МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ Олександр ЛИТВИНЕНКО
"_____"_____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Програмний модуль для аналізу даних польотів малої авіації

Виконавець: Олександр СЛЮСАРЕНКО

Керівник: Олена НЕЧИПОРУК

Нормоконтролер: Євгеній ТУПОТА

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій

Кафедра комп'ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія

Освітньо-професійна програма «Системне програмування»

Форма навчання денна

ЗАТВЕРДЖУЮ

Завідувач кафедри

Олександр ЛИТВИНЕНКО

«_____»_____2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Слюсаренка Олександра Костянтиновича

1. Тема кваліфікаційної роботи Програмний модуль для аналізу даних польотів малої авіації

затверджена наказом ректора від «28» 08 2023 р. № 1494/ст

2. Термін виконання роботи (проєкту): з 02.10.2023 р. по 31.12.2023 р.

3. Вихідні дані до роботи (проєкту): програмна документація, ДСТУ, мова програмування *Python*, середовище розробки *PyCharm*

4. Зміст пояснювальної записки: вступ, аналіз предметної області, вимоги та проєктування програмного модуля, реалізація програмного модуля.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) структурна схема програмного модуля;

2) діаграма послідовності *UML*;

3) діаграма класів *UML*;

4) схема алгоритму обробки даних;

5) схема алгоритму аналізу *PWA*.

6. Календарний план

№ п/п	Етапи виконання кваліфікаційної роботи	Термін виконання етапів	Примітка
1	Пошук та аналіз джерел для аналізу предметної області за темою кваліфікаційної роботи	02.10-21.10	
2	Розробка плану кваліфікаційної роботи	22.10-02.10	
3	Розробка розділу 1: Аналіз предметної області	03.11-13.11	
4	Розробка розділу 2: Вимоги та проєктування програмного модуля	14.11-27.11	
5	Розробка розділу 3: Реалізація програмного модуля	28.11-08.12	
6	Оформлення пояснювальної записки та написання висновків	11.12–18.12	
7	Розробка презентації для захисту роботи	19.12–24.12	
8	Підговка до захисту	25.12–31.12	

7. Дата отримання завдання: «02» 10 2023 р.

Керівник кваліфікаційної роботи _____ Олена НЕЧИПОРУК

Завдання прийняв до виконання _____ Олександр СЛЮСАРЕНКО

РЕФЕРАТ

Кваліфікаційна робота на тему «Програмний модуль для аналізу даних польотів малої авіації». Записка до кваліфікаційної роботи містить: 83 с., 10 рис., 2 табл., 25 літературних джерел, 1 додаток.

Ключові слова: БПЛА, ЛОГ-ФАЙЛ, *GPS*, *API*, *MAVLink*, ІНСТРУМЕНТ АНАЛІЗУ.

Об’єктом дослідження є аналіз даних польотної інформації.

Предметом дослідження є методи аналізу даних польотної інформації.

Метою кваліфікаційної роботи є розробка програмного модуля для аналізу польотів малої авіації.

Технічними та програмними засобами що використовувалися при розробці кваліфікаційної роботи є персональний комп’ютер з операційною системою *macOS* та середовище розробки *PyCharm*.

Для створення програмного модулю використовувався об’єктно орієнтований метод програмування, алгоритми побудовані відповідно міжнародним і державним стандартам. Основними характеристиками та показниками в роботі є зручність інтерфейсу, швидкість та функціональність розробленого програмного забезпечення.

В свою чергу науковою новизною полягає в інтеграції передових технологій та методик обробки даних, що в сумі забезпечить більшу точність та зручність у роботі з вихідними даними польотів малої авіації.

Головною рекомендацією використання програмного модуля є доступ користувачів без глибоких знань у авіаційній галузі для аналізу польотних даних малої авіації через веб-браузер.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

СОП – документ управління якістю, набір інструкцій з конкретного аспекту роботи.

БПЛА – літальний апарат, який може злітати, здійснювати політ і сідати без фізичної присутності пілота на його борту.

Лог-файл – файл із записами про події в хронологічному порядку.

GPS – радіоелектронних засіб, для визначати положення та швидкість руху об’єкта.

API – це набір чітко визначених методів для взаємодії різних компонентів.

MAVLink – протокол для комунікації безпілотного апарату із наземною станцією.

SDK – набір із засобів розробки, утиліт і документації.

СУБД – набір взаємопов’язаних даних і програм для доступу до цих даних.

PWM – процес керування шириною високочастотних імпульсів.

JSON – це текстовий формат обміну даними між комп’ютерами.

DOM – програмний інтерфейс для *HTML* і *XML* документів.

ВСТУП

У сучасному світі, де роль малої авіації та, зокрема, безпілотних літальних апаратів стрімко зростає розвиток технологій відбувається надзвичайно швидко, проте з цим виникають нові виклики, проблеми та завдання, серед яких ключове місце займає аналіз польотних даних. Проблема аналізу є актуальною з огляду на зростаючу кількість операторів дронів, складності маршрутів та великої різноманітності сценаріїв використання безпілотних літальних апаратів.

За останнє десятиліття суспільство зіткнулося з накопиченням цифрової інформації, зокрема в сфері авіації. Через це виникають потреба розробки ефективних інструментів для аналізу та обробки великих обсягів даних. Наукове завдання полягає у створенні програмного модуля, який би дозволяв ефективно обробляти, аналізувати та візуалізувати польотні дані, забезпечуючи краще розуміння та контроль над станом безпілотних літальних апаратів.

З розвитком малої авіації та безпілотних літальних апаратів виникла потреба у детальному аналізі польотних даних, необхідність пов’язана з підвищенням безпеки польотів, ефективності використання авіатехніки та оптимізації її експлуатаційних характеристик. Аналіз даних дозволяє глибше зрозуміти поведінку літальних апаратів в різних умовах, виявляти потенційні небезпеки та вносити корективи в процеси планування та виконання польотів.

Проблематика розробки програмного модуля для аналізу даних польотів малої авіації тісно пов’язана з сучасним станом розвитку цієї галузі, зокрема з активним використанням безпілотних літальних апаратів.

Однією з ключових проблем є складність інтерфейсів існуючих аналітичних систем, через виникають бар’єри для операторів БПЛА та інших фахівців, які могли б використовувати ці системи для контролю та управління польотами але не мають навичок або знань аналізу.

Крім того, багато існуючих рішень мають обмежий функціонал, бо вони часто проектуються з огляду на конкретні типи літальних апаратів або специфічні сценарії

використання, що робить їх менш ефективними у випадках, коли потрібно адаптувати систему під різноманітні потреби користувачів.

У контексті динаміки розвитку науки, техніки та суспільства, важливо підкреслити, що розробка програмного модуля відповідає сучасним вимогам безпеки, ефективності та інноваційності. Впроваджений модуль стає відповіддю на недостатність опрацювання цієї проблеми на сучасному етапі та вимагає дослідження проблеми з використанням нових інструментів для аналізу польотів безпілотних літальних апаратів.

Виходячи з цього, стає зрозуміло, що потреба в розробці нових, більш гнучких і інтуїтивно зрозумілих інструментів для аналізу польотних даних малої авіації є актуальною і важливою. Такий інструмент дозволить ефективно реагувати на можливі проблеми, відхилення у польотних даних та забезпечить вищий рівень безпеки та ефективності польотів.

Розробка програмного модуля відповідає на ці виклики, пропонуючи рішення, яке не тільки покращує процес аналізу даних, але й відкриває нові можливості для досліджень користувачів у цій галузі.

Об’єктом дослідження є аналіз даних польотної інформації.

Предмет є методи аналізу даних польотної інформації

Вивчення цих аспектів відіграє вирішальну роль у забезпеченні безпеки та ефективності польотів.

Мета дослідження – розробити програмний модуль для аналізу даних польотів малої авіації. Цей модуль передбачає впровадження зручного, інтуїтивно зрозумілого та ефективного інструменту без необхідності встановлення додатку чи мов програмування для аналізу польотних даних, з акцентом на користувацький досвід і точність обробки даних.

Наукова новизна отриманих результатів. Динамічний розвиток малої авіації та збільшення використання безпілотних літальних апаратів вимагає нових підходів до аналізу та оптимізації польотних даних. Запропонований програмний модуль включає нові методи та інструменти, що враховують унікальні вимоги та особливості БПЛА. Оскільки сучасні інструменти мають вимоги по операційних

систем та встановлених мов програмування програмний модуль надає можливість операторам БПЛА отримувати детальний та точний аналіз польотних даних незалежно в веб-браузері незалежно від операційної системи. Програмний модуль дозволяє адаптувати інструменти аналізу під індивідуальні сценарії, строювати нові більш ефективних стратегій польоту.

Практичне значення отриманих результатів. Програмний модуль сприяє оптимізації процесів планування та аналізу польотних місій малої авіації. Даний модуль може використовуватися в комерційних цілях та навчальних закладах для демонстрації та вивчення аспектів польотної аналітики.

Вступ відображає основні цілі та завдання кваліфікаційної роботи, а також підкреслює важливість і актуальність теми. Кваліфікаційна робота складається з наступних основних розділів:

1) аналіз предметної області:
   - розглядається основні поняття в аналізі даних польотів;
   - аналізується особливості аналізу польотів малої авіації;
   - виконання порівняльного аналізу існуючих інструментів для аналізу даних польотів безпілотних літальних апаратів;
   - визначення та постановка задачі.
2) вимоги та проєктування програмного модуля:
   - проведення аналізу та вибір технологій для розробки програмного модуля;
   - визначення функціональних та нефункціональних вимог;
   - розробка архітектури програмного модуля.
3) реалізація програмного модуля:
   - розробки серверної частини з скаладанням схем алгоритмів;
   - розробка клієнтської частини;
   - тестування та валідація програмного модуля.

Кожен розділ завершується висновками, що підкреслюють основні результати та наступні кроки в процесі розробки програмного модуля.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Основні поняття в аналізі даних польотів

1.1.1. Огляд та мета аналізу даних польотів

Аналіз даних польотів – це процес збору, аналізу та інтерпретації даних з бортових реєстраторів літака з метою вдосконалення безпеки та ефективності використання. Аналіз даних використовується для виявлення ризиків, навчанню пілотів, документації та оптимізації польотів.

Основна мета аналізу даних польотів малої авіації є підвищення безпеки польотів. За допомогою аналізу виявляється небезпечні практики і процедури, а також механічні проблеми з повітряними судами. Виявляючи і вирішуючи ці проблеми, авіакомпанії та інші авіаційні організації знизжуть ризик аварій та інцидентів [6].

За допомогою аналізу даних польотів компанії аналізують такі події, як жорсткі посадки, надмірний злітний розворот, експлуатація під час аварійних ситуацій, шляхом завантаження та інтерпретації записаних даних. Для аналіз доповнюється побудовою графіків обов’язкових параметрів, а згодом і додаткових, що надає корисну інформацію для розслідування відділу авіаційної безпеки. Крім цього можна використовувати ці графікі під час тренувальних завдань або дебрифінгу з екіпажем.

Якщо не аналізувати такі параметри, то виявити тенденцію можна лише після проведення судової експертизи, тобто коли літак вже зазнав пошкоджень. Через це на сьогоднішній день багато організацій почали запроваджувати предиктивний аналізу з метою фіксацій аномальних тенденцій в експлуатації літальних апаратів. Наприклад, збільшення швидкості на зльоті призводить до тривоги і завдяки чому є можливість запобігти інциденту із занесенням хвостової частини літака. Багато

інших прикладів пов’язані з технікою польоту, які також виявляються з допомогою статистичних методів, які вивчають дані польоту на макрорівні, а не на деталях польоту. Таким чином, залежно від правил з якими акредитований експлуатант, вкрай важливо, щоб аналіз проводився в рамках офіційної програми.

Аналіз даних польоту допомогає поліпшити продуктивність компаній за рахунок зниження витрат, збільшення доходів та підвищення безпеки.

Аналіз польотів в авіації зменшує витрати визначаючи і вирішуючи питання пов’язані з ефективність. Наприклад, аналіз використовують для виявлення рейсів, де використовується пального більше ніж потрібно, або час рейсу більше ніж запланований [7].

Також, збільшення доходів для компаній реалізується за рахунок підвищення задоволеності клієнтів. Наприклад, аналіз можна використовувати для виявлення часто затримуваних або скасованих рейсів, завдяки чому авіакомпанії своєчасно вирішуватимуть проблеми та покращувати рівень якості обслуговування [5].

Так як основна мета аналізу польотів авіації є підвищення безпеки, то виявляючи та усуваючи небезпечні практики і процедури, та механічні проблеми авіакомпанії або інші авіаційні організації знизують ризик аварій або небезпечних інцидентів. Без цього компанії отримують значні економічні витрати, що пов’язані з аварійними інцидентами або ремонтом.

Також, аналіз польотних даних використовується для розслідування нещасних випадків та інцидентів. Аналізуючи літальних апарат з бортового самописця, комісія визначає причину аварії або інциденту і вжити заходи, для того усунення ситуації в майбутньому [24].

Типові проблеми якості, що впливають на параметри польоту включають:

- стрибки – коли виникають раптові зміни параметрів польоту, які наближаються до аномальних значень;
- несправність – коли польотний параметр не реєструється протягом польоту через несправність в системі;
- замороження – коли у певний момент польоту параметр перестає змінюватися і більше не реєструється належним чином;

− зсув – коли виникає постійна різниця між значенням параметру польоту і фактичним фізичним значенням. Воно виникає коли параметр польоту потребує калібрування.

Загалом, аналіз польотних даних авіації є потужним інструментом, який використовують в авіаційній галузі не тільки для підвищення безпеки, а також для ефективності та зменшенню витрат для компаній, що його використовують [11].

Беручи до уваги параметри польоту, що використовуються в більшості програмного забезпечення для аналізу польотів, вони повинні бути точно записані та декодовані. Наприклад, це правильне визначення знаків або визначення одиниць.

1.1.2. Огляд алгоритмів аналізу польотів

При аналіз польотів авіації використовуються такі алгоритми, як:

1) широкі алгоритми – алгоритми, що розроблені для допомогти виявити небезпечних подій, такі як ризик *CFIT* або вихід за межі злітно-посадкової смуги, коли під час посадки літак дуже швидко наближається до кінця злітно-посадкової смуги, тобто його наземна швидкість здається занадто високою для відстані, що залишилася;
2) спеціалізовані алгоритми – алгоритми які розроблені для виявлення конкретних відхилень від стандартних операційних процедур або прийнятих практик, навіть коли вони не обов’язково призводять до небезпечного результату. Спеціалізовані алгоритми авіаційного аналізу польоту є корисними для відстеження конкретної проблеми, наприклад, пов’язаної з технікою пілотування або виконанням певного СОП. Так як сфера використання спеціалізованих алгоритмів вузька, через що непередбачувані проблеми безпеки залишитися непоміченими, якщо програмне забезпечення використовує тільки їх.

Широкі алгоритми охоплюють всі події, пов’язані з конкретним ризиком для безпеки. Вони корисні для початкової оцінки проблем безпеки.

Так як, широкий алгоритм є досить неспецифічним, вихідні дані, згенеровані таким алгоритмом, необхідно проаналізувати, щоб розділити їх на менші категорії. Тому наведений алгоритм не дуже підходить для моніторингу конкретних питань безпеки, в загальному він є відправною точкою для впровадження аналізу [9].

При розробці програмного забезпечення для аналізу польотів авіації впроваджують спочатку широкі алгоритми, що охоплюють всі етапи польоту та основні авіаційні події, і після цього поступово доповнюють набір спеціалізованими алгоритмами, які аналізують та рахують більш специфічні ризики експлуатаційної безпеки.

При роботі з великими обсягами даних методи статистичного аналізу допомагають виділити корисну інформацію, тому використання відповідних методів статистичного аналізу в програмному забезпеченні для аналізу польотів важливо для визначення тенденцій і нових областей операційного ризику. Це також включає використання методів візуалізації даних для кращого розуміння складних взаємозв’язків між різними факторами. Це допомагає авіакомпаніям і регулюючим органам краще реагувати на виклики безпеки і оптимізувати процедури для зниження ризиків [10].

1.2. Особливості аналізу польотів безпілотних літальних апаратів

1.2.1. Визначення безпілотного літального апарату

Безпілотні літальні апарати – це літальні апарати, які злітають, здійснювати політ і сідати без фізичної присутності пілота на його борту. Безпілотні літальні апарати на сьогоднішній день все частіше використовуються в агрономії, інспекції інфраструктури, доставок посилок, військових операцій, рятувальних операції тощо.

Через те, що з кожним роком все більше дронів використовується у громадському повітряному просторів зростає занепокоєння щодо безпеки його використання [11].

Під час польотної місії безпілотного літального апарату на його фізичну стійкість впливають такі фактори, як вразливість перевірки вхідних даних у програмі контролера, несправність датчиків через що навколишні перешкоди неправильно інтерпретуються, або кібератака, яка навмисно створює шуми, щоб збити системи дрони з пантелику, через що це призводить до падіння на громадську територію або створення небезпечної ситуації [23].

До прикладу, в програмах керування *ArduPilot*, *PX4* та інших є багато помилок валідації вхідних даних, через що гіроскопічні датчики безпілотних літальних апаратів легко виводяться з ладу шляхом створення шкідливих звукових шумів з резонансними частотами. Тому, проведення польотних місій над населеними пунктами ризики для населення. Через що, виявлення аномалій у фізичному стані БПЛА під час виконання польотних місій або після них є вкрай важливим [13].

