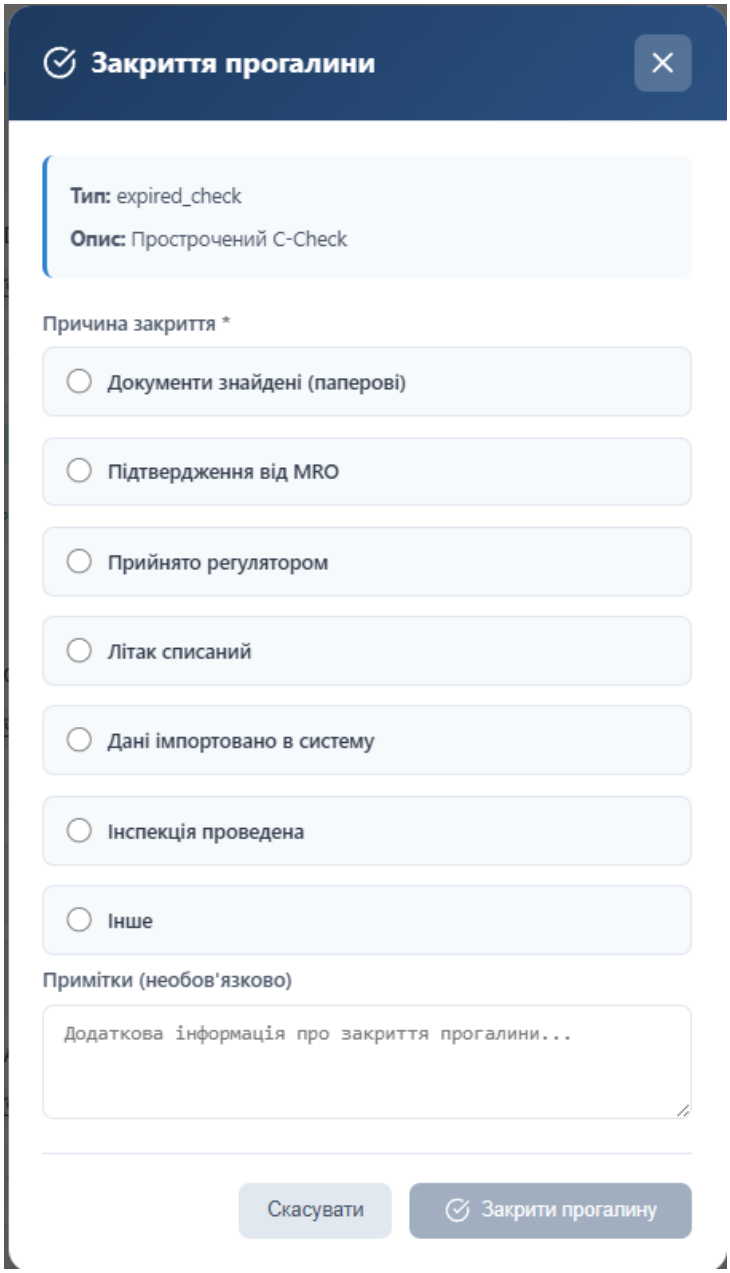


Рисунок 3.18 – Скріншот модального вікна закриття прогалин

Система пропонує користувачу вибрати одну з семи попередньо визначених причин закриття: знайдені паперові документи, отримано підтвердження від *MRO*-організації, прогалину прийнято регулятором як допустиму, дані успішно імпортовано в систему, проведено фізичну інспекцію, літак списано, або інша причина з можливістю вказати деталі.

Такий структурований підхід до документування причин закриття забезпечує консистентність даних та спрощує подальший аудит процесу відновлення. Користувач також може додати довільний текстовий коментар для деталізації обставин закриття прогалини.

Секція компонентів дозволяє переглядати та управляти всіма встановленими на літаку агрегатами та вузлами (див. рисунок 3.19).

