Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)
Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система моніторингу радіосигналів в системах обізнаності
(тема)

Виконав:
здобувач 4 року навчання,
групи ПЗПІ-21-8
Микита ХАМБУР
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. каф. ПІ Володимир ЛЕЩИНСЬКИЙ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту

Завідувач кафедри ПІ _____ Кирило СМЕЛЯКОВ
(підпис) (власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 25 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Хамбуру Микиті Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система моніторингу радіосигналів в системах обізнаності

затверджена наказом університету від 19 травня 20 25 р. № 397Ст

2. Термін подання студентом роботи до екзаменаційної комісії 24.06.2025

3. Вихідні дані до роботи Розробка програмного рішення, яке забезпечує оперативну оцінку радіообстановки та веде журнал появи й зникнення джерел сигналів.

4. Перелік питань, що потрібно опрацювати в роботі Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	14.05.2025	*виконано*
2	Створення специфікації ПЗ	16.05.2025	*виконано*
3	Проектування ПЗ	20.05.2025	*виконано*
4	Розробка ПЗ	23.05.2025	*виконано*
5	Тестування ПЗ	25.05.2025	*виконано*
6	Оформлення пояснювальної записки	31.05.2025	*виконано*
7	Підготовка презентації та доповіді	12.06.2025	*виконано*
8	Попередній захист	14.06.2025	*виконано*
9	Нормоконтроль, рецензування	17.06.2025	*виконано*
10	Здача роботи у електронний архів	20.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	24.06.2025	*виконано*

Дата видачі завдання «09» «квітня» 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. каф. ПІ Володимир ЛЕЩИНСЬКИЙ
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 67 стор., 13 рис., 5 табл., 11 джерел.

РАДІОМОНІТОРИНГ, PYTHON, NUMPY, PYQT, FFT, SPECTRUM ANALYSIS, SQLITE

Об’єкт розробки – програмна система моніторингу радіосигналів (ПСМР) для підвищення ситуаційної обізнаності. Система приймає потоки даних як від програмного генератора тестових сигналів, так і (за потреби) від апаратних SDR-приймачів, виконує спектральний аналіз у діапазоні 30 кГц – 6 ГГц, автоматично визначає активні передавачі й відображає їхні характеристики – частоту, потужність, тривалість роботи та азимут – в інтерактивному графічному інтерфейсі оператора.

Мета розробки – створити надійне та розширюване програмне рішення, яке забезпечує оперативну оцінку радіообстановки, веде журнал появи й зникнення джерел сигналів та може слугувати основою для подальших досліджень і інтеграції в комплексні системи безпеки, транспорту чи аварійно-рятувальних служб.

Метод рішення – реалізація крос-платформного застосунку на Python 3.12. Ядро обробки використовує NumPy та SciPy для синтезу й FFT-аналізу сигналів, а asyncio забезпечує безперервний часовий потік даних. Графічний інтерфейс створено на PyQt6 та PyQtGraph (спектр- і водоспад-діаграми, таблиця характеристик), дані подій зберігаються у SQLite. Модульна архітектура «ядро – GUI – сховище» дозволяє легко підключити реальний SDR-приймач або зовнішній шлюз у майбутніх версіях системи.

У результаті розробки створено ПСМР, здатну одночасно відстежувати до 256 незалежних радіоджерел, оновлювати спектр із частотою 20 кадрів/с та формувати журнал подій. Система спрощує аналіз спектра завдяки інтерактивним інструментам і працює на типовому обладнанні без спеціалізованих радіопристроїв.

ABSTRACT

RADIO MONITORING, PYTHON, NUMPY, PYQT, FFT, SPECTRUM ANALYSIS, SQLITE

The subject of this work is a software system for radio-signal monitoring (SSRM) designed to enhance situational awareness. The system ingests data streams both from a software test-signal generator and—when required—from hardware SDR receivers, performs spectral analysis over the 30 kHz–6 GHz band, automatically detects active transmitters, and displays their characteristics—frequency, power, duration, and azimuth—in an interactive operator GUI.

The aim of the development is to create a reliable and extensible software solution that provides real-time assessment of the radio environment, maintains a log of signal-source appearances and disappearances, and can serve as a basis for further research or integration into comprehensive security, transportation, or emergency-response systems.

Solution approach – a cross-platform application implemented in Python 3.12. The processing core uses NumPy and SciPy for signal synthesis and FFT analysis, while asyncio ensures a continuous temporal data stream. The graphical interface is built with PyQt6 and PyQtGraph (spectrum and waterfall plots, characteristics table), and event data are stored in SQLite. A modular “core–GUI–storage” architecture allows easy integration of a real SDR receiver or an external gateway in future system versions.