Процеси управління безпілотного літального апарату складається з наземної станції управління, пульта дистанційного керування, комп'ютера-компаньйона (це пристрій, який знаходиться на борту транспортного засобу і контролює та комунікує з автопілотом через канал зв'язку з низькою затримкою), польотного контролера, датчиків та двигуна (рис. 1.1).

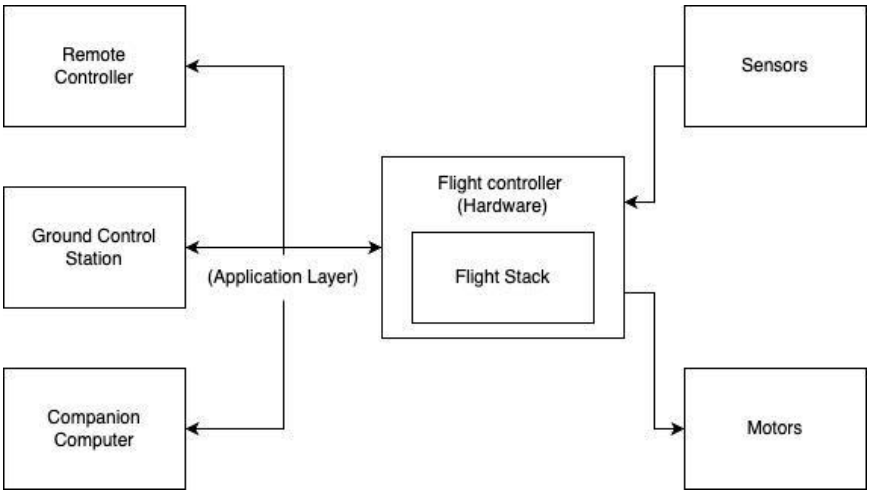

Рис. 1.1. Процеси упавління безпілотного літального апарату

Польотний стек (програмна частина польотного контролера) – це те, що керує рухом і операціями БПЛА , для цього воно використовує інформацію з датчиків, що

відображають фізичний стан БПЛА , до приклади такі як: системи глобального позиціонування (*GPS*), акселерометр, гіроскоп, термометр, і відповідно керуючи двигунами на дроні. Більшість польотних стеків з відкритим вихідним кодом надають *API*, які підтримують протокол *MAVLink*, який де-факто на сьогодні є стандартом зв'язку дронів на прикладному рівні. Протокол використовується дистанційними контролерами, наземними станціями управління та комп'ютерами-компаньйонами.

Комп'ютер-супутник – це, як правило, легкий комп'ютер, наприклад, *Raspberry PI*, всередині якого спеціально написаний бізнес-додаток викликає високорівневі *SDK*, такі як *MAVSDK*, який використовує протокол *MAVLink* для зв'язку з контролером польоту для досягнення своїх бізнес-цілей, таких як місії з пошуку маршрутних точок, виявлення об'єктів і структурних оглядів [16].

1.2.2. Використання лог-файлу для запису бортової інформації

Лог-файл – спеціальний файл, у якому накопичується зібрана службова та статистична інформація про події в системі або програмі. Операційні системи та серверне програмне забезпечення зазвичай мають розвинуту систему ведення логів, які записують всі дії в операційній системі, програмних додатках і пристроях, такі як повідомлення, звіти про помилки, запити до файлів, передача файлів і запити на вхід/вихід, фактично усі події автоматично документується в лог-файлах.

З допомогою аналіз лог-файлів знаходяться потенційні вразливості, такі як порушення безпеки або неминучий вихід з ладу обладнання, надаючи інформацію про продуктивність системи [15].

Програми, мережі, пристрої та операційні системи часто створюють лог-файли у спеціалізованих програмах, які представляються у вигляді часових рядів та активностей.

Для аналізу логів у реальному часі їх зазвичай зберігають у файлі, базі даних або в спеціалізованому додатку, який називається збирачем логів [18].

Через величезні обсяги даних, що генеруються в сучасному цифровому світі, для *IT*-фахівців стало неможливим вручну відстежувати параметри систем та аналізувати лог-файли. Через це складні системи управління лог-файлами стали розвиватися, а також стратегії, які автоматизують основні операції зі збору, форматування та аналізу даних [8].

До методів, що використовують на сьогоднішній день по системі управління та аналізу лог-файлами включають:

1) штучний інтелект – завдяки цьому методу сучасні системи аналізу логів автоматично розпізнають, видаляти або ігнорувати дані логів, які не допомагають виявити проблеми або порушення безпеки, також називають «штучним ігноруванням», дозволяє аналізу логів видавати тривоги, коли заплановані рутинні події не відбуваються тоді, коли вони повинні [12];
2) кореляція – програмні системи об’єднують лог-файли з різних джерел, для чіткого розшифрування події. Це особливо при кібератаках, коли кореляція журналів мережевих пристроїв, серверів, брандмауерів і систем зберігання даних надає можливості виявляти дані, пов’язані з атакою, і закономірності, які важко замітити в ізольованому файлі;
3) нормалізація – програмні системи аналізу перетворює різноманітні дані елементів лог-файлі в стандартний формат. Це допомагає створення централізованого зберігання та індексування даних незалежно від джерела журналу [14];
4) розпізнавання шаблонів – сучасні технології машинного навчання (*ML*) можна використовувати для пошуку шаблонів у даних журналів, які вказувати на аномалії, наприклад, порівняння повідомлень, прихованих у зовнішньому списку, щоб побачити, чи є загроза, прихована в шаблоні. Це допомагає відфільтрувати звичайні записи журналу, дозволяючи зосередити аналіз на виявленні проблем;
5) структурування – цей метод забезпечує форматування файлів таким чином, щоб дані могли інтерпретувати як люди, так і машини. Завдяки цьому значна частина важкої роботи виконується автоматично;

6) гегування та класифікація – цей метод допомогає позначити ключовими словами і класифікувати за типами файли, для тощо, можливості застосовування фільтрів, що прискорює пошук важливої інформації [12].

Аналіз лог файлів надає наступні переваги:

1) відповідність – багато урядових або регуляторних органів вимагають доказів відповідності безлічі нормативних актів, які впливають практично на кожен бізнес. Аналіз лог-файлів надає можливість компаніям підтвердити про дотримання вимог *HIPAA, PCI, GDPR* та інших нормативних актів;
2) ефективність – система аналізу логів допомагає компанії стати більш ефективною, бо програмна система використовує єдине сховище лог-файлів, при аналізі якиї помилки або інші проблеми системі знаходяться швидко, що дозволяє ефективно виправити помилки;
3) висока доступність – своєчасні дії, засновані на інформації з аналізу допомагає запобігти переростанню проблеми в вразливість системи;
4) підвищення безпеки – оскільки кіберзлочинність стає все більш структурованою, зростає і попит на більш ефективний захист. Якщо стався злом або втрата даних, аналіз журналу подій надає важливі інструменти для прийняття превентивних заходів, а також для проведення судового аналізу після факту. Аналіз журналів можна використовувати для виявлення спроб незаконного доступу та перевірки правильності налаштування операцій безпеки та брандмауерів;
5) допомога в пошуку вразливостей системи та можливості системи працювати під піковим навантаженням.

Завдання програмної системи аналізу логів полягає в тому, щоб допомогти проаналізувати широкий спектр даних і повідомлень в контексті, що потребує нормалізації даних логів [17].

У більшості випадків лог-файли передаються програмній системі з метою очищення, форматування або нормалізації, та отримання експертного аналізу, щоб знайти тенденції або виявити аномалії, такі як кібератака або витік даних.

Аналіз лог-файлів виконується наступним чином:

1) збір даних – це етап, на якому система збирає дані з апаратних і програмних датчиків;
2) індексація даних – це етап, на якому дані з усіх джерел централізуються та індексуються для швидкого пошуку алгоритмам аналізу;
3) аналіз – це етап нормалізація, розпізнавання шаблонів, кореляція і тегування;
4) моніторинг – на цьому етапі при виявленні аномалій програмна система аналізу логів надсилає сповіщення в режимі реального часу;
5) звітність – на цьому етапі програмна система аналізу логів генерує звіт по виконаному аналізу.

Безпілотний літальний апарат це сучасний та надійні пристрої, але, як і будь-яка машина виходять з ладу, якщо одна з їхніх систем перестане працювати як потрібно. Коли система є життєво важливою для роботи дрона, наприклад, система двигуна, то польотна місія закінчується розбитим літальним апаратом [25].

Більшість дронів, особливо промислових реєструють польотні дані під час польоту. Ці дані зазвичай включають статус польоту, показання датчиків та іншу інформацію, таку як значення параметрів конфігурації.

1.2.3. Структура лог-файлу

Кожна компанія по виготовленню безпілотних літальних апаратів використовують різні формати лог-файлів (наприклад: *DAT, TXT, CSV*, тощо) та структуру даних для запису даних [3].

Структуру даних лог-файлів компанії *DJI* складається заголовку «*Packet Type*», що вказує на тип пакету. В даному лог-файлі пакет буває 8 типів: *GPS*, двигун, точка відправлення, пульт дистанційного керування, місцезнаходження планшета, батарея, положення і статус польоту.

Пакет «Двигун» вказує на швидкість і навантаження на двигуни та інші характеристики, пакет «точка відправлення» відображають координати початкової точки дрона в момент запуску.

Пакет «Пульт дистанційного керування» вказує на стан пульта дистанційного керування, наприклад, дроселя, керма та елеватора.

Пакет «Місцезнаходження планшета» відображає широту і довготу планшета під час місії «За мною», тобто коли безпілотний літальний апарат слідкує за планшетом.

Пакет «Батарея» показує стан акумулятора, наприклад такі характеристики як ємність, температуру, струм і напруга.

Пакет «Положення» вказує значення кута нахилу, крену та рискання (коли безпілотний літальний апарат розвертається навколо своєї вертикальної осі).

Пакет «*GPS*» відображає місцезнаходження БПЛА за *GPS*, висоту, 3-осьове прискорення, гіроскоп та інші показники датчиків.

Пакет «Статус польоту» відображає режим польоту, наприклад, автопілот, повернення додому, зависання тощо. Та інші параметри конфігурації, такі як коефіцієнт підсилення, маневреність, максимальна швидкість тощо, а також час польоту в мілісекундах з моменту початку польоту [19].

Лог-файли в безпілотних літальних апаратах зберігаються автопілотами безпосередньо в бортову пам’ять, незалежно від наявності телеметричного з’єднання. Системи дронів здатні записувати дані зі значно більшою швидкістю, ніж телеметричні журнали, які відображають в додатку по керуванню безпілотних літальних апаратів.

По суті, лог-файл – це чорний ящик дрона. Інформація, що міститься в ньому відрізняється залежно від контролера польоту і типу літального апарату, але журнал даних незалежно від виробника містить високочастотні дані всіх основних систем дрона.

Всі регістри записуються у вигляді повідомлень, кожне з яких має відповідну мітку часу та ідентифікаційний код - *RCOU* ідентифікує *RC*-вихід, наприклад лог-файл *DJI* (табл 1.1). Таку саме інформацію можна отримати з інших типів дронів, але вона буде в іншому форматі або структурі даних [20].

Таблиця 1.1

Фрагмент лог-файлу *DJI*

velocity_x	**velocity_x**	**velocity_z**	**status_mode**	**rtk_connect_status**
1.49978	1.00E-06	-0.001255	6	1
1.499783	1.00E-06	-0.001428	6	1
1.499787	1.00E-06	-0.001608	6	1
1.499792	1.00E-06	-0.001815	6	1

Значення «*velocity x/y/z*» – відповідає за швидкість двигуна у *м/с*.

Значення «*status mode*» – режим, в якому зараз знаходиться безпілотний літальний апарат, значення 6 з таблиці – це симуляція.

Значення «*rtk connect status*» – це статус з'єднання з *GPS*, це значення знаходиться в діапазоні від 0 до 1.

Згідно з базами даних про інциденти з безпілотниками, причини інцидентів з безпілотниками можна поділити на наступні категорії [4]:

1) людський фактор: оператор втрачає контроль над безпілотником;
2) порушення правил: наприклад, політ над забороненою для польотів зоною або перевищення дозволеної висоти;
3) проблеми зі зв'язком: наприклад, втрата зв'язку між наземною станцією управління та БПЛА;
4) апаратна несправність: наприклад, несправні датчики або перешкоди;
5) проблема з конфігурацією програмного забезпечення: встановлення невірної конфігурації значення параметрів;
6) погані погодні умови: наприклад, екстремальна температура та вітер.

Аналіз лог-файлів польотів вирішує ці проблеми, за винятком людської помилки, так як всі проблеми характеризуються відхиленням від нормального стану.

Важливим є розуміння конкретних умов польоту дрона, бо це має визначне значення при інтерпретації лог-файлу. Різні фактори, включаючи завдання дрона, зовнішні умови, такі як вітер, температура і рельєф місцевості впливають на продуктивність дрона, що призводить до очікуваних втрат в ефективності.

Щоб провести аналіз безпілотного літального апарату в ідеальних умовах, як правило, передбачають політ дрону за сіткою. Така стратегія польоту максимізує покриття датчиків, гарантуючи, що зібрані дані мають бажане перекриття, що має вирішальне значення для точного аналізу [21].

Польоти по сітці також підвищують експлуатаційну ефективність дрона: підтримуючи прямолінійні траєкторії протягом більшої частини місії, дрон економить енергію і оптимізує свій маршрут, максимально використовуючи заряд акумулятора.

В ідеальному випадку – це коли дрон, який висить у контрольованому середовищі, наприклад, в ангарі, ізольованому від погодних умов. За таких контрольованих умов можна очікувати, що якщо цей дрон зависатиме на місці на постійній висоті, двигуни повинні реагувати однаково. Іншими словами, вони повинні демонструвати схожі значення вихідних параметрів [22].

Зрозуміло, що ці ідеальні умови рідко збігаються з реальними сценаріями, особливо під час картографічних місій. Під час таких місій дрони працюють в різних умовах навколишнього середовища і рельєфу, що призводить до варіацій в роботі двигунів.

1.3. Порівняльний аналіз існуючих систем аналізу даних польотів

1.3.1. Аналіз інструменту *MAVExplorer*

Проведено аналіз двох основних інструментів для аналізу лог-файлів дронів – це *MAVExplorer* та *Mission Planner*.

MAVExplorer – це інструмент, розроблений для детального аналізу лог-файлів з дронів, що базується на *MAVLink* та *DataFlash*. Його основна мета – це надати користувачам глибокий аналіз даних, отриманих з різних типів дронів.

Основний функціонал *MAVExplorer*:

1) завантаження та аналіз лог-файлів з автопілотів після польоту, що дозволяє глибоко зрозуміти поведінку літального апарату;

2) різноманітні опції для зберігання та аналізу даних, що забезпечують гнучкість у використанні інструменту;
3) можливість генерації автоматизованих звітів для зручності подальшого аналізу;
4) детальний ручний аналіз логів, який дозволяє виявити специфічні проблеми чи тенденції;
5) гнучке налаштування параметрів логування, що дозволяє користувачам налаштовувати запис даних під свої потреби;

З переваг інструменту *MAVExplorer* можна виділити:

- інтерфейс *MAVExplorer* є дуже інтуїтивним, що робить процес аналізу логів зручним та ефективним (рис. 1.2);
- велика гнучкість у створенні графіків: можливості створювати налаштовані графіки з використанням *Python*, та завантажувати графіки інших користувачів;
- також інструмент дозволяє обмежувати графіки на основі заданих умов;
- можливість візуалізувати траєкторію польоту.

До недоліки інструменту *MAVExplorer* можна виділити:

- вимоги до встановлення, бо інструмент потребує встановлення додаткових залежностей, таких як python та інших в залежності від обраних графіків;
- необхідні знання *Python* для повноцінного використання даного інструменту;
- необхідно встановити додаток *MAVProxy* на операційну систему, бо *MAVExplorer* є пакетом цього додатку;
- додаток *MAVProxy* та *MAVExplorer* немає можливості встановити на *MacOS*.