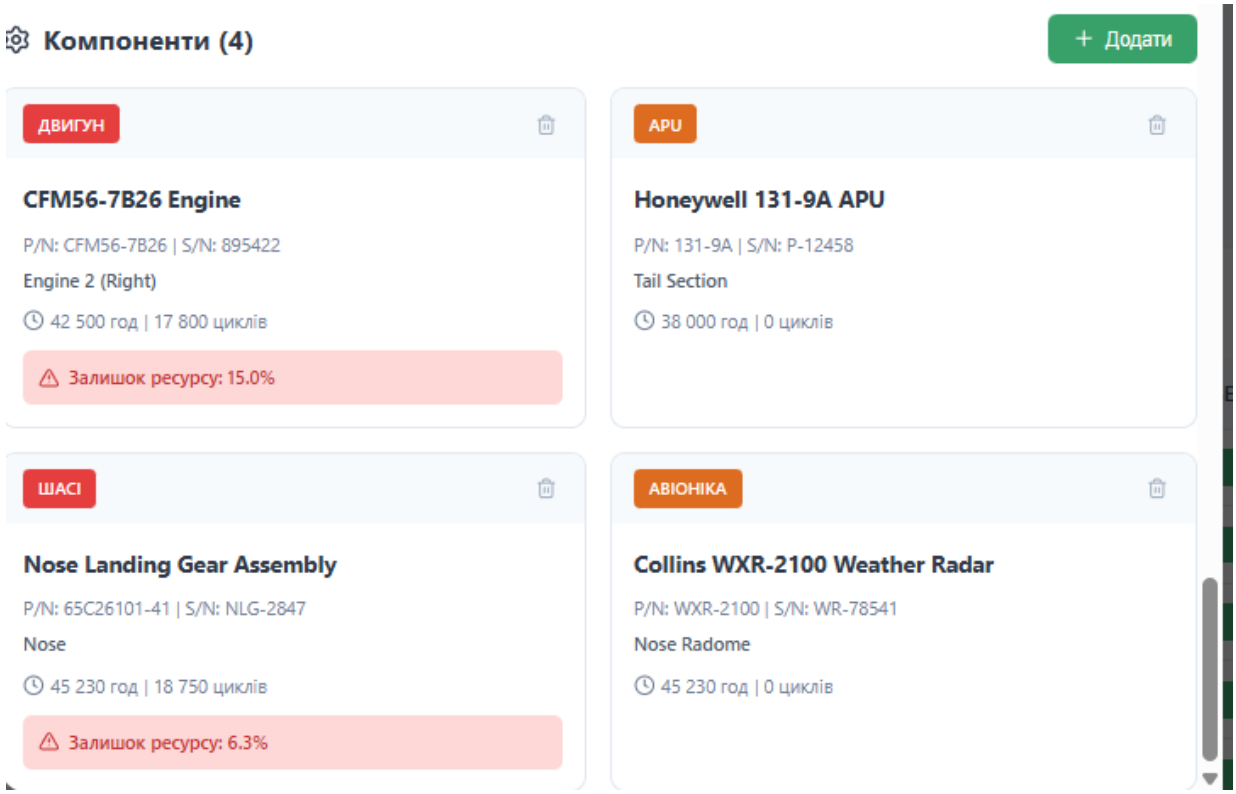


Рисунок 3.19 – Скріншот відображення компонентів літака

Система відображає компоненти у вигляді адаптивної сітки карток, де кількість колонок автоматично підлаштовується під ширину екрану. Якщо для

літака ще не додано жодного компонента, відображається порожній стан з відповідним повідомленням та можливістю швидко додати перший компонент.

Кожна картка компонента містит повну ідентифікаційну інформацію: категорію компонента згідно з класифікацією систем повітряного судна, найменування з каталожним та серійним номерами, позицію встановлення на літаку, поточне напрацювання в годинах та циклах. Для компонентів з обмеженим ресурсом система автоматично розраховує та відображає залишок ресурсу у відсотках. Візуальний індикатор використовує кольорове кодування: зелений колір сигналізує про достатній запас ресурсу, червоний – про критичну необхідність планування заміни або капітального ремонту. Це дозволяє інженерам швидко ідентифікувати компоненти, що наближаються до граничного напрацювання, без необхідності виконувати ручні розрахунки.

Форма додавання нового компонента розгортається безпосередньо в інтерфейсі при натисканні відповідної кнопки. Форма організована логічно з розділенням на ідентифікаційні дані (назва, категорія, номери), параметри встановлення (позиція, критичність) та дані про напрацювання.

Окрема увага приділена компонентам з обмеженим ресурсом – при встановленні відповідного прапорця з'являються додаткові поля для введення граничних значень напрацювання за годинами та циклами, які система використовуватиме для автоматичного моніторингу залишку ресурсу.

Модальне вікно додавання літака надає повний набір полів для внесення інформації про нове повітряне судно до системи (див. рисунок 3.20). Форма оптимізована для швидкого введення даних з автоматичною валідацією обов'язкових полів. Реєстраційний номер автоматично конвертується у верхній регістр для дотримання стандарту. Випадаючі списки для виробника та статусу запобігають введенню некоректних значень. Календарний вибір дати виробництва забезпечує правильний формат. Після збереження новий літак автоматично з'являється у загальній сітці та доступний для подальшої роботи.

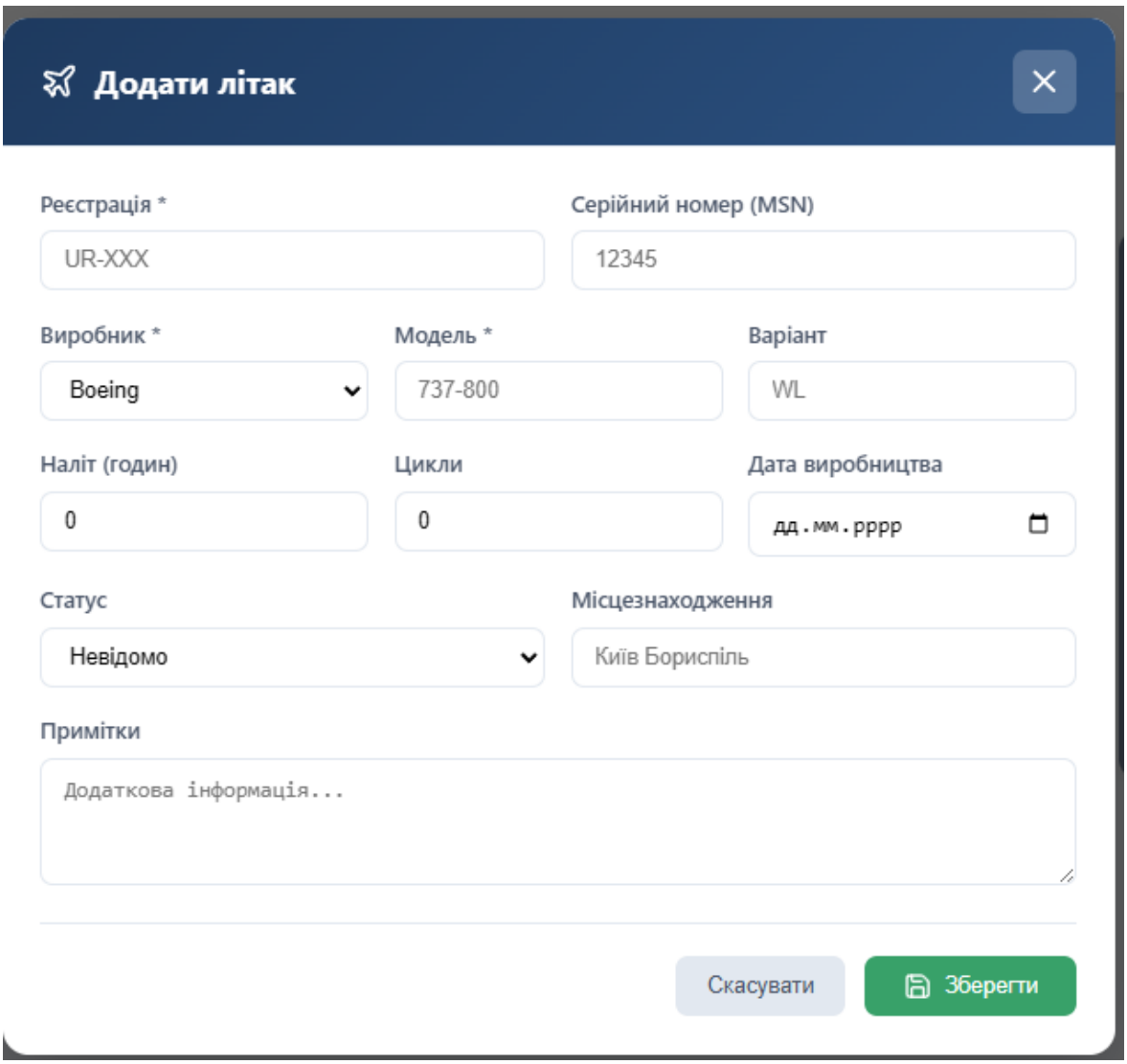


Рисунок 3.20 – Скріншот форми додавання нового літака до системи

Таблиця пріоритетів відновлення є ключовим інструментом для планування роботи з документацією флоту. Система автоматично ранжує всі літаки (див. рисунок 3.21) за пріоритетом відновлення, враховуючи кількість та критичність прогалин.