As a result of this development, a SSRM has been produced that can simultaneously track up to 256 independent radio sources, update the spectrum at 20 frames per second, and generate an event log. The system simplifies spectrum analysis through interactive tools and operates on standard hardware without specialized radio equipment.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПСМР (Програмна система моніторингу радіосигналів) – програмний комплекс для прийому, спектрального аналізу та відстеження радіосигналів у режимі реального часу.

SDR (Software-Defined Radio) – архітектура радіоприймачів/передавачів, у якій більша частина обробки сигналу виконується програмно.

FFT (Fast Fourier Transform) – алгоритм швидкого дискретного перетворення Фур'є для отримання спектра сигналу.

RSA (Real-Time Spectrum Analysis) – метод спектрального аналізу у реальному часі з охопленням широкої смуги без послідовного сканування.

DSP (Digital Signal Processing) – цифрова обробка сигналів із використанням чисельних алгоритмів.

IoT (Internet of Things) – мережа фізичних пристроїв, здатних обмінюватися даними через інтернет.

FPGA (Field-Programmable Gate Array) – програмована користувачем вентильна матриця для апаратного прискорення обробки даних (ПЛІС).

UML (Unified Modeling Language) – уніфікована мова моделювання для візуалізації структури й поведінки системи.

MVC (Model-View-Controller) – архітектурний шаблон розподілу на модель, представлення та контролер.

ER (Entity-Relationship) – модель «сутність-зв'язок» для опису структури реляційної бази даних.

GUI (Graphical User Interface) – графічний інтерфейс користувача для візуальної взаємодії з програмою.

BLAS (Basic Linear Algebra Subprograms) – набір низькорівневих підпрограм для виконання лінійної алгебри.

ORM (Object-Relational Mapping) – технологія відображення об'єктів програми на записи в реляційній базі даних.

ВСТУП

Сучасна епоха цифрової трансформації супроводжується вибуховим зростанням кількості бездротових сервісів – від стільникових мереж п’ятого покоління й супутникового інтернету до мільярдів IoT-пристроїв та навігаційних систем. Усе це суттєво підвищує навантаження на радіочастотний спектр – обмежений природний ресурс, безперебійну роботу якого критично важливо забезпечувати для безпеки авіації, мореплавства, екстрених служб і цивільної інфраструктури. Актуальність розробки доступної програмної системи, здатної оперативно аналізувати радіообстановку в реальному часі без залучення дорогого вимірювального обладнання, не викликає сумнівів.

Метою роботи є розробка доступної програмної системи моніторингу радіосигналів (ПСМР) з використанням недорогих SDR-пристроїв та Python-стека для візуалізації та обробки сигналів у реальному часі. Передбачається, що рішення працюватиме на звичайних робочих станціях, а модульна архітектура дасть змогу легко перейти від емулятора сигналів до фізичного SDR-приймача.

Для досягнення цієї мети насамперед необхідно проаналізувати сучасний стан галузі та існуючі програмно-апаратні рішення радіомоніторингу, на цій основі сформулювати функціональні й нефункціональні вимоги до ПСМР з урахуванням обмежень апаратної платформи, спроєктувати архітектуру та обрати відповідний технологічний стек. Далі потрібно реалізувати модулі генерації або приймання сигналів, візуалізації спектра та журналювання подій, після чого провести тестування та оцінити точність виявлення передавачів.

Очікується, що отримані результати сприятимуть підвищенню оперативності й достовірності оцінки радіообстановки, забезпечать доступну платформу для навчання основ цифрової обробки сигналів та закладуть підґрунтя для подальших досліджень у сфері радіочастотної безпеки й моніторингу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Радіочастотний спектр є обмеженим ресурсом, що інтенсивно використовується різними службами та технологіями. Сьогодні існують мільярди бездротових пристроїв (мобільні гаджети, IoT-сенсори, передавачі 5G тощо), і радіоефір стає дедалі більш насиченим сигналами та зашумленим. Бездротовий спектр суворо регулюється державними органами, адже неконтрольовані передавачі можуть створювати взаємні перешкоди. Наявність небажаних або заважаючих сигналів у робочому діапазоні – буденне явище, що становить загрозу для критичних радіосистем. З огляду на перевантаженість спектра та зростання кількості радіовипромінювачів, виникла потреба у постійному моніторингу спектра, щоб своєчасно виявляти несанкціоновані або проблемні передавальні сигнали [1]. Таким чином, радіомоніторинг сьогодні відіграє ключову роль у забезпеченні ефективного і безпечного використання РЧ-ресурсу.