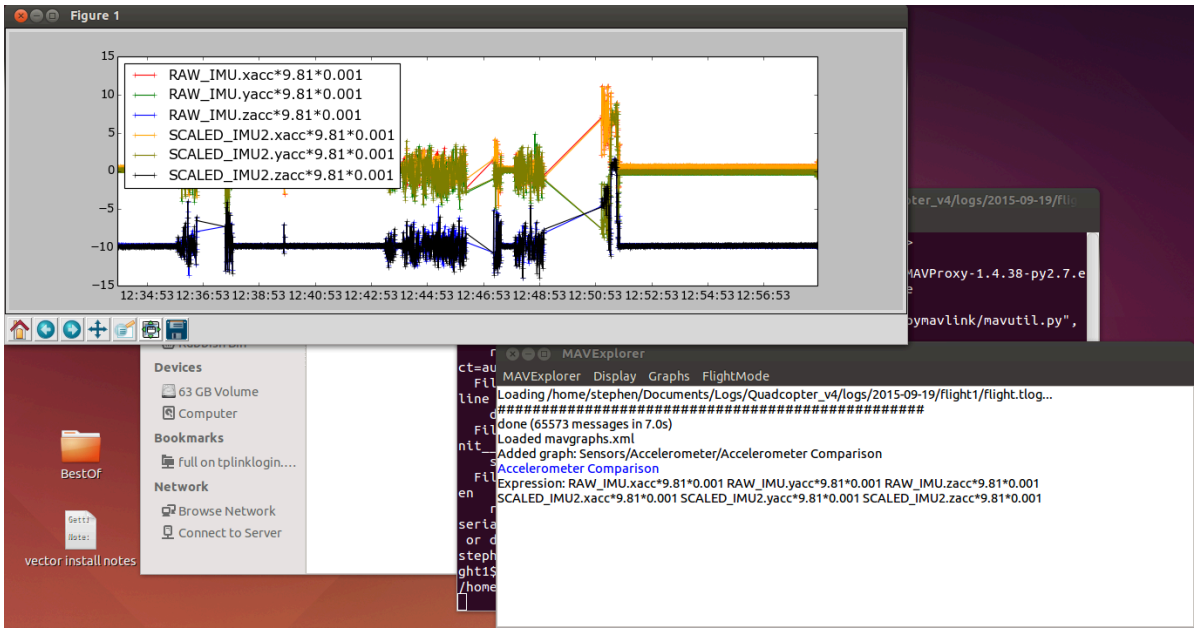

Рис. 1.2. Інтерфейс користувача *MAVExplorer*

MAVExplorer є потужним інструментом з багатими функціональними можливостями для детального аналізу даних. Однак, вимоги до встановлення та висока крива навчання є перешкодою для новачків.

1.3.2. Аналіз інструменту *Mission Planner*

Mission Planner – це інструмент є відкритим рішенням (*opensource*), тобто де кожний користувач має можливість внести свій вклад в розвиток інструменту. Інструмент є частиною проєкту *ArduPilot*, який займається розробкою вільного програмного забезпечення для автономних транспортних засобів (рис. 1.3).

Основний функціонал *Mission Planner*:

1) завантаження та аналіз лог-файлів з автопілотів після польоту;

2) різноманітність опцій для зберігання та аналізу даних;

3) генерація автоматизованих звітів;

4) детальний ручний аналіз логів;

5) гнучке налаштування параметрів логування.

Рис. 1.3. Інтерфейс користувача *Mission Planner*

З переваг інструменту *Mission Planner* можна виділити:

- забезпечує генерування автоматизованих звітів;
- дозволяє глибокий ручний аналіз логів за певними характеристиками;
- підтримка аналізу *FPV* дронів;
- легкість використання для налаштувань та планування місій;
- активна підтримка від спільноти *ArduPilot*.

До недоліки інструменту *Mission Planner* можна виділити:

- незручність у відображенні графіків та повідомлень;
- проблеми з завантаженням карт;
- інструмент сумісний тільки з операційною системою *Windows*.

Обидва інструменти мають свої унікальні переваги та недоліки. *MAVExplorer* є більш гнучким інструментом для користувачів, які мають технічні знання та бажають детального аналізу даних. Інструмент *MAVExplorer* краще підходить для тих, хто шукає гнучкість у відображенні графіків та аналізі даних. *Mission Planner* краще підходить для новачків або користувачів, які не хочуть заглиблюватися в технічні деталі, бо цей інструмент пропонує більш простий та інтуїтивний підхід до

аналізу лог-файлів. Описані інструменти вимагають певного рівня технічної компетенції та знань для використання всього функціоналу.

1.4. Постановка завдання дослідження

На основі детального теоретичного аналізу та ретельного вивчення існуючих програмних рішень у сфері аналізу польотних даних малої авіації, виявлено ряд недоліків та обмежень у цих системах.

Мета дослідження – розробити програмний модуль для аналізу даних польотів малої авіації, створення зручного, інтуїтивно зрозумілого та ефективного інструменту без необхідності встановлення додатку чи мов програмування для аналізу польотних даних, з акцентом на користувацький досвід і точність обробки даних.

Виходячи з мети дослідження та аналізу існуючих рішень визначено наступні завдання:

- проаналізувати та обрати технології для дослідження програмного модуля;
- визначити функціональні та нефункціональні вимоги для досягнення мети дослідження;
- дослідити архітектури програмного модуля, спроєктувати структурну схему, діаграму послідовності та класів;
- розробити схеми алгоритму до серверної частини програмного модуля;
- розробити інтеграцію клієнтської частини з серверної;
- провести тестування програмного модуля.

Впровадження програмного модуля полегшить інтерпретацію даних, адаптування інструментів аналізу під індивідуальні сценарії, стрення більш ефективних стратегій польоту та стане доступними для широкого кола користувачів без необхідності глибоких технічних знань.

1.5. Висновки до розділу

1) Аналіз даних польотів малої авіації відіграє важливу роль у підвищенні безпеки та ефективності використання літаків. Цей процес включає збір, аналіз та інтерпретацію даних із систем літака. Основною метою якого є підвищення безпеки польотів шляхом виявлення ризиків, небезпечних застосувань та механічних проблем.
2) Оскільки популярність безпілотних літальних апаратів зростає, вони стикаються з унікальними проблемами при аналізі польотних даних. Основні фактори, що впливають на стабільність БПЛА включають: вразливості програмного забезпечення, відмову датчиків і кібератаки. Ефективний аналіз даних допомогає виявити та запобігти інцидентам, пов'язаним із фізичним станом безпілотних літальних апаратів.
3) Проведено аналіз структури лог-файлів безпілотних літальних апаратів компанії *DJI*. Особливість цих лог-файлів полягає у їх складності та різноманітності типів даних, що відображають різні аспекти роботи від *GPS* даних та параметрів польоту до стану батареї та системи управління. Ця інформація є критично важливою для забезпечення безпечного та ефективного використання безпілотних літальних апаратів. Визначено, що кожен пакет даних містить унікальну інформацію, необхідну для повного розуміння стану та поведінки дрона під час польоту. Наприклад, дані про швидкість і навантаження на двигуни, інформація про стан пульта дистанційного керування та *GPS*-координати не тільки відображають поточний стан БПЛА, але й допомагають у виявленні та аналізі потенційних проблем. Інформація, що міститься в лог-файлі відрізняється залежно від контролера і типу літального апарату, але журнал даних незалежно від виробника містить високочастотні дані всіх основних систем дрона.
4) Досліджено умови польоту дрона для точної інтерпретації лог-файлів, визначено, що вітер, температура і рельєф місцевості визначають істотний вплив на продуктивність безпілотних літальних апаратів. Тому потрібно

враховувати ці параметри, де очікувані втрати в ефективності є нормою. Для створення ідеальних умов аналізу використовують стратегія польоту по сітці. Метод не тільки гарантує точність аналізу системи, так як ця місія підвищує експлуатаційну ефективність дрона, оптимізуючи використання акумулятора за рахунок підтримки прямолінійних траєкторій польоту.

5) Проведено аналіз існуючих інструментів для аналізу лог-файлів БПЛА – *MAVExplorer* та *Mission Planner*. *MavExplorer* виявився більш гнучким, але використання його є складнішим через необхідність встановлення *Python* та основного додатку для його запуску, тоді як *Mission Planner* став простішим та інтуїтивнішим, але менш гнучким. Також для ефективного використання обох інструментів користувач повинен мати певний рівень технічної компетенції та знань для оптимального використання.

РОЗДІЛ 2
ВИМОГИ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ

2.1. Аналіз та вибір технологій для розробки програмного модуля

2.1.1. Вибір технологій для клієнсткої частини

Вибір технологій для розробки програмного модуля для аналізу даних польотів малої авіації має враховувати сучасні стандарти розробки, вимоги до продуктивності, безпеки та масштабованості. Обрано такі технології як:

– клієнтська частина – *HTML, CSS, JavaScript, React.JS*;
– серверна частина – *Python, RESTful API, HTTPS, OAuth 2.0, JWT, pytest;*
– СУБД – *PostgreSQL*.

Розробка програмного модуля для аналізу даних польотів малої авіації вимагає ретельного вибору технологій, що відповідають сучасним стандартам ефективності, безпеки та масштабованості.

Клієнтська інтеграція веб-додатків за допомогою *HTML*, *CSS*, *JavaScript* та *React.JS* є важливим аспектом розробки сучасних веб-сайтів та веб-додатків. Процес розробки починається з *HTML*, який є структурою будь-якого веб-сайту, він використовує різні теги для відображення інтерактивних елементів, таких як текст, зображення, посилання, кнопки та форми, завдяки тегам створюється структура веб-сторінки.

CSS відіграє важливу роль у визначенні візуального стилю веб-додатку. Завдяки *CSS* можна змінювати кольори, шрифти, розміри елементів, поля та розташування елементів на сторінці. Також *CSS* забезпечує адаптивність дизайну, дозволяючи правильно відображати веб-додаток на різних пристроях.

JavaScript додає інтерактивності та динаміки веб-додатку. Дана мова програмування дозволяє реалізовувати складні функції, такі як обробка подій інтерфейсу користувача, анімація, асинхронне завантаження даних із сервера та інші

динамічні елементи.Це робить веб-додаток більш активним і реактивним. *JavaScript* ефективно взаємодіє з *HTML* і *CSS*, змінюючи *DOM*, щоб динамічно змінювати вміст і стиль сторінки без перезавантаження.

Ввикористання бібліотеки *React.JS* покращує процес розробки веб-додатків, дана бібліотека використовує декларативний підхід, щоб полегшити створення інтерактивного інтерфейсу користувача. Завдяки чому можна змінювати дані без перезавантаження сторінки. *React.js* використовує компонентний підхід, який розбиває інтерфейс на окремі частини багаторазового використання. Завдяки цьому код є впорядкованим і простим для модифікації. Це також полегшує управління станами для веб-додатків, де дані та інтерфейси тісно пов’язані.

Інтеграція цих технологій забезпечує міцну основу для розробки сучасних, ефективних та зручних для користувача веб-додатків. *HTML* і *CSS* визначають основу і стиль, *JavaScript* додає інтерактивності і реагує на зміни, *React.js* оптимізує процес розробки та управління станами. Всі ці технології працюють разом, щоб створити високофункціональний, інтуїтивно зрозумілий і візуально привабливий користувальницький інтерфейс.

2.1.2. Вибір технологій для серверної частини

Для серверної частини програмного модуля обрано мову *Python* версії 3.8. *Python* є високорівневою мовою програмування з дуже гнучною та широкою підтримкою бібліотек. Особливо *Python* підходить для роботи з науковими та аналітичними даними завдяки потужним бібліотекам, таким як *NumPy*, *Pandas*, та *Matplotlib*, що дозволяють ефективно обробляти та візуалізувати великі об’єми даних.

Саме завдяки своїй простоті, широким бібліотекам та потужним можливостям аналізу даних *Python* широко використовується для аналізу польотних даних, підвищення безпеки та покращення експлуатаційних характеристик літальних апаратів.

Python має такі особливості, які вирішують завдання при дослідженні програмного модуля для аналізу польотних даних малої авіації:

1) *Python* чудово справляється з обробкою великої кількості даних, від читання та очищення необроблених даних до виконання складних перетворень та обчислень;
2) зручні можливості візуалізації даних, за допомогою таких бібліотек, як *Matplotlib* та *Seaborn*. Наведенні бібліотеки дозволяють інженерам та аналітикам створювати чіткі та інформативні візуалізації польотних даних;
3) фреймворки машинного навчання *Python*, такі як *scikit-learn* і *TensorFlow*, дозволяють розробляти прогностичні моделі для виявлення несправностей і оптимізації продуктивності;
4) широкі статистичні бібліотеки *Python* підтримують поглиблене дослідження даних та перевірку гіпотез;
5) *Python* легко інтегрується з іншим аерокосмічним програмним забезпеченням та інструментами, підвищуючи ефективність робочого процесу.

Окрім базового аналізу, *Python* застосовують в великій кількості аерокосмічних завдань, такі як:

1) системи на основі *Python* безперервно відстежують стан літака, виявляють аномалії та сповіщають про необхідність технічного обслуговування;
2) *Python* використовується для розробки реалістичних симуляторів польотів для навчання пілотів та тестування літаків;
3) інженери використовують *Python* для оптимізації дизайну літаків, враховуючи такі фактори, як аеродинаміка, вага та паливна ефективність;
4) *Python* відіграє важливу роль в аналізі погодних даних, що є критично важливим для планування польотів та безпеки.

Тому *Python* міцно зарекомендував себе як цінний інструмент в аерокосмічній галузі, особливо в аналізі польотних даних. Його простота, велика кількість бібліотек та активна спільнота сприяють його успіху. Незалежно від того, чи йдеться про забезпечення безпеки польотів, оптимізацію продуктивності або покращення

дизайну літаків, роль *Python* в аерокосмічній галузі продовжує зростати, роблячи небо безпечнішим, а авіацію ефективнішою.

Для роботи з базою даних в програмного модулі обрано систему управління базами даних *PostgreSQL*. Наведенна СУБД забезпечує високу продуктивність, надійність, та гнучкість. *PostgreSQL* підтримує складні запити, транзакції, а також масштабування, що є критично важливим для аналітичних додатків.

Інтеграція *PostgreSQL* в при розробці програмного модуля полягає не тільки у зберіганні даних, а також у їх ефективному управлінні, записі та аналізі. Це включає роботу з різними типами даних, оптимізацію запитів, продуктивності та забезпечення безпеки. Завдяки цьому *PostgreSQL* є міцною основою впровадженню програмного модуля для аналізу польотів малої авіації.

2.1.2. Вибір інструментів інфраструктури програмного модуля

Обрано *PyCharm* як основний редактора коду, оскільки цей редактор має широкі можливості для кастомізації, підтримки різноманітних розширень, та інтеграцію з системами контролю версій. При розробці програмного модуля важливо вибрати ефективні інструменти, що підвищують продуктивність розробників і *PyCharm* повністю їм відповідає.

У контексті розробки програмного модуля для аналізу даних польотів, міжсерверне спілкування відіграє ключову роль. *RESTful API* є оптимальним вибором для цього завдання, оскільки воно забезпечує стандартизований інтерфейс для обміну даними між різними системами. *RESTful API* дозволяє легко інтегрувати різні сервіси та додатки, при цьому забезпечуючи високу гнучкість та масштабованість.

Для аутентифікації та авторизації користувачів обрано сервіс *OAuth 2.0* та *JSON Web Tokens (JWT)*, що забезпечують надійний механізм захисту даних користувачів та контролю доступу. Безпека є критично важливою для будь-якого веб-додатку. Використання *HTTPS* для шифрування даних, що передаються між клієнтом та сервером, є обов’язковим.

Обрано фреймворк для модульного тестування *pytest*, який дозволяє ефективно виявляти та виправляти помилки на ранніх стадіях розробки. Автоматизація тестування є ключовим аспектом для забезпечення якості та надійності програмного забезпечення.

Всі ці технології разом створюють міцну та безпечну серверну архітектуру для веб-додатків. *Python* забезпечує гнучку та потужну платформу для розробки серверної логіки, в той час як *RESTful API* дозволяє цій логіці спілкуватися з клієнтом та іншими сервісами. *HTTPS* та *OAuth 2.0* надають необхідні рівні безпеки, а *JWT* забезпечує надійну аутентифікацію та авторизацію користувачів. Разом, ці технології дозволяють створювати високофункціональні, безпечні та ефективні веб-додатки, які ефективно працюють з великими обсягами даних.

2.2. Визначення функціональних та нефункціональних вимог

Важливим є етапом дослідження програмного модуля для аналізу польотів малої авіації є визначення функціональних і нефункціональних вимог, що дозволяє забезпечити якість та ефективність кінцевого продукту. Особливо це стосується області малої авіації, де рівень відповідальності та вимог до точності є високими.

Розробка програмного модуля починається з чіткого розуміння того, що саме досліджувана система робити та які проблеми вирішує.

Світ дронів та малої авіації є надзвичайно динамічним, із швидкими змінами у технологіях та методах використання. Це вимагає від програмного забезпечення гнучкості та можливості швидкої адаптації до нових вимог. Функціональні та нефункціональні вимоги повинні відображати цю гнучкість, дозволяючи легко вносити зміни та оновлення.

Формування вимог для програмного модуля для аналізу польотів малої авіації, зокрема дронів, ґрунтується на детальному аналізі потреб користувачів та особливостей галузі. Малі авіаційні апарати, такі як дрони, використовуються в широкому спектрі застосувань – від агрономічного моніторингу до фотографії та

зйомки. Кожен із цих сценаріїв має унікальні вимоги до функціональності та продуктивності програмного забезпечення.

На основі проведеного порівняльного аналізу існуючих систем аналізу даних польотів сформовано наступні функціональні вимоги до програмного модуля:

1) збір та обробка польотних даних:
   - можливість завантаження лог-файлів польотів для подальшого аналізу;
   - автоматизована обробка даних для виявлення стандартних показників польоту (швидкість, висота, *GPS* координати тощо).
2) візуалізація даних:
   - використання графіків для наглядного представлення даних польотів.
3) аналітичні інструменти:
   - генерація детальних звітів за кожним польотом;
   - аналіз показників для оцінки якості польоту та виявлення можливих проблем.
4) користувацький інтерфейс:
   - простий та зручний інтерфейс для доступу до функцій аналізу;
   - можливість користувача налаштовувати відображення даних за своїми потребами.
5) безпека та конфіденційність:
   - забезпечення безпеки даних під час передачі та зберігання;
   - механізми аутентифікації та авторизації для доступу до даних.

