Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Програмно-апаратний засіб для інсталяції програмного забезпечення у БПЛА
Назва теми

КвРКІ 210493.21.04.04 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент IV курсу, група КІ2-21-4 _____ Олексій ШПИЛЮК
Підпис — Ініціали, прізвище

Керівник _____ Катерина БЕРЕЗЬКА
Підпис, дата — Ініціали, прізвище

Нормоконтролер _____ Тетяна КИСІЛЬ
Підпис, дата — Ініціали, прізвище

До захисту допускаю:
зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Ольга ПАВЛОВА
Підпис — Ініціали, прізвище

«12» червня 2025 р.

Хмельницький 2025

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прий
Нормоконтроль	Тетяна КИСІЛЬ, доцент кафедри КІІС		
Антиплагіат	Андрій НІЧЕПОРУК, доцент кафедри КІІС		

7. Дата видачі завдання «_10_»_____01_____2025 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Прим
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2025	викон
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2025	викон
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2025	викон
4	Робота над розділом 2 – проектування програмно-технічного засобу	01.04.2025	викон
5	Робота над розділом 3 – програмна реалізація та тестування програмно-технічного засобу	29.04.2025	викон
6	Оформлення пояснювальної записки згідно вимог	25.05.2025	викон
7	Попередній захист ВКР	26.05.2025	викон
8	Захист ВКР на засіданні ЕК	Червень 2025 року	

Студент

Керівник роботи

Підпис Олексій Шпилюк
Ініціали, прізвище

Підпис Березька Катерина
Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 210493.21.04.04 ПЗ	Пояснювальна записка	55		
			Графічні матеріали			
2		КвРКІ 210493.21.04.04_Е8	Архітектура ПЗ проєкту	1		
3		КвРКІ 210493.21.04.04_Е8	Архітектура ПЗ для системи	1		
4		КвРКІ 210493.21.04.04_Е8	Реалізація проєкту	1		

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 210493.21.04.04 ПЗ		
Розробив		Шпилюк			Відомість проекту	Літера	Аркуш 1 / Аркушів 1
Перевір.		Березька				У	
Н. контр.		Кисіль		20.06.25		ХНУ, КІ2-21-4	
Затв.		Павлова		20.06.25			

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмно-апаратний засіб для інсталяції програмного забезпечення у БПЛА
».

Автор роботи: Олексій ШПИЛЮК.

Керівник роботи: Березька Катерина

Пояснювальна записка: 55 с., 3 дод., 43 джерел.

Графічна частина: 3 креслення.

Метою дипломної роботи є розробка програмного забезпечення для спрощеної та пришвидшеної прошивки польотних контролерів.

Об'єктом дослідження є функціональні вимоги сучасних виробництв та як можна пришвидшити процес.

Під час проведення даного дослідження був використаний метод систематичного огляду літератури для вивчення і аналізу предметної області даного дослідження з текстових джерел інформації.

Підпис студента

30.05.2025

Дата

ЗМІСТ

ВСТУП 3
1 ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ 4
1.1 Аналіз предметної області і виявлення наявних проблем і завдань .. 4
1.2 Порівняльний аналіз переваг та недоліків існуючих рішень 8
1.3 Методологічні підходи до вирішення задачі за темою дослідження 14
1.4 Постановка задачі 16
1.5 ВИСНОВКИ 17
2 ПРОЕКТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ 19
2.1 Обґрунтування вибору мов програмування та програмного забезпечення 19
2.2 Функційні вимоги програмного забезпечення 28
2.3 Проектування логіки взаємодії з контролером БПЛА 32
2.4 Проєктування файлової та параметричної структури системи 35
2.5 Вибір та опис апаратної частини 37
2.6 Висновки 41
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ 42
3.1 Принцип роботи та архітектура клієнтської частини програмного засобу 42
3.2 Реалізація інтерфейсу користувача 44
3.3 Логіка обробки даних і команд 48
3.4 Реалізація логування та повідомлень 52
3.5 Запуск та тестування веб-застосунку 53
ВИСНОВКИ 58
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ 59
ДОДАТОК А 63
ДОДАТОК Б 64
ДОДАТОК В 65

ВСТУП

У сучасних умовах стрімкого розвитку безпілотних літальних апаратів (БПЛА) особливу увагу привертає питання ефективного керування їх функціональністю за рахунок встановлення та оновлення відповідного програмного забезпечення. Програмно-апаратні засоби, що забезпечують цей процес, виступають ключовим елементом у системах автоматизованого обслуговування дронів, зокрема в умовах польових випробувань, де необхідна оперативність та надійність.

На сьогодні розроблено низку спеціалізованих конфігураторів та утиліт, які дозволяють прошивати контролери польоту, однак більшість із них орієнтовані на професійних користувачів і мають складний інтерфейс або вимагають специфічних знань. У результаті виникає потреба у створенні простого у використанні, стабільного і адаптованого до конкретних умов засобу встановлення ПЗ на БПЛА.

Метою даної дипломної роботи є розробка програмно-апаратного комплексу, що дозволяє здійснювати прошивку контролерів польоту дронів у напівавтоматичному режимі, з урахуванням специфіки польових умов експлуатації. Основна увага приділяється створенню зручного користувацького інтерфейсу, реалізації підтримки популярних форматів прошивок та забезпеченню стійкості системи до типових помилок, що виникають у процесі оновлення мікропрограм.

У ході виконання роботи буде проведено аналіз наявних рішень, розроблено програмне забезпечення, спроектовано інтерфейс користувача, а також здійснено тестування розробленого засобу на базі типових контролерів, що застосовуються в БПЛА. Запропоноване рішення має стати ефективною альтернативою існуючим інструментам і сприяти підвищенню автономності та керованості безпілотних систем.

1 ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ

1.1 Аналіз предметної області і виявлення наявних проблем і завдань

У сучасному світі безпілотні літальні апарати (БПЛА) стали невід'ємною частиною багатьох сфер діяльності, включаючи військову, цивільну та комерційну. Їх використання охоплює розвідку, спостереження, доставку вантажів, а також наукові дослідження.

Основним компонентом, що забезпечує функціональність БПЛА, є програмне забезпечення, або прошивка, яка контролює всі аспекти польоту та взаємодії з навколишнім середовищем. Прошивка визначає поведінку дрона, його стабільність, реакцію на команди та здатність виконувати складні маневри.

З розвитком технологій з'явилися різноманітні платформи для прошивки та налаштування БПЛА. Серед них варто відзначити такі програми, як Betaflight [11] та INAV [12], які дозволяють користувачам налаштовувати параметри польоту, оновлювати прошивку та оптимізувати роботу дронів відповідно до конкретних завдань.

Крім того, зростає популярність веб-застосунків для прошивки дронів, які забезпечують зручний інтерфейс та можливість оновлення програмного забезпечення без необхідності встановлення додаткових програм на комп'ютер. Це особливо актуально для користувачів, які шукають швидкі та ефективні способи обслуговування своїх БПЛА.

Таким чином, предметна область, пов'язана з прошивкою та налаштуванням БПЛА, є динамічною та вимагає постійного вдосконалення інструментів для забезпечення безпеки, ефективності та зручності використання дронів у різних сферах діяльності.

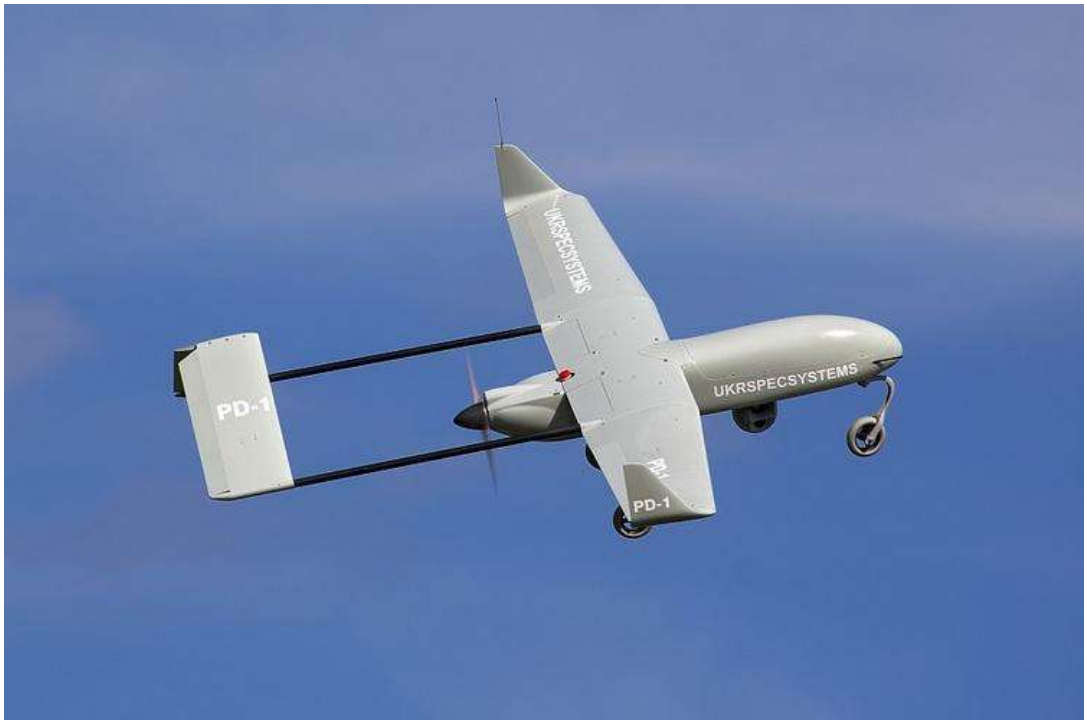


Рисунок 1.1 – Сучасний військовий БПЛА

У сфері прошивки безпілотних літальних апаратів (БПЛА) існує низка програмних рішень, які забезпечують оновлення та налаштування програмного забезпечення польотних контролерів. Серед найбільш поширених інструментів варто виділити Betaflight Configurator та INAV Configurator, які використовуються для прошивки та конфігурації відповідних прошивок на польотних контролерах.

Betaflight Configurator - це кросплатформенний додаток, який дозволяє користувачам прошивати та налаштовувати польотні контролери з прошивкою Betaflight. Процес прошивки включає підключення контролера до комп'ютера через USB, вибір відповідної версії прошивки та її завантаження на пристрій. Для успішної прошивки необхідно встановити відповідні драйвери, зокрема STM32 VCP або Silabs CP210x, залежно від типу контролера. У випадку виникнення проблем з драйверами рекомендується використовувати утиліту **ImpulseRC Driver Fixer** для автоматичного встановлення необхідних драйверів.

INAV Configura**tor** - це інструмент, призначений для прошивки та налаштування польотних контролерів з прошивкою INAV. Процедура прошивки схожа на Betaflight і включає підключення контролера в режимі **DFU** (Device

Firmware Upgrade), вибір відповідної прошивки та її завантаження. Для цього також необхідно встановити відповідні драйвери, зокрема **WinUSB**, які можна інсталювати за допомогою утиліти Zadig[31].

Таблика 1.1 – Наявні утиліти

Критерій	Betaflight Configurator	INAV Configurator
Підтримка ОС	Windows, MacOS, Linux	Windows, MacOS, Linux
Необхідність драйверів	Так	Так
Режим прошивки	DFU	DFU
Утиліти для драйверів	ImpulseRC	Zadig

Обидва інструменти мають схожий функціонал та вимоги до системи. Проте, для новачків процес встановлення драйверів та прошивки може бути складним, що підкреслює необхідність у більш інтуїтивно зрозумілих та автоматизованих рішеннях для прошивки БПЛА.

У процесі аналізу існуючих рішень для прошивки безпілотних літальних апаратів (БПЛА) було виявлено низку проблем, що ускладнюють ефективне та безпечне оновлення програмного забезпечення польотних контролерів. Ці проблеми стосуються як технічних аспектів, так і зручності використання наявних інструментів.

Основні проблеми при прошивці БПЛА:

Складність встановлення драйверів та підключення пристроїв: Процес прошивки часто вимагає встановлення специфічних драйверів, таких як STM32 VCP або Silabs CP210x, що може бути складним для користувачів без технічного досвіду. Неправильне встановлення або відсутність необхідних драйверів призводить до неможливості підключення польотного контролера до комп'ютера.

Залежність від операційної системи та сумісність: Деякі інструменти для прошивки мають обмежену підтримку різних операційних систем, що створює

труднощі для користувачів, які використовують, наприклад, macOS або Linux. Це обмежує доступність процесу прошивки для широкого кола користувачів.

Відсутність інтуїтивно зрозумілого інтерфейсу: Багато існуючих програм мають складний інтерфейс, що не забезпечує достатньої зручності для користувачів без глибоких технічних знань. Це ускладнює процес прошивки та збільшує ризик помилок.

Реалізація зазначених завдань дозволить створити ефективний інструмент для прошивки БПЛА, що відповідає сучасним вимогам та забезпечує зручність і безпеку для користувачів.

У процесі розробки веб-застосунку для прошивки безпілотних літальних апаратів (БПЛА) особливу увагу приділено формулюванню функціональних та нефункціональних вимог. Ці вимоги забезпечують відповідність програмного забезпечення очікуванням користувачів та технічним стандартам. Застосунок повинен забезпечувати підтримку широкого спектру моделей польотних контролерів, включаючи STM32, F4, F7 та інші. Інтерфейс користувача має бути інтуїтивно зрозумілим, з чіткими інструкціями для користувачів з різним рівнем технічної підготовки. Можливість оновлення прошивки через веб-інтерфейс без потреби у додатковому програмному забезпеченні є обов'язковою. Застосунок повинен автоматично розпізнавати підключені польотні контролери та надавати відповідні опції для їх прошивки. Усі дії під час прошивки мають фіксуватися в журналі для подальшого аналізу та виявлення можливих помилок. Застосунок має бути кросплатформенним, доступним через сучасні веб-браузери на різних операційних системах, таких як Windows, macOS та Linux. Процес прошивки повинен бути захищеним від несанкціонованого доступу, з використанням сучасних протоколів шифрування та автентифікації. Архітектура застосунку повинна дозволяти легке додавання нових функцій та підтримку нових моделей БПЛА в майбутньому. Застосунок повинен забезпечувати швидке завантаження та обробку даних, мінімізуючи час очікування користувача. Інтерфейс застосунку має

бути доступним на кількох мовах, включаючи українську та англійську, для забезпечення зручності користувачів з різних регіонів.

Визначені функціональні та нефункціональні вимоги створюють основу для розробки веб-застосунку, який буде відповідати потребам користувачів та забезпечить ефективний процес прошивки БПЛА.

У результаті проведеного аналізу предметної області, що охоплює процеси прошивки безпілотних літальних апаратів (БПЛА), було виявлено низку суттєвих проблем та обмежень у наявних програмних рішеннях. Зокрема, складність встановлення драйверів, обмежена кросплатформеність, відсутність інтуїтивно зрозумілого інтерфейсу та нестабільність процесу оновлення прошивки створюють бар'єри для ефективного використання БПЛА.

Для подолання зазначених проблем сформульовано вимоги до майбутнього програмного засобу, який має бути кросплатформенним, безпечним, надійним та зручним у використанні. Реалізація такого веб-застосунку дозволить спростити процес прошивки БПЛА, зменшити кількість помилок та підвищити ефективність використання дронів у різних сферах діяльності.

Таким чином, розробка нового програмного засобу для прошивки БПЛА є актуальним завданням, що сприятиме підвищенню ефективності та безпеки використання безпілотних літальних апаратів.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

Для обґрунтованого аналізу існуючих програмних засобів, призначених для прошивки безпілотних літальних апаратів (БПЛА), необхідно визначити чіткі критерії оцінювання. Ці критерії дозволяють систематизувати порівняння та виявити переваги й недоліки кожного з розглянутих рішень.

Оцінюється здатність програмного засобу працювати з різними моделями польотних контролерів, такими як STM32, F4, F7 та інші. Важливо враховувати,

наскільки швидко та ефективно програмне забезпечення адаптується до нових моделей контролерів, що з'являються на ринку.

Аналізується можливість використання програмного засобу на різних операційних системах, включаючи Windows, macOS та Linux. Кросплатформеність забезпечує ширший спектр користувачів та підвищує зручність використання.

Визначається ступінь інтуїтивності та зручності інтерфейсу. Програмне забезпечення повинно мати зрозумілу навігацію, чіткі інструкції та доступні елементи керування, що особливо важливо для користувачів з обмеженим технічним досвідом.

Оцінюється наявність функцій, таких як автоматичне виявлення підключених пристроїв, підтримка різних режимів прошивки, можливість резервного копіювання та відновлення налаштувань, а також інші додаткові опції, що підвищують ефективність роботи.

Аналізується рівень захисту даних під час процесу прошивки, наявність механізмів виявлення та запобігання помилкам, а також можливість відновлення після збоїв. Важливо, щоб програмне забезпечення забезпечувало стабільну та безпечну роботу з мінімальним ризиком для обладнання.

Розглядається частота оновлень програмного забезпечення, доступність технічної підтримки та активність спільноти користувачів. Регулярні оновлення свідчать про постійний розвиток продукту та врахування зворотного зв'язку від користувачів.