Для кожного літака відображається детальна статистика (див. рисунок 3.22): розбивка прогалин за рівнями критичності, розрахований показник готовності у відсотках з візуальним індикатором, оцінка загального часу необхідного для відновлення всієї документації.

Пріоритет відновлення

#	РЕЄСТРАЦІЯ	МОДЕЛЬ
1	UR-PSE	Boeing 737-800
2	UR-PSF	Boeing 737-900ER
3	UR-SQC	Airbus A319-100
4	UR-PSB	Boeing 737-900ER
5	UR-PSA	Boeing 737-800
6	UR-TEST	Boeing 737-800
7	UR-PSC	Boeing 737-800

Рисунок 3.21 – Скріншот таблиці рейтингу пріоритетів: реєстрація та модель літаків

КРИТИЧНІ	ВАЖЛИВІ	НЕЗНАЧНІ	РІВЕНЬ ВІДНОВЛЕННЯ	ЧАС (ГОД)
0	2	1	75%	38
0	2	1	75%	38
0	2	1	75%	38
0	1	0	90%	12
0	0	0	100%	0
0	0	0	100%	0
0	0	0	100%	0

Рисунок 3.22 – Скріншот таблиці рейтингу пріоритетів: розподіл прогалин та показник готовності

Показник готовності дозволяє швидко визначити які літаки найближче до повного відновлення документації, а які потребують найбільших зусиль. Реєстраційні номери в таблиці є активними посиланнями, що дозволяє швидко перейти до детального перегляду конкретного літака для початку роботи з його прогалинами.

Модальне вікно імпорту даних реалізує функціонал масового завантаження інформації з зовнішніх джерел. Система підтримує три найпоширеніших формати файлів та автоматично визначає структуру даних. Інтерфейс організовано у вигляді вкладок для різних типів імпорту (див. рисунок 3.23).

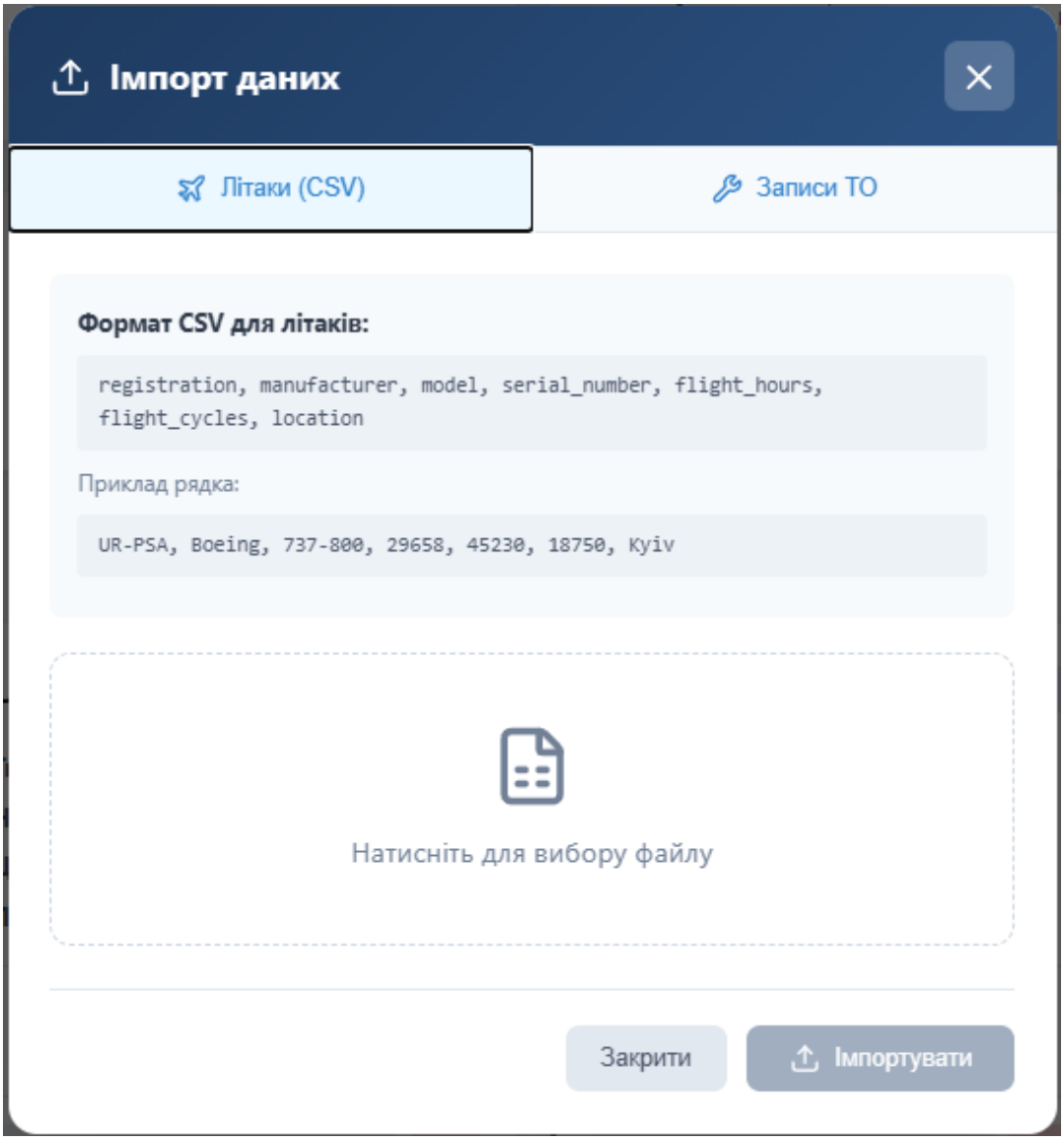


Рисунок 3.23 – Скріншот меню імпорту літаків з *CSV*

Особливу увагу приділено метаданим джерела – користувач обов'язково вказує походження даних та оцінює їх надійність (див. рисунок 3.24). Ця оцінка в подальшому використовується системою при *gap*-аналізі та допомагає розрізняти дані від офіційних сертифікованих *MRO*-організацій від неперевірених джерел.