На основі проведеного порівняльного аналізу існуючих систем аналізу даних польотів сформовано наступні нефункціональні вимоги до програмного модуля:

1) продуктивність:
   - висока швидкість аналізу та візуалізації даних;
   - ефективне використання системних ресурсів.
2) надійність:
   - висока доступність системи та мінімальний час простою;
   - механізми для швидкого відновлення роботи після збоїв.
3) масштабованість:

- здатність системи обробляти зростаючий об’єм даних і користувачів.

4) безпека:

- захист від несанкціонованого доступу та кібератак;
- забезпечення конфіденційності даних за допомогою сучасних методів шифрування.

5) користувацький досвід:

- легкий у навігації та використанні інтерфейс;
- інтерфейс, який підлаштовується під різні типи пристроїв (комп’ютери, мобільні пристрої).

Функціональні вимоги включають збір та обробку польотних даних, візуалізацію даних, аналітичні інструменти, користувацький інтерфейс, а також забезпечення безпеки та конфіденційності даних. Нефункціональні вимоги охоплюють продуктивність, надійність, масштабованість, безпеку, інтероперабельність та користувацький досвід. Врахування цих вимог на етапі дослідження забезпечить створення ефективного, надійного та відповідного програмного рішення для аналізу польотів малої авіації.

2.3. Проектування архітектури програмного модуля

2.3.1. Структурна схема програмного модуля

Архітектура програмного модуля служить каркасом для системи, визначаючи її структуру, взаємодію компонентів та способи інтеграції з зовнішніми сервісами. Ефективна архітектура гарантує не лише стабільність та продуктивність системи, але й її масштабованість та гнучкість в майбутньому.

На основі сформованих функціональних та нефункціональних вимог програмного модуля архітектура програмного модуля розділена на дві ключові частини: серверну та клієнтську.

Серверна частина використовує мову програмування *Python* і відповідає за збір, обробку та аналіз польотних даних. Клієнтська частина використовує мову

програмування *JavaScript* та бібліотеку створення інтерфейсів *React.js* і відповідає за користувацький інтерфейс для візуалізації даних та взаємодії з користувачем.

При побудові архітектури використовуються принципи чистого коду та модульності. Кожен компонент системи розробляється таким чином, щоб бути незалежним, з легкою інтеграцією та заміною. Це забезпечує високу гнучкість системи та спрощує процес розвитку та масштабування.

Продуктивність системи забезпечується завдяки оптимізації запитів до бази даних *PostgreSQL* та ефективному управлінню ресурсами сервера.

Архітектура підтримує горизонтальне масштабування та використання мікросервісів для підвищення ефективності системи.

Мікросервісна архітектура програмного модуля надає можливості реалізувати визначені функціональні та нефункціональні вимоги і також забезпечує стабільність, гнучкість та легкість у розширенні системи. Використання сучасних технологій та методів розробки гарантує високу продуктивність та надійність програмного модуля.

На основі проведеного аналізу функціональних вимог програмного модуля для аналізу даних польотів малої авіації побудовано структурну схему. Структурна схема складається з двох частин – структурна схема серверної частини (рис. 2.1) та структурна схема клієнтської частини (рис. 2.2).

До структурної схеми серверного блоку входять наступні модулі:

1) модуль обробки даних польотів – за допомогою якого програмна система перетворює завантажений лог-файл користувача у формат *JSON*, який є стандартом для аналізу. Цей компонент включає фільтрацію шумів та екстракцію важливих характеристик;
2) модуль аналізу – за допомогою якого сервер взаємодиє з усіма модуля для конкретного аналізу, такі як модуль діагностики вібрацій, діагностики батареї, діагностики *PWM*, діагностики *GPS* та модуль перешкод;
3) модуль діагностики вібрацій – за допомогою якого програмна система створює аналіз на основі даних акселерометра для оцінки рівня вібрацій та їх впливу на роботу БПЛА;

4) модуль діагностики батареї – за допомогою якого програмна система створює аналіз стану батареї з використанням аналізу часових рядів;
5) модуль діагностики *PWM* – за допомогою якого програмна система створює аналіз пілотного контролеру дрона, який регулює швидкість обертання моторів;
6) модуль діагностики *GPS* – за допомогою якого програмна система створює комплексний аналіз точності *GPS*;
7) модуль перешкод – за допомогою якого програмна система створює аналіз шуму і перешкод *GPS*;
8) модуль взаємодії з базою даних – за допомогою якого програмна система встановлює з'єднання з базою даних, керує запитами та оновленнями даних, гарантує консистентність та цілісність даних;
9) модуль генерування звітів – за допомогою якого програмна система створює готовий звіт для користувача на основі проведеного аналізу;

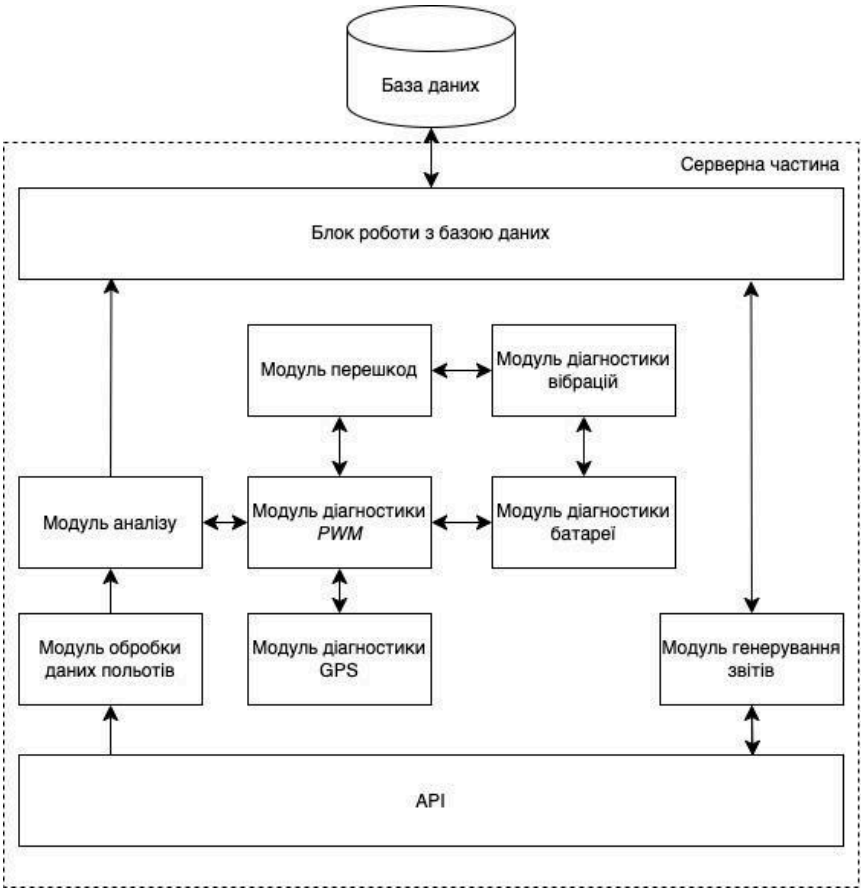

Рис. 2.1. Структурна схема сервеної частини

Наведена структура орієнтована на ефективне оброблення та аналіз даних, забезпечуючи високу якість діагностики та надійність системи. Вона створена для забезпечення гнучкості в аналізі даних, зручності інтеграції з клієнтською частиною, а також для забезпечення високих стандартів безпеки та конфіденційності даних.

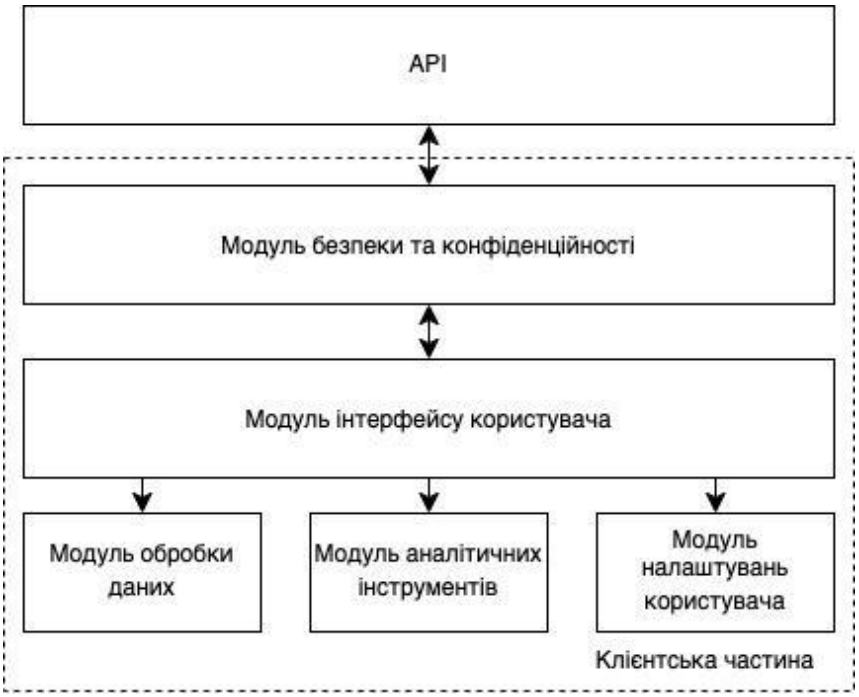

Рис. 2.2. Структурна схема клієнтскої частини

До структурної схеми клієнтського блоку входять наступні компоненти:

1) модуль інтерфейсу користувача – забезпечує користувачам доступ до різноманітних функцій системи через зрозумілий та зручний інтерфейс;
2) модуль обробки даних – здійснює передачу та обробку даних, отриманих від сервера, для подальшого використання в клієнтській частині;
3) модуль аналітичних інструментів – використовується для генерації звітів та детального аналізу даних;
4) модуль налаштувань користувача – дозволяє користувачам кастомізувати відображення та параметри системи;

5) модуль безпеки та конфіденційності – забезпечує захист даних та авторизацію доступу до системи.

Наведена структура орієнтована на забезпечення зручного та ефективного використання системи користувачами, з особливим акцентом на безпеку та гнучкість візуалізації даних.

2.3.2. Діаграма послідовності програмного модуля

На основі проведеного аналізу функціональних вимог програмного модуля для аналізу даних польотів малої авіації побудовано діаграму послідовності. Діаграма послідовності відображає процеси або об’єкти програмної систети та їх взаємодію під час виконання сценаріїв (рис. 2.3).

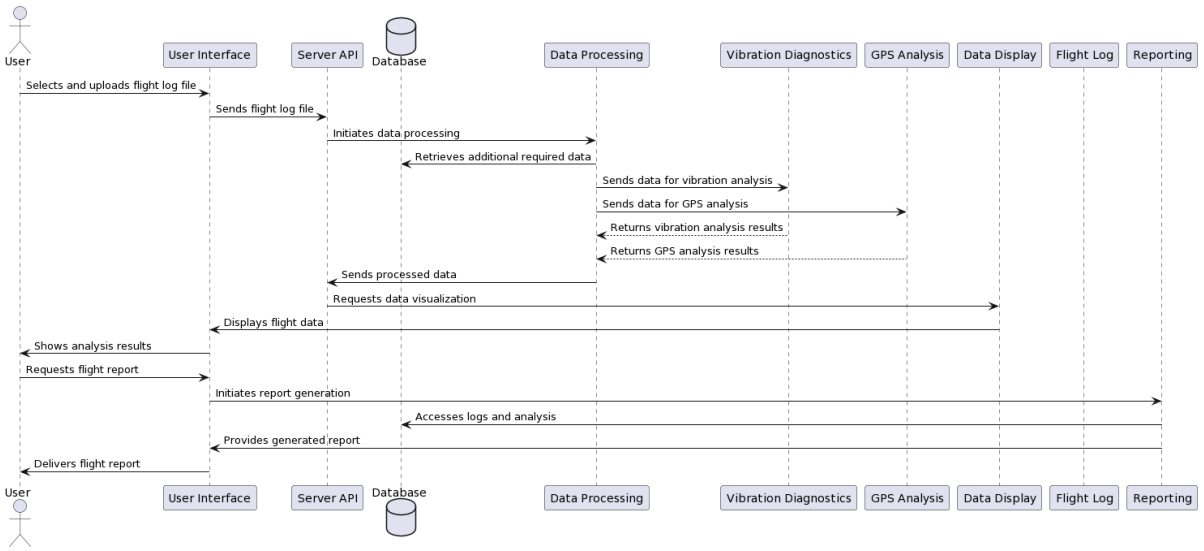

Рис. 2.3. Діаграма послідовності процесу завантаження лог-файлу польоту дрона

Діаграма послідовності складається з наступних етапів:

1) користувач обирає файл – через інтерактивний інтерфейс користувача на клієнтській стороні, користувач обирає лог-файл з даними польотів свого БПЛА .

2) завантаження файлу на сервер – клієнтська частина надсилає запит на сервер через захищене *API*, після чого серверна частина отримує файл і передає його модулю обробки даних;
3) попередня обробка даних – на серверній стороні модуль обробки даних ініціює процес конвертування лог-файлу після чого виконує фільтрації шуму та виявлення ключових необхідних для аналізу даних польоту;
4) аналіз даних польотів – наступним кроком є модуль аналізу даних, який використовує алгоритми для оцінки параметрів польоту, таких як вібрація, швидкість, траєкторія тощо;
5) діагностика вібрацій – модуль аналізу вібрацій використовує дані акселерометра для оцінки впливу вібрацій на літальний апарат;
6) *GPS* аналіз – модуль перевіряє точність даних *GPS*, кількість супутників, та визначає наявність перешкод чи шумів;
7) відображення даних користувачу – серверна частина відправляє оброблені та аналізовані дані на клієнтську частину, де дані візуалізують через модуль відображення результатів клієнтської частини;
8) збереження історії польотів – наступним етапом є збереження результатів аналізу в історії польотів користувача в базі даних для можливості їх перегляду в майбутньому;
9) генерація звіту – наступний і заключний етап, на якому модуль звітності створює звіт, що включає всі ключові показники та аналітичні дані, який можна завантажити або експортувати.

2.3.3. Діаграма класів програмного модуля

На основі описаної архітектури та визначених функціональних вимог побудовано діаграму класів (рис. 2.4) програмного модуля, яка має на меті ілюструвати структуру та взаємодію різних компонентів системи. Наведена діаграма є необхідною для розуміння внутрішньої архітектури програмного модуля та відіграє важливу роль у проектуванні та реалізації системи.

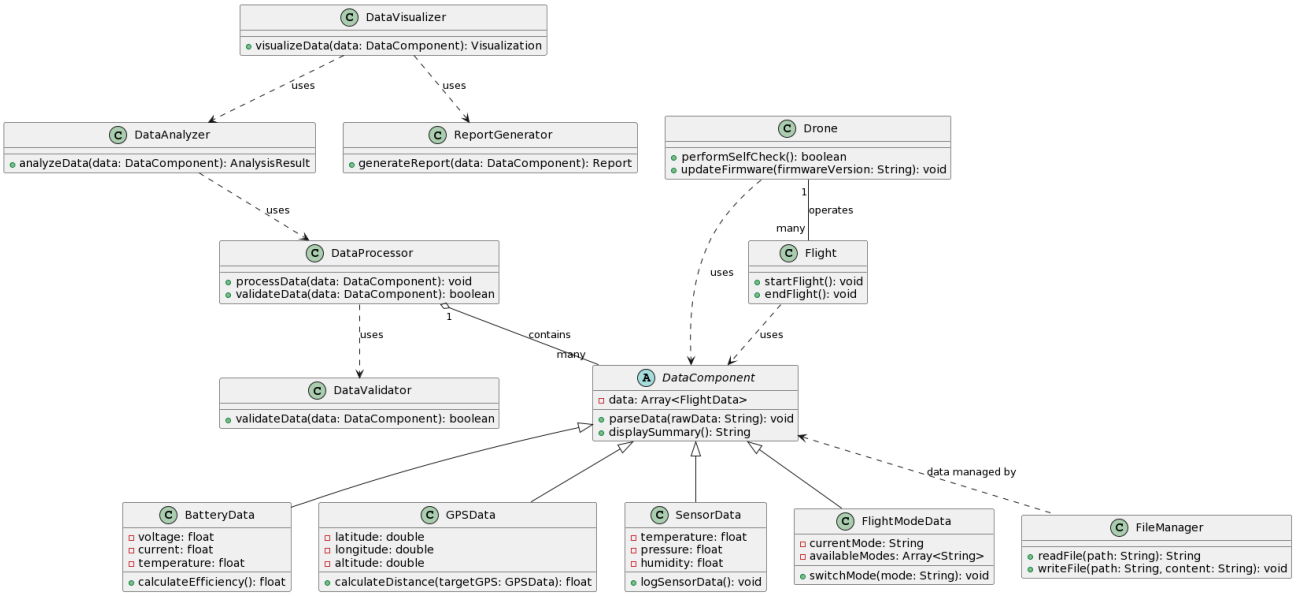

Рис. 2.4. Діаграма класів програмного модуля

Наведена діаграма класів об’єднує класи у загальну ієрархію для репрезентації програмного модуля, що включає взаємодію між компонентами для збору, обробки, аналізу та візуалізації даних польотів дронів. Діаграма складається з таких класів їх атрибутів та методів з наведеної діаграми:

1) *DataComponent* – абстрактний клас, є основою для зберігання і обробки даних польотів. Він реалізує стандартний інтерфейс для конвертування отриманих даних та представлення зведеної інформації. Всі класи, що містять специфічні типи даних, наслідуються від цього класу. Наведений клас містить наступні атрибути та методи:
   - Атрибути: Масив даних польотів.
   - *parseData(rawData: any)* – метод для конвертування та обробки даних користувача.
   - *displaySummary()* – повертає об’єкт з відповідними конвертованими даними.
2) *BatteryData* – клас, відповідальність якого є представлення детальної інформацію про стан батареї аналізованого безпілотного літального апарату, включаючи напругу, струм та температуру. Також він включає метод для обчислення ефективності батареї, що є корисним для оцінки її

продуктивності та терміну служби. Наведений клас містить наступні атрибути та методи:

- Атрибути: Напруга, струм, температура батареї.
- *calculateEfficiency()* – метод для обчислення ефективності батареї.