Таким чином, визначення зазначених критеріїв дозволяє здійснити всебічний аналіз існуючих програмних засобів для прошивки БПЛА та обґрунтувати вибір оптимального рішення або необхідність розробки нового програмного продукту, що відповідатиме сучасним вимогам та очікуванням користувачів.

У сфері прошивки та налаштування безпілотних літальних апаратів (БПЛА) існує кілька популярних програмних засобів, які забезпечують користувачам можливість конфігурувати польотні контролери. Серед них варто виділити Betaflight Configurator [11], INAV Configurator [12] та SpeedyBee App [13].

Betaflight Configurator є кросплатформеним застосунком для налаштування польотних контролерів з прошивкою Betaflight. Він підтримує операційні системи Windows, macOS та Linux, що забезпечує широку доступність для користувачів. Інтерфейс програми дозволяє здійснювати конфігурацію параметрів польоту, налаштування режимів, калібрування сенсорів та оновлення прошивки. Користувачі можуть змінювати напрямок обертання моторів, налаштовувати телеметрію та інші функції .

INAV Configurator призначений для налаштування польотних контролерів з прошивкою INAV. Цей інструмент також є кросплатформеним і працює як додаток у браузері Google Chrome. INAV підтримує різні типи літальних апаратів, включаючи квадрокоптери, гексакоптери, октокоптери, літаки з фіксованим крилом та ровери . Програма дозволяє користувачам налаштовувати параметри польоту, калібрувати сенсори та оновлювати прошивку.

SpeedyBee App є мобільним застосунком для пристроїв на базі iOS та Android, який дозволяє налаштовувати польотні контролери з прошивками Betaflight, INAV та EmuFlight. Застосунок підтримує підключення до контролерів через Bluetooth або OTG-кабель, що забезпечує зручність налаштування в польових умовах без потреби в ноутбуці . Користувачі можуть оновлювати прошивку, аналізувати дані чорної скриньки та налаштовувати параметри польоту безпосередньо з мобільного пристрою.

Таким чином, кожен з розглянутих інструментів має свої особливості та переваги, що дозволяє користувачам обирати найбільш відповідний варіант залежно від конкретних потреб та умов використання.

З метою обґрунтування необхідності розробки нового веб-застосунку для прошивки безпілотних літальних апаратів (БПЛА) було проведено порівняльний аналіз трьох популярних програмних засобів: Betaflight Configurator, INAV Configurator та SpeedyBee App. Аналіз здійснено за ключовими критеріями, визначеними у підрозділі 1.2.

Betaflight Configurator підтримує широкий спектр польотних контролерів, зокрема STM32 F3, F4, F7 [14], що забезпечує сумісність з більшістю сучасних БПЛА. INAV Configurator орієнтований на контролери, які використовуються в апаратах з фіксованим крилом та гібридних моделях. SpeedyBee App забезпечує підтримку контролерів з прошивками Betaflight [11], INAV [12] та EmuFlight [27], що робить його універсальним інструментом для різних типів БПЛА.

Betaflight Configurator є кросплатформеним застосунком, доступним для Windows, macOS та Linux. INAV Configurator також підтримує ці операційні системи, проте для macOS можуть виникати труднощі з запуском через відсутність підпису додатку. SpeedyBee App працює на мобільних пристроях з iOS та Android, що забезпечує зручність використання в польових умовах.

Betaflight Configurator має інтуїтивно зрозумілий інтерфейс з логічною структурою меню та візуалізацією параметрів. INAV Configurator пропонує більш складний інтерфейс, що може вимагати додаткового навчання для новачків. SpeedyBee App відзначається простотою та зручністю інтерфейсу, оптимізованого для мобільних пристроїв.

Betaflight Configurator дозволяє налаштовувати параметри польоту, калібрувати сенсори, оновлювати прошивку та аналізувати дані чорної скриньки. INAV Configurator орієнтований на налаштування функцій для апаратів з фіксованим крилом, таких як автопілот та GPS-навігація. SpeedyBee App забезпечує можливість оновлення прошивки, налаштування параметрів польоту та аналізу даних безпосередньо з мобільного пристрою через Bluetooth або OTG-кабель.

Betaflight Configurator та INAV Configurator забезпечують стабільну роботу та мають механізми захисту від помилок під час прошивки. SpeedyBee App, хоча і зручний у використанні, може мати обмеження щодо стабільності з'єднання через бездротові інтерфейси, що може впливати на надійність процесу прошивки.

Betaflight Configurator та INAV Configurator мають активні спільноти розробників та користувачів, що забезпечує регулярні оновлення та підтримку.

SpeedyBee App також оновлюється, проте частота оновлень може бути нижчою порівняно з іншими інструментами.

Таким чином, проведений порівняльний аналіз дозволяє виявити переваги та недоліки кожного з розглянутих програмних засобів, що є основою для формування вимог до нового веб-застосунку для прошивки БПЛА.

У процесі аналізу наявних програмних засобів для прошивки безпілотних літальних апаратів (БПЛА) було виявлено низку суттєвих обмежень та недоліків, які впливають на ефективність та зручність їх використання. Ці недоліки стосуються як технічних аспектів, так і користувацького досвіду, що обґрунтовує необхідність розробки нового програмного продукту, який враховуватиме виявлені проблеми.

Одним із основних недоліків Betaflight Configurator є складність процесу встановлення та оновлення програмного забезпечення. Користувачі часто стикаються з необхідністю ручного встановлення драйверів, що може бути проблематичним для осіб без технічного досвіду. Крім того, відсутність можливості запуску декількох екземплярів програми одночасно обмежує гнучкість при налаштуванні кількох пристроїв.

INAV Configurator має обмежену сумісність з різними версіями операційних систем, зокрема, останні версії не підтримують Windows 7 та вимагають додаткових дій для запуску на macOS через відсутність підпису додатку . Також спостерігаються проблеми з відображенням деяких елементів інтерфейсу, що може ускладнювати процес налаштування для користувачів.

Хоча SpeedyBee App забезпечує зручність налаштування БПЛА через мобільні пристрої, вона має обмеження щодо функціональності. Зокрема, можливість прошивки контролерів доступна лише для певних версій прошивки, а для повноцінного оновлення необхідно використовувати комп'ютер з Betaflight Configurator . Крім того, нестабільність з'єднання через Bluetooth або Wi-Fi може призводити до переривання процесу налаштування.

Таким чином, виявлені недоліки існуючих програмних засобів свідчать про необхідність створення нового веб-застосунку для прошивки БПЛА, який буде позбавлений зазначених обмежень та забезпечить більш зручний та надійний процес налаштування для користувачів з різним рівнем технічної підготовки.

У результаті проведеного аналізу існуючих програмних засобів для прошивки безпілотних літальних апаратів (БПЛА) було виявлено низку обмежень, які знижують ефективність та зручність їх використання. Зокрема, Betaflight Configurator не підтримує одночасний запуск декількох екземплярів програми, що ускладнює налаштування кількох пристроїв одночасно. INAV Configurator вимагає додаткових дій для запуску на macOS через відсутність підпису додатку та не підтримує Windows 7. SpeedyBee App має обмеження щодо функціональності, зокрема, можливість прошивки контролерів доступна лише для певних версій прошивки, а для повноцінного оновлення необхідно використовувати комп'ютер з Betaflight Configurator.

Таким чином, існує потреба у розробці нового веб-застосунку для прошивки БПЛА, який буде кросплатформенним, зручним у використанні та позбавленим зазначених обмежень. Такий застосунок дозволить спростити процес прошивки, зменшити кількість помилок та підвищити ефективність використання дронів у різних сферах діяльності.

Слід зазначити, що виявлені недоліки існуючих рішень, такі як складність встановлення драйверів, проблеми сумісності з операційними системами та обмежена гнучкість, є не стільки помилками реалізації цих програм, скільки наслідком їхньої фундаментальної архітектури. Будучи нативними десктопними або мобільними застосунками, вони неминуче стикаються з необхідністю глибокої інтеграції з операційною системою для доступу до апаратного забезпечення, що й породжує залежність від специфічних драйверів та платформних обмежень. Таким чином, вирішення цих ключових проблем лежить не в площині незначного покращення існуючих інструментів, а у принциповій зміні підходу. Перехід до архітектури веб-застосунку з використанням сучасних браузерних API, таких як

Web Serial API, дозволяє усунути саму першопричину цих недоліків, переносячи відповідальність за взаємодію з апаратним забезпеченням на рівень стандартизованого браузерного середовища. Це обґрунтовує доцільність розробки саме веб-орієнтованого програмного засобу як наступного логічного кроку в еволюції інструментів для обслуговування БПЛА.

1.3 Методологічні підходи до вирішення задачі за темою дослідження

У контексті виявлених обмежень існуючих програмних засобів для прошивки безпілотних літальних апаратів (БПЛА), виникає необхідність у розробці нового рішення, яке б забезпечувало ефективність, надійність та зручність у використанні. Основними критеріями для вибору технологій та інструментів розробки є:

Кросплатформеність: забезпечення сумісності з різними операційними системами для розширення кола користувачів.

Інтуїтивно зрозумілий інтерфейс: спрощення процесу прошивки для користувачів з різним рівнем технічної підготовки.

Надійність та безпека: мінімізація ризиків помилок під час прошивки та захист даних користувача.

Можливість розширення функціоналу: забезпечення гнучкості для додавання нових функцій у майбутньому.

З огляду на ці критерії, оптимальним рішенням є розробка веб-застосунку для прошивки БПЛА. Вибір на користь веб-застосунку був зроблений після порівняльного аналізу з іншими можливими архітектурними підходами. Традиційні нативні десктопні застосунки, хоч і пропонують максимальну продуктивність та прямий доступ до системних ресурсів, мають суттєві недоліки: вони вимагають розробки та підтримки окремих версій для кожної операційної системи (Windows, macOS, Linux), а головне – змушують кінцевого користувача проходити складний процес встановлення та налаштування специфічних драйверів, що і є однією з ключових проблем, яку покликана вирішити дана

робота. Інша альтернатива, мобільний застосунок, є привабливою завдяки високій портативності, що ідеально підходить для польових умов. Проте, цей підхід також вимагає розробки під дві окремі платформи (iOS та Android) і стикається зі значними обмеженнями мобільних операційних систем щодо прямого доступу до USB-пристроїв, що ускладнює надійну реалізацію процесу прошивки. На цьому тлі, веб-застосунок з використанням Web Serial API пропонує унікальний баланс: він забезпечує кросплатформеність "з коробки", не потребує встановлення, усуває проблему драйверів для кінцевого користувача, і при цьому надає достатньо низькорівневий доступ до апаратного забезпечення для реалізації надійного процесу інсталяції ПЗ. Використання веб-технологій дозволяє забезпечити кросплатформеність, оскільки доступ до застосунку можливий через будь-який сучасний веб-браузер. Інтерфейс користувача можна реалізувати за допомогою сучасних фреймворків, таких як React або Vue.js, що забезпечить інтуїтивність та зручність у використанні.

Для забезпечення надійності та безпеки процесу прошивки, слід впровадити механізми валідації даних, обробки помилок та резервного копіювання налаштувань перед прошивкою. Крім того, використання модульної архітектури дозволить легко розширювати функціонал застосунку, додаючи нові можливості без значних змін у базовій структурі. Таким чином, розробка веб-застосунку для прошивки БПЛА відповідає всім визначеним критеріям та дозволить створити ефективний, надійний і зручний інструмент для користувачів. Для управління процесом розробки було обрано ітеративну модель, що поєднує елементи гнучких (Agile) методологій. Робота над проєктом була розбита на короткі цикли (ітерації), кожен з яких включав етапи планування, проєктування, реалізації та тестування конкретного функціонального блоку. Такий підхід дозволив забезпечити високий рівень гнучкості та швидко адаптуватися до технічних викликів, що неминуче виникають при інтеграції програмного забезпечення з апаратними компонентами. Регулярний зворотний зв'язок наприкінці кожної ітерації давав змогу своєчасно коригувати вимоги та архітектурні рішення, мінімізуючи ризики та забезпечуючи

поступове, контрольоване нарощування функціональності системи. Це рішення забезпечить покращення процесу прошивки БПЛА та сприятиме підвищенню ефективності їх використання у різних сферах діяльності.

1.4 Постановка задачі

З метою реалізації кросплатформеного веб-застосунку для прошивки безпілотних літальних апаратів (БПЛА) було обрано набір технологій та інструментів, які забезпечують ефективну взаємодію з апаратним забезпеченням через браузер, стабільну роботу з послідовними інтерфейсами та зручність у розробці та підтримці системи.

Основні технології та інструменти

JavaScript (ES6+): використовується як основна мова програмування для реалізації логіки застосунку, включаючи обробку даних, взаємодію з апаратним забезпеченням та управління інтерфейсом користувача.

Web Serial API [2, 5]: забезпечує доступ до послідовних портів пристроїв безпосередньо з браузера, що дозволяє здійснювати прошивку та налаштування БПЛА без необхідності встановлення додаткового програмного забезпечення.

WebUSB API [23]: дозволяє взаємодіяти з USB-пристроями через браузер, що є критично важливим для роботи з контролерами БПЛА та іншими периферійними пристроями.

Vite [17]: використовується як інструмент для збирання проєкту, забезпечуючи швидке завантаження модулів та ефективну розробку.

jQuery [18]: застосовується для спрощення маніпуляцій з DOM, обробки подій та AJAX-запитів, що сприяє швидшій розробці інтерфейсу користувача.

MSP (MultiWii Serial Protocol) [3, 7]: протокол, що використовується для обміну даними між застосунком та контролером польоту БПЛА, забезпечуючи передачу команд та отримання телеметрії.

Web Workers [19]: використовуються для обробки ресурсомістких операцій, таких як парсинг файлів прошивки, у фоновому режимі, що забезпечує плавність роботи інтерфейсу користувача.

Node.js [20]: застосовується для розробки допоміжних інструментів та скриптів, необхідних для обробки даних та автоматизації процесів розробки.

Структура проєкту

Проєкт організовано у вигляді модульної структури, що сприяє легкості в підтримці та розширенні функціональності:

src/js/: містить основну логіку застосунку, включаючи обробку даних, взаємодію з апаратним забезпеченням та управління інтерфейсом користувача.

src/components/: містить компоненти інтерфейсу користувача, реалізовані з використанням Vue.js, що забезпечує реактивність та модульність інтерфейсу.

public/: містить статичні ресурси, такі як зображення, шрифти та іконки, необхідні для відображення інтерфейсу користувача.

styles/: містить CSS-файли, що визначають стильове оформлення інтерфейсу користувача.

Таким чином, обраний стек технологій та інструментів забезпечує ефективну реалізацію веб-застосунку для прошивки БПЛА, що відповідає вимогам кросплатформеності, зручності у використанні та можливості взаємодії з апаратним забезпеченням безпосередньо через браузер.

1.5 Висновки

У рамках першого розділу кваліфікаційної роботи було проведено комплексний аналіз предметної області, пов'язаної з обслуговуванням безпілотних літальних апаратів, що дозволило виявити ключові проблеми і сформулювати обґрунтовані вимоги до розробки нового програмного засобу.

Дослідження показало, що сучасні БПЛА є складними програмно-апаратними комплексами, функціональність яких значною мірою визначається

прошивкою польотного контролера. Існуючі на ринку програмні рішення, такі як Betaflight Configurator, INAV Configurator та SpeedyBee App, надають інструменти для оновлення та налаштування цих прошивок, однак їх використання пов'язане з низкою системних недоліків. Головною проблемою, виявленою в ході аналізу, є надмірна складність процесу для кінцевого користувача, особливо для тих, хто не має глибоких технічних знань. Ця складність проявляється, насамперед, у необхідності ручного встановлення специфічних драйверів (наприклад, STM32 VCP, Silabs CP210x, WinUSB), що є обов'язковою умовою для підключення польотного контролера до комп'ютера. Процес ускладнюється залежністю від операційної системи, що створює додаткові перешкоди для користувачів macOS та Linux. Порівняльний аналіз існуючих рішень виявив, що ці проблеми мають не стільки характер помилок реалізації, скільки є наслідком їхньої фундаментальної архітектури як нативних десктопних або мобільних застосунків. Такі програми неминуче вимагають глибокої інтеграції з операційною системою для доступу до апаратного забезпечення, що і породжує залежність від драйверів та платформних обмежень. Наприклад, Betaflight Configurator не дозволяє запускати декілька екземплярів програми одночасно, що обмежує гнучкість при роботі з кількома пристроями , а SpeedyBee App, хоч і пропонує зручність мобільного використання, має обмежену функціональність повноцінної прошивки, вимагаючи для цього використання комп'ютера На основі виявлених недоліків було зроблено висновок про доцільність розробки програмного засобу на базі принципово іншого архітектурного підходу — веб-застосунку. Перехід до архітектури веб-застосунку дозволяє усунути першопричину існуючих проблем, переносячи відповідальність за взаємодію з апаратним забезпеченням на рівень стандартизованого браузерного середовища. Таке рішення забезпечує кросплатформеність "з коробки", не вимагає від користувача жодних дій зі встановлення програми чи драйверів, а також гарантує, що він завжди працює з актуальною версією інструменту.