Область завантаження файлу підтримує як традиційний вибір через діалогове вікно, так і сучасний метод перетягування файлу. Після вибору файлу система відображає його параметри та активує кнопку імпорту. Процес імпорту виконується з детальною валідацією кожного рядка даних.

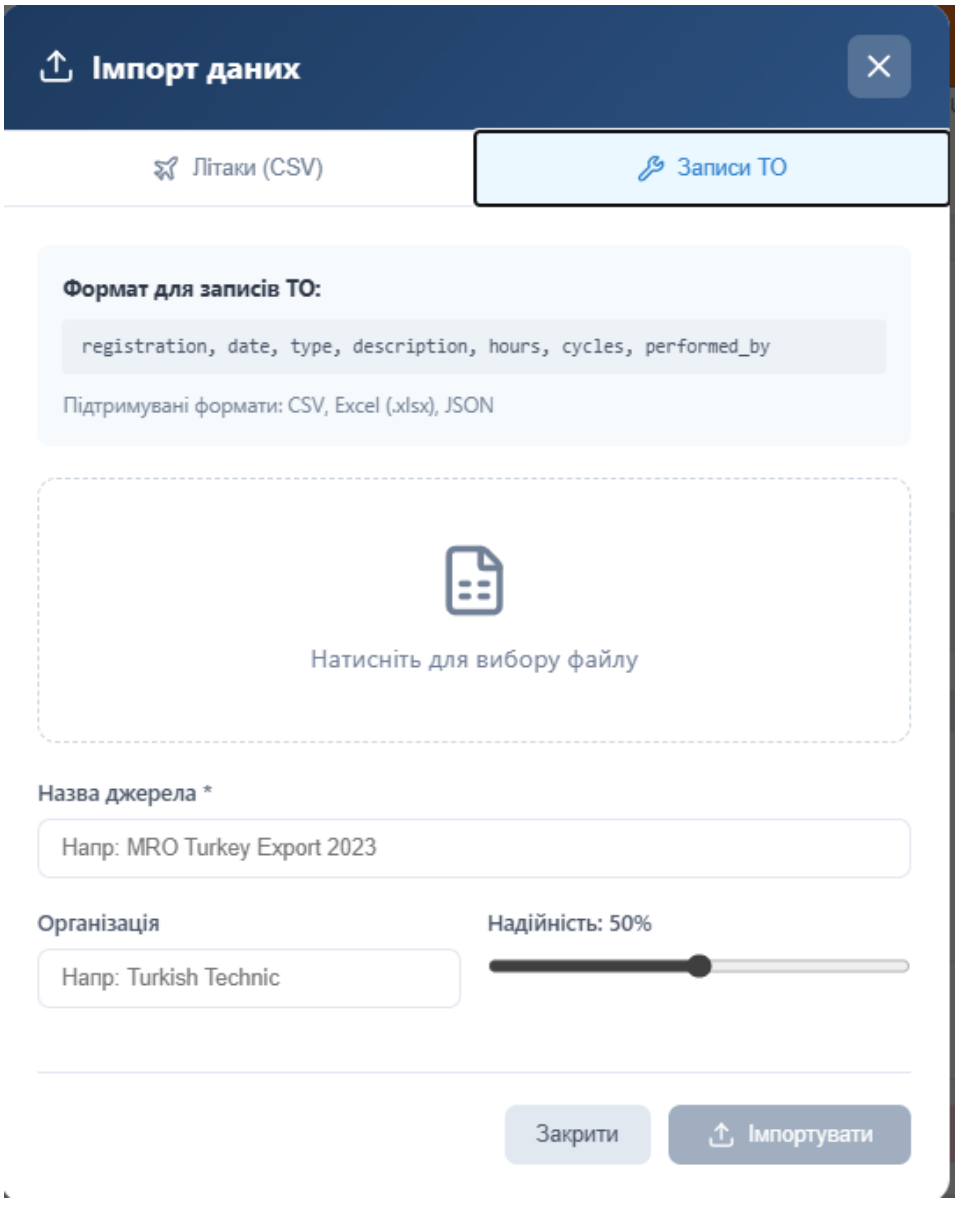


Рисунок 3.24 – Скріншот меню імпорту літаків ТО з налаштуваннями джерела

Результати імпорту відображаються у зрозумілому форматі з підсумковою статистикою та детальним переліком помилок якщо вони виникли. Кожна помилка вказує конкретний рядок файлу та опис проблеми, що дозволяє користувачу швидко виправити дані та повторити імпорт.

Висновки до розділу

У третьому розділі виконано програмну реалізацію системи відновлення даних технічного обслуговування повітряних суден та проведено комплексне тестування.

Обґрунтовано вибір технологій розробки. Для серверної частини обрано мову *Python* з фреймворком *FastAPI*, що забезпечує високу продуктивність та автоматичну генерацію документації *API*. Для роботи з базою даних використано *ORM SQLAlchemy* та СУБД *PostgreSQL*. Для клієнтської частини обрано бібліотеку *React* з *Recharts* для візуалізації. Обрані технології є відкритим програмним забезпеченням з активними спільнотами та якісною документацією.

Реалізовано базу даних з *ORM*-моделями для всіх сутностей предметної області. Моделі *Aircraft*, *Component*, *MaintenanceRequirement*, *MaintenanceRecord*, *DataGap* та *DataSource* забезпечують зберігання інформації про флот, компоненти, вимоги ТО, записи про виконані роботи, виявлені прогалини та джерела даних. Налаштовано зв'язки між моделями та індекси для оптимізації запитів.