3) *GPSData* – клас відповідальний за роботу з геолокаційними даними, такі як широта, довгота та висота. Наведений клас включає метод для обчислення відстані до іншої *GPS* координати, описаний метод є критично важливим для навігації та маршрутизації дрона. Наведений клас містить наступні атрибути та методи:
   - Атрибути: Широта, довгота, висота.
   - *calculateDistance(targetGPS: GPSData)* – метод для обчислення відстані до цільової координати *GPS*.
4) *SensorData* – клас відповідає за збір даних з параметрів різних датчиків дрона, включаючи температуру, тиск та вологість. Наведений клас включає метод для реєстрації цих даних, що використовується для моніторингу умов польоту та стану дрона. Наведений клас містить наступні атрибути та методи:
   - Атрибути: Температура, тиск, вологість.
   - *logSensorData()* – метод для збору даних лог-файлу з датчиків.
5) *FlightModeData* – Клас відповідальний за моніторингом режимів політу безпілотного літального апарату. Цей клас важливий для аналізу поведінки дрона при різних сценаріїв політу, наприклад, для балансування між швидкістю та енергоефективністю. Наведений клас містить наступні атрибути та методи:
   - Атрибути: Поточний режим польоту, доступні режими.
   - *getCurrentMode(flight: Flight)* – повертає поточний режим польоту.
6) *DataProcessor* – клас відповідальний за обробку зібраних даних. Клас містить такі процеси, як виклик валідації даних, їх перетворення та підготовку до подальшого аналізу. Наведений клас містить наступні атрибути та методи:

- *processData(data: DataComponent)* – метод для обробки даних.
- *validateData(data: DataComponent)* – метод для валідації даних.

7) *DataValidator* – клас, який відіграє роль аудитора даних, перевіряючи їх на відповідність визначеним критеріям валідності, обраним фільтрам та статусу польоту. Це корисно для забезпечення валідності даних перед тим, як вони будуть використані в критичних обчисленнях або звітах. Наведений клас містить наступні атрибути та методи:
   - *validateData(data: DataComponent)* – валідація даних.
8) *DataAnalyzer* – клас для аналізу оброблених даних для виявлення тенденцій, визначення відхилень. Результати аналізу ефективно використовувати для прийняття рішень по налаштуванню, планування маршрутів або підвищення ефективності використання дронів. Наведений клас містить наступні атрибути та методи:
   - *analyzeData(data: DataComponent)* – метод який отримує нормалізовані дані, та повертає готовий результат аналізу.
9) *DataVisualizer* – Клас для перетворення складних аналітичних даних в графічне представлення. Це включає створення графіків та інших видів діаграм. Наведений клас містить наступні атрибути та методи:
   - *visualizeData(data: DataComponent, mode: string)* – метод який отримує дані та режим, і повертає відповідне графічне представлення аналізу.
10) *ReportGenerator* – клас для компонування аналізованих даних в структуровані звіти. Звіти включають різноманітні метрики, оцінки ефективності. Наведений клас містить наступні атрибути та методи:
   - *generateReport(data: DataComponent* – метод для генерування звіту на основі наведених даних.
11) *Drone* – клас-сутність для опису дрона. Включає аналіз функцій самодіагностики та оновлення прошивки, що є ключовими для підтримки дрона в робочому стані та забезпечення його функціональності. Наведений клас містить наступні атрибути та методи:

- *performSelfCheck()* – проведення аналізу самодіагностики.
- *updateFirmware(firmwareVersion: String)* – проведення аналізу оновлення БПЛА .

12) *FileManager* – клас для роботи з файловою системою, який виконує читання та запис файлів. Наведений клас містить наступні атрибути та методи:
    - *readFile(path: String)* – метод для зчитування файлу.
    - *writeFile(path: String, content: String)* – метод для зберігання файлу.
13) *Flight* – клас-сутність польоту, який описує абстрактний політ завантажений користувачем. Клас дозволяє контролювати політний цикл та збирати відповідні дані для кожного польоту. Наведений клас містить наступні атрибути та методи:
    - *startFlight()* – метод для отримання даних на початок польоту.
    - *endFlight()* – метод для отримання даних на кінець польоту.

При складані діаграми класів описано наступну взаємодію між класами:

1) Клас *DataProcessor* містить множину екземплярів *DataComponent* і використовує *DataValidator* для перевірки даних.
2) Клас *DataAnalyzer* використовує клас *DataProcessor* для обробки даних перед аналізом.
3) Клас *DataVisualizer* використовує клас *DataAnalyzer* для отримання даних, що будуть візуалізовано, та клас *ReportGenerator* для створення звітів.
4) Клас *Drone* пов’язаний з декількома екземплярами *Flight*.
5) Клас *FileManager* управляє даними для класу *DataComponent*.
6) Клас *Flight* використовує клас *DataComponent* для збору даних польоту.

Наведена діаграма класів відображає чітке розділення між класами, завдяки чому програмний модуль є легким в розширенні та підтримці.

Кожен клас зосереджений на конкретному процесі польоту безпілотного літального апарату, збору та обробки даних, що сприяє модульності та зменшує залежності.

2.4. Висновки до розділу

1) Детально проаналізовано інструменти та технологій, визначено сучасні стандарти, які є необхідними для розробки програмного модуля для аналізу даних польотів малої авіації.
2) Досліджено інтеграцію клієнсткої та серверної частини, та обрано технології для клієнтської та серверної частини, а також СУБД. Для клієнтської частини обрано: *HTML, CSS, JavaScript, React.JS*, для серверної частини обрано: *Python, RESTful API, HTTPS, OAuth 2.0, JWT, pytest*, та СУБД – *PostgreSQL.*
3) На основі проведеного порівняльного аналізу існуючих систем аналізу даних польотів сформовано наступні функціональні вимоги до програмного модуля:
   - можливість завантаження лог-файлів польотів для подальшого аналізу;
   - автоматизована обробка даних для виявлення стандартних показників польоту (швидкість, висота, *GPS* координати тощо);
   - використання графіків для наглядного представлення даних польотів;
   - генерація детальних звітів за кожним польотом;
   - аналіз показників для оцінки якості польоту та виявлення можливих проблем;
   - простий та зручний інтерфейс для доступу до функцій аналізу;
   - можливість користувача налаштовувати відображення даних за своїми потребами;
   - забезпечення безпеки даних під час передачі та зберігання;
   - механізми аутентифікації та авторизації для доступу до даних.
4) На основі проведеного порівняльного аналізу існуючих систем аналізу даних польотів сформовано наступні нефункціональні вимоги до програмного модуля:
   - висока швидкість аналізу та візуалізації даних;
   - ефективне використання системних ресурсів;

- висока доступність системи та мінімальний час простою;
- механізми для швидкого відновлення роботи після збоїв;
- здатність системи обробляти зростаючий об’єм даних і користувачів;
- захист від несанкціонованого доступу та кібератак;
- забезпечення конфіденційності даних за допомогою сучасних методів шифрування;
- легкий у навігації та використанні інтерфейс;
- інтерфейс, який підлаштовується під різні типи пристроїв (комп’ютери, мобільні пристрої).

5) побудовано та описано структурні схеми для клієнтської та серверної частини. Побудовано та описано сценарій завантаження та отримання аналізу польоту користувача в діаграмі послідовності *UML*. Побудовано та описано діаграму класів *UML* програмного модуля, яка відображає взаємодію між компонентами для збору, обробки, аналізу та візуалізації даних польотів дронів.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОГРАМНОГО МОДУЛЯ

3.1. Розробка серверної частини

3.1.1. Визначення функціональності серверної частини

На основі визначених функціональних вимог та спроектованій архітектурі розроблено серверну частину програмного модуля аналізу даних польотів малої авіації. Зона відповідальності полягає в обробці, аналізі завантажених файлів журналу польотів, зберіганні та генеруванні звіту.

Після завершення аналізу, результати надсилаються серверною частиною на клієнтську, яка вже відповідальна за візуалізації даних та можливості фільтрування результатів.

В серверній частині є можливості розширеного інтерфейсу взаємодії з базою даних, завдяки чому це дозволяє не тільки зберігати результати аналізу, але й ефективно управляти історією користувачів. Завдяки механізмів кешування та оптимізації запитів забезпечується висока швидкий доступ до історії.

Не менш важливим аспектктом серверної частини є безпека. В програмному модулі весь обмін даними між серверною та клієнтською частинами захищений протоколами шифрування *HTTPS*, що забезпечує конфіденційність та цілісність персональних даних користувачі.

Таким чином, серверна частина стає міцною основою для системи аналізу польотних даних, готовою вирішувати складні завдання з обробки, зберігання та аналізу великих обсягів даних, забезпечуючи при цьому високу продуктивність і надійність.

3.1.2. Алгоритм роботи з завантаженими файлами

Розроблено схема алгоритму алгоритму для взаємодії серверної частини з завантаженим файлом користувача (рис. 3.3).

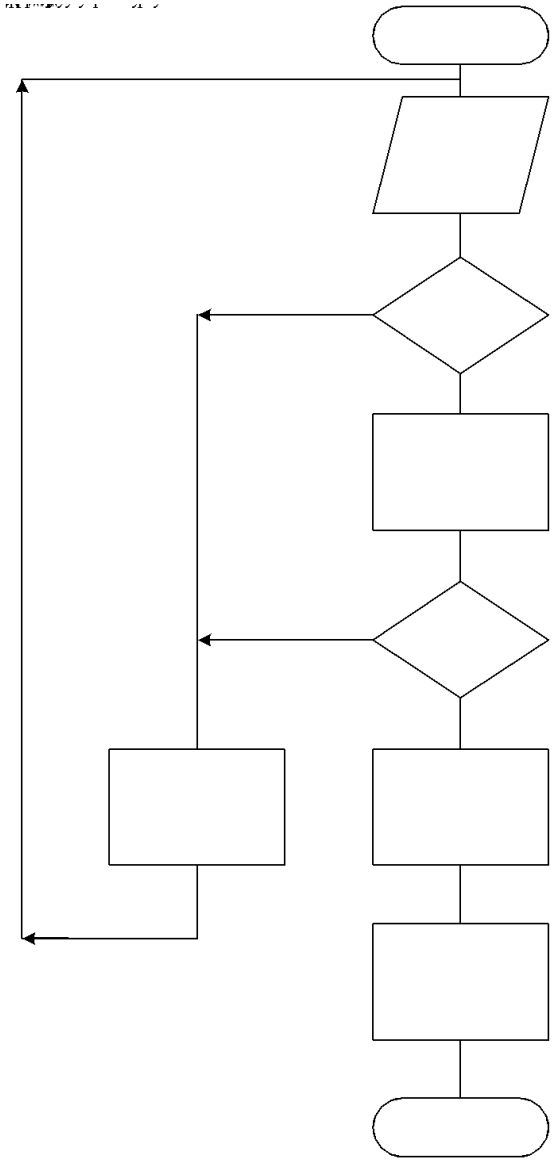

Рис. 3.3. Схема алгоритму обробки файлів

На основі схеми алгоритмів розроблено наступний функціонал модуля обробки файлів користувачів, що складається з:

1) імпорт бібліотек та модулів – в програмний модуль завантажуються необхідних бібліотек для обробки файлів, взаємодії з базою даних та запитів від серверної частини;

2) визначення функції *update_vehicle_db_entry* для взаємодії з базою даних, реалізує наступний функціонал:
   - з’єднання з базою даних;
   - оновлення записів у базі даних на основі завантаженого файлу користувача;
   - закриття з’єднання з базою даних.
3) впровадження класу *UploadHandler*, що застосовується для:
   - підготовка змінних для обробника;
   - підготовка до обробки *POST*-запиту;
   - встановлення ліміту розміру тіла запиту;
   - обробка отриманих даних;
   - обробка *POST*-запиту, який містить файл логу, його аналіз та збереження результатів в базі даних;
   - валідація файлу;
   - завантаження файлу у файлову систему;
   - генерація безпечного токену для файлу.

На основі визначеного функціоналу розроблено модуль та його вихідний код:

Наведена функція відповідає за обробку завантажених даних:

```
def process_submission(self, *args, **kwargs):
if self.data_stream:
try:
self.data_stream.complete_stream()
submission_data = self.data_stream.fetch_data(
['description', 'contact_email', 'allow_analysis', 'hidden', 'source', 'submission_type',
'user_feedback', 'wind_velocity', 'user_rating', 'media_link', 'is_public',
'vehicle_identifier'])
```

За допомогою методу *sanitize* виконується отримання полів з файлу, декодування та перетворення в рядки:

```
description_text = sanitize(submission_data['description'].decode("utf-8"))
```

contact_email = submission_data[‘contact_email’].decode("utf-8")

submission_category = ‘individual’

if ‘submission_type’ in submission_data:

submission_category = submission_data[‘submission_type’].decode("utf-8")

Визначення джерело завантаження (за замовчуванням – *web interface)*:

submission_source = ‘web_interface’

if ‘source’ in submission_data:

submission_source = submission_data[‘source’].decode("utf-8")

submission_title = “

is_hidden = False

if ‘hidden’ in submission_data and submission_data[‘hidden’].decode("utf-8") == ‘true’:

is_hidden = True

consent_for_analysis = False

Визначення, чи надано згоду на аналіз даних:

if ‘allow_analysis’ in submission_data and

submission_data[‘allow_analysis’].decode("utf-8") == ‘true’:

consent_for_analysis = True

Очистка та декодування відгуку користувача:

if ‘user_feedback’ in submission_data:

user_feedback = sanitize(submission_data[‘user_feedback’].decode("utf-8"))

Ініціалізація допоміжних змінних:

wind_velocity = -1

user_rating = “

email_recorded = “

media_link = “

submission_public = False

vehicle_name = “

error_messages = “

Якщо кагеторія відноситься до категорії *flight report*, тоді необідно виконати наступне:

if submission_category == ‘flight_report’:

try:

Отримання даних про швидкості вітру:

wind_velocity = int(sanitize(submission_data[‘wind_velocity’].decode("utf-8")))

except ValueError:

wind_velocity = -1

Отримання рейтингу та запис контактного email.

user_rating = sanitize(submission_data[‘user_rating’].decode("utf-8"))

if user_rating == ‘unset’:

user_rating = “

email_recorded = contact_email

Видалення посилань на медіафайл, для зменшення об’єму:

media_link = sanitize(submission_data[‘media_link’].decode("utf-8"), quote=True)

if not is_valid_url(media_link):

media_link = “

Отримання ідентифікатору транспортного засобу:

if ‘vehicle_identifier’ in submission_data:

vehicle_name = sanitize(submission_data[‘vehicle_identifier’].decode("utf-8"))

За замовчуванням встановлюємо аналіз для статистики:

consent_for_analysis = True

Визначення, чи є подання публічним.

if ‘is_public’ in submission_data and submission_data[‘is_public’].decode("utf-8")

== ‘true’:

Отримання інформації про завантажений файл

submission_public = True

file_part = self.data_stream.retrieve_parts(‘file_input’)[0]

uploaded_file_name = file_part.get_filename()

while True:

Генерація унікального імя та шляху для збереження файлу:

log_identity = str(uuid.uuid4())

new_file_path = create_log_path(log_identity)

if not os.path.exists(new_file_path):

break

Перевірка заголовка файлу на тип лог-файлу:

header_size = len(ULogFile.HEADER_BYTES)

if file_part.get_partial_payload(header_size) != ULogFile.HEADER_BYTES:

Перевірка формату завантаженого файлу:

if uploaded_file_name.lower().endswith('.log'):

raise InvalidFileError(400,

'Incorrect file format. This appears to be a log file. '

'Please upload to <a href="http://example.com" '

'target="_blank">example.com.')

raise InvalidFileError(400, 'Incorrect file format')

Перенос файлу до буферу сервісу:

print('Transferring uploaded file to', new_file_path)

file_part.transfer_to(new_file_path)

Генерація безпечного випадкового токена для *URL*:

access_token = str(binascii.hexlify(os.urandom(16)), 'ascii')

ulog_file = None

if submission_source != 'CI':

Завантаження *log* файлу.

ulog_file_path = create_log_path(log_identity)

ulog_file = load_ulog(ulog_file_path)