Радіомоніторинг передбачає безперервне вимірювання та аналіз частотного спектра сигналів, що надходять на антену приймача. Традиційно для цього застосовувалися апаратні спектральні аналізатори та спеціалізовані приймачі: вони сканували діапазони частот і будували спектр (графік потужності сигналу від частоти). Однак класичні аналізатори зі схемою послідовного перебору частот мають обмеження – вони можуть пропустити короткочасні імпульсні сигнали, оскільки не охоплюють одразу весь діапазон. Сучасний підхід – це використання реального часу спектрального аналізу (Real-Time Spectrum Analysis, RSA): система одночасно охоплює широкий частотний діапазон і оновлює спектр із високою частотою кадрів, забезпечуючи 100% ймовірність виявлення навіть коротких сигналів. Одним із найбільших технічних викликів є відстеження швидких змін сигнального середовища: аби надійно фіксувати

варіації сигналів у часі, недостатньо повільного сканування – потрібна паралельна обробка широкої смуги частот у режимі реального часу [2].

Програмно-визначене радіо (Software-Defined Radio, SDR) стало ключовою технологією у галузі радіомоніторингу. SDR-пристрої – це гнучкі радіоприймачі/передавачі, в яких більшість радіочастотної обробки винесено в програмне забезпечення. Типовий SDR складається з аналогового фронт-енду (антена, фільтри, підсилювач, перетворювач частоти) та цифрового бек-енду (АЦП/ЦАП і програмований процесор). На аналоговому етапі сигнал оцифровується, після чого цифровий процесор виконує всі необхідні операції: демодуляцію, спектральний аналіз, фільтрацію, виявлення сигналів тощо. Цифрова обробка сигналів дозволяє легко змінювати логіку обробки, оновлюючи програмне забезпечення замість фізичної заміни апаратури. У високопродуктивних SDR на ПЛІС (FPGA) реалізовано паралельні алгоритми цифрової обробки, що забезпечує мінімальні затримки та одночасний аналіз кількох каналів. Це означає, що один SDR може динамічно переналаштовуватися на різні діапазони і типи сигналів без заміни обладнання, що надзвичайно зручно для моніторингу широкого спектру.

На ринку представлено широкий спектр SDR-рішень – від дорогих лабораторних приймачів до масових дешевих адаптерів. Наприклад, плата Ettus USRP є модульною SDR-платформою для досліджень і може покривати діапазон до кількох ГГц, але коштує сотні чи тисячі доларів. Натомість у 2010-х роках було відкрите неочікуване бюджетне рішення: USB-тюнер DVB-T на чипі RTL2832U, вартістю $20, виявився здатним працювати як SDR-приймач [3]. Цей простий пристрій (відомий як RTL-SDR) при підключенні до ПК дозволяє приймати та аналізувати сигнали на частотах ~24–1766 МГц. Хоча його технічні параметри поступаються професійним аналізаторам, поява RTL-SDR здійснила революцію, зробивши радіомоніторинг доступним широкому колу інженерів і радіоаматорів за невелику ціну. Отже, сучасний стан технологій характеризується переходом від вузькоспеціалізованих апаратних засобів до

гнучких апаратно-програмних комплексів. Програмна реалізація (на ПК або вбудованих системах) у поєднанні з SDR дає змогу виконувати спектральний аналіз, демодуляцію та інші операції суто програмно, використовуючи алгоритми цифрової обробки (наприклад, швидке перетворення Фур'є, FFT для отримання спектра сигналу). Це відкриває нові можливості для моніторингу: відстеження використання частот у реальному часі, виявлення несанкціонованих передавачів, контроль перешкод та інші завдання як у цивільній сфері (управління спектром, тестування радіосумісності, моніторинг якості зв'язку), так і у військовій та силовій сферах (радіоелектронна розвідка, придушення радіовипромінювань, системи ситуаційної обізнаності тощо).

Таким чином, сучасні методи радіомоніторингу спираються на програмно-визначені приймачі та потужні алгоритми DSP. В реальних системах це реалізовано або у вигляді спеціалізованих приладів, або у вигляді програмних інструментів на ПК з підключеними SDR-пристроями. Важливо зазначити, що зростання обсягів даних, які потрібно обробляти, диктує високі вимоги до обчислювальної інфраструктури. Наприклад, моніторинг широкої смуги частот означає генерування потоків даних у сотні МБ/с і більше, що потребує ефективних алгоритмів обробки та швидкісних інтерфейсів передачі даних до комп'ютера [4]. Тому сфера радіомоніторингу активно розвивається у напрямку вдосконалення як апаратних компонентів (швидкодіючі АЦП, FPGA для попередньої обробки), так і програмного забезпечення (оптимізовані алгоритми аналізу сигналів, системи штучного інтелекту для автоматичної інтерпретації спектра тощо). Все це створює передумови для появи нових програмних систем моніторингу радіосигналів, здатних працювати в реальному часі з мінімальними затратами.