Реалізовано *REST API* з ендпоінтами для *CRUD*-операцій над сутностями, імпорту даних з файлів *CSV*, *Excel* та *JSON*, виконання *gap*-аналізу та отримання аналітичних даних для дашборду. *API* документовано через автоматично згенеровану специфікацію *Swagger*.

Реалізовано сервіси бізнес-логіки. Сервіс *DataImporter* забезпечує імпорт даних з автоматичним розпізнаванням структури, маппінгом полів та нормалізацією значень. Сервіс *GapAnalyzer* реалізує алгоритм виявлення

прогалин та розрахунок *scoring*-оцінок за методикою з чотирма факторами критичності.

Реалізовано веб-інтерфейс користувача. Дашборд відображає зведену статистику та візуалізацію стану флоту. Сторінка повітряного судна надає детальну інформацію про літак та його прогалини. Сторінка управління прогалинами забезпечує фільтрацію, сортування та зміну статусу. Сторінка імпорту дозволяє завантажувати файли з даними.

ВИСНОВКИ

У кваліфікаційній роботі проведено комплексне дослідження та практичну розробку системи відновлення інформаційного забезпечення технічного обслуговування та ремонту літаків авіакомпанії, адаптованої до умов післявоєнного періоду, з акцентом на консолідацію фрагментованих даних та автоматизоване виявлення прогалин у документації, що є критичною умовою для відновлення сертифікації льотної придатності флоту.

Актуальність теми зумовлена наслідками повномасштабного вторгнення, внаслідок якого українські авіакомпанії втратили доступ до серверів, архівів та баз даних технічного обслуговування. Дані про виконані роботи фрагментовані: частина залишилась у резервних копіях, частина в Excel-файлах на комп'ютерах співробітників, частина у сторонніх MRO-провайдерів за кордоном. Відсутність документального підтвердження виконаних робіт унеможливлює отримання сертифіката льотної придатності, що призводить до заземлення технічно справних повітряних суден. Традиційні MRO-системи орієнтовані на штатну експлуатацію з безперервним потоком структурованих даних і не надають інструментів для роботи з фрагментованими джерелами, що робить запропоноване рішення своєчасним і практично значущим для українських авіакомпаній.

Метою роботи є створення програмного засобу для консолідації фрагментованих даних технічного обслуговування, автоматичного виявлення прогалин та визначення пріоритетності їх відновлення. Для досягнення мети вирішено ряд завдань: проаналізовано предметну область технічного обслуговування повітряних суден та регуляторні вимоги до документації, досліджено існуючі MRO-системи та виявлено їх обмеження, спроєктовано архітектуру системи та структуру бази даних, розроблено алгоритм gap-аналізу та методику scoring-оцінки критичності прогалин, реалізовано веб-інтерфейс користувача з візуалізацією стану флоту.

Аналіз предметної області виявив, що документування технічного обслуговування регламентується міжнародними стандартами ICAO та європейським Регламентом 1321/2014, які визначають обов'язкові елементи записів та вимоги до їх зберігання протягом усього терміну експлуатації повітряного судна. Порівняльний аналіз провідних систем AMOS, Ramco, TRAX, OASES та flydocs показав, що жодна з них не поєднує функції імпорту з різнорідних фрагментованих джерел, автоматичного gap-аналізу та scoring-пріоритезації в єдиному рішенні, орієнтованому на післякризові сценарії.

У процесі розробки визначено функціональні та нефункціональні вимоги до системи та обрано сучасні технології реалізації: мову програмування Python з фреймворком FastAPI для серверної частини, СУБД PostgreSQL для надійного зберігання даних з підтримкою ACID-транзакцій, бібліотеку React для побудови інтерактивного клієнтського інтерфейсу. Система побудована за трирівневою клієнт-серверною архітектурою з чітким розділенням рівнів представлення, бізнес-логіки та даних. Взаємодія між компонентами реалізована через REST API з обміном даними у форматі JSON.

Розроблена система включає нормалізовану базу даних з шести взаємопов'язаних таблиць для зберігання інформації про повітряні судна, компоненти, вимоги програми технічного обслуговування, записи про виконані роботи, виявлені прогалини та джерела імпортованих даних. Бізнес-логіка реалізована у вигляді окремих модулів: імпорту даних з підтримкою форматів CSV та Excel, нормалізації даних для приведення до єдиної структури, gap-аналізу для виявлення відсутніх записів та scoring-оцінки для визначення критичності прогалин. Розроблена методика оцінки критичності враховує п'ять факторів: критичність компонента, час з останнього обслуговування, тип роботи, наявність обмеженого ресурсу та надійність джерела даних, що дозволяє отримати інтегральний показник від 0 до 100 балів з класифікацією за чотирма рівнями критичності.

Веб-інтерфейс системи забезпечує зручну роботу користувачів та включає дашборд з ключовими показниками стану флоту та діаграмами розподілу

прогалин за категоріями критичності, сторінку централізованого перегляду та управління прогалинами з можливістю фільтрації, сортування та зміни статусу, сторінку імпорту даних з підтримкою завантаження файлів методом drag-and-drop та автоматичним розпізнаванням структури. Кольорова індикація забезпечує швидку візуальну оцінку стану документації без необхідності детального аналізу числових показників.

Результати роботи демонструють, що запропонована система дозволяє суттєво скоротити час консолідації фрагментованих даних технічного обслуговування з різнорідних джерел, оптимізувати використання обмежених ресурсів авіакомпанії через пріоритезацію відновлення документації за рівнем критичності та підготувати структуровані дані для подальшої міграції у промислові MRO-системи після стабілізації операційної діяльності. Використання технологій з відкритим вихідним кодом забезпечує мінімізацію витрат на впровадження та незалежність від комерційних постачальників програмного забезпечення.

Перспективи подальшого розвитку системи охоплюють розширення переліку підтримуваних форматів імпорту для роботи з більшою різноманітністю джерел даних, реалізацію інтеграції з промисловими MRO-системами через програмні інтерфейси для автоматизації обміну даними, застосування методів машинного навчання для автоматичного розпізнавання структури вхідних файлів та класифікації записів, а також розробку мобільного застосунку для роботи в польових умовах з обмеженим доступом до мережевої інфраструктури.