Збереження аних в базі даних:

db_connection = sqlite3.connect(get_database_path())

db_cursor = db_connection.cursor()

db_cursor.execute(

'INSERT INTO LogEntries (Id, Title, Description, '

‘Filename, DateCreated, AnalysisConsent, Hidden, ‘
‘Source, Email, WindVelocity, Rating, Feedback, Type, ‘
‘MediaLink, ErrorMessages, Public, AccessToken) VALUES ‘
[log_identity, submission_title, description_text, uploaded_file_name,
datetime.datetime.now(), consent_for_analysis,
is_hidden, submission_source, email_recorded, wind_velocity, user_rating,
user_feedback, submission_category, media_link, error_messages,
submission_public, access_token])

Оновлення інформації про транспортний засіб в базі даних:

if ulog_file:
vehicle_info = update_vehicle_entry(db_cursor, ulog_file, log_identity,
vehicle_name)
vehicle_name = vehicle_info.name
db_connection.commit()

Формування посилання для видалення запису:

plot_url = ‘/plot_app?log=‘ + log_identity
full_plot_url = get_http_protocol() + ‘://’ + get_domain() + plot_url
print(full_plot_url)
delete_link = get_http_protocol() + ‘://’ + get_domain() +
‘/modify_entry?action=delete&log=‘ + log_identity + ‘&token=‘ + access_token

Інформація для клієнтської частини:

email_info = {}
email_info[‘description’] = description_text
email_info[‘feedback’] = user_feedback
email_info[‘uploaded_filename’] = uploaded_file_name
email_info[‘type’] = ‘‘
email_info[‘frame’] = ‘‘
email_info[‘hardware’] = ‘‘
email_info[‘uuid’] = ‘‘
email_info[‘software’] = ‘‘

email_info[‘rating’] = user_rating

if vehicle_name:

email_info[‘vehicle’] = vehicle_name

Отримання додаткових даних з лог файлу для відправлення на клієнтську частину сповіщення:

if ulog_file:

converted_log = LogConvert(ulog_file)

email_info[‘type’] = px4_log.get_type()

frame_data = get_frame(ulog_file)

if frame_data:

frame_name, frame_id = frame_data

email_info[‘frame’] = frame_name or frame_id

hardware_info = “

if ‘ver_hw’ in ulog_file.msg_info:

hardware_info = sanitize(ulog_file.msg_info[‘ver_hw’])

email_info[‘hardware’] = hardware_info

if ‘sys_uuid’ in ulog_file.msg_info and hardware_info != ‘SITL’:

email_info[‘uuid’] = sanitize(ulog_file.msg_info[‘sys_uuid’])

software_branch = “

if ‘ver_sw_branch’ in ulog_file.msg_info:

software_branch = ‘ (branch: ‘+ulog_file.msg_info[‘ver_sw_branch’]+’)’

if ‘ver_sw’ in ulog_file.msg_info:

software_version = sanitize(ulog_file.msg_info[‘ver_sw’])

email_info[‘software’] = software_version + software_branch

Перевірка чи категорія відноситься до *flight_report* і є публічною (тобто користувач дав згоду на обробку):

if submission_category == ‘flight_report’ and submission_public:

Відправлення звіту на клієнтську частину:

email_recipients = set(email_config[‘public_reports’])

if user_rating in [‘poor’, ‘crash_sw_hw’, ‘crash_pilot’]:

email_recipients |= set(email_config[‘critical_reports’])

send_report_email(

list(email_recipients), full_plot_url, DBData.format_rating(user_rating),

Додатковий запис до бази даних:

DBData.format_wind(wind_velocity), delete_link,

email_recorded, email_info)

generate_db_entry_from_log(log_identity, db_connection)

IOLoop.current().add_callback(create_overview_image, log_identity)

Закінчення роботи з базою даних та закриття підключення.

db_connection.commit()

db_cursor.close()

db_connection.close()

send _notification(contact_email, full_plot_url, delete_link, email_info)

3.1.3. Алгоритм нормалізацій та аналізу

Розроблено схеми алгоритму для нормалізацій та аналізу серверної частини звіту на основі завантаженого лог-файлу користувача (рис. 3.4).

На основі схеми алгоритмів розроблено наступний функціонал модуля нормалізації файлів користувачів, який призначений для аналізу лог-файлів, отриманих від буфера сервісної частини.

Клас містить методи для обробки завантаженого файлу (отримання необхідних ключів для аналізу), нормалізація ключів і аналіз різних аспектів даних файлу, таких як *PWM* сигнали моторів, статус батареї, вібрації, точність *GPS*, робота *PID* контролера і інші.

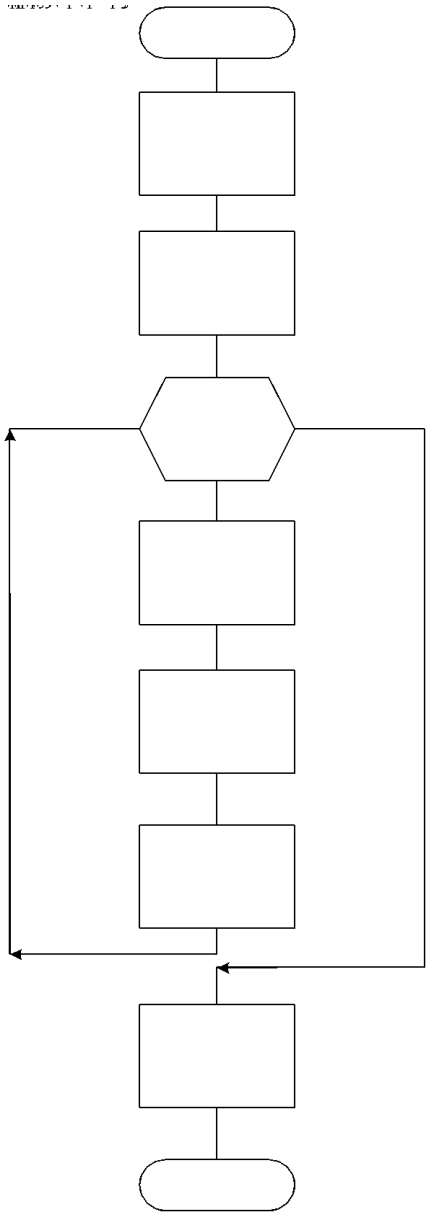

Рис. 3.4. Схема алгоритму нормалізацій та аналізу

На основі визначеного функціоналу розроблено модуль та його вихідний код:

```
import pandas as pd
import numpy as np
from scipy import signal
from sklearn.preprocessing import StandardScaler
class DroneLogAnalyzer:
def __init__(self, log_file):
self.log_file = log_file
self.converted_data = self.load_and_prepare_data()
```

Отримання та нормалізація даних з файлу:

def load_and_prepare_data(self):

data = pd.read_csv(self.log_file)

Нормалізація часових міток:

data['timestamp'] = pd.to_datetime(data['timestamp'])

return data

Метод для обробки даних пакетами для покращення ефективності:

def batch_process(self, func, column_name, batch_size=1000):

results = []

for start in range(0, len(self.converted_data), batch_size):

end = start + batch_size

batch_data = self.converted_data[column_name][start:end]

result = func(batch_data)

results.append(result)

return pd.concat(results)

Метод для нормалізації ряду даних:

def normalize_data(self, series):

scaler = StandardScaler()

return scaler.fit_transform(series.values.reshape(-1, 1))

Метод для аналізу *PWM* сигналів моторів:

def analyze_motor_pwm(self, pwm_data):

normalized_pwm = self.normalize_data(pwm_data)

return np.mean(normalized_pwm), np.std(normalized_pwm)

Метод для аналіз стану батареї з використанням аналізу часових рядів:

def analyze_battery_status(self, battery_voltage):

normalized_voltage = self.normalize_data(battery_voltage)

battery_voltage_fft = np.fft.fft(normalized_voltage)

return normalized_voltage, battery_voltage_fft

Метод для аналіз вібрацій:

def analyze_vibrations(self, vibration_data):

normalized_vibrations = self.normalize_data(vibration_data)

freqs, times, spectrogram = self.fourier_transform(normalized_vibrations)

return freqs, times, spectrogram

Метод комплексного аналізу точності *GPS*:

def analyze_gps_accuracy(self, gps_data):

normalized_gps_accuracy = self.normalize_data(gps_data)

accuracy_fft = np.fft.fft(normalized_gps_accuracy)

return normalized_gps_accuracy, accuracy_fft

Метод для аналізу роботи *PID* контролера:

def analyze_pid_performance(self, pid_data):

normalized_pid = self.normalize_data(pid_data)

correlation = np.corrcoef(normalized_pid['estimated'],

normalized_pid['setpoint'])[0, 1]

return correlation

Метод для аналізу шуму та перешкод *GPS:*

def analyze_gps_noise_and_jamming(self, gps_noise_data):

normalized_gps_noise = self.normalize_data(gps_noise_data)

noise_peaks, _ = signal.find_peaks(normalized_gps_noise, height=50)

return normalized_gps_noise, noise_peaks

Використання даного модуля:

log_file = 'path_to_log_file.csv'

analyzer = DroneLogAnalyzer(log_file)

Зміна *pwm_analysis* містить результати аналізу *PWM* використовуючи розбиття файлу по пакетам для отримання часових проміжків:

pwm_analysis = analyzer.batch_process(analyzer.analyze_motor_pwm, 'motor_pwm')

У наведеному коді програмного модуля використовуються такі підходи:

1) Пакетна обробка – завантажений файл обробляються пакетами з метою отримання часових проміжків та зменшити використання пам'яті і поліпшити продуктивність, особливо при обробці великих масивів даних.

2) Нормалізація – для даного підходу використовується метод *StandardScaler* з бібліотеки *scikit-learn*.
3) Гнучкість методів аналізу – в реалізованому модулі кожен метод аналізу приймає конкретний набір даних, дозволяючи легко застосувати різні види обробки без змін вихідного коду інших класів або методів.

Завдяки впроваджених підходів наведений модуль є відкритим до розширення в залежності від конкретних вимог.

3.1.4. Алгоритм аналізу *PWA*

Розроблено схему алгоритму для аналізу *PWA* на основі завантаженого лог-файлу користувача (рис. 3.5).

На основі наведенії схемі алгоритмів розроблено наступний функціонал модуля аналізу *PWA* метрики, який призначений для для обробки кадру даних, що містить як значення *PWA*, так і часові мітки для кожного двигуна. Також модуль обчислює середні значення *PWA* за весь політ, а також перевірки значень задніх двигунів відносно передніх.

На основі визначеного функціоналу розроблено модуль та його вихідний код:

import pandas as pd

Метод для аналізу *PWM* двигуна для безпілотного літального апарату протягом усього польоту:

def analyze_motor_pwm(df):

Переданий параметр *DataFrame* містіть значення: *time, C1, C2, C3, C4*, що представляють час і значення *PWM* для кожного двигуна.

Наведена функція повертає звіт, що містить середню *PWM* для кожного двигуна, середню *PWM* для кожної частини польоту і мітки, що вказують чи поведінка знаходить в межах очікуваної поведінки:

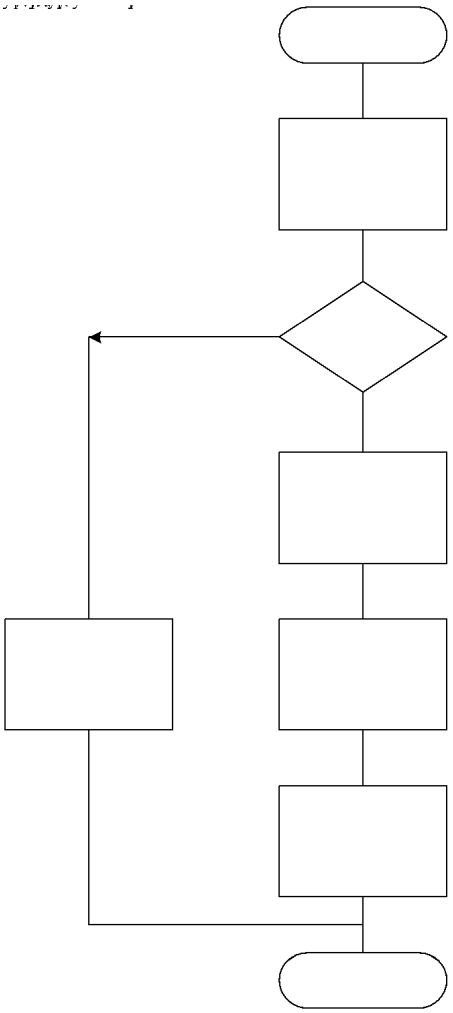

Рис. 3.5. Алгоритм аналізу *PWA*

Вихідний код досліджуваної функції:

df (pd.DataFrame):

required_columns = {‘time’, ‘C1’, ‘C2’, ‘C3’, ‘C4’}

if not required_columns.issubset(df.columns):

raise ValueError(f"DataFrame must contain columns: {required_columns}")

Розрахувати середнє значення *PWM* для кожного двигуна за весь політ:

averages = df[[‘C1’, ‘C2’, ‘C3’, ‘C4’]].mean()

Політ польоту на частини залежно від часу:

halfway_point = df[‘time’].median()

first_half = df[df[‘time’] <= halfway_point]

second_half = df[df[‘time’] > halfway_point]

Розраховує середнє значення *PWM* для кожного двигуна для кожної часової частини:

averages_first_half = first_half[[‘C1’, ‘C2’, ‘C3’, ‘C4’]].mean()

averages_second_half = second_half[[‘C1’, ‘C2’, ‘C3’, ‘C4’]].mean()

Перевірте очікуваної поведінку, що задні двигуни повинні мати вищі середні показники, ніж передні:

expected_behavior_first_half = averages_first_half[‘C2’] >

averages_first_half[‘C1’] and averages_first_half[‘C4’] >

averages_first_half[‘C3’]

expected_behavior_second_half = averages_second_half[‘C2’] >

averages_second_half[‘C1’] and averages_second_half[‘C4’] >

averages_second_half[‘C3’]

Повернення результату:

results = {

‘overall_averages’: averages.to_dict(),

‘first_half_averages’: averages_first_half.to_dict(),

‘second_half_averages’: averages_second_half.to_dict(),

‘expected_behavior_first_half’: expected_behavior_first_half,

‘expected_behavior_second_half’: expected_behavior_second_half}

return results

Наведений метод враховує часові мітки і аналізує розділяючи дані на ці часові ряді. Завдяки чому визначається чи змінюється очікувана поведінка в будь-який момент польоту.

Для використання наведеного методу потрибно передати їй значення *DataFrame*. Структура *DataFrame* має такий вигляд:

data = {

‘time’: [1, 2, 3, 4, 5],

‘C1’: [1429, 1403, 1668, 1400, 1401],

‘C2’: [1498, 1490, 1704, 1495, 1496],

‘C3’: [1422, 1402, 1639, 1405, 1403],

‘C4’: [1529, 1518, 1700, 1520, 1521]}

df = pd.DataFrame(data)

Отримання результату виконується наступним чином:

results = analyze_motor_pwm(df)

print(results)

Наведена функція передбачає, що стовпець «*time*» має формат, який можна розділити числовим способом (наприклад, секунди або монотонний ряд). Якщо стовпчик «*time*» має інший формат, наприклад, рядок або об’єкт типу *datetime*, перед використанням цієї функції його потрібно перетворити в числовий формат.

3.2. Розробка клієнтської частини

3.2.1. Опис функціональності клієнтської частини

На основі визначених функціональних вимог та спроектованій архітектурі розроблено клієнтську частину програмного модуля аналізу даних польотів малої авіації. Зона відповідальності полягає в ефективній візуалізації результатів аналізу, отриманих від сервера, та наданні інтуїтивно зрозумілих інструментів для взаємодії з цими даними.

Ключові характеристики клієнтської частини включають:

1) інтерактивні інструменти візуалізації – до цих інструментів входять графіки, картографічні зображення та інші візуальні елементи, які дозволяють користувачам швидко оцінювати і аналізувати польотні дані;
2) можливість фільтрування даних – фільтрування даних згідно з різними параметрами, що полегшує пошук конкретної інформації в великому звіті;
3) розробка враховує різні розміри екранів і пристроїв, забезпечуючи однаково зручне використання як на десктопах, так і на мобільних пристроях;
4) інтеграція з серверною частиною для отримання та відправлення даних, забезпечуючи актуальність інформації та швидкість відгуку системи.

Таким чином, клієнтська частина становить необхідну ланку в загальній архітектурі системи аналізу польотних даних, забезпечуючи кінцевим користувачам потужний інструмент для інтерпретації та вивчення даних польотів.

3.2.2. Компонент для відображення графіків

Відповідно до визначених характеристик клієнтської частині, реалізовано компонент *AdvancedGraphComponent*. Цей компонент є універсальним і призначений для динамічного представлення даних через різні типи графіків (лінійний, стовпчиковий, круговий) в залежності від потреб користувача. Компонент забезпечує обробку та відображення даних, використовуючи передані параметри від батьківського компоненту, включаючи тип графіка та його налаштування. Така гнучкість дозволяє ефективно адаптуватися до різних аналітичних сценаріїв та вимог системи, забезпечуючи зручний інтерфейс для інтерактивної візуалізації даних.