2 ПРОЕКТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

2.1 Обґрунтування вибору мов програмування та програмного забезпечення

У процесі проєктування програмно-апаратного засобу для прошивки безпілотних літальних апаратів (БПЛА) було обрано підхід реалізації функціональності у вигляді веб-застосунку, що працює безпосередньо у браузері. Такий вибір був зумовлений низкою факторів, серед яких основну роль відіграли кросплатформеність, спрощення користувацького досвіду, а також наявність сучасних браузерних API для роботи з апаратним забезпеченням.

На відміну від традиційних десктопних застосунків, веб-застосунок не потребує встановлення та оновлення вручну, що спрощує його розгортання й експлуатацію кінцевим користувачем. . Це дозволяє реалізувати повноцінний процес прошивки контролерів БПЛА в середовищі, зручному для широкого кола користувачів.

Переваги такого підходу проявляються у відмові від необхідності встановлювати драйвери або додаткові утиліти, які зазвичай потрібні при роботі з апаратними контролерами. Таким чином, забезпечується гнучкість та доступність застосунку незалежно від операційної системи (Windows, macOS, Linux), а єдиною вимогою є підтримка Web Serial API у браузері (Google Chrome, Microsoft Edge тощо).

Отже, веб-платформа була обрана як оптимальне середовище для розробки засобу прошивки, що забезпечує універсальність, портативність та технологічну сучасність, необхідну для створення інноваційного рішення у сфері обслуговування БПЛА.

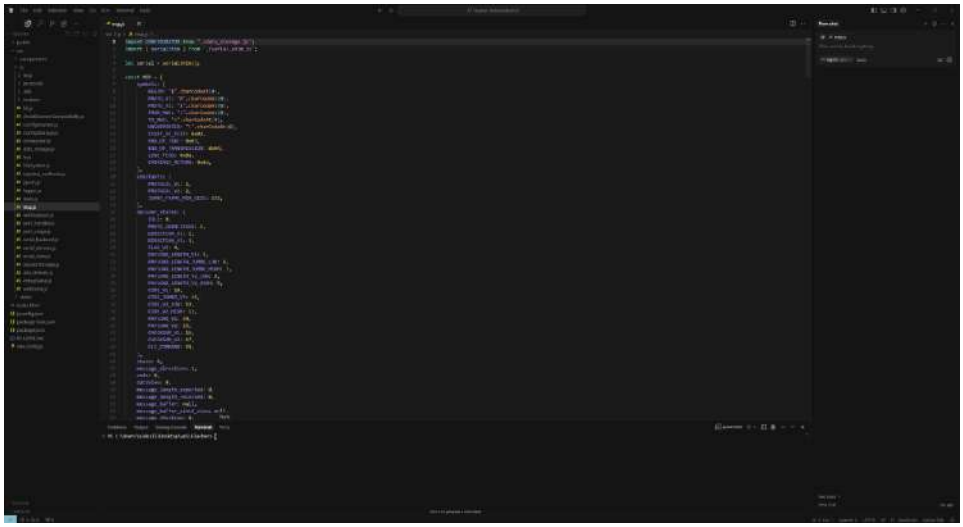


Рисунок 2.1 – Графічний інтерфейс інтегрованого середовища розробки програмного забезпечення Cursor

Для програмування засобу було обрано новітній редактор коду Cursor, що є редактором коду з функціоналом інтегрованого середовища розробки програмного забезпечення.

Cursor є дуже легким у навантаженні та простим у використанні. На рисунку 2.1 зображено графічний інтерфейс Cursor.

Для реалізації клієнтської частини програмного забезп ечення було обрано мову JavaScript (ECMAScript 6+), яка є стандартом де-факто для розробки веб-застосунків. Вибір зумовлено не лише широкою підтримкою цієї мови у сучасних браузерах, але й наявністю потужної екосистеми бібліотек, фреймворків та засобів автоматизації, що значно пришвидшують розробку інтерактивного та функціонального інтерфейсу.

JavaScript забезпечує повну інтеграцію з браузерними API, зокрема з Web Serial API та WebUSB API, що є критично важливим для взаємодії з апаратними засобами, такими як контролери польоту БПЛА. Завдяки цьому стало можливим організувати безпосередній обмін даними з мікроконтролером без необхідності встановлення проміжного програмного забезпечення чи драйверів.

Мова JavaScript також підтримує об’єктно-орієнтовану парадигму, що дозволяє модульно структурувати програмний код. У результаті кодова база

застосунку залишається масштабованою, придатною до розширення та легшою в обслуговуванні. Було використано переваги синтаксису ES6, зокрема модулі, стрілкові функції, async/await, деструктуризацію, що сприяє підвищенню читабельності й надійності коду.

Таким чином, вибір JavaScript як основної мови програмування був зумовлений як технічними, так і архітектурними перевагами, які забезпечують оптимальні умови для реалізації функціональності веб-застосунку прошивки БПЛА.

Для реалізації архітектури веб-застосунку було використано фреймворк Vue.js версії 2.7 [1, 8], що забезпечує реактивність, компонентну структуру та зручність у створенні динамічного інтерфейсу користувача. Обрану версію було визначено з урахуванням сумісності з існуючими модулями та стабільністю екосистеми Vue 2.x. Застосування Vue.js дозволило ізолювати логіку інтерфейсів у компоненти - наприклад, окремий компонент PortPicker.vue відповідає за вибір порту підключення до пристрою.

Крім Vue, у проєкті було використано бібліотеку jQuery, яка застосовується для реалізації операцій низького рівня взаємодії з DOM-елементами, обробки подій та швидкої побудови HTML-структур. Попри широку критику в сучасній розробці, включення jQuery у даному контексті було доцільним через наявність готових фрагментів коду, які використовуються для обробки станів та подій у рамках обміну даними між фронтендом і апаратним контролером.

У якості інструменту збирання було використано Vite - сучасний білдер, орієнтований на швидку розробку з модульною підтримкою ES6 та гарячим оновленням змін. Vite дозволяє зменшити час запуску середовища розробки та забезпечити високу швидкодію навіть при значному обсязі модулів. У package.json також було підключено плагін @vitejs/plugin-vue2 для повної підтримки синтаксису Vue 2 у Vite-конфігурації.

З метою підготовки до потенційного автономного використання застосунку у майбутньому, до складу проєкту також було включено плагін vite-plugin-pwa [22], що дозволяє в подальшому реалізувати офлайн-доступність засобу.

Таким чином, обрані інструменти - Vue.js, jQuery, Vite та супутні плагіни - дозволяють реалізувати масштабований, зручний у підтримці та швидкий у розробці інтерфейс користувача, який інтегрується з модулями взаємодії з апаратною частиною БПЛА. Однією з ключових особливостей розробленого веб-застосунку є його здатність здійснювати безпосередню взаємодію з апаратним забезпеченням - польотними контролерами БПЛА - через послідовний інтерфейс. З цією метою у браузерному середовищі було використано Web Serial API, що забезпечує прямий доступ до COM-портів пристроїв з використанням JavaScript.

У проєкті реалізацію доступу до серійного порту було виконано в модулі webSerial.js, де передбачено ініціалізацію порту, конфігурацію швидкості з'єднання та обробку подій зчитування/запису даних. У файлі serial_backend.js реалізовано більш високорівневу логіку взаємодії з пристроєм, включаючи обробку повідомлень, конвертацію даних у потрібний формат та організацію двостороннього обміну. Ці компоненти відповідають за формування запитів до контролера та отримання відповідей у режимі реального часу.

Для роботи з вмістом прошивки використовується окремий модуль hex_parser.js, в якому реалізовано обробку файлів у форматі HEX. Завдяки цьому модулю можлива підготовка даних до передавання в контролер у відповідному форматі з дотриманням структури сторінок пам’яті та перевіркою контрольних сум.

Особливу роль у системі взаємодії відіграє протокол MSP (MultiWii Serial Protocol), підтримка якого реалізована у модулі MSPConnector.js. Він забезпечує уніфікований обмін командами управління та діагностики між застосунком та контролером. Протокол MSP дозволяє формувати пакети даних із заголовком, довжиною та контрольними байтами, що підвищує надійність передавання.

Система обміну повідомленнями всередині застосунку реалізована через механізм eventBus.js, що дозволяє компонувально розділити UI-логіку та транспортний рівень взаємодії з обладнанням.

Таким чином, обрані інструменти та модулі забезпечують повноцінну та безпечну взаємодію із апаратною частиною БПЛА, що дозволяє здійснювати прошивку пристрою без використання сторонніх утиліт та встановлення драйверів, виключно у середовищі браузера.

З метою забезпечення ефективного процесу створення та підтримки веб-застосунку було використано низку допоміжних інструментів, що спрощують збірку, перевірку працездатності, розгортання та супровід проєкту. Зокрема, увагу зосереджено на засобах збирання, контролю версій та автоматизації типових задач розробника.

Базовою інфраструктурною одиницею проєкту виступає Vite - сучасний засіб для збирання JavaScript/TypeScript-застосунків, що забезпечує миттєвий запуск розробницького сервера (vite dev), збірку (vite build) та попередній перегляд (vite preview). У файлі package.json ці команди представлені як скрипти, що дозволяє швидко запускати потрібний режим за допомогою одного рядка у терміналі.

Для організації структури проєкту використано модульний підхід: кожна окрема підсистема винесена у відповідну директорію (components, js, workers, msp, protocols), що дозволяє легко масштабувати систему без порушення її цілісності. Всі компоненти, що мають взаємодію, пов’язані через подієву шину (eventBus.js), яка забезпечує слабкий зв’язок між модулями.

Контроль версій реалізовано з використанням Git, що є стандартом для сучасного командного розроблення. Репозиторій дозволяє фіксувати кожну зміну, повертатися до попередніх версій, а також працювати над окремими частинами системи паралельно без втрати стабільності основної гілки.

Для можливості подальшої адаптації застосунку під офлайн-режим або встановлення у вигляді псевдо-додатку, до проєкту інтегровано vite-plugin-pwa. Цей плагін дозволяє у перспективі реалізувати кешування файлів та можливість

роботи без підключення до мережі, що особливо важливо у польових умовах експлуатації БПЛА.

Таким чином, поєднання сучасних інструментів збирання, керування залежностями, контролю версій і підтримки майбутнього розширення функціональності забезпечує повноцінне середовище розробки для реалізації надійного програмного забезпечення прошивки БПЛА.

Обраний для розробки програмного засобу модульний підхід дозволяє забезпечити високий рівень гнучкості та масштабованості системи. Кожен функціональний блок реалізований у вигляді окремого модуля, що відповідає за чітко визначене коло завдань. Такий поділ не лише спрощує процес розробки та тестування окремих частин системи, але й полегшує її подальшу модифікацію та розширення. Наприклад, модуль аналізу прошивки hex_parser.js може бути доопрацьований для підтримки нових форматів файлів без значного впливу на інші компоненти, як-от модуль взаємодії з серійним портом webSerial.js. Це також покращує загальну читабельність коду та структуру проєкту, що є важливим для довгострокової підтримки.

Для забезпечення ефективної та слабкозв'язаної комунікації між різними модулями та компонентами інтерфейсу було використано подієву шину, реалізовану в модулі eventBus.js. Шина подій дозволяє модулям генерувати повідомлення про певні зміни стану або завершення операцій, на які можуть підписатися інші зацікавлені частини системи, не маючи прямих залежностей один від одного. Такий підхід мінімізує зв'язність компонентів та сприяє створенню більш гнучкої архітектури.

Яскравим прикладом використання eventBus.js у розробленому засобі є обробка події підключення до порту. Після успішного встановлення з'єднання з COM/USB-портом через Web Serial API, модуль webSerial.js генерує подію "port-connected". На цю подію реагує графічний компонент PortPicker.vue, який оновлює свій стан для відображення факту підключення та блокує кнопку вибору порту. Одночасно модуль serial_backend.js, отримавши подію, ініціює наступні кроки,

наприклад, надсилання запиту версії прошивки до підключеного контролера через MSPConnector.js. Крім того, модуль logger.js фіксує факт успішного підключення у системному журналі. Таким чином, модулі реагують на подію, не знаючи безпосередньо про існування один одного, що робить систему більш гнучкою до змін та розширення.

Основними логічними компонентами розробленого програмного засобу є такі: PortPicker.vue — компонент користувацького інтерфейсу, що відповідає за надання користувачеві можливості вибору активного COM/USB-порту для взаємодії з польотним контролером; webSerial.js — низькорівневий модуль, який інкапсулює логіку роботи з Web Serial API браузера, забезпечує ініціалізацію з'єднання з обраним портом, налаштування параметрів передачі, а також асинхронне читання та запис даних; serial_backend.js — центральний координуючий модуль, який керує логікою процесу прошивки, отримує дані від hex_parser.js, формує запити через MSPConnector.js та організовує їх послідовну передачу через webSerial.js, обробляючи відповіді від контролера.

Модуль hex_parser.js відповідає за аналіз файлу прошивки у форматі HEX, перевірку коректності структури, виділення адресних ділянок і формування послідовностей байтів, які можна безпосередньо передавати контролеру. MSPConnector.js реалізує логіку протоколу MSP, зокрема формування пакетів команд згідно зі специфікацією, обчислення контрольних сум та обробку відповідей від пристрою. logger.js забезпечує централізоване ведення журналу подій: усі операції, повідомлення про успіх, попередження та помилки, що виникають у системі, фіксуються і передаються в інтерфейс користувача. Нарешті, eventBus.js реалізує шину подій, що дозволяє згаданим модулям взаємодіяти без створення жорстких залежностей.

Розробка програмного засобу базується на модульному підході, що забезпечує високий рівень гнучкості, масштабованості та спрощує супровід системи. Кожен логічний блок інкапсульований у відповідному файлі або директорії, що дозволяє чітко розмежувати відповідальності та мінімізувати

взаємний вплив компонентів при внесенні змін. Наприклад, логіка парсингу файлів прошивки винесена в окремий потік (worker) src/js/workers/hex_parser.js, що дозволяє обробляти потенційно великі файли без блокування основного потоку інтерфейсу користувача. Можливість незалежної розробки та тестування таких модулів, як реалізація протоколу MSP в директорії src/js/msp/ (що включає MSPCodes.js, MSPConnector.js, MSPHelper.js) або підтримка різних протоколів взаємодії з пристроями в src/js/protocols/ (наприклад, webstm32.js, webusbdfu.js), значно підвищує адаптивність системи до нових вимог.

Ключовим елементом, що забезпечує слабку зв'язність та ефективну комунікацію між різними частинами застосунку, є подієва шина, реалізована в src/components/eventBus.js. Цей механізм дозволяє модулям генерувати та підписуватися на події, обмінюючись інформацією без необхідності прямих імпортів або викликів. Це особливо важливо для взаємодії між UI-компонентами (наприклад, PortPicker.vue) та модулями бізнес-логіки.

Прикладом слугує процес підключення до порту: компонент PortPicker.vue може ініціювати запит на підключення. Модуль webSerial.js, відповідальний за взаємодію з Web Serial API, після успішного встановлення з'єднання генерує подію, наприклад, "port:connected", через eventBus.js. На цю подію реагують інші модулі: той же PortPicker.vue для оновлення свого візуального стану, serial_backend.js для початку обміну даними з контролером, та logger.js для запису інформації про успішне підключення до системного журналу.

Функціонально архітектуру програмного засобу можна представити наступними основними компонентами та модулями:

- Компоненти користувацького інтерфейсу (UI) (директорія src/components/):
- PortPicker.vue: Забезпечує візуальний вибір COM/USB-порту для підключення до польотного контролера.
- eventBus.js: розташований у src/components/, виконує центральну роль подійного посередника для всього застосунку.

– init.js (src/components/): Відповідає за ініціалізацію певних UI-компонентів та загальних налаштувань інтерфейсу.

– Ядро системи та логіка (переважно в src/js/):

– main.js: Вхідна точка застосунку, де відбувається ініціалізація Vue.js та основних модулів.

– serial_backend.js: Оркеструє процес прошивки, взаємодіючи з UI, парсером прошивки, модулями протоколів та модулем серійного порту.

– connection.js: Містить узагальнену логіку управління станом з'єднання.

Взаємодія з апаратним забезпеченням (переважно в src/js/ та src/js/protocols/):

– webSerial.js: Інкапсулює взаємодію з Web Serial API для роботи з послідовними портами.

– port_handler.js, serial_devices.js, usb_devices.js: Допоміжні модулі для управління та ідентифікації доступних портів.

– Модулі протоколів (src/js/protocols/): webstm32.js, webusbdfu.js забезпечують підтримку специфічних протоколів для різних типів завантажувачів мікроконтролерів.

– Обробка даних прошивки (в src/js/workers/ та src/js/):

– hex_parser.js (src/js/workers/): Виконує ресурсоємне завдання парсингу файлів прошивки у форматі Intel HEX в окремому потоці.

– FileSystem.js (src/js/): Забезпечує взаємодію з файловою системою для завантаження файлів прошивки.

– Реалізація протоколу MSP (директорія src/js/msp/):

– MSPConnector.js: Формує та розбирає пакети даних згідно з MultiWii Serial Protocol.