Таким чином, робота пропонує практичне та теоретично обґрунтоване рішення для відновлення інформаційного забезпечення технічного обслуговування повітряних суден в умовах післявоєнного періоду, що може бути використане українськими авіакомпаніями при відновленні діяльності та слугувати основою для подальших досліджень у сфері інформаційних технологій для кризових ситуацій в авіаційній галузі.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *World Bank. (2025, February 25). Updated Ukraine recovery and reconstruction needs assessment released.* [Електронний ресурс] *URL:https://www.worldbank.org/en/news/press-release/2025/02/25/updated-ukraine-recovery-and-reconstruction-needs-assessment-released.*

2. *European Union Aviation Safety Agency. (2024). Easy Access Rules for Continuing Airworthiness (Regulation (EU) No 1321/2014). EASA.* [Електронний ресурс] *URL:https://www.easa.europa.eu/en/document-library/easy-access-rules/easy-access-rules-continuing-airworthiness-regulation-eu-no.*

3. *ICAO. (2022). Annex 6: Operation of Aircraft. Part I: International Commercial Air Transport – Aeroplanes (12th ed.). International Civil Aviation Organization.* [Електронний ресурс]. *URL:https://www.swiss-as.com/amos-mro URL: https://www.icao.int/publications/pages/publication.aspx?docnum=AN%206-1.*

4. *ICAO. (2022). Doc 9760: Airworthiness Manual (5th ed.). International Civil Aviation Organization.* [Електронний ресурс] *URL: https://www.icao.int/publications/pages/publication.aspx.*

5. *Swiss AviationSoftware. (2024). AMOS: Aviation Maintenance & Engineering System. URL:*

6. *Ramco Systems. (2024). Ramco Aviation Suite: MRO Software Solutions. URL: https://www.ramco.com/aviation .*

7. *TRAX. (2024). eMRO – Enterprise Maintenance, Repair & Overhaul Solution. URL: https://www.trax.aero).*

8. *Commsoft. (2024). OASES – Open Aviation Strategic Engineering System. URL: https://www.oases.aero.*

9. *flydocs. (2024). Aircraft Records Management Platform. URL: https://www.flydocs.aero.*

10. Python Software Foundation. (2024). What is Python? Executive Summary. URL: https://www.python.org/doc/essays/blurb/.

11. Dou.ua. (2025, February 10). Рейтинг мов програмування 2025. TypeScript і Python. URL: https://dou.ua/lenta/articles/language-rating-2025/.

12. FastAPI. (2024). FastAPI Documentation.
URL: https://fastapi.tiangolo.com .

13. Pydantic. (2024). Pydantic Documentation. Version 2.10.
URL: https://docs.pydantic.dev .

14. SQLAlchemy. (2024). SQLAlchemy 2.0 Documentation.
URL: https://docs.sqlalchemy.org .

15. Alembic. (2024). Alembic Documentation. Version 1.14.
URL: https://alembic.sqlalchemy.org .

16. PostgreSQL Global Development Group. (2024). PostgreSQL 16 Documentation. URL: https://www.postgresql.org/docs/16 .

17. Pandas Development Team. (2024). Pandas Documentation. Version 2.2.
URL: https://pandas.pydata.org/docs .

18. Openpyxl. (2024). Openpyxl Documentation. Version 3.1.
URL: https://openpyxl.readthedocs.io .

19. React. (2024). React Documentation. Meta Platforms.
URL: https://react.dev .

20. Recharts. (2024). Recharts Documentation. URL: https://recharts.org .

21. Axios. (2024). Axios Documentation. URL: https://axios-http.com .

22. Node.js. (2024). Node.js Documentation. Version 22. OpenJS Foundation.
URL: https://nodejs.org/docs/latest-v22.x/api .

23. McKinney W. (2022). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Jupyter (3rd ed.). O'Reilly Media. ISBN: 978-1098104030.

24. Kleppmann M. (2023). Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media. ISBN: 978-1098119058.

25. Richards M., Ford N. (2021). Software Architecture: The Hard Parts. O'Reilly Media. ISBN: 978-1492086895.

26. Lauret A. (2023). The Design of Web APIs (2nd ed.). Manning Publications. ISBN: 978-1617299919.

27. Geewax J.J. (2021). API Design Patterns. Manning Publications. ISBN: 978-1617295850.

28. Elmasri R., Navathe S.B. (2022). Fundamentals of Database Systems (8th ed.). Pearson. ISBN: 978-0137502523.

29. Banks A., Porcello E. (2020). Learning React: Modern Patterns for Developing React Apps (2nd ed.). O'Reilly Media. ISBN: 978-1492051725.

30. Vite. (2024). Vite Documentation. Version 5.0. URL: https://vitejs.dev .

31. ISO/IEC 25010:2023. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE). Geneva: ISO.

32. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ: ДП «УкрНДНЦ», 2015.

33. Uvicorn. (2024). Uvicorn Documentation. URL: https://www.uvicorn.org.

34. Liskovych N., Shevchuk D., Cherednichenko K. (2024). Strategic adaptation of Ukrainian airlines to the challenges of post-war air traffic organization. CEUR Workshop Proceedings. URL: https://ceur-ws.org/Vol-3925/short11.pdf .

35. Reuters. (2025, February 25). Ukraine needs $524 billion to recover, rebuild after three years of war, World Bank says. URL: https://www.reuters.com/world/europe/ukraine-needs-524-billion-recover-rebuild-after-three-years-war-world-bank-says-2025-02-25/ .

36. Leonardo. (2025, July 10). Leonardo, Enav and UkSATSE sign an agreement to restore Ukraine's air traffic management system. URL: https://www.leonardo.com/en/press-release-detail/-/detail/10-07-2025-leonardo-enav-and-uksatse-sign-an-agreement-to-restore-ukraine-s-air-traffic-management-system .

37. Federal Aviation Administration. (2023). Aviation Maintenance Technician Handbook – General (FAA-H-8083-30B). U.S. Department of Transportation. URL:https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/media/amt_general_handbook.pdf .