1.2 Виявлення та вирішення проблем

Незважаючи на досягнутий прогрес, у галузі радіомоніторингу досі існує низка суттєвих проблем і викликів. До ключових проблем, що стримують ефективність радіомоніторингу, належать такі:

– висока вартість обладнання;
– обмеження реального часу;
– складність візуалізації та інтерпретації даних;
– масштабованість системи.

Висока вартість обладнання. Більшість професійних пристроїв для аналізу спектра (включно з висококласними SDR) є дорогими. Спеціалізовані апаратні комплекси коштують десятки тисяч доларів, що обмежує їх широке застосування. Наприклад, портативні багатоканальні SDR-приймачі лабораторного рівня можуть коштувати більше 10000 доларів. Традиційний підхід до моніторингу часто полягав у розгортанні мережі стаціонарних постів або у ручному об'їзді територій фахівцями з обладнанням – так звані «вартові спектра». Це потребує значних фінансових і людських ресурсів. В результаті збір даних про використання спектра в режимі реального часу на великій території стає вкрай складним і дорогим. Шляхи вирішення – застосування доступних за ціною SDR-пристроїв разом із програмним аналізом на ПК дозволяє різко знизити витрати. З появою дешевих USB-SDR радіомоніторинг став економічно доцільним навіть для невеликих організацій чи приватних осіб. Програмна реалізація на звичайних комп'ютерах усуває потребу в дорогих апаратах: багато функцій спектрального аналізу можна виконати у програмі, використовуючи потужність сучасних процесорів. Таким чином, програмно-орієнтовані рішення значно знижують бар'єр входження в сферу радіомоніторингу за рахунок здешевлення обладнання.

Обмеження реального часу. Моніторинг радіосигналів генерує великий потік даних, і обробити їх без затримок – непросте завдання. Для широкосмугового аналізу потрібно опрацьовувати мільйони семплів за секунду,

виконуючи спектральне перетворення (FFT) та інші алгоритми на льоту. Традиційні платформи не забезпечують достатньої швидкодії: наприклад, як вже зазначалося, послідовне сканування спектра «пропускає» швидкоплинні сигнали. Також обмеженою є пропускна здатність інтерфейсів: не кожен приймач може передати весь сирий потік IQ-даних на ПК без втрат. Шляхи вирішення: поява SDR з апаратним прискоренням DSP частково знімає цю проблему – сучасні SDR вбудовують FPGA, що виконує попередню обробку сигналу паралельно та з мінімальною затримкою. З боку програмного забезпечення, оптимізація алгоритмів і використання ефективних бібліотек відіграють ключову роль. Мова Python, хоч інтерпретована, має в арсеналі високопродуктивні бібліотеки (NumPy, SciPy тощо), реалізовані на C/Fortran, які дозволяють виконувати операції типу FFT над великими масивами даних дуже швидко. Наприклад, обчислення спектра через numpy.fft використовує оптимізовані алгоритми і багатоядерні обчислення, що дає змогу наблизити аналіз до реального часу. Крім того, паралельна обробка і буферизація даних допомагають не втрачати інформацію: можна застосувати багатопотоковість (окремий потік для прийому даних з SDR, окремий – для обчислення та візуалізації). У результаті програмна система, написана на Python, здатна забезпечити прийнятну швидкодію для помірних смуг пропускання (наприклад, кілька МГц) на сучасному ПК. Практичні реалізації демонструють, що інструменти на кшталт GNU Radio або власних Python-програм у поєднанні з SDR можуть обробляти потоки даних у реальному часі з мінімальними затримками.

Складність візуалізації та інтерпретації даних. Отримання спектральних даних – лише частина задачі; не менш важливо представити їх користувачеві у наочній та зрозумілій формі. При моніторингу широкого діапазону частот виникає велика кількість інформації: десятки або сотні спектральних ліній, які змінюються щомиті. Інтерпретувати ці дані без зручних інструментів важко – фахівець може пропустити аномальний сигнал серед шуму або не побачити

тенденцію у зміні спектра. Традиційні апарати зазвичай відображають тільки поточний спектр на екрані, і користувач змушений вручну спостерігати за змінами. Шляхи вирішення: програмні системи дозволяють створити багатий графічний інтерфейс користувача (GUI) для аналізу сигналів