Вихідний код компоненту:

Встановлення залежностей:

import React, { useState, useEffect } from 'react';

import { Line, Bar, Pie } from 'react-chartjs-2';

import { processDataForGraphType, defaultOptions } from './GraphUtils';

import './GraphStyles.css';

Створення компоненту та опис необіхдних параметрів для його роботи, а саме тип графіку, дані аналізу та параметру графіку:

const AdvancedGraphComponent = ({ graphType, data, customOptions }) => {

Оголошення змінних компонентів через метод життєвого циклу *useState*, який буде оновлювати інтерфейс, якщо змінні зміняться:

const [graphData, setGraphData] = useState({});

const [options, setOptions] = useState(defaultOptions);

Оголошення методу *useEffect,* який буде виконуватись при встановленню компоненту в *DOM* сторінки:

useEffect(() => {

Встановлення змінних необхідних для роботи компоненту:

const processedData = processDataForGraphType(graphType, data);

setGraphData(processedData);

setOptions({ ...options, ...customOptions });

}, [graphType, data, customOptions]);

Функція яка повертає відповідний графік в залежності від обраного типу:

const renderGraph = () => {

switch (graphType) {

case 'line':

return <Line data={graphData} options={options} />;

case 'bar':

return <Bar data={graphData} options={options} />;

case 'pie':

return <Pie data={graphData} options={options} />;

default:

return Unsupported graph type }};

Відображення графіку компонентом:

return {renderGraph()} };

export default AdvancedGraphComponent;

Для ізолювання відображення від моделі створено утилітарні методи.

Функція для обробки даних для лінійного графіка, приймає параметри *data* - це масив даних для графіка. Та повертає – об'єкт оброблених даних для лінійного графіка.

Дані приходять у вигляді масиву об'єктів *{ x: value, y: value }*

export const processLineGraphData = (data) => {

Сортування даних за значенням *x* (час, дата тощо):

const sortedData = data.sort((a, b) => a.x - b.x);

return {

labels: sortedData.map(item => item.x),

datasets: [{

label: 'Демонстрація даних',

data: sortedData.map(item => item.y),

backgroundColor: 'rgba(255, 99, 132, 0.2)',

borderColor: 'rgba(255, 99, 132, 1)',

borderWidth: 1}]}};

Функція для обробки даних для стовпчикового графіка, приймає параметри *data, які представленні в* вигляді масиву даних для графіка. Та повертає – об'єкт оброблених даних для стовпчикового графіка.

export const processBarGraphData = (data) => {

Агрегація даних за фільтрованою категорією:

const aggregatedData = data.reduce((acc, item) => {

if (!acc[item.category]) {

acc[item.category] = 0}

acc[item.category] += item.value;

return acc}, {});

Повертає масив об'єктів для графіку:

return {

labels: Object.keys(aggregatedData),

datasets: [{

label: 'Категорії',

data: Object.values(aggregatedData),

backgroundColor: Object.keys(aggregatedData).map(() =>

*`rgba(${Math.floor(Math.random() * 255)}, ${Math.floor(Math.random() * 255)},*

*${Math.floor(Math.random() * 255)}, 0.5)`),*

borderColor: Object.keys(aggregatedData).map(() => `rgba(0, 0, 0, 1)`),

borderWidth: 1}]}};

Функція для обробки даних для кругового графіка, приймає параметри *data* - це масив даних для графіка. Та повертає – об'єкт оброблених даних для кругового графіка.

```
export const processPieGraphData = (data) => {
```

Розподіл даних за кагегоріямі:

```
const categoryCounts = data.reduce((acc, item) => {
acc[item.category] = (acc[item.category] || 0) + 1;
return acc}, {});
return {
labels: Object.keys(categoryCounts),
datasets: [{
label: 'Розподіл за категоріями',
data: Object.values(categoryCounts),
backgroundColor: Object.keys(categoryCounts).map(() =>
`rgba(${Math.floor(Math.random() * 255)}, ${Math.floor(Math.random() * 255)},
${Math.floor(Math.random() * 255)}, 0.7)`),
hoverOffset: 4}]}};
```

Функція для генерації опцій для графіка за замовчуванням. Повертає масив об'єктів опцій для графіку:

```
export const defaultOptions = {
responsive: true,
scales: {
y: {
beginAtZero: true}},
plugins: {
legend: {
position: 'top'},
title: {
display: true,
text: 'Приклад Графіка'}}};
```

Наведені функції *processLineGraphData, processBarGraphData, processPieGraphData* необхідні для обробки даних для різних типів графіків. Вони

включають логіку для сортування, агрегації та розподілу даних відповідно до потреб графіка.

Метод *defaultOptions* – повертає загальні налаштування для графіків, включаючи відповідність розміру контейнера, масштабування осей та налаштування легенди та заголовка.

3.2.2. Сервіс для відправки запитів до серверу

Відповідно до заданих параметрів інтерфейсу програмного модуля впроваджено сервісу, який призначений для виконання *HTTP*-запитів до серверної частини.

Вихідний код компоненту:

Опис типу параметрів, які потрібно передати в сервіс для успішного виконання запиту на сервер:

```
type TApiFetch = {
method?: HttpMethods;
endPoint: string;
body?: unknown};
export const apiFetch = async ({ method = 'GET', endPoint, body }: TApiFetch) {
```

Сервіс створює *HTTP*-запит на основі заданих параметрів *method*, *endPoint*, *body*. Це дозволяє відправляти різні типи запитів (наприклад, *GET* або *POST*) на будь який шлях *API* серверу, що корисно для взаємодії з різними даними польоту (наприклад, *GPS*-аналіз, аналіз вібрації), які будуть отримуватися ізольовано:

```
return await fetch(`${API_LINK}${endPoint}`, {
credentials: 'include',
method,
headers: {
'Content-type': 'application/json'},
body: JSON.stringify(body),}).then(async (res: Response) => {
```

Після відправлення запиту функція обробляє відповідь. Якщо відповідь не є успішною («*res.ok*» є *false*), то вона обробляє помилку, використовуючи стандартні *HTTP*-коди помилок.

if (!res.ok) {

const payload: TErrorResponse = await res.json().catch(() => null);

У разі помилки сервера, функція створюватиме помилку *HTTPError* з детальною інформацією про помилку. Це дозволяє ефективно обробляти помилки на стороні клієнта, наприклад, показувати повідомлення про помилки користувачам інтерфейсу.

throw new HTTPError(payload.error.status, res.status, payload.error.message,

res.statusText, payload)}

if (res.status === ResponseStatusEnum.NO_CONTENT) {

return null}

Якщо помилок немає, то повертає результат отриманий з серверної частина, та також приводимо його до типу *JSON*.

return res.json()}).catch(err => {

Якщо користувач не авторизований, він буде перенаправлений на головну сторінку:

if (err.status === ResponseStatusEnum.Unauthorized) {

window.location.replace('/');}

if (err instanceof HTTPError) {

throw {

message: err.message ?? 'Unknown error',

status: err.status,

statusText: err.statusText,

payload: err.payload,

stack: err.stack}}})};

Наведений сервіс є ключовою для взаємодії між клієнтською і серверною частиною, що надає дані для аналізу польотів. Наведена функція дозволяє ефективно

передавати, отримувати та оновлювати дані, необхідні для відображення графіків та інших аналітичних інструментів.

3.2.3. Компонент для відображення аналізу батареї

Для відображення взаємодії між сервісом для отримання даних з серверної частини та компоненту графіків створенно компонент для відображення аналізу батареї у вигляді лінійного графіку. Для цього при створенні компоненту виконано запит до серверу та налаштовано сам графік, після чого створено компонент з графіками, в який передано отриманий аналіз та налаштування.

Вихідний код компоненту:

Додавання основного компоненту для відображення графіків:.

import AdvancedGraphComponent from ‘./AdvancedGraphComponent’;

import React, { useState, useEffect } from ‘react’;

Додавання сервісу для відправки запиту на аналізу даних

import { apiFetch } from ‘./apiFetch’;

import { processLineGraphData } from ‘./GraphUtils’;

const BatteryStatusAnalyzer = () => {

Компонент використовує методи життєвого циклу компоненту *useState* і *useEffect* для керування станом даних батареї, отримання цих даних з сервера та перезавантаженням компоненту, якщо дані були змінені:

const [batteryData, setBatteryData] = useState(null);

useEffect(() => {

const fetchData = async () => {

try {

Виклик сервісу для отримання даних аналізу батареї безпілотного літального апарату з сервера:

const data = await apiFetch({ endPoint: ‘/battery-status’ });

Виклик функції *processLineGraphData* для підготовки даних для лінійного графіка:

setBatteryData(processLineGraphData(data))}

catch (error) {

console.error(‘Error fetching battery status data:’, error)}};

Виклик функції для отримання даних при першому завантаженню компонента в *DOM* сторінки:

fetchData()}, []);

Конфігурація заголовка, підписів осей та інші параметри для візуального представлення даних.

const graphOptions = {

title: {

display: true,

text: 'Аналіз стану батареї'},

scales: {

y: {

title: {

display: true,

text: 'Заряд (%)'}},

x: {

title: {

display: true,

text: 'Час'}}}};

return (

* *

<h2>Аналіз стану батареї</h2>

{batteryData ? (

Якщо дані з серверу вілідні, тоді відображуємо компоненет *AdvancedGraphComponent* для рендерингу графіка з переданими даними аналізу стану батареї:

<AdvancedGraphComponent graphType='line' data={batteryData}

customOptions={graphOptions} />) : (

* Завантаження даних...)}*

*)};*

export default BatteryStatusAnalyzer;

Цей компонент ефективно візуалізує дані стану батареї, дозволяючи користувачам легко аналізувати їх зміни у часі, для відображення аналізу він використовує сервіс для отримання даних аналізу та компонент графіку, в який він передає отримані дані та налаштування для графіку. Також, наведений компонент має гнучкі налаштування та структурований підхід до обробки та відображення даних.

3.3. Тестування та валідація програмного модуля

3.3.1. Формування вимог до тестування

Для тестування та валідації програмного модуля обрано типову описану раніше функцію для генерування аналізу *PWM* моторів.

Для ретельного тестування функції *analyze_motor_pwm* необхідно створити серію тестових випадків з використанням бібліотеки *pytest*, які охоплюють різні сценарії, включаючи типові випадки, малоімовірні випадки та ситуації з потенційними помилками, для того щоб перевірити поведінку функції під різними ситуаціями.

На основі вихідного коду методу *analyze_motor_pwm* сформовано наступні тестові поведінки:

1) тест з типовими даними – перевірка, що функція правильно розраховує середні значення та ідентифікує очікувану поведінку, коли значення *PWM* задніх моторів вищі ніж передніх;
2) тест з однаковими значеннями – перевірка, коли всі мотори мають однакові значення *PWM* (що не є очікуваною поведінкою);
3) тест з вищими значеннями *PWM* передніх моторів – перевірка сценарію, де передні мотори мають вищі значення *PWM*, ніж задні мотори (що також не є очікуваною поведінкою);
4) тест без даних – перевірка, що функція правильно обробляє порожній *DataFrame*;

5) тест надійності даних – перевірка функції з нечисловими даними, щоб переконатися в коректності сценарію дій функцій під час помилки;
6) тест з відсутністю деяких стовпців – перевірка, що функція викликає помилку, якщо відсутні хоча б один з стовпців.

3.3.2. Опис вихідного коду тестового файлу для функції аналізу *PWM*

На основі визначених вимог до тестових випадків та вихідного коду функції сформовано наступний код тестових випадків використовуючи бібліотеку для тестування *pytest:*

Встановлення необхідних бібліотек для виконання тесту та створення даних для тесту аналізу:

```
import pytest
import pandas as pd
from analyzesModules import analyze_motor_pwm
@pytest.fixture
def typical_data():
return pd.DataFrame({
‘time’: range(1, 10001),
‘C1’: [1400 + i % 5 for i in range(10000)],
‘C2’: [1500 + i % 5 for i in range(10000)],
‘C3’: [1400 + i % 5 for i in range(10000)],
‘C4’: [1500 + i % 5 for i in range(10000)]})
```

Опис стандарного шаблону вхідних даних:

```
@pytest.fixture
def equal_pwm_data():
return pd.DataFrame({
‘time’: range(1, 10001),
‘C1’: [1500] * 10000,
‘C2’: [1500] * 10000,
```

*‘C3’: [1500] * 10000,*

*‘C4’: [1500] * 10000})*

Тест з типовими даними для перевірки поведінку функції з типовими даними польоту, де значення *PWM* задніх моторів вищі. Даний тестовий випадок перевіряє, що функція правильно ідентифікує очікувану поведінку та розраховує середні значення:

def test_with_typical_data(typical_data):

result = analyze_motor_pwm(typical_data)

assert result[‘expected_behavior_first_half’] is True

assert result[‘expected_behavior_second_half’] is True

assert result[‘overall_averages’][‘C2’] > result[‘overall_averages’][‘C1’]

assert result[‘overall_averages’][‘C4’] > result[‘overall_averages’][‘C3’]

Тест з однаковими значеннями *PWM* для перевірки поведінку функції, коли всі мотори мають однакові значення *PWM*. Даний тестовий випадок перевіряє, що функція визначає це як ненормальну поведінку:

def test_with_equal_pwm_values(equal_pwm_data):

result = analyze_motor_pwm(equal_pwm_data)

assert result[‘expected_behavior_first_half’] is False

assert result[‘expected_behavior_second_half’] is False

Тест з вищими значеннями *PWM* для передніх моторів для перевірки поведінку функції, коли передні мотори мають вищі значення *PWM*, ніж задні. Даний тестовий випадок перевіряє, що функція Модифікує дані для інвертування значень *PWM* і перевіряє відповідну реакцію функції:

def test_with_front_motors_higher(typical_data):

typical_data[‘C1’], typical_data[‘C2’] = typical_data[‘C2’], typical_data[‘C1’]

typical_data[‘C3’], typical_data[‘C4’] = typical_data[‘C4’], typical_data[‘C3’]

result = analyze_motor_pwm(typical_data)

assert result[‘expected_behavior_first_half’] is False

assert result[‘expected_behavior_second_half’] is False

Тест з без даних для перевірки поведінку функції з порожнім *DataFrame*. Даний тестовий випадок перевіряє, що функція викликає помилку «*ValueError*»:

def test_with_no_data():

empty_data = pd.DataFrame({‘time’: [], ‘C1’: [], ‘C2’: [], ‘C3’: [], ‘C4’: []})

with pytest.raises(ValueError):

analyze_motor_pwm(empty_data)

Тест з недійсними даними для перевірки поведінку функції з нечисловими даними. Даний тестовий випадок перевіряє, що функція викликає помилку «*TypeError*»:

def test_with_invalid_data():

invalid_data = pd.DataFrame({‘time’: [‘a’, ‘b’, ‘c’], ‘C1’: [1, 2, 3], ‘C2’: [4, 5, 6], ‘C3’: [7, 8, 9], ‘C4’: [0, 1, 2]})

with pytest.raises(TypeError):

analyze_motor_pwm(invalid_data)

Тест з відсутніми стовпцями для перевірки поведінку функції при відсутності деяких стовпців. Даний тестовий випадок перевіряє, що функція викликає помилку «*ValueError*»:

def test_with_missing_columns():

incomplete_data = pd.DataFrame({‘time’: range(1, 10001), ‘C1’: [1400 + i % 5 for

i in range(10000)]})

with pytest.raises(ValueError):

analyze_motor_pwm(incomplete_data)

Кожний з наведених тестових випадків перевіряє коректність роботи та аналізу даних, за винятком тестів, що обробляють умови помилок, які створюють власні тестові дані. Тестові випадки перевіряють правильність обчислення середніх значень та очікувану поведінку та перевіряють коректність обробки помилок, коли функція закінчує своє виконання через помилкою.

3.3.3. Результат тестування

Виконання автотестів для перевірки результів коректності роботи функції аналізу *PWM* (табл. 3.1).

Таблиця 3.1

Тестування методу

Дія	**Очікуваний результат**	**Фактичний результат**
Тест з типовими даними	Середні значення *PWM* для передніх і задніх моторів відповідають очікуваному (передні > задні).	Пройдено
Тест з однаковими значеннями *PWM*	Метод вказує на ненормальну поведінку (*False*).	Пройдено
Тест з вищими значеннями *PWM* передніх моторів	Метод вказує на ненормальну поведінку (*False*).	Пройдено
Тест без даних	Викликається помилка *ValueError*, так як переданий порожній *DataFrame*.	Пройдено
Тест з недійсними даними	Викликається помилка *TypeError*, так як в даних є нечислові значення.	Пройдено
Тест з відсутніми стовпцями	Викликається помилка *ValueError*, так як в даних відсутні необхідні стовпці.	Пройдено

Усі тести пройшли успішно і відповідають очікуваному результату, включаючи випадки, де функція повинна створювати помилку при неправильних даних або відсутності необхідних стовпців.