– MSPCodes.js: Містить визначення кодів команд MSP.

– MSPHelper.js: Надає допоміжні функції для роботи з MSP.

Допоміжні сервіси та утиліти (переважно в src/js/ та src/js/utils/):

– logger.js: Централізоване логування подій та помилок системи.

- ConfigStorage.js, data_storage.js, SessionStorage.js: Модулі для управління конфігурацією, зберігання даних сесії та інших проміжних даних.

notifications.js (в src/js/ та src/js/utils/): Система сповіщень для користувача.

checkBrowserCompatibilty.js: Перевірка сумісності браузера з необхідними API.

Така структуризація дозволяє чітко розділити відповідальності та забезпечує гнучкість для подальшого розвитку програмного засобу. Нижче на рисунку 2.1 представлена компонентна діаграма, що візуалізує основні модулі та їхні ключові взаємозв'язки.

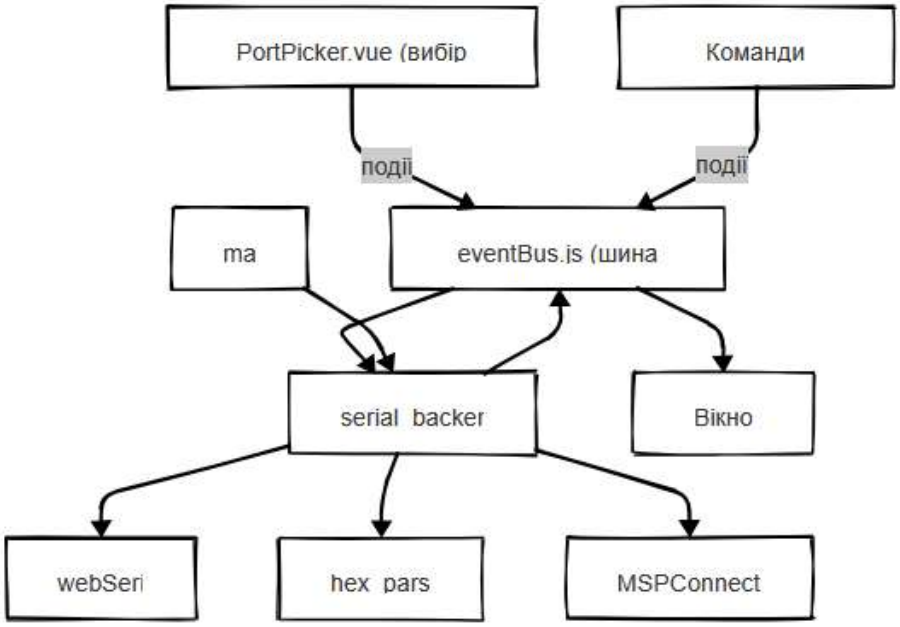


Рисунок 2.1 – Модулі та їхні ключові взаємозв'язки

2.2 Функційні вимоги програмного забезпечення

Розроблюваний програмний засіб призначений для забезпечення ефективної та надійної інсталяції програмного забезпечення, зокрема прошивок, на польотні контролери безпілотних літальних апаратів. Цей розділ детально описує ключові вимоги, які визначають функціональні можливості системи та критерії її якості,

слугуючи основою для подальшого проєктування, розробки та тестування. Для наочного представлення основних сценаріїв взаємодії користувача з системою на рисунку 2.X буде зображено відповідну діаграму варіантів використання.

Система повинна надавати користувачеві можливість керувати файлами прошивок, що включає завантаження файлів у форматі Intel HEX та їх базову перевірку на відповідність вказаному формату, з інформуванням користувача у випадку виявлення невідповідностей. Після завантаження система має аналізувати (парсити) HEX-файл, вилучаючи з нього адреси пам'яті та безпосередньо дані для запису.

Важливою частиною функціональності є взаємодія з польотним контролером. Передбачається, що система автоматично виявлятиме та відображатиме список доступних COM/USB-портів, до яких може бути підключений БПЛА, використовуючи для цього Web Serial API. Користувач матиме змогу обрати необхідний порт зі списку, після чого система повинна встановлювати та підтримувати з'єднання. Також система має вміти ініціювати переведення контролера в режим прошивки, якщо це передбачено його апаратними та програмними можливостями, наприклад, через специфічні MSP-команди.

Центральною функцією є сам процес інсталяції програмного забезпечення. Користувач повинен мати можливість ініціювати цей процес після вибору файлу та успішного підключення до контролера. Система має поблочно передавати підготовлені дані прошивки на польотний контролер. Ця передача має здійснюватися з використанням протоколу MSP (MultiWii Serial Protocol) або іншого сумісного протоколу, що забезпечує надійну доставку та контроль цілісності даних, наприклад, через перевірку підтверджень від контролера про успішний запис кожного блоку. Користувачеві також має бути надана можливість скасувати процес прошивки на будь-якому етапі до його повного завершення, якщо це технічно можливо з боку контролера.

Для забезпечення прозорості роботи та діагностики, система повинна надавати користувачеві актуальний зворотний зв'язок. Це включає відображення в

реальному часі детального статусу процесу прошивки, поточного етапу (наприклад, підключення, стирання, запис, верифікація) та загального прогресу виконання. Користувач має отримувати чіткі та зрозумілі повідомлення про успішне завершення операцій або про виникнення помилок із зазначенням можливих причин. Усі ключові операції, системні події, попередження та помилки повинні фіксуватися у файлі журналу (лозі) з часовими мітками та можливістю його перегляду в інтерфейсі системи.

Окрім функціональних аспектів, важливими є нефункціональні вимоги. Щодо продуктивності, час ініціалізації системи та виявлення портів не повинен перевищувати 5 секунд, а швидкість завантаження та парсингу файлу прошивки середнього розміру (наприклад, 512 КБ) – 10 секунд. Інтерфейс користувача має залишатися відгукливим під час виконання фонових операцій, що досягається використанням Web Workers для парсингу та асинхронних операцій для Web Serial API.

Надійність системи передбачає стабільну роботу та коректне завершення процесу прошивки за умови справності обладнання. Система повинна адекватно обробляти помилки, такі як втрата з'єднання, помилки читання/запису порту чи файлів, невідповідність відповідей контролера, та у випадку критичного збою безпечно завершувати взаємодію з контролером.

Сумісність програмного засобу має забезпечуватися з останніми стабільними версіями веб-браузерів Google Chrome та Microsoft Edge, що підтримують Web Serial API, а також роботу на операційних системах Windows 10/11, macOS та популярних дистрибутивах Linux без необхідності встановлення специфічних драйверів. Система орієнтована на сумісність з польотними контролерами на базі мікроконтролерів STM32 F4/F7, що підтримують прошивку через послідовний порт та протокол MSP.

Аспекти безпеки включають базову перевірку файлу прошивки для зменшення ризику завантаження невідповідного ПЗ та використання механізмів контролю цілісності даних, як-от контрольні суми в протоколі MSP, для

запобігання спотворенню даних під час передачі. Додатково, в рамках вимог безпеки, система повинна реалізовувати механізми валідації вхідних даних на рівні парсера прошивки. Це включає перевірку не тільки контрольних сум, але й відповідності типів записів та діапазонів адрес специфікації формату Intel HEX. Такий підхід має на меті унеможливити обробку зловмисно сформованих файлів, які можуть містити аномально великі блоки даних або некоректні адресні вказівники, здатні викликати відмову в обслуговуванні (DoS) на рівні веб-застосунку або, в гіршому випадку, призвести до непередбачуваної поведінки мікроконтролера

Зручність використання (Usability) є пріоритетом: графічний інтерфейс має бути простим, логічним, інтуїтивно зрозумілим та не перевантаженим. Користувач повинен отримувати чіткий візуальний зворотний зв'язок на свої дії, а термінологія в інтерфейсі має бути зрозумілою.

Нарешті, підтримуваність та розширюваність системи забезпечуються добре структурованою, модульною кодовою базою, що полегшує розуміння, модифікацію та виправлення помилок. Архітектура повинна дозволяти відносно легко додавати підтримку нових типів польотних контролерів, протоколів зв'язку або форматів файлів прошивок у майбутньому.

На рисунку 2.2 наведено загальну функціональну структуру взаємодії компонентів системи:

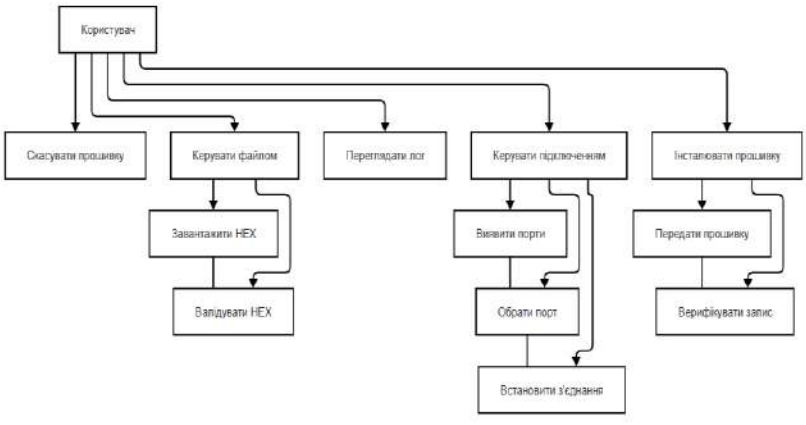


Рисунок 2.2 – Функціональна структура взаємодії компонентів системи

2.3Проектування логіки взаємодії з контролером БПЛА

Ефективна та надійна інсталяція програмного забезпечення на польотний контролер БПЛА є ключовою функцією розроблюваного програмного засобу. Цей розділ детально описує спроєктовану логіку взаємодії між веб-застосунком та апаратним забезпеченням контролера, включаючи послідовність операцій, залучені програмні модулі та використовувані протоколи. Основою для комунікації слугує Web Serial API, що дозволяє браузеру безпосередньо взаємодіяти з COM/USB-портом, до якого підключено контролер.

Процес взаємодії розпочинається після того, як користувач обрав відповідний COM/USB-порт через компонент інтерфейсу PortPicker.vue та ініціював підключення, яке встановлюється модулем webSerial.js. Після успішного встановлення з'єднання та завантаження користувачем файлу прошивки (наприклад, у форматі HEX через FileSystem.js), наступним кроком є підготовка та передача даних прошивки.

Центральну роль у координації цього процесу відіграє модуль serial_backend.js. Він отримує дані прошивки, попередньо оброблені модулем hex_parser.js, який аналізує HEX-файл, вилучає адреси та байти даних, перевіряє контрольні суми рядків та формує послідовні блоки, готові для запису в пам'ять мікроконтролера.

Для безпосередньої комунікації з польотним контролером та передачі команд і даних прошивки використовується протокол MSP (MultiWii Serial Protocol). Формування та розбір пакетів даних згідно зі специфікацією MSP реалізовано в модулі MSPConnector.js. Цей модуль відповідає за створення MSP-повідомлень, що містять команди для контролера (наприклад, ініціація режиму прошивки, запит на стирання сектору пам'яті, передача блоку даних, запит статусу операції) та власне дані прошивки. MSPConnector.js також обробляє відповіді від контролера,

перевіряючи їх коректність та контрольні суми. Коди команд MSP визначені у MSPCodes.js.

Послідовність взаємодії модулів під час інсталяції прошивки детально зображена Файл прошивки, завантажений через FileSystem.js, передається до hex_parser.js для аналізу. Оброблені дані надходять до serial_backend.js, який, використовуючи MSPConnector.js, формує MSP-пакети. Ці пакети потім передаються на фізичний рівень через модуль webSerial.js, який надсилає їх до підключеного польотного контролера.

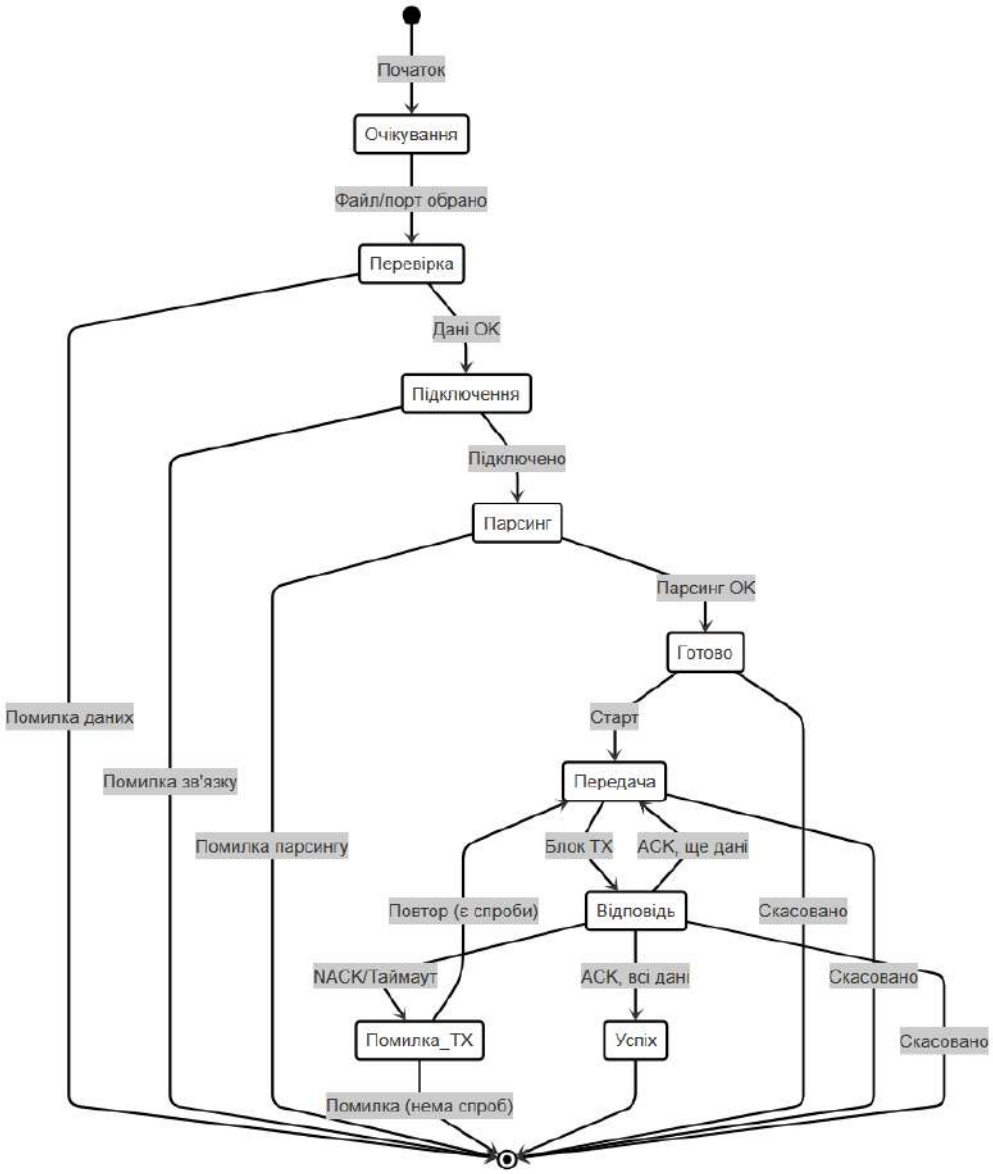


Рисунок 2.3 – Діаграма станів застосунку

Загальний алгоритм процесу прошивки контролера, представлений на рисунку 2.3 включає наступні кроки:

Ініціалізація системи та вибір порту: Користувач обирає активний COM/USB-порт, після чого система перевіряє та встановлює з'єднання.

Завантаження та парсинг файлу прошивки: Користувач завантажує .hex файл прошивки. Модуль hex_parser.js аналізує цей файл, перевіряє його структуру та розбиває на блоки даних, придатні для передачі.

Підготовка та передача блоків даних: Кожен блок даних прошивки інкапсулюється в MSP-пакет (за допомогою MSPConnector.js) та передається через webSerial.js на польотний контролер.

Отримання відповіді та контроль: Після передачі кожного блоку система очікує на відповідь (підтвердження) від контролера. У випадку отримання позитивного підтвердження, передається наступний блок.

Обробка помилок та повторні спроби: Якщо виникають збої (наприклад, відсутність відповіді, негативне підтвердження, помилка контрольної суми), система може реалізувати механізм повторної передачі блоку або, після декількох невдалих спроб, аварійно завершити процес прошивки, повідомивши користувача про помилку.

Завершення процесу: Після успішної передачі всіх блоків прошивки та отримання фінального підтвердження від контролера, процес інсталяції вважається завершеним, про що користувач також інформується.

Така архітектура взаємодії забезпечує поблоковий контроль над процесом передачі даних, дозволяє обробляти помилки на кожному етапі та підвищує загальну надійність інсталяції програмного забезпечення на БПЛА. Вся комунікація між UI-компонентами та модулями бізнес-логіки, що керують цим процесом, здійснюється через подієву шину eventBus.js, що забезпечує слабку зв'язність компонентів системи.