38. Python.org. (2024). Welcome to Python.org. URL: https://www.python.org .

Додаток А

Модуль ORM-моделей

```
from sqlalchemy import Column, Integer, String, Float, DateTime, Date, ForeignKey,
Text, Boolean, Enum as SQLEnum
from sqlalchemy.orm import relationship
from datetime import datetime
import enum

from app.core.database import Base

class AircraftStatus(enum.Enum):
    ACTIVE = "active"
    STORED = "stored"
    IN_MAINTENANCE = "in_maintenance"
    GROUNDED = "grounded"
    DESTROYED = "destroyed"
    UNKNOWN = "unknown"

class ComponentCategory(enum.Enum):
    ENGINE = "engine"
    APU = "apu"
    LANDING_GEAR = "landing_gear"
    AVIONICS = "avionics"
    FLIGHT_CONTROLS = "flight_controls"
    HYDRAULICS = "hydraulics"
    PNEUMATICS = "pneumatics"
    FUEL_SYSTEM = "fuel_system"
```

```
    ELECTRICAL = "electrical"
    STRUCTURES = "structures"
    INTERIOR = "interior"
    OTHER = "other"

class MaintenanceType(enum.Enum):
    A_CHECK = "a_check"
    B_CHECK = "b_check"
    C_CHECK = "c_check"
    D_CHECK = "d_check"
    LINE_MAINTENANCE = "line_maintenance"
    ENGINE_OVERHAUL = "engine_overhaul"
    COMPONENT_REPLACEMENT = "component_replacement"
    AD_COMPLIANCE = "ad_compliance"
    SB_COMPLIANCE = "sb_compliance"
    REPAIR = "repair"
    INSPECTION = "inspection"
    MODIFICATION = "modification"
    OTHER = "other"
class CriticalityLevel(enum.Enum):
    CRITICAL = "critical"
    MAJOR = "major"
    MINOR = "minor"
    COSMETIC = "cosmetic"

class GapStatus(enum.Enum):
    OPEN = "open"
    IN_PROGRESS = "in_progress"
```

```
    RESOLVED = "resolved"
```



```
    ACCEPTED = "accepted"

class Aircraft(Base):
    __tablename__ = "aircraft"

    id = Column(Integer, primary_key=True, index=True)
    registration = Column(String(20), unique=True, nullable=False, index=True)
    serial_number = Column(String(50))
    manufacturer = Column(String(100))
    model = Column(String(100))
    variant = Column(String(50))

    total_flight_hours = Column(Float, default=0)
    total_cycles = Column(Integer, default=0)

    manufacture_date = Column(Date)
    status = Column(SQLEnum(AircraftStatus), default=AircraftStatus.UNKNOWN)
    last_known_location = Column(String(200))
    notes = Column(Text)

    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime, default=datetime.utcnow,
onupdate=datetime.utcnow)
    components = relationship("AircraftComponent", back_populates="aircraft",
cascade="all, delete-orphan")
    maintenance_records = relationship("MaintenanceRecord",
back_populates="aircraft", cascade="all, delete-orphan")
```

```
    data_gaps = relationship("DataGap", back_populates="aircraft", cascade="all, delete-orphan")

class AircraftComponent(Base):
    __tablename__ = "aircraft_components"

    id = Column(Integer, primary_key=True, index=True)
    aircraft_id = Column(Integer, ForeignKey("aircraft.id"), nullable=False)

    name = Column(String(200), nullable=False)
    part_number = Column(String(100))
    serial_number = Column(String(100))
    category = Column(SQLEnum(ComponentCategory), default=ComponentCategory.OTHER)
    position = Column(String(100))

    is_life_limited = Column(Boolean, default=False)
    life_limit_hours = Column(Float)
    life_limit_cycles = Column(Integer)

    hours_since_new = Column(Float, default=0)
    cycles_since_new = Column(Integer, default=0)
    hours_since_overhaul = Column(Float)
    cycles_since_overhaul = Column(Integer)

    installation_date = Column(Date)
    criticality = Column(SQLEnum(CriticalityLevel), default=CriticalityLevel.MINOR)
    notes = Column(Text)
```

```
    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)
    aircraft = relationship("Aircraft", back_populates="components")
    maintenance_records = relationship("MaintenanceRecord", back_populates="component", cascade="all, delete-orphan")

class MaintenanceRecord(Base):
    __tablename__ = "maintenance_records"

    id = Column(Integer, primary_key=True, index=True)
    aircraft_id = Column(Integer, ForeignKey("aircraft.id"), nullable=False)
    component_id = Column(Integer, ForeignKey("aircraft_components.id"))

    maintenance_type = Column(SQLEnum(MaintenanceType), nullable=False)
    description = Column(Text)
    performed_date = Column(Date, nullable=False)

    flight_hours_at_maintenance = Column(Float)
    cycles_at_maintenance = Column(Integer)

    performed_by = Column(String(200))
    work_order_number = Column(String(100))
    certificate_number = Column(String(100))

    data_source_id = Column(Integer, ForeignKey("data_sources.id"))
    confidence_score = Column(Float, default=1.0)
    notes = Column(Text)
```

```
    created_at = Column(DateTime, default=datetime.utcnow)

    aircraft = relationship("Aircraft", back_populates="maintenance_records")
    component = relationship("AircraftComponent",
back_populates="maintenance_records")
    data_source = relationship("DataSource",
back_populates="maintenance_records")

class DataGap(Base):
    __tablename__ = "data_gaps"

    id = Column(Integer, primary_key=True, index=True)
    aircraft_id = Column(Integer, ForeignKey("aircraft.id"), nullable=False)
    component_id = Column(Integer, ForeignKey("aircraft_components.id"))

    gap_type = Column(String(100), nullable=False)
    maintenance_type = Column(SQLEnum(MaintenanceType))

    priority_score = Column(Float, default=0)
    criticality = Column(SQLEnum(CriticalityLevel), default=CriticalityLevel.MINOR)

    description = Column(Text)
    recommended_action = Column(Text)
    estimated_recovery_hours = Column(Float)
    status = Column(SQLEnum(GapStatus), default=GapStatus.OPEN)
    resolved_at = Column(DateTime)
    resolution_reason = Column(String(100))
    resolution_notes = Column(Text)
```

```
    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime, default=datetime.utcnow,
onupdate=datetime.utcnow)
    aircraft = relationship("Aircraft", back_populates="data_gaps")
    component = relationship("AircraftComponent")

class DataSource(Base):
    __tablename__ = "data_sources"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(200), nullable=False)
    source_type = Column(String(50))
    file_name = Column(String(255))
    organization = Column(String(200))
    reliability_score = Column(Float, default=0.5)
    records_imported = Column(Integer, default=0)
    records_with_errors = Column(Integer, default=0)
    import_date = Column(DateTime, default=datetime.utcnow)
    notes = Column(Text)
    maintenance_records = relationship("MaintenanceRecord",
back_populates="data_source")
class MaintenanceRequirement(Base):
    __tablename__ = "maintenance_requirements"

    id = Column(Integer, primary_key=True, index=True)
    aircraft_model = Column(String(100), nullable=False)
```