3.4. Висновки до розділу

1) Розроблено надійну та ефективну серверну частину для програмного модуля аналізу даних польотів малої авіації на основі визначених функціональних вимог та ретельно спроектованій архітектурі програмного модуля. Серверна частина відповідає за завдання обробки, аналізу, зберігання даних журналів польотів та генерації звітів. Вона забезпечує ефективне управління даними, включаючи розширені можливості взаємодії з базою даних, кешування та оптимізацію запитів, що значно підвищує швидкість доступу до інформації.
2) Розроблено схема алгоритму алгоритму для взаємодії серверної частини з завантаженим файлом користувача. На основі схеми алгоритму розроблено вихідний код до модуля взаємодії серверної частини з лог-файлами, який включає етапи прийому файлу, перевірки на валідність, наприклад формату та цілісності, обробки даних та подальшої передачі оброблених даних на наступні етапи аналізу.
3) Розроблено схему алгоритму алгоритму для нормалізації та аналізу звіту, в наведеній схемі алгоритму відображено послідовність кроків, які виконуються для нормалізації та аналізу даних, отриманих із файлу користувача.
4) Розроблено схема алгоритму алгоритму для аналізу *PWA* на основі завантаженого файлу користувача, наведена схема алгоритму відображає методи аналізу, для обробки та інтерпретації даних із файлів користувачів для визначення параметрів *PWA*.
5) Реалізовано ключові компоненти для клієнтської частини програмного модуля. Розроблено компонент для відображення графіків в різних форматах в залежності від вибору користувача – *AdvancedGraphComponent*. На основі цього компоненту реалізовано компонент *BatteryStatusAnalyzer* який надсилає запит на серверну частину і передає отриманні дані в компонент графіку, та налаштування для відображення.

6) Розроблено сервіс для виконання *HTTP*-запитів до серверної частини, що є важливим елементом для забезпечення зв'язку між клієнтською та серверною частинами.
7) Для забезпечення надійності та точності програмного модуля, проведено ретельне автоматичне тестування з використанням бібліотеки *pytest*. Для тестування та валідації програмного модуля обрано типовий метод аналізу – генерування аналізу *PWM* моторів. Описано серію тестових випадків, які охоплюють різні сценарії, включаючи типові випадки, малоімовірні випадки та ситуації з потенційними помилками, для того щоб перевірити поведінку функції під різними ситуаціями.

ВИСНОВКИ

1) Аналіз даних польотів малої авіації відіграє важливу роль у підвищенні безпеки та ефективності використання літаків. Цей процес включає збір, аналіз та інтерпретацію даних із систем літака. Основною метою якого є підвищення безпеки польотів шляхом виявлення ризиків, небезпечних застосувань та механічних проблем.
2) Оскільки популярність безпілотних літальних апаратів зростає, вони стикаються з унікальними проблемами при аналізі польотних даних. Основні фактори, що впливають на стабільність БПЛА включають: вразливості програмного забезпечення, відмову датчиків і кібератаки. Ефективний аналіз даних допомагає виявляти та запобігає інцидентам, пов'язаним із фізичним станом безпілотних літальних апаратів.
3) Проведено аналіз існуючих інструментів для аналізу лог-файлів БПЛА – *MAVExplorer* та *Mission Planner*. *MavExplorer* виявився більш гнучким, але використання його є складнішим через необхідність встановлення *Python* та основного додатку для його запуску, тоді як *Mission Planner* став простішим та інтуїтивнішим, але менш гнучким. Також для ефективного використання обох інструментів користувач повинен мати певний рівень технічної компетенції та знань для оптимального використання.
4) Детально проаналізовано інструменти та технологій, визначено сучасні стандарти, які є необхідними для розробки програмного модуля для аналізу даних польотів малої авіації.
5) Спроєктовано інтеграцію клієнсткої та серверної частини, та обрано технології для клієнтської та серверної частини, а також СУБД. Для клієнтської частини обрано: *HTML, CSS, JavaScript, React.JS*, для серверної частини обрано: *Python, RESTful API, HTTPS, OAuth 2.0, JWT, pytest*, та СУБД – *PostgreSQL.*

6) На основі проведеного порівняльного аналізу існуючих систем аналізу даних польотів сформовано наступні функціональні вимоги до програмного модуля:
   - можливість завантаження лог-файлів польотів для подальшого аналізу;
   - автоматизована обробка даних для виявлення стандартних показників польоту (швидкість, висота, *GPS* координати тощо);
   - використання графіків для наглядного представлення даних польотів;
   - генерація детальних звітів за кожним польотом;
   - аналіз показників для оцінки якості польоту та виявлення можливих проблем;
   - простий та зручний інтерфейс для доступу до функцій аналізу;
   - можливість користувача налаштовувати відображення даних за своїми потребами;
   - забезпечення безпеки даних під час передачі та зберігання;
   - механізми аутентифікації та авторизації для доступу до даних.
7) На основі проведеного порівняльного аналізу існуючих систем аналізу даних польотів сформовано наступні нефункціональні вимоги до програмного модуля:
   - висока швидкість аналізу та візуалізації даних;
   - ефективне використання системних ресурсів;
   - висока доступність системи та мінімальний час простою;
   - механізми для швидкого відновлення роботи після збоїв;
   - здатність системи обробляти зростаючий об’єм даних і користувачів;
   - захист від несанкціонованого доступу та кібератак;
   - забезпечення конфіденційності даних за допомогою сучасних методів шифрування;
   - легкий у навігації та використанні інтерфейс;
   - інтерфейс, який підлаштовується під різні типи пристроїв (комп’ютери, мобільні пристрої).

8) Спроектовано архітектуру програмного модуля. Побудовано та описано структурні схеми для клієнтської та серверної частини. Побудовано та описано процес в *UML* діаграмі послідовності для сценарію завантаження та отримання аналізу польоту користувача. Побудовано та описано *UML* діаграму класів програмного модуля, яка відображає взаємодію між компонентами для збору, обробки, аналізу та візуалізації даних польотів дронів.
9) Розроблено надійну та ефективну серверну частину для програмного модуля аналізу даних польотів малої авіації на основі визначених функціональних вимог та ретельно спроектованій архітектурі програмного модуля. Серверна частина відповідає за завдання обробки, аналізу, зберігання даних журналів польотів та генерації звітів. Вона забезпечує ефективне управління даними, включаючи розширені можливості взаємодії з базою даних, кешування та оптимізацію запитів, що значно підвищує швидкість доступу до інформації.
10) Розроблено схема алгоритму алгоритму для взаємодії серверної частини з завантаженим файлом користувача. На основі схеми алгоритму розроблено вихідний код до модуля взаємодії серверної частини з лог-файлами, який включає етапи прийому файлу, перевірки на валідність, наприклад формату та цілісності, обробки даних та подальшої передачі оброблених даних на наступні етапи аналізу.
11) Розроблено схема алгоритму алгоритму для нормалізації та аналізу звіту, в якій наведеній відображено послідовність кроків, які виконуються для нормалізації та аналізу даних, отриманих із файлу користувача.
12) Розроблено схема алгоритму алгоритму для аналізу *PWA* на основі завантаженого файлу користувача, наведена схема алгоритму відображає методи аналізу, для обробки та інтерпретації даних із файлів користувачів для визначення параметрів *PWA*.
13) Реалізовано ключові компоненти для клієнтської частини програмного модуля. Розроблено компонент для відображення графіків в різних

форматах в залежності від вибору користувача – *AdvancedGraphComponent*. На основі цього компоненту реалізовано компонент *BatteryStatusAnalyzer* який надсилає запит на серверну частину і передає отриманні дані в компонент графіку, та налаштування для відображення.

14) Розроблено сервіс для виконання *HTTP*-запитів до серверної частини, що є важливим елементом для забезпечення зв'язку між клієнтською та серверною частинами.
15) Для забезпечення надійності та точності програмного модуля проведено ретельне автоматичне тестування з використанням бібліотеки *pytest*. Для тестування та валідації програмного модуля обрано типовий метод аналізу – генерування аналізу *PWM* моторів. Описано серію тестових випадків, які охоплюють різні сценарії, включаючи типові випадки, малоімовірні випадки та ситуації з потенційними помилками, для того щоб перевірити поведінку функції під різними ситуаціями.
16) Наукова новизна отриманих результатів. Динамічний розвиток малої авіації та збільшення використання безпілотних літальних апаратів вимагає нових підходів до аналізу та оптимізації польотних даних. Запропонований програмний модуль включає нові методи та інструменти, що враховують унікальні вимоги та особливості БПЛА. Оскільки сучасні інструменти мають вимоги по операційних систем та встановлених мов програмування програмний модуль надає можливість операторам БПЛА отримувати детальний та точний аналіз польотних даних незалежно в веб-браузері незалежно від операційної системи. Програмний модуль дозволяє адаптувати інструменти аналізу під індивідуальні сценарії, строювати нові більш ефективних стратегій польоту.
17) Практичне значення отриманих результатів. Програмний модуль сприяє оптимізації процесів планування та аналізу польотних місій малої авіації. Даний модуль може використовуватися в комерційних цілях та навчальних закладах для демонстрації та вивчення аспектів польотної аналітики.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ПОЛОЖЕННЯ ПРО ДИПЛОМНІ РОБОТИ (ПРОЕКТИ) ВИПУСКНИКІВ НАЦІОНАЛЬНОГО АВІАЦІЙНОГО УНІВЕРСИТЕТУ. Київ: НАУ, 2017.
2. ДОКУМЕНТАЦІЯ. ЗВІТИ У СФЕРІ НАУКИ І ТЕХНІКИ. Структура і правила оформлення. ДСТУ 3008-95. Київ.
3. Класифікація дронів. *URL: http://surl.li/srwm* (дата звернення: 26.11.2023).
4. Луцький М.Г., Харченко В.П., Бугайко Д.О. Розвиток міжнародного регулювання та нормативної бази використання безпілотних літальних апаратів. Вісник НАУ, 2011, № 2, с. 5-14.
5. Родіонов П. Ю. Управління інформаційною діяльністю авіакомпанії: дисертація на здобуття наукового ступеня кандидата економічних наук. Київ: Національний авіаційний університет, 2015. 239 с.
6. Юровіч О. Системи управління безпілотними літальними апаратами для здійснення моніторингу наземних об’єктів. Системи управління, навігації та зв’язку. Збірник наукових праць. ПНТУ, Полтава, 2018, 3(49), сс. 33-38.
7. *Aghassibake N. Supporting data visualization services in academic libraries. The Journal of Interactive Technology and Pedagogy,* 2020, № 18.
8. *Bao L., Busany N., Lo D., Maoz S. Statistical log differencing. In:* 2019 34*th IEEE/ACM International Conference on Automated Software Engineering (ASE),* 2019*, pp*. 851–862.
9. *Chen M., Mozaffari M., Saad W., Yin C., Debbah M., Hong C.S. Caching in the Sky: Proactive Deployment of Cache-Enabled Unmanned Aerial Vehicles for Optimized Quality-of-Experience. IEEE Journal of Selected Areas in Communications,* 2017*, vol.* 35*, no.* 5*, pp.* 1046-1061.
10. *Condomines J.-P., Zhang R., Larrieu N. Network intrusion detection system for UAV ad-hoc communication: From methodology design to real test validation. Ad Hoc Networks,* 2019.

11. *Du M., Li F., Zheng G., Srikumar V. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the* 2017 *ACM SIGSAC conference on computer and communications security,* 2017*, pp.* 1285–1298.
12. *Gudla C., Rana M.S., Sung A.H. Defense techniques against cyber attacks on unmanned aerial vehicles. In: Proceedings of the international conference on embedded systems, cyber-physical systems, and applications (ESCS),* 2018*, pp.* 110–116.
13. *Huang Y., Xiu J., Qiu L., Zhang R. Cognitive UAV Communication Using Joint Trajectory and Power Control. In: Proceedings of the 19th IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Kalamata, Greece, June* 2018*, pp.* 1-5.
14. *Kartashov V.M., Oleynikov V.N., Sheiko S.A., Korytsev I.V., Babkin S.I., Zubkov O.V., Anokhin M.A. Information characteristics of sound radiation of small unmanned aerial vehicles. Telecommunications and Radio Engineering,* 2018*, vol.* 77*, no.* 10*, pp*. 915-924.
15. *Ki J.S. Study on Developing a Flight Data Visualization. Journal of the Society of Korea Industrial and Systems Engineering,* 2003*, vol.* 26*, no.* 3*, pp.* 74-78*, September.*
16. *Kumar S., Nanda M., Rao P.R., Jose L.K. Model-based approach for design and development of avionics display application. In: ICETE,* 2019.
17. *Liu Q., Qiao Z., Lv Y. PyVT: A python-based open-source software for visualization and graphic analysis of fluid dynamics datasets. Aerospace Science and Technology,* 2021*, no.* 117.
18. *Makolkina M., Kirichek R., Teltevskaya V., Surodeeva E. Research of interaction between applications of augmented reality and control methods of UAVs. Lecture Notes in Computer Science (LNCS),* 2017*, vol.* 10372*, pp.* 186–193.
19. *Moses A., Rutherford M.J., Valavanis K.P. Radar-based detection and identification for miniature air vehicles. In: IEEE International Conference on Control Applications,* 2011*, September* 28-30.

20. *Shim J.-I., Jo G.-S. Flight Data Visualization and Post-test Flight Data Analysis System by Using Database. International Journal of Mechanical & Mechatronics Engineering IJMME-IJENS,* 2016*, vol.* 16*, no.* 02.
21. *Silalahi S., Ahmad T., Studiawan H. DFLER: Drone Flight Log Entity Recognizer to support forensic investigation on drone device. Softw. Impacts,* 2022.
22. *Silalahi S., Ahmad T., Studiawan H. Drone flight logs sequence mining. In:* 2022 *IEEE International Conference on Cybernetics and Computational Intelligence,* 2022*, pp.* 107–111.
23. *Unmanned Aircraft Systems (UAS): ICAO Cir 328 AN/190. ICAO, Montreal, Canada,* 2011*,* 66 *p.*
24. *Wu J., Wang H., Li N., Yao P., Huang Y., Yang H. Path Planning for Solar-Powered UAVs in Urban Environments. Neurocomputing,* 2018*, vol.* 275*, pp.* 2055-2065.
25. *Xiaojun C., Xianpeng L., Peng X. IoT-based air pollution monitoring and forecasting system. In: Computer and Computational Sciences (ICCCS). Proceedings IEEE International Conference,* 2015.

1.

Додаток А

Лістинг коду візуальної частини

```
import Chart, {CategoryScale, Chart as ChartJS, Filler,
LinearScale, LineElement, PointElement, Tooltip} from 'chart.js';
import React, { ReactElement, useEffect, useState } from 'react';
import { Line } from 'react-chartjs-2';
ChartJS.register(CategoryScale, LinearScale, PointElement, LineElement, Filler, Tooltip);
export const LineChart = ({values, hoverCallback}): ReactElement => {
const [hoveredIndex, setHoveredIndex] = useState<number | null>(null);
useEffect(() => {
const delayInputTimeoutId = setTimeout(() => {
hoverCallback && hoverCallback(hoveredIndex);
}, 10);
return (): void => {
clearTimeout(delayInputTimeoutId)}}, [hoveredIndex]);
const options = {
maintainAspectRatio: false,
scales: {
x: {
display: false,
grid: {
display: false}},
y: {
title: {
display: false},
ticks: {
display: false},
border: {
dash: [2, 4],
```

```
display: false},
grid: {
tickLength: 0,
tickWidth: 0,
borderColor: 'red',
borderWidth: 5,
color: (context: { tick: { value: number } }) => {
if (context.tick.value === 0) {
return '#771e00'}}}},
elements: {
point: {
pointRadius: 0,
pointBackgroundColor: '#771e00'}},
hoverRadius: 3,
interaction: {
mode: 'nearest',
intersect: false,
axis: 'x'},
plugins: {
tooltip: {
displayColors: false,
backgroundColor: 'rgba(1, 1, 1, 0)',
titleColor: 'rgba(1, 1, 1, 0)',
bodyColor: 'rgba(1, 1, 1, 0)'},
legend: {
display: false,
},
title: {
display: false}}};
const data = {
```

```
labels: Array.from({ length: values.length }).map((_, index) => index),
datasets: [{
data: values,
backgroundColor: context => {
const bgColor = ['rgba(255, 56, 0, 0.30)', 'rgba(255, 56, 0, 0.00)'];
if (!context.chart.chartArea) {
return}
const {ctx, chartArea: { top, bottom }} = context.chart;
const gradientBg = ctx.createLinearGradient(0, top, 0, bottom);
const colorTR = 1 / (bgColor.length - 1);
for (let i = 0; i < bgColor.length; i++) {
gradientBg.addColorStop(0 + i * colorTR, bgColor[i])}
return gradientBg},
fill: {
target: 'start'},
borderColor: ['#FF3800'],
spanGaps: true,
borderWidth: 1}]};
return (
<Line
options={options}
data={data}
plugins={[
{
id: 'mouseLine',
afterDatasetDraw(chart: Chart) {
const {
ctx,
tooltip,
scales: { x, y },
```

```
chartArea: { bottom, top },
} = chart;
if (tooltip._active.length > 0 && ctx) {
const xCoor = x.getPixelForValue(tooltip.dataPoints[0].dataIndex);
const yCoor = y.getPixelForValue(tooltip.dataPoints[0].parsed.y);
ctx?.save();
ctx?.beginPath();
ctx.lineWidth = 1;
ctx.strokeStyle = '#771e00';
ctx?.setLineDash([2, 4]);
ctx?.moveTo(xCoor, yCoor);
ctx?.lineTo(xCoor, top);
ctx?.lineTo(xCoor, bottom);
ctx?.stroke();
ctx?.closePath();
ctx?.setLineDash([]);
setHoveredIndex(tooltip.dataPoints[0].parsed.x)} else {
setHoveredIndex(null)}}}]}/>)};
```