2.4Проєктування файлової та параметричної структури системи

Ефективна робота програмного засобу для інсталяції програмного забезпечення у БПЛА вимагає продуманої організації зберігання як самих файлів прошивок, так і конфігураційних та сесійних даних. Для забезпечення простоти архітектури, гнучкості та можливості автономної роботи застосунку без залежності від зовнішніх баз даних або серверних API було обрано підходи до управління даними, що базуються на файловій обробці та використанні браузерних механізмів зберігання.

Основним об'єктом даних, з яким працює система, є файл прошивки мікроконтролера, зазвичай у форматі HEX. Процес обробки такого файлу розпочинається із завантаження користувачем файлу зі свого локального сховища, за що відповідає модуль FileSystem.js. Цей модуль забезпечує читання вмісту файлу та його передачу для подальшого аналізу та структурування, яке виконує модуль hex_parser.js, розташований, як зазначено у структурі проєкту, в директорії src/js/workers/ для асинхронної обробки. hex_parser.js здійснює детальний розбір кожного рядка файлу, перевіряє його відповідність специфікації формату HEX, вилучає адреси пам'яті, байти даних та перевіряє контрольні суми рядків. Результатом роботи парсера є структурований набір послідовних блоків даних, готових для передачі безпосередньо в пам'ять мікроконтролера. Такий підхід дозволяє заздалегідь підготувати дані та мінімізувати обчислення під час безпосередньої взаємодії з контролером.

Для налаштування поведінки програмного засобу та адаптації до різних умов використання передбачено зберігання конфігураційних параметрів у модулі ConfigStorage.js. Тут можуть зберігатися значення за замовчуванням, такі як швидкість передачі даних через серійний порт, значення таймаутів для очікування відповіді від контролера, налаштування рівня деталізації логування та параметри відображення повідомлень для користувача. Ці параметри завантажуються при старті застосунку.

Для забезпечення безперервності роботи та збереження проміжного стану взаємодії з користувачем і пристроєм використовуються механізми сесійного зберігання. Інформація про поточний стан підключення до польотного контролера, обраний користувачем COM/USB-порт, а також ознаки валідності завантаженої прошивки зберігаються за допомогою модуля data_storage.js. Для тимчасового зберігання даних протягом активного сеансу роботи користувача з веб-застосунком використовується SessionStorage.js, який надає зручний інтерфейс для роботи з браузерним sessionStorage. Такий підхід дозволяє кешувати критично важливі параметри без необхідності їх передачі на зовнішні сервери, що забезпечує високий рівень приватності та автономності роботи застосунку.

Невід'ємною частиною системи є механізм логування, реалізований у модулі logger.js. Цей модуль відповідає за фіксацію всіх ключових етапів роботи застосунку, включаючи події взаємодії з користувачем, статус підключення до контролера, етапи передачі даних прошивки, а також повідомлення про помилки, попередження та інформацію про успішне завершення операцій. Логи, що генеруються, можуть відображатися в інтерфейсі користувача, надаючи йому детальну інформацію про хід роботи системи.

Обрана модель управління даними та параметрами, що базується на файловій обробці прошивок та використанні браузерних механізмів зберігання для конфігураційних і сесійних даних, забезпечує необхідну гнучкість, мінімізує зовнішні залежності та робить систему простою у розгортанні та використанні, особливо в умовах, де доступ до мережевих ресурсів може бути обмеженим

На рівні проєктування, дані, якими оперують ці модулі, мають чітку структуру. Конфігураційні параметри, що зберігаються у ConfigStorage.js, представляють собою статичний об'єкт JavaScript, де ключами є назви параметрів, а значеннями – їхні встановлені величини, наприклад: { baudRate: 115200, mspTimeout: 250, retryAttempts: 3 }. Це дозволяє легко отримувати доступ до будь-якого налаштування за його ключем з будь-якого модуля системи. Натомість, проміжні параметри та стан системи, що зберігаються у data_storage.js, є

динамічним реактивним об'єктом, який відображає поточну ситуацію. Його структура може включати поля, такі як { isConnected: false, selectedPortInfo: null, firmwareFile: { name: 'firmware.hex', size: 512000, isValid: true }, currentStatusMessage: 'Готово до підключення' }. Централізоване зберігання такого об'єкта стану дозволяє різним компонентам інтерфейсу (наприклад, кнопкам або індикаторам) реактивно оновлюватися при зміні будь-якого з цих полів, що є однією з ключових переваг використання фреймворку Vue.js..

2.5 Вибір та опис апаратної частини

Для забезпечення успішної інсталяції програмного забезпечення на безпілотні літальні апарати, розроблюваний програмний засіб орієнтований на взаємодію з поширеними та доступними апаратними платформами, що використовуються в сучасних БПЛА. Ключовим апаратним елементом, з яким взаємодіє система, є мікроконтролер польотного контролера.

Програмний засіб спроєктовано з урахуванням сумісності з мікроконтролерами сімейства ARM Cortex-M, зокрема, широко розповсюдженими STM32 F4 / F7. Ці мікроконтролери використовуються у більшості популярних польотних контролерів від таких виробників, як Matek, Kakute, SpeedyBee, Holybro та інших. Такий вибір забезпечує широке охоплення потенційно сумісних пристроїв.

Підключення польотного контролера до комп'ютера користувача, на якому запущено веб-застосунок, здійснюється через стандартний інтерфейс USB-UART. Це дозволяє встановити послідовний канал зв'язку, необхідний для передачі команд та даних прошивки. Важливою особливістю розробленого програмного засобу є використання Web Serial API, що надається сучасними веб-браузерами. Ця технологія дозволяє веб-застосунку напряму взаємодіяти з COM/USB-портом операційної системи без необхідності встановлення користувачем спеціалізованих

VCP-драйверів (Virtual COM Port), що значно спрощує процес налаштування та використання програмного засобу.

Для ініціації процесу прошивки польотні контролери повинні підтримувати відповідні режими завантажувача (bootloader). Розроблюваний засіб орієнтований на підтримку стандартних механізмів завантаження прошивки, таких як DFU (Device Firmware Upgrade) [15], BLHeli bootloader [16] або завантаження через MSP (MultiWii Serial Protocol) [3], якщо контролер підтримує такий режим для оновлення свого програмного забезпечення. При цьому, архітектура програмного засобу спроєктована таким чином, щоб бути гнучкою до конкретної реалізації завантажувача на стороні польотного контролера. Система не робить жорстких припущень щодо апаратної реалізації, а покладається на стандартизовані протоколи взаємодії. Це означає, що якщо польотний контролер коректно ідентифікує себе в операційній системі як віртуальний COM-порт та підтримує відповідний протокол команд (наприклад, MSP), розроблений засіб зможе з ним взаємодіяти, незалежно від конкретного виробника або моделі. Такий підхід забезпечує високий рівень сумісності та довговічність розробки, оскільки вона не прив'язана до специфічних апаратних рішень, а до відкритих стандартів комунікації.

Важливо розуміти практичні відмінності між цими режимами завантажувача (bootloader) та їх вплив на архітектуру програмного засобу. Режим DFU (Device Firmware Upgrade) є стандартним протоколом, вбудованим у ROM-пам'ять багатьох мікроконтролерів STM32. Він забезпечує швидкий та низькорівневий доступ до пам'яті пристрою, але зазвичай вимагає переведення контролера у спеціальний режим (часто за допомогою фізичної кнопки "boot") та використання іншого стеку драйверів на стороні операційної системи (наприклад, WinUSB), що краще підходить для взаємодії через WebUSB API, а не Web Serial API. На противагу цьому, завантаження через протокол MSP або інші кастомні завантажувачі, що працюють поверх віртуального COM-порту (VCP), є більш високорівневим підходом. Він не вимагає від користувача перемикання режимів

вручну, оскільки команда на перезавантаження в режим прошивки може бути надіслана через той самий послідовний порт, що й звичайні команди. Саме цей підхід є більш зручним для користувача і краще узгоджується з використанням Web Serial API. Розроблений програмний засіб фокусується саме на цьому другому підході для забезпечення максимальної простоти та універсальності, хоча архітектурно система є достатньо гнучкою для потенційної інтеграції підтримки DFU через WebUSB у майбутньому.

Сумісні контролери, як правило, мають можливість переходу в режим прошивки або автоматично при виявленні відповідного запиту від програмного засобу, або шляхом виконання користувачем певних дій, наприклад, натискання кнопки "boot" на платі контролера під час підключення. Пристрої такого класу дозволяють завантажувати файли прошивки у форматах .hex або .bin за стандартними протоколами обміну, логіку яких реалізовано в програмних модулях веб-застосунку, зокрема, з використанням MSP для передачі даних.

Проєктування програмного засобу також враховує необхідність гнучкого налаштування параметрів послідовного з'єднання для забезпечення стабільної комунікації з різними апаратними конфігураціями. Хоча більшість сучасних польотних контролерів використовують стандартну швидкість передачі даних, наприклад, 115200 біт/с, реалізована архітектура передбачає можливість зміни цього та інших параметрів, таких як кількість бітів даних, наявність біта парності та кількість стоп-бітів. Ця конфігурація задається при виклику методу port.open() об'єкта Web Serial API. Такий підхід дозволяє системі адаптуватися до вимог специфічних або застарілих пристроїв, де можуть використовуватися нестандартні налаштування зв'язку. На рівні програмної реалізації це означає, що модуль ConfigStorage.js містить значення цих параметрів за замовчуванням, які використовуються модулем webSerial.js під час ініціалізації порту, що забезпечує як простоту використання для типових сценаріїв, так і гнучкість для нестандартних випадків.

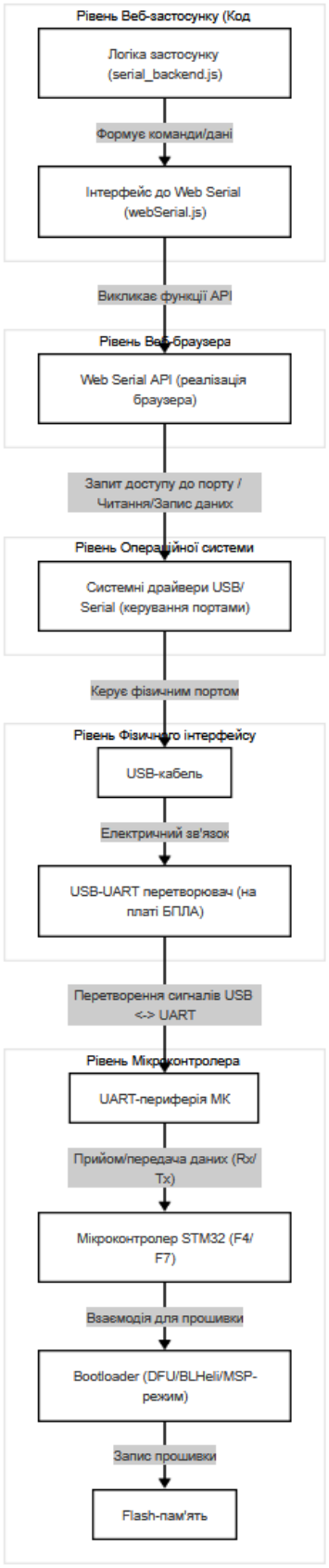


Рисунок 2.4 – Блок схема взаємодії програмного забезпечення

Спрощена блок-схема взаємодії програмного забезпечення з апаратною частиною показана на рисунку 2.4. Вона ілюструє, як веб-застосунок, що виконується у браузері, через послідовний інтерфейс (Serial), реалізований за допомогою Web Serial API, передає дані прошивки (Hex-файл) на мікроконтролер STM32 F4/F7, який, у свою чергу, записує ці дані у свою Flash-пам'ять. Хоча для пристроїв на STM32 із завантажувачем DFU також можлива робота через WebUSB API, в рамках поточної реалізації основний акцент зроблено на Web Serial API як на більш універсальному та стабільному варіанті для широкого спектра польотних контролерів, що працюють через віртуальний COM-порт.

2.6 Висновки

У ході проєктування програмно-апаратного засобу для прошивки безпілотних літальних апаратів було обґрунтовано вибір технологічного стеку, інструментів розробки та визначено функціональні вимоги до системи. Особливу увагу приділено підтримці Web Serial API, що забезпечує прямий доступ до апаратного забезпечення через веб-браузер без необхідності встановлення сторонніх драйверів.

Розроблено логіку взаємодії з польотним контролером, у якій реалізовано багаторівневу обробку: від завантаження та розбору прошивки до передачі даних у структурованому вигляді з підтвердженням на рівні протоколу. Показано механізми обробки збоїв, повторних спроб та логування дій користувача.

Спроєктовано внутрішню файлову структуру системи, що дозволяє зберігати конфігураційні параметри, сеансові дані. Відмова від використання повноцінної бази даних забезпечила простоту розгортання системи та підвищену автономність.

У результаті реалізовано гнучку, модульну архітектуру, що відповідає сучасним вимогам до веб-застосунків, інтегрованих із фізичними пристроями. Це створює передумови для переходу до етапу безпосередньої реалізації інтерфейсу та розробки функціональних компонентів, описаних у наступному розділі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

3.1 Принцип роботи та архітектура клієнтської частини програмного засобу

Програмний засіб для інсталяції програмного забезпечення у БПЛА реалізовано як веб-застосунок, що використовує сучасні браузерні технології для взаємодії з апаратним забезпеченням. Його клієнтська частина побудована на базі фреймворку Vue.js версії 2.7, а для організації та збирання проєкту застосовано інструмент Vite. Файлова структура проєкту логічно організована для забезпечення модульності та зручності розробки. Конфігураційні файли, такі як vite.config.js та package.json, розташовані в кореневій директорії разом із стартовою HTML-сторінкою index.html. Статичні ресурси, включаючи зображення, шрифти та іконки, розміщені в директорії public, а стилі інтерфейсу згруповані в директорії styles, де знаходяться основні CSS-файли main.css та secondary.css. Основна логіка застосунку зосереджена в директорії src/js, яка містить окремі JavaScript-модулі, згруповані за їхнім функціональним призначенням. Наприклад, директорія src/js/msp/ містить модулі для роботи з протоколом MSP (MSPCodes.js, MSPConnector.js, MSPHelper.js), директорія src/js/protocols/ призначена для реалізації специфічних протоколів взаємодії (websocket.js, webstm32.js, webusbdfu.js), а директорія src/js/workers/ містить допоміжні процеси, зокрема hex_parser.js для аналізу файлів прошивок. Компоненти користувацького інтерфейсу, розроблені на Vue.js, розташовані в директорії src/components/, яскравим прикладом яких є PortPicker.vue. Взаємодія між різними модулями та компонентами відбувається через подієву шину eventBus.js, що також знаходиться в src/components/. Принцип роботи програмного засобу від моменту запуску до

початку інсталяції прошивки можна описати наступною послідовністю активації та взаємодії ключових модулів

Запуск веб-застосунку (main.js): Процес ініціалізації інтерфейсу розпочинається з основного модуля main.js. Цей файл є центральним вузлом, що забезпечує завантаження Vue-компонентів, підключення eventBus.js, ініціалізацію модуля serial_backend.js та підготовку до взаємодії з Web Serial API. Саме звідси активуються всі ключові залежності, зокрема init.js (з src/components/), а також здійснюється підключення головного інтерфейсу до index.html. Таким чином, main.js виступає координатором запуску всіх компонентів застосунку та основою його початкової активації.

Ініціалізація середовища (init.js, checkBrowserCompatibilty.js): Наступним кроком є перевірка середовища виконання та сумісності браузера. Модуль init.js, розташований у src/components/, виконує системну перевірку, використовуючи checkBrowserCompatibilty.js (з src/js/) для виявлення підтримки Web Serial API у поточному браузері. У разі відсутності необхідної підтримки, користувачеві відображається відповідне попередження. Роль init.js полягає в оцінці готовності браузера до взаємодії з апаратним забезпеченням, що є критичною передумовою для коректної роботи функціоналу прошивки.

Вибір порту для з'єднання (PortPicker.vue, serial_devices.js, webSerial.js, notifications.js): Після успішного проходження перевірки сумісності, користувачеві пропонується обрати порт для з'єднання з польотним контролером. Компонент PortPicker.vue відповідає за відображення списку доступних COM/USB портів. Для отримання цього списку та ініціалізації підключення активуються модулі serial_devices.js (з src/js/), що відповідає за виявлення пристроїв, та webSerial.js (з src/js/), який реалізує безпосередню взаємодію з Web Serial API. При натисканні кнопки оновлення списку відбувається повторне сканування доступних портів. Успішне встановлення з'єднання підтверджується користувачеві через модуль сповіщень notifications.js (з src/js/ або src/js/utils/). Отже, PortPicker.vue формує

інтерфейс для вибору пристрою, тоді як інші модулі забезпечують технічну реалізацію сканування, підключення та інформування.

Завантаження та підготовка файлу прошивки (FileSystem.js, hex_parser.js): Після вибору порту та встановлення з'єднання користувач має можливість завантажити файл прошивки. Модуль FileSystem.js (з src/js/) дозволяє обрати локальний .hex файл. Далі цей файл обробляється модулем hex_parser.js (з src/js/workers/), який виконує аналіз його структури, розбиває на блоки даних, перевіряє контрольні суми та готує дані у форматі, придатному для подальшої передачі на мікроконтролер, наприклад, через протокол MSP. Функціональність обробки файлу прошивки автоматизована та спрямована на підготовку коректних даних для інсталяції.

Передача прошивки до БПЛА (serial_backend.js, MSPConnector.js, webSerial.js): На завершальному етапі підготовки відбувається фактична передача прошивки до підключеного БПЛА. Модуль serial_backend.js (з src/js/) координує цей процес. Він використовує MSPConnector.js (з src/js/msp/) для формування MSP-пакетів, що містять блоки даних прошивки. Фізичне надсилання цих пакетів через обраний COM/USB-порт реалізовано у модулі webSerial.js. Після передачі кожного блоку даних система очікує на відповідь-підтвердження (ACK/NACK) від контролера. У випадку помилки передачі, serial_backend.js може ініціювати повторні спроби. Таким чином, процес прошивки виконується поетапно з контролем успішності передачі кожного блоку, що забезпечує надійність інсталяції програмного забезпечення.

3.2 Реалізація інтерфейсу користувача

Інтерфейс користувача (UI) розробленого програмного засобу спроєктований з акцентом на простоту, інтуїтивність та надання користувачеві чіткого зворотного зв'язку на всіх етапах інсталяції програмного забезпечення на БПЛА. Основою для

побудови UI слугує фреймворк Vue.js версії 2.7, що забезпечує реактивність, компонентний підхід до розробки та ефективне управління станом інтерфейсу.

Загальна структура сторінки веб-застосунку визначена у файлі index.html, який містить базовий контейнер div з ідентифікатором id="app". Вся подальша візуалізація та логіка інтерфейсу керується через Vue.js, зокрема через головний компонент застосунку, де згруповані ключові елементи керування

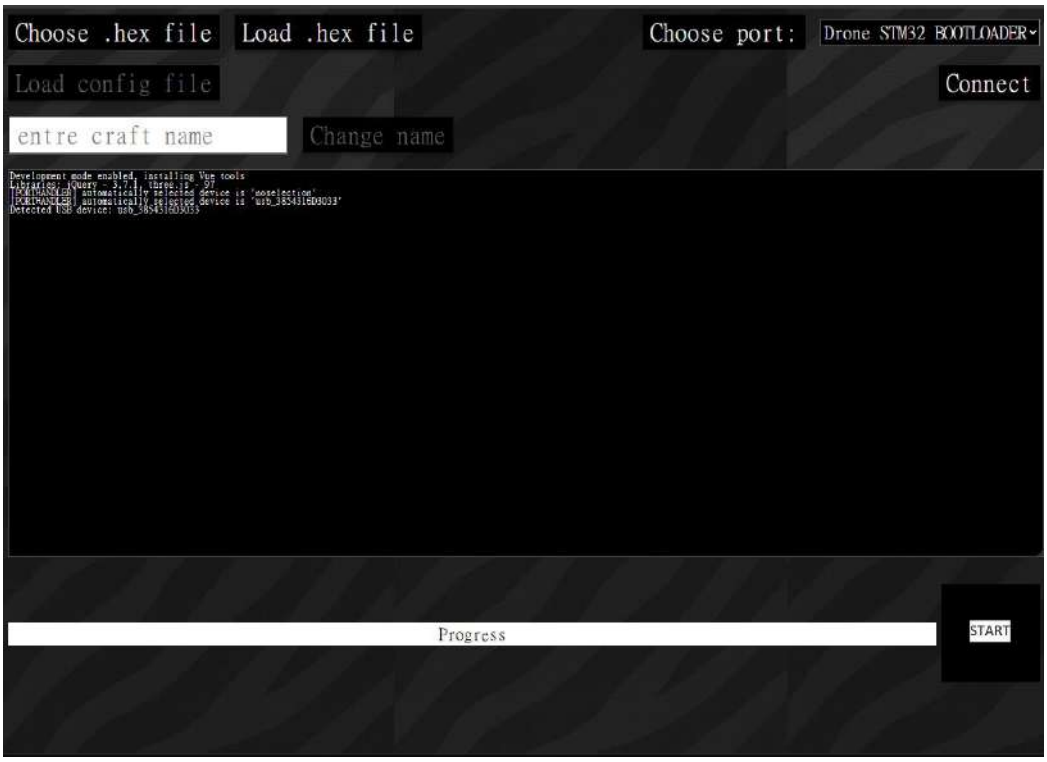


Рисунок 3.1 – Загальний вигляд головного вікна веб-застосунку

Ключовими функціональними блоками та відповідними їм UI-компонентами або елементами є елемент для завантаження файлу прошивки, що ініціюється кнопкою load_hex_file; Vue-компонент PortPicker.vue для вибору порту підключення; кнопка load_conf_file для завантаження конфігураційного файлу; кнопка connectbutton для встановлення з'єднання з пристроєм; елементи для зміни назви пристрою changeCraftName; кнопка flash_firmware для запуску процесу прошивки; текстове поле console для відображення логів та повідомлень; та індикатор progress_bar для візуалізації ходу виконання процесу прошивки.

Більшість цих компонентів не приймають вхідні параметри (props), а оперують глобальними або локальними станами, такими як connected, firmwareLoaded, progress та logMessages . Вони активно взаємодіють із системою через подієву шину eventBus.js, генеруючи події на кшталт "port-selected", "hex-loaded", "start-flash", "log-message", а також підписуючись на події для оновлення власного стану, наприклад, статусу з'єднання або нових повідомлень у лозі . Типовим прикладом такої взаємодії є кнопка flash_firmware, яка після натискання надсилає подію "start-flash", що обробляється модулем serial_backend.js для ініціації передачі прошивки .

Структурно, у головному компоненті Vue.js відображаються логічно організовані групи елементів: блок з кнопкою для завантаження прошивки, блок з компонентом PortPicker.vue, блок з кнопкою connectbutton, індикатор прогресу progress_bar та консоль повідомлень console . Ці блоки можуть бути згруповані за допомогою CSS-класу button_group для забезпечення візуальної узгодженості та вирівнювання елементів, наприклад, в одному рядку.

Динамічне відображення та доступність елементів інтерфейсу реалізовано за допомогою директив Vue.js, таких як v-if або v-show, що залежать від поточного стану застосунку. Наприклад, доступність кнопки запуску прошивки (flash_firmware), видимість індикатора прогресу (progress_bar) або поля для введення нової назви пристрою контролюються такими реактивними прапорами, як isConnected (стан підключення) або firmwareLoaded (файл прошивки завантажено). Зокрема, кнопка інсталяції прошивки стає активною лише після успішного завантаження файлу прошивки та вибору активного порту, а індикатор прогресу з'являється виключно під час безпосередньої передачі даних на пристрій

Типовий сценарій взаємодії користувача з інтерфейсом виглядає наступним чином: користувач натискає кнопку load_hex_file, що відкриває стандартний діалог вибору файлу. Після вибору .hex файлу, він обробляється у фоновому режимі модулем hex_parser.js, а інформація про успішне завантаження (подія "hex-loaded") фіксується в системному журналі через logger.js. Далі користувач обирає COM/USB-порт у компоненті PortPicker.vue. Ця дія ініціює процес сканування

доступних пристроїв модулем serial_devices.js та спробу ініціалізації обраного порту через webSerial.js. Після успішного вибору порту кнопка connectbutton стає активною, і після встановлення з'єднання з пристроєм генерується подія "port-connected". За умови успішного підключення та завантаженого файлу прошивки, кнопка flash_firmware стає доступною для натискання. Ініціація процесу прошивки цією кнопкою запускає передачу блоків даних на контролер . Усі ключові події, статуси операцій та можливі помилки фіксуються та відображаються в текстовому полі console, а стан виконання процесу прошивки динамічно оновлюється на індикаторі progress_bar .

Реактивність інтерфейсу, що є однією з ключових переваг Vue.js, активно використовується для забезпечення актуального відображення стану системи. Наприклад, індикатор progress_bar прив'язаний до реактивного поля progress (наприклад, через v-bind:style="{width: progress + '%'}"), значення якого оновлюється в реальному часі після надсилання кожного MSP-пакета даних. Аналогічно, текстове поле console динамічно відображає масив повідомлень logMessages, який поповнюється через події, що надходять від eventBus.js; кожен новий запис у лозі додається до списку, що відображається (наприклад, за допомогою директиви v-for="msg in logMessages"). Загальні реактивні прапори, такі як isConnected, firmwareReady та errorOccurred, керують умовним рендерингом та активністю різних кнопок та блоків інтерфейсу, забезпечуючи логічну послідовність дій користувача .

Стилізація інтерфейсу реалізована за допомогою CSS-файлів main.css та secondary.css, використовуючи класичний підхід без застосування специфічних методологій типу БЕМ або CSS-in-JS. Клас button_group використовується для горизонтального вирівнювання пов'язаних елементів керування та структурування блоків на сторінці, застосовуючи властивості display: flex, gap та margin-bottom. Хоча мобільна адаптивність не була першочерговою метою, деякі стилі, такі як max-width та overflow-wrap, забезпечують базову гнучкість відображення на різних пристроях. Окрему увагу під час реалізації інтерфейсу було приділено питанням

доступності (accessibility, a11y). Для забезпечення можливості роботи з застосунком для користувачів з обмеженими можливостями було впроваджено підтримку навігації за допомогою клавіатури. Усі інтерактивні елементи, такі як кнопки та поля вибору, доступні для фокусування через клавішу Tab у логічній послідовності. Для ключових елементів керування були додані ARIA-атрибути (Accessible Rich Internet Applications), що надають допоміжним технологіям, таким як екранні диктори, додатковий контекст про призначення та поточний стан елемента (наприклад, aria-disabled для неактивних кнопок). Також було враховано вимоги до контрастності тексту та фону для покращення читабельності.

3.3 Логіка обробки даних і команд

У розробленому програмному засобі реалізовано комплексну та багаторівневу логіку обробки даних і команд, що забезпечує надійну та ефективну взаємодію з польотним контролером через серійний інтерфейс. Ця логіка охоплює декілька ключових етапів: глибокий аналіз та підготовка файлів прошивки у форматі Intel HEX [4, 9, детальне формування та розбір команд протоколу MSP, а також безпосередня передача даних через Web Serial API з обробкою станів та помилок.

Обробка файлів прошивки є першим та критично важливим етапом підготовки до інсталяції. Вона здійснюється за допомогою модуля hex_parser.js, розташованого у Web Worker для уникнення блокування інтерфейсу користувача під час обробки великих файлів. Детальна реалізація передбачає створення функції, що аналізує кожен рядок індивідуально. Ця функція спочатку перевіряє, чи починається рядок з обов'язкового символу ":". Далі, вона послідовно вилучає підрядки, що відповідають за кількість байтів даних, 16-бітну адресу та тип запису, і перетворює їх із шістнадцяткового текстового формату в числові значення. Після цього функція у циклі обробляє основну частину рядка, де містяться дані, перетворюючи кожну пару символів у відповідний байт. Під час цього процесу

також обчислюється контрольна сума: вона ініціалізується сумою байта кількості даних, байтів адреси та типу запису, а потім до неї послідовно додаються всі байти даних. Відповідно до специфікації, фінальна контрольна сума, вказана у файлі, має бути дводоповненням до молодшого байта обчисленої суми. Реалізація перевіряє цю умову, і якщо суми не збігаються, генерується помилка, що запобігає використанню пошкоджених даних. Важливим аспектом є обробка різних типів записів: записів даних (тип 00), записів кінця файлу (тип 01), що сигналізує про завершення прошивки, а також записів розширеної лінійної адреси (тип 04), які необхідні для коректної адресації пам'яті у 32-бітних мікроконтролерах. Парсер коректно обробляє ці записи для формування правильної карти пам'яті. Результатом роботи парсера є не просто набір байтів, а структурована карта пам'яті, що представляє собою масив об'єктів. Кожен об'єкт містить стартову адресу та відповідний масив байтів, призначений для запису за цією адресою. Така структура дозволяє модулю serial_backend.js гнучко керувати процесом прошивки, наприклад, записувати дані не послідовно, а за конкретними адресами.

Після успішного парсингу прошивки, наступним кроком є її передача на контролер. Формування команд та обробка відповідей за протоколом MSP [3, 7] реалізовано в модулях MSPConnector.js, MSPCodes.js та msp.js. Перед початком передачі основних блоків прошивки, serial_backend.js ініціює "рукостискання" з контролером. Цей процес починається з надсилання команди MSP_API_VERSION, щоб перевірити сумісність версій протоколу. Після отримання коректної відповіді, може надсилатися команда MSP_BOARD_INFO для отримання ідентифікатора плати та MSP_BUILD_INFO для отримання дати збірки та версії прошивки. Ця інформація логується та може бути використана для додаткової валідації сумісності обраного файлу прошивки з апаратним забезпеченням. Лише після успішного "рукостискання" система переходить до команд, що безпосередньо стосуються запису в пам'ять. Ключовою частиною реалізації є функція у модулі MSPConnector.js для створення MSP-пакету. Ця функція приймає на вхід код команди та опціональний масив байтів даних. Всередині функції створюється

буфер, куди послідовно записуються байти преамбули $M<, далі – байт, що вказує розмір даних, та байт коду команди. Після цього до буфера додаються самі байти даних. Далі реалізовано обчислення контрольної суми: ініціалізується змінна-акумулятор, і для кожного байта, починаючи з розміру даних і до кінця, виконується операція побітового XOR. Отримане значення контрольної суми додається як останній байт у пакет. На виході функція повертає сформований пакет у вигляді масиву байтів Uint8Array, який повністю відповідає специфікації MSP і готовий для передачі.

Передача даних через Web Serial API є фінальним етапом у ланцюгу обробки команд, і ця логіка реалізована в модулях serial_backend.js та webSerial.js. webSerial.js відповідає за низькорівневу взаємодію з API. Центральною функцією тут є асинхронна функція для встановлення з'єднання. Її реалізація починається з виклику navigator.serial.requestPort(). Цей виклик є критично важливим для безпеки, оскільки він ініціює діалогове вікно браузера, в якому користувач має свідомо надати дозвіл на доступ до конкретного підключеного пристрою. Програма не може отримати доступ до портів без цієї явної дії користувача. Оскільки вибір користувача є асинхронною операцією, її результат очікується за допомогою конструкції await. Після того, як об'єкт порту успішно отримано, на ньому викликається метод port.open(), якому передається об'єкт з параметрами з'єднання, такими як швидкість передачі (baudRate), наприклад, 115200 біт/с. Ця операція також є асинхронною. Вся логіка підключення обгорнута у блок try...catch для коректної обробки можливих помилок. Після успішного відкриття порту ініціалізуються два незалежні цикли для читання та запису за допомогою port.readable.getReader() та port.writable.getWriter(). Цикл читання безперервно очікує на вхідні дані від контролера, які акумулюються в буфері та передаються для аналізу в MSPConnector.js на предмет наявності повних MSP-пакетів. Цикл запису реалізує чергу для відправки даних, що запобігає спробам одночасного запису та гарантує послідовність надсилання пакетів. Таким чином, реалізована логіка обробки даних і команд забезпечує надійну та ефективну взаємодію з

польотним контролером, дозволяючи виконувати операції з прошивкою та налаштуванням пристрою безпосередньо з веб-застосунку. Особливу увагу в реалізації логіки обробки команд було приділено механізму обробки помилок та відновлення після збоїв на рівні протоколу MSP. Модуль serial_backend.js реалізує скінченний автомат, що керує станами процесу прошивки. Після надсилання кожного блоку даних, система переходить у стан очікування відповіді (AWAITING_ACK). Для цього стану встановлюється таймер (використовуючи setTimeout), тривалість якого визначена в конфігураційному файлі (наприклад, 250 мс). Якщо протягом цього часу від контролера надходить очікуване підтвердження (ACK), скінченний автомат переходить до стану надсилання наступного блоку (SENDING_CHUNK), а таймер скасовується.

Якщо ж надходить негативне підтвердження (NACK) або спрацьовує таймер (timeout), система не перериває процес одразу, а переходить у стан повторної спроби (RETRYING). У цьому стані лічильник спроб для поточного блоку даних інкрементується. Якщо кількість спроб не перевищує встановленого ліміту (наприклад, 3 спроби, що також є конфігураційним параметром), система повторно надсилає той самий блок даних і знову переходить у стан очікування. Цей цикл дозволяє подолати поодинокі збої передачі, спричинені короткочасними завадами на лінії зв'язку. Якщо ж усі спроби виявляються невдалими, скінченний автомат переходить у фінальний стан помилки (ERROR_STATE), процес прошивки негайно припиняється, а користувачу через модулі logger.js та notifications.js виводиться детальне повідомлення про помилку із зазначенням адреси блоку, на якому стався збій.

Такий підхід із застосуванням скінченного автомата, таймерів та механізму повторних спроб значно підвищує надійність процесу прошивки, роблячи його стійким до типових проблем, що виникають при роботі з послідовними інтерфейсами.

Окрім основного процесу прошивки, реалізовано логіку для виконання допоміжних команд. Наприклад, перед початком запису нової прошивки система

може надсилати команду на стирання необхідних секторів флеш-пам'яті контролера. Ця операція також є критично важливою і виконується за аналогічним принципом: надсилається команда MSP, і система очікує підтвердження її успішного виконання. Це гарантує, що запис нової прошивки буде здійснюватися на коректно підготовлену область пам'яті. Також передбачено можливість надсилання команди для перезавантаження контролера після успішного завершення всіх операцій, що дозволяє пристрою одразу запуститися з новим програмним забезпеченням без необхідності ручного перепідключення живлення. Усі ці кроки чітко логуються, надаючи користувачеві повну картину процесу взаємодії з пристроєм.

3.4 Реалізація логування та повідомлень

У розробленому веб-застосунку реалізовано систему логування та повідомлень, яка забезпечує інформування користувача про хід виконання операцій, виникнення помилок та інші важливі події. Ця система є невід'ємною частиною інтерфейсу користувача та сприяє зручності використання програмного засобу. Основним компонентом системи логування є модуль logger.js, який відповідає за формування та виведення повідомлень різного рівня важливості. Повідомлення класифікуються за рівнями: інформаційні, попередження та помилки. Кожне повідомлення супроводжується часовою міткою, що дозволяє відстежувати послідовність подій.

Для відображення повідомлень у інтерфейсі користувача використовується текстове поле з ідентифікатором console, розміщене на головній сторінці застосунку. Повідомлення додаються до цього поля у хронологічному порядку, зберігаючи історію подій протягом сесії роботи користувача.

Модуль notifications.js реалізує механізм сповіщень, які відображаються у вигляді спливаючих вікон або банерів. Ці сповіщення використовуються для інформування користувача про успішне завершення операцій, виникнення

помилок або необхідність виконання певних дій. Сповіщення мають різні стилі оформлення залежно від типу повідомлення, що дозволяє швидко ідентифікувати їхню важливість. На рівні реалізації, кожне повідомлення, що передається до модуля logger.js, є об'єктом з визначеною структурою, наприклад: { timestamp: new Date().toISOString(), level: 'INFO', message: 'Порт успішно відкрито.' }. Модуль logger.js форматує цей об'єкт у рядок стандартизованого вигляду, додаючи часову мітку та рівень важливості, перед тим як додати його до масиву повідомлень, що відображається у консолі. Своєю чергою, модуль notifications.js реалізований як функція, що приймає не лише текст повідомлення, але й його тип ('success', 'error', 'warning'). Залежно від переданого типу, до спливаючого вікна сповіщення динамічно додається відповідний CSS-клас (наприклад, .notification-success, .notification-error). Ці класи, визначені у файлі main.css, застосовують різне візуальне оформлення, зокрема колір фону (наприклад, зелений для успіху, червоний для помилки) та іконку, що дозволяє користувачеві миттєво оцінити характер системного повідомлення.

Взаємодія між модулями логування та іншими компонентами застосунку здійснюється за допомогою подієвої системи, реалізованої у модулі eventBus.js. Цей модуль забезпечує механізм підписки на події та їх обробки, що дозволяє централізовано керувати повідомленнями та забезпечувати їх своєчасне відображення у інтерфейсі користувача.

Таким чином, реалізована система логування та повідомлень забезпечує ефективне інформування користувача про стан системи та хід виконання операцій, що сприяє підвищенню зручності та надійності використання веб-застосунку.

3.5 Запуск та тестування веб-застосунку

Для забезпечення коректної роботи та перевірки функціональності розробленого веб-застосунку було проведено процедури запуску та комплексного тестування на різних етапах розробки. Основною метою тестування було

підтвердження працездатності всіх заявлених функцій, перевірка сумісності з різними операційними системами та браузерами, а також оцінка стабільності, продуктивності та зручності використання програмного засобу в типових сценаріях експлуатації.

Запуск веб-застосунку в режимі розробки здійснюється за допомогою інструменту Vite, який забезпечує швидке збирання проєкту та запуск локального сервера розробки. Для цього необхідно виконати стандартні команди в терміналі: спочатку встановити всі залежності проєкту за допомогою npm install, а потім запустити сервер командою npm run dev. Після виконання цих команд веб-застосунок стає доступним у веб-браузері за локальною адресою, зазвичай http://localhost:5173 або аналогічною, що дозволяє розробнику інтерактивно перевіряти зміни та функціональність.

Методологія тестування включала кілька рівнів перевірки. На рівні модульного тестування перевірялася коректність роботи окремих функцій в ізоляції. Наприклад, для модуля hex_parser.js були розроблені тестові сценарії, що подавали на вхід функції парсингу як коректні, так і навмисно пошкоджені рядки Intel HEX для перевірки правильності обробки даних та механізмів валідації контрольних сум. На рівні інтеграційного тестування перевірялася взаємодія між модулями, наприклад, коректність передачі даних від парсера до serial_backend.js та правильність викликів методів webSerial.js для надсилання сформованих MSP-пакетів. Системне тестування проводилося на реальному апаратному забезпеченні для перевірки повного циклу роботи застосунку. Окрему увагу було приділено тестуванню зручності використання (Usability Testing). Для цього було залучено користувача, що має базовий досвід роботи з БПЛА, але не є розробником. Йому було поставлено завдання виконати повний цикл прошивки контролера, а процес фіксувався для виявлення моментів, що викликали труднощі. Результати цього тестування дозволили покращити текстові повідомлення про помилки, зробивши їх більш зрозумілими для кінцевого користувача та додавши рекомендації щодо можливих дій.

Тестове середовище включало персональні комп'ютери під управлінням операційних систем Windows 11 та Linux (Ubuntu 24.04). Тестування проводилося в останніх стабільних версіях веб-браузерів Google Chrome та Microsoft Edge. Як апаратна платформа використовувався польотний контролер Matek F722-SE з встановленою прошивкою Betaflight 4.4. Під час тестування продуктивності проводилися заміри часу виконання ключових операцій. Зокрема, час парсингу файлу прошивки розміром 1 МБ не перевищував 3-4 секунд, що підтверджує ефективність використання Web Worker. Загальний час прошивки тестового контролера (запис 512 КБ даних) при швидкості 115200 біт/с склав у середньому 1 хвилину 15 секунд, що є порівнянним з нативними аналогами.

Під час ручного системного тестування було перевірено низку ключових сценаріїв. Основний позитивний сценарій полягав у повній процедурі прошивки, що підтвердив коректну взаємодію всіх модулів. Також було проведено тестування негативних сценаріїв. Тестувалася реакція системи на спробу підключення до порту, який вже зайнятий іншою програмою, або на фізичне відключення USB-кабелю під час роботи. Очікуваним результатом, який було підтверджено, є відображення відповідного повідомлення про помилку та коректне оновлення стану інтерфейсу. Проводилися спроби завантаження файлів з пошкодженою структурою. Система коректно ідентифікувала ці проблеми на етапі парсингу і виводила помилку, не допускаючи початку процесу прошивки. Окремо перевірялася обробка помилок комунікації, наприклад, отримання негативного підтвердження (NACK) від контролера, що підтвердило працездатність механізму повторних спроб у serial_backend.js. В рамках тестування безпеки було проведено низку перевірок на відмовостійкість. Зокрема, здійснювались спроби завантаження файлів прошивки, що містили некоректні типи записів або навмисно порушені контрольні суми на різних ділянках файлу. Система продемонструвала очікувану поведінку, коректно ідентифікуючи помилку на етапі парсингу та блокуючи подальші дії. Також було перевірено стійкість до введення невалідних даних у поля, де це можливо, та реакцію на несподівані відповіді від контролера, емулюючи

їх на програмному рівні. Це дозволило переконатись, що застосунок адекватно обробляє виняткові ситуації, не допускаючи неконтрольованих збоїв або зависання інтерфейсу.

Окрім запуску в режимі розробки, зібрану версію веб-застосунку, отриману командою vite build, можна розгорнути на будь-якому статичному веб-сервері. Розгортання в локальній мережі може бути легко реалізовано за допомогою http-server з пакетного менеджера npm. Після збирання проєкту достатньо запустити сервер у директорії dist, після чого застосунок стає доступним для всіх пристроїв у мережі за IP-адресою сервера. Це може бути корисним для командної роботи або в умовах, де необхідно надати доступ до інструменту декільком спеціалістам. Інтеграція плагіну vite-plugin-pwa також створює передумови для майбутньої адаптації застосунку під офлайн-режим. Можливість "встановлення" застосунку на робочий стіл, а також кешування основних ресурсів через Service Worker, дозволить запускати інструмент миттєво, навіть без доступу до мережі, що є вирішальною перевагою в польових умовах.

Проведене функціональне та нефункціональне тестування підтвердило, що розроблений веб-застосунок повністю відповідає поставленим вимогам і готовий до практичного використання. Окрім запуску в локальному режимі розробки через команду npm run dev, для розгортання та розповсюдження програмного засобу передбачено створення оптимізованої продакшн-версії. Цей процес ініціюється командою vite build, яка виконує компіляцію, мініфікацію та збирання всього коду та ресурсів проєкту в статичні файли, що розміщуються у директорії dist. Такий підхід забезпечує максимальну продуктивність та сумісність з будь-яким статичним веб-сервером.

Сценарії розгортання зібраної версії є гнучкими та можуть бути адаптовані під різні потреби. Для роботи в умовах локальної майстерні або для спільного використання в команді, найпростішим рішенням є розгортання в локальній мережі. Це можна легко реалізувати за допомогою будь-якого простого HTTP-сервера, наприклад, http-server з пакетного менеджера npm. Після запуску сервера

в директорії dist, застосунок стає доступним для всіх пристроїв у межах цієї мережі за IP-адресою комп'ютера-сервера. Такий підхід є надзвичайно корисним для командної роботи, оскільки дозволяє кільком спеціалістам одночасно працювати з єдиною версією інструменту без необхідності налаштовувати його на кожному окремому робочому місці. Для ширшого розповсюдження, наприклад, у рамках відкритого проєкту, директорію dist можна так само легко розгорнути на глобальних хостингових платформах для статичних сайтів, таких як GitHub Pages або Netlify, зробивши інструмент доступним для спільноти по всьому світу. Детальніше розглядаючи процес тестування безпеки, варто відзначити, що перевірка не обмежувалася лише валідацією структури HEX-файлу. Було проведено стрес-тестування, спрямоване на виявлення потенційних вразливостей типу "відмова в обслуговуванні" (DoS) на рівні веб-застосунку. Для цього було створено спеціальні тестові файли, що містили аномально довгі послідовності даних в одному рядку або надвелику кількість записів розширеної адреси, які, хоч і були синтаксично коректними, могли б призвести до надмірного споживання оперативної пам'яті браузера під час парсингу. Тести показали, що реалізація у Web Worker ефективно ізолює цей процес, і навіть при обробці таких файлів основний потік інтерфейсу користувача залишався відгукливим, а після досягнення певного ліміту на розмір оброблюваних даних процес парсингу коректно завершувався з помилкою, запобігаючи "зависанню" вкладки браузера.

Також у рамках тестування сумісності було проведено експерименти з різними перетворювачами USB-UART на базі мікросхем CP2102 та CH340, які часто зустрічаються в недорогих або кастомних польотних контролерах. Результати підтвердили, що завдяки стандартизації на рівні Web Serial API, застосунок успішно взаємодіяв з обома типами перетворювачів без необхідності встановлення будь-яких специфічних драйверів в операційній системі, що є значною перевагою порівняно з нативними конфігураторами, де часто виникають проблеми саме на цьому етапі.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено веб-застосунок, призначений для взаємодії з польотним контролером безпілотного літального апарата. Основною метою розробки було створення інструменту, який дозволяє здійснювати прошивку та налаштування пристрою без необхідності використання спеціалізованого програмного забезпечення, забезпечуючи зручність та доступність для користувача.

Розроблений веб-застосунок реалізовано з використанням фреймворку Vue.js та Web Serial API, що дозволяє здійснювати безпосередню взаємодію з пристроєм через серійний інтерфейс без потреби в додаткових драйверах чи програмному забезпеченні. Інтерфейс користувача забезпечує інтуїтивно зрозуміле управління процесом прошивки, включаючи вибір порту, завантаження файлу прошивки, відображення логів та індикаторів прогресу.

Логіка обробки даних включає парсинг файлів прошивки у форматі Intel HEX, формування команд протоколу MSP та передачу даних через Web Serial API. Система логування та повідомлень забезпечує інформування користувача про хід виконання операцій та виникнення помилок, що сприяє підвищенню надійності та зручності використання застосунку.

Проведене тестування веб-застосунку на різних операційних системах та браузерах підтвердило його стабільну та надійну роботу, що свідчить про успішність реалізації поставлених завдань.

Таким чином, розроблений веб-застосунок є ефективним інструментом для взаємодії з польотним контролером безпілотного літального апарата, забезпечуючи зручність, доступність та надійність процесу прошивки та налаштування пристрою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Vue.js: The Progressive JavaScript Framework. URL: https://vuejs.org/guide/introduction. (дата звернення: 02.04.2025).
2. Web Serial API – MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Serial_API. (дата звернення: 03.04.2025).
3. MultiWii Serial Protocol (MSP) – ArduPilot Documentation. URL: https://ardupilot.org/copter/docs/common-msp-overview.html. (дата звернення: 04.04.2025).
4. Intel HEX File Format – Arm Developer. URL: https://developer.arm.com/documentation/ka003292/latest/. (дата звернення: 05.04.2025).
5. Web Serial API – GitHub Pages. URL: https://wicg.github.io/serial/. (дата звернення: 07.04.2025).
6. Intel HEX-record Format – AWS. URL: https://cdck-file-uploads-europe1.s3.dualstack.eu-west-1.amazonaws.com/arduino/original/3X/3/e/3e9c5b1da740f9e28e97098c8f97c8628a9aa314.pdf. (дата звернення: 08.04.2025).
7. Reefwing MultiWii Serial Protocol (MSP) – GitHub. URL: https://github.com/Reefwing-Software/Reefwing-MSPGitHub. (дата звернення: 09.04.2025).
8. Vue.js Documentation – GitHub. URL: https://github.com/vuejs/docs. (дата звернення: 10.04.2025).
9. Intel HEX File Specification – Microchip Online Docs. URL: https://onlinedocs.microchip.com/oxy/GUID-BB433107-FD4E-4D28-BB58-9D4A58955B1A-en-US-7/GUID-DF9E479D-6BA8-49E3-A2A5-997BBA49D34D.html. (дата звернення: 11.04.2025).
10. Getting started with the Web Serial API – Google Codelabs. URL: https://codelabs.developers.google.com/codelabs/web-serial. (дата звернення: 12.04.2025).

11. Betaflight Flight Controller Firmware Project. URL: https://betaflight.com/. (дата звернення: 14.04.2025).

12. INAV Flight Controller Firmware. URL: https://inavflight.com/. (дата звернення: 14.04.2025).

13. SpeedyBee App – Official Website. URL: https://www.speedybee.com/speedy-bee-app/. (дата звернення: 15.04.2025).

14. STM32 32-bit Arm Cortex MCUs – STMicroelectronics. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html. (дата звернення: 16.04.2025).

15. Device Class Specification for Device Firmware Upgrade (DFU), Version 1.1 – USB Implementers Forum. URL: https://www.usb.org/document-library/device-class-specification-device-firmware-upgrade-version-11. (дата звернення: 17.04.2025).

16. BLHeli Firmware for ESCs. URL: https://github.com/bitdump/BLHeli. (дата звернення: 18.04.2025).

17. Vite: Next Generation Frontend Tooling. URL: https://vitejs.dev/. (дата звернення: 21.04.2025).

18. jQuery Official Website. URL: https://jquery.com/. (дата звернення: 22.04.2025).

19. Web Workers API – MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API. (дата звернення: 23.04.2025).

20. Node.js Official Website. URL: https://nodejs.org/. (дата звернення: 24.04.2025).

21. Git Official Website. URL: https://git-scm.com/. (дата звернення: 25.04.2025).

22. vite-plugin-pwa – npm. URL: https://www.npmjs.com/package/vite-plugin-pwa. (дата звернення: 28.04.2025).

23. WebUSB API – MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebUSB_API. (дата звернення: 29.04.2025).

24. Web Serial API – Web APIs – MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Serial_API. (дата звернення: 30.04.2025).

25. STM32 Programming via USB (DFU) – Phil's Lab #72 – FEDEVEL. URL: https://fedevel.com/blog/stm32-programming-via-usb-dfu-phils-lab-72. (дата звернення: 01.05.2025).

26. Read from and write to a serial port – Capabilities – Chrome Dev. URL: https://developer.chrome.com/docs/capabilities/serial. (дата звернення: 02.05.2025).

27. Tutorial: Restart STM32F in DFU mode – ST Community. URL: https://community.st.com/t5/stm32-mcus-embedded-software/tutorial-restart-stm32f-in-dfu-mode/td-p/640149. (дата звернення: 05.05.2025).

28. Using the WebSerial API to communicate with the microcontroller – yarkov.tech. URL: https://yarkov.tech/en/blog/2022-11-04/using-the-web-serial-api-to-communicate-with-the-microcontroller/. (дата звернення: 06.05.2025).

29. STM32 force boot mode via USB? – EEVblog forum. URL: https://www.eevblog.com/forum/microcontrollers/stm32-force-boot-mode-via-usb/. (дата звернення: 07.05.2025).

30. Web Serial API PDF Guide – MIT. URL: https://web.mit.edu/6.115/www/document/Web%20Serial%20API%20-%20Chrome%20for%20Developers.pdf. (дата звернення: 08.05.2025).

31. How to fix dfu-util, STM, WinUSB, Zadig... – Hanselman's Blog. URL: https://www.hanselman.com/blog/how-to-fix-dfuutil-stm-winusb-zadig-bootloaders-and-other-firmware-flashing-issues-on-windows. (дата звернення: 12.05.2025).

32. Web Serial API – GitHub Pages (specification). URL: https://wicg.github.io/serial/. (дата звернення: 13.05.2025).

33. Getting started with STM32 built-in USB DFU Bootloader – Electronics Specifier. URL: https://www.electronicspecifier.com/news/tech-videos/getting-started-with-stm32-built-in-usb-dfu-bootloader. (дата звернення: 14.05.2025).

34. r/WebSerialAPI – Reddit community. URL: https://www.reddit.com/r/WebSerialAPI/. (дата звернення: 15.05.2025).

35. STM32 USB programming jumper – StackOverflow. URL: https://stackoverflow.com/questions/66538837/stm32-usb-programming-jump-to-bootloader-for-dfu. (дата звернення: 16.05.2025).

36. Getting started with the Web Serial API – Google Codelabs. URL: https://codelabs.developers.google.com/codelabs/web-serial. (дата звернення: 19.05.2025).

37. STM32 DFU Programming – TKJ Electronics blog. URL: https://blog.tkjelectronics.dk/2010/01/stm32-dfu-programming/. (дата звернення: 20.05.2025).

38. Read USB serial port data with Web Serial API – StackOverflow. URL: https://stackoverflow.com/questions/70920727/read-usb-serial-port-data-with-web-serial-api-in-javascript. (дата звернення: 21.05.2025).

39. STM32 Nucleo and DFU USB Bootloading – MobileWill. URL: https://mobilewill.us/stm32-nucleo-and-dfu-usb-bootloading/. (дата звернення: 22.05.2025).

40. STM32 USB DFU circuit guidance – EE StackExchange. URL: https://electronics.stackexchange.com/questions/675956/stm32-usb-dfu-circuit. (дата звернення: 23.05.2025).

41. Web Serial example demo – GitHub. URL: https://github.com/funnierinspanish/web-serial-example. (дата звернення: 26.05.2025).

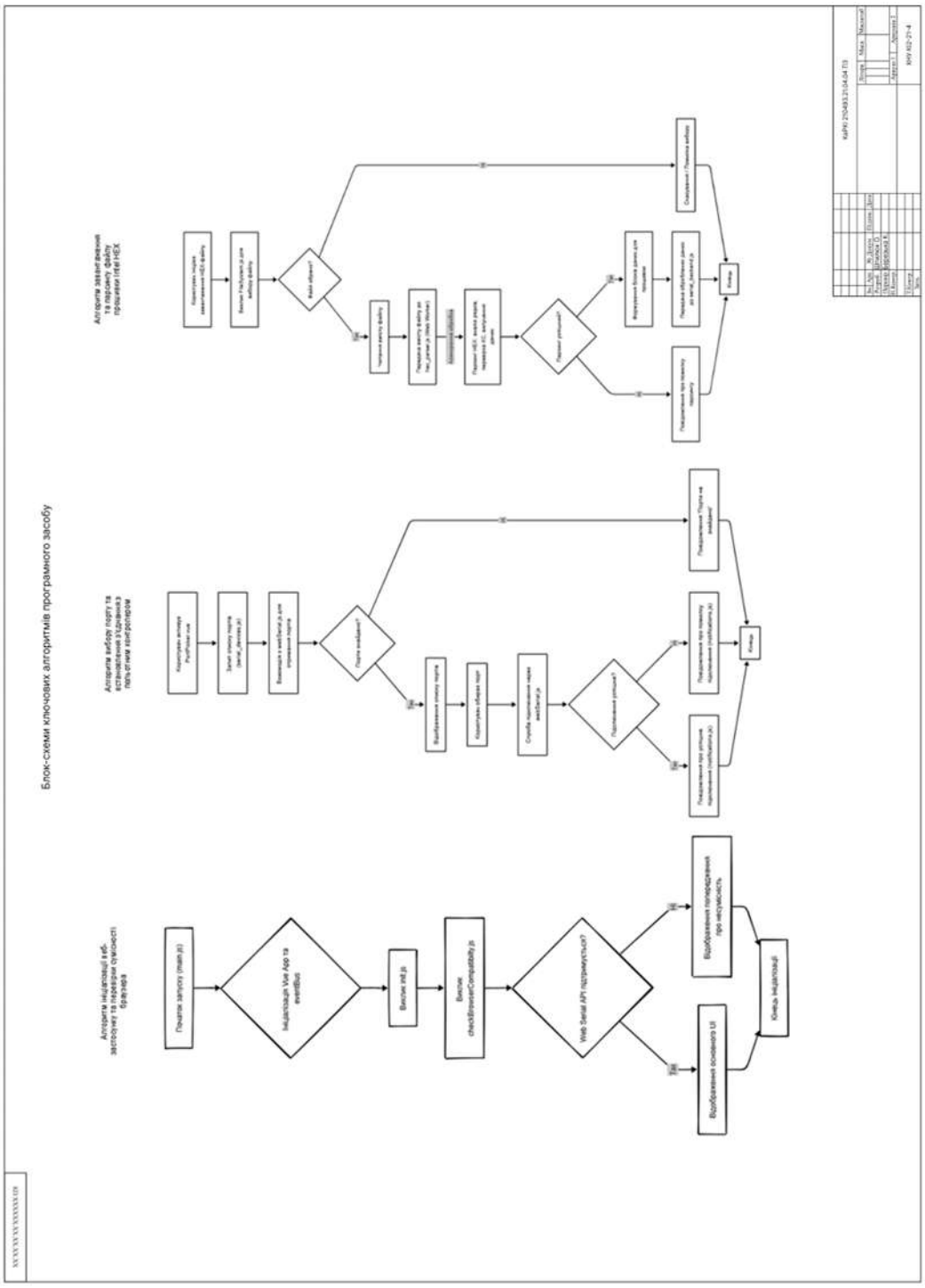
42. Assistance Understanding STM32 DFU Descriptors – r/embedded. URL: https://www.reddit.com/r/embedded/comments/um1g7b/assistance_understanding_stm32_dfu_descriptors/. (дата звернення: 27.05.2025).

43. Choosing the Right Communication Protocol for your Web Application – arXiv. URL: https://arxiv.org/abs/2409.07360. (дата звернення: 30.05.2025).

Додаток А

(обов’язковий)

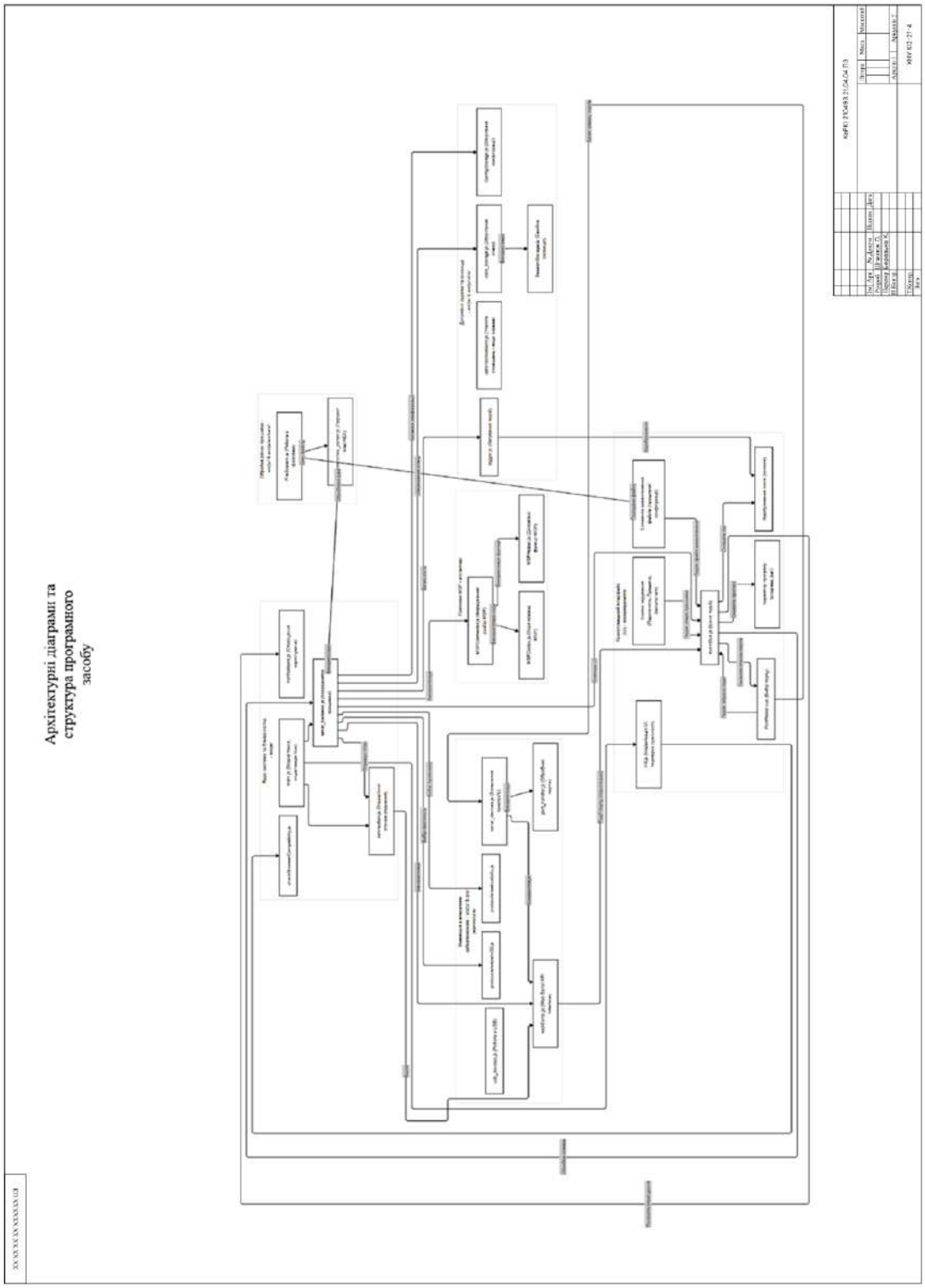
Копія креслення «БЛОК-СХЕМИ КЛЮЧОВИХ АЛГОРИТМІВ ПРОГРАМНОГО ЗАСОБУ»



Додаток Б

(обов’язковий)

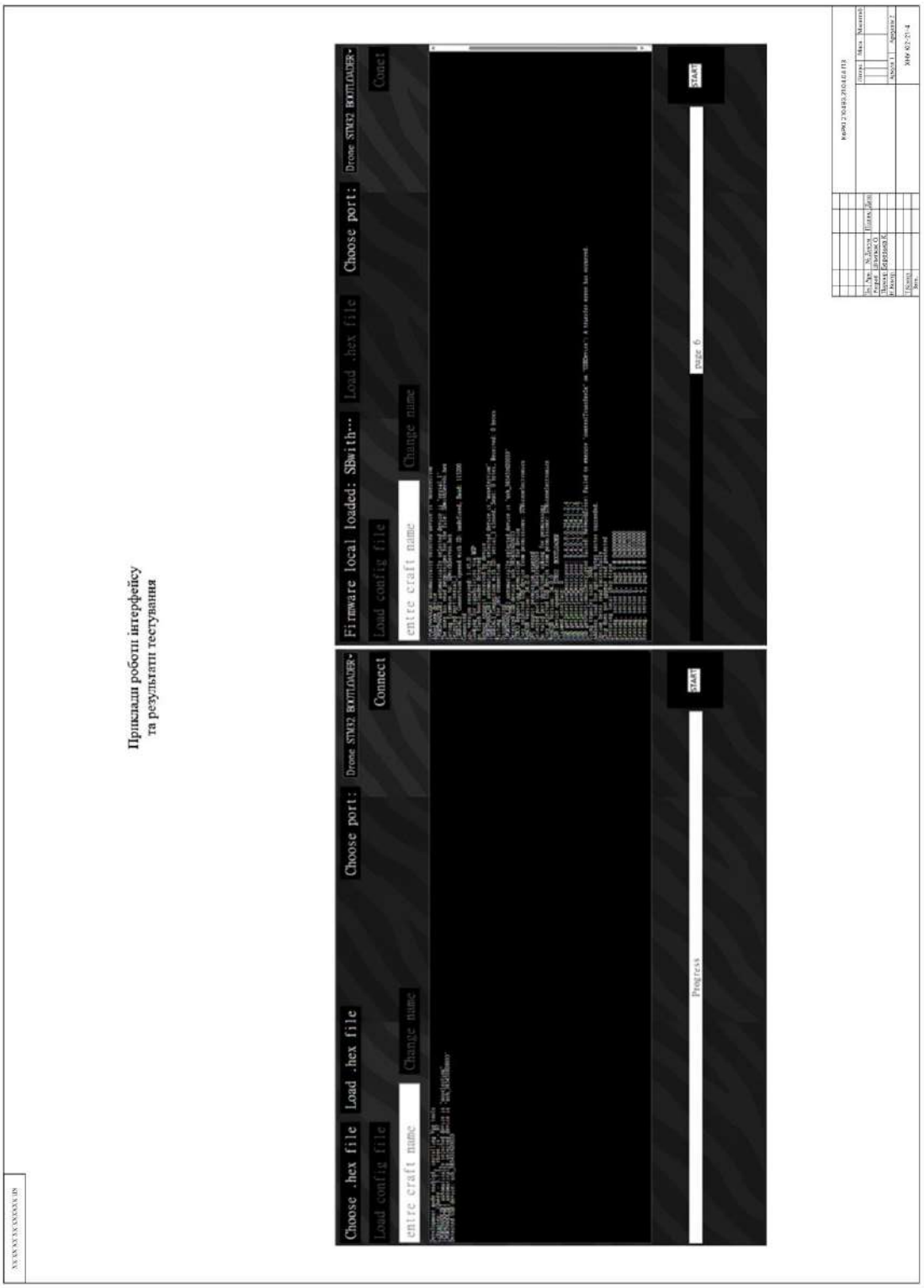
КОПІЯ КРЕСЛЕННЯ «АРХІТЕКТУРНА ДІАГРАМА ТА СТРУКТУРА ПРОГРАМНОГО ЗАСОБУ»



Додаток В

(обов'язковий)

КОПІЯ КРЕСЛЕННЯ «ПРИКЛАДИ РОБОТИ ІНТЕРФЕЙСУ ТА РЕЗУЛЬТАТИ ТЕСТУВАННЯ»



Tue Jun 10 18:53:12 EEST 2025, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 13%**

	Document		Sum coincidence on the DB	
ID: 244768 Title: БКР Програмно-апаратний засіб для інсталяції програмного забезпечення у БПЛА Added in a DB: 2025-06-10 Authors: Олексій ШПИЛЮК Heads: Катерина БЕРЕЗЬКА Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	99858	664	939 (1%)	16 (2%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Олексій ШПИЛЮК

Співавтор:

Назва: Шпилюк_Програмно-апаратний засіб для інсталяції програмного забезпечення у БПЛА

Експерт:

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:1%

Коефіцієнт подібності 2:0%

Мікропробіли: 6

Заміна букв: 0

Інтервали: 0

Білі знаки: 1

Дата створення звіту: 2025-06-11 01:14:37.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2025-06-11 Доцент Андрій Нічепорук

Дата експерт

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Олексій ШПИЛЮК

Тема: Програмно-апаратний засіб для інсталяції програмного забезпечення у БПЛА

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 Кількість сторінок записки _____

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є розробка програмного забезпечення для спрощеної та пришвидшеної прошивки польотних контролерів.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі кваліфікаційної роботи проведено аналіз предметної області та теоретичних основ досліджуваної проблеми. В другому розділі кваліфікаційної роботи виконано проєктування програмно-технічного засобу. В третьому розділі кваліфікаційної роботи описано програмну реалізацію та тестування програмно-технічного засобу.

4. Позитивні сторони роботи: Висока практична цінність роботи.

5. Негативні сторони роботи:

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на високому інженерно-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: відмінно (4.75/A)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Онищко О.Г., доцент каф. ІПЗ

"12" 06 2025 р. _____ (підпис)

Завідувачу кафедри КІІС
д-р. філософії, доц. Ользі ПАВЛОВІЙ

Олексія ШПИЛЮКА

ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2-21-4

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Strike-Plagiarism та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

10.06 2025 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Програмно-апаратний засіб для інсталяції програмного забезпечення у БПЛА

Автор: Олексій ШПИЛЮК

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Катерина БЕРЕЗЬКА к.т.н., доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 1% і адресується до 16 першоджерела; та системою Anti-Plagiarism складає 1%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Катерина БЕРЕЗЬКА

Гарант ОП _____ Андрій Нічепорук

Завідувач кафедри КІС _____ Ольга ПАВЛОВА