requirement_type = Column(SQLEnum(MaintenanceType), nullable=False)

interval_hours = Column(Float)

interval_cycles = Column(Integer)

interval_days = Column(Integer)

description = Column(Text)

regulatory_reference = Column(String(200))

is_mandatory = Column(Boolean, default=True)

created_at = Column(DateTime, default=datetime.utcnow)

Додаток Б

Модуль аналізу прогалин

Модуль складається з двох класів: ScoringEngine для розрахунку пріоритетів та GapAnalyzer для виявлення прогалин.

Вагові коефіцієнти та таблиці оцінювання:

```
class ScoringEngine:
    WEIGHTS = {
        'component_criticality': 0.35,
        'time_since_maintenance': 0.25,
        'maintenance_type': 0.20,
        'life_limited': 0.15,
        'data_reliability': 0.05
    }

    CRITICALITY_SCORES = {
        ComponentCategory.ENGINE: 1.0,
        ComponentCategory.FLIGHT_CONTROLS: 0.95,
        ComponentCategory.LANDING_GEAR: 0.90,
        ComponentCategory.AVIONICS: 0.80,
        # ... інші категорії
    }

    MAINTENANCE_TYPE_SCORES = {
        MaintenanceType.AD_COMPLIANCE: 1.0,
        MaintenanceType.D_CHECK: 0.95,
        MaintenanceType.ENGINE_OVERHAUL: 0.90,
        MaintenanceType.C_CHECK: 0.85,
        # ... інші типи
    }
```

```
def calculate_priority_score(self, component_category, maintenance_type,
                             days_since_maintenance, is_life_limited=False,
                             life_remaining_percent=None) -> float:
    score = 0.0

    if component_category:
        score += self.CRITICALITY_SCORES.get(component_category, 0.5) \
                 * self.WEIGHTS['component_criticality']

    if maintenance_type:
        score += self.MAINTENANCE_TYPE_SCORES.get(maintenance_type, 0.5) \
                 * self.WEIGHTS['maintenance_type']

    if days_since_maintenance:
        score += min(days_since_maintenance / 365, 1.0) \
                 * self.WEIGHTS['time_since_maintenance']

    if is_life_limited and life_remaining_percent and life_remaining_percent < 20:
        score += (1.0 - life_remaining_percent / 20) * self.WEIGHTS['life_limited']

    return round(score * 100, 1)
```

Визначення рівня критичності за балом:

```
def determine_criticality(self, score: float) -> CriticalityLevel:
    if score >= 70: return CriticalityLevel.CRITICAL
    elif score >= 50: return CriticalityLevel.MAJOR
```

```
    elif score >= 30: return CriticalityLevel.MINOR
    return CriticalityLevel.COSMETIC
```

Перевірка періодичних форм ТО (A/C/D-Check):

```
def _check_periodic_maintenance(self, aircraft: Aircraft) -> List[DataGap]:
    gaps = []
    checks = [
        (MaintenanceType.A_CHECK, "A-Check"),
        (MaintenanceType.C_CHECK, "C-Check"),
        (MaintenanceType.D_CHECK, "D-Check")
    ]
    for maint_type, check_name in checks:
        last_record = self.db.query(MaintenanceRecord).filter(
            MaintenanceRecord.aircraft_id == aircraft.id,
            MaintenanceRecord.maintenance_type == maint_type
        ).order_by(MaintenanceRecord.performed_date.desc()).first()

        if not last_record:
            gap = self._create_gap(
                aircraft_id=aircraft.id,
                gap_type=f"missing_{maint_type.value}",
                maintenance_type=maint_type,
                description=f"Відсутні записи про {check_name}",
                recommended_action=f"Знайти документацію {check_name}"
            )
            gaps.append(gap)
    return gaps
```

```
def _calculate_life_remaining(self, component) -> Optional[float]:
    if not component.is_life_limited:
        return None

    remaining = []
    if component.life_limit_hours and component.hours_since_new:
        remaining.append((component.life_limit_hours - component.hours_since_new)
                         / component.life_limit_hours * 100)
    if component.life_limit_cycles and component.cycles_since_new:
        remaining.append((component.life_limit_cycles - component.cycles_since_new)
                         / component.life_limit_cycles * 100)

    return min(remaining) if remaining else None
```

Формування рейтингу пріоритетів флоту:

```
def get_fleet_priority_ranking(self) -> List[Dict[str, Any]]:
    ranking = []

    for aircraft in self.db.query(Aircraft).all():
        gaps = self.db.query(DataGap).filter(
            DataGap.aircraft_id == aircraft.id,
            DataGap.status == GapStatus.OPEN
        ).all()

        critical = sum(1 for g in gaps if g.criticality == CriticalityLevel.CRITICAL)
        major = sum(1 for g in gaps if g.criticality == CriticalityLevel.MAJOR)
```

```
    minor = sum(1 for g in gaps if g.criticality == CriticalityLevel.MINOR)

    recovery_score = max(0, 100 - (critical * 10 + major * 5 + minor * 2))

    ranking.append({
        "registration": aircraft.registration,
        "model": f"{aircraft.manufacturer} {aircraft.model}",
        "critical_gaps": critical,
        "major_gaps": major,
        "minor_gaps": minor,
        "recovery_score": recovery_score
    })

return sorted(ranking, key=lambda x: x["recovery_score"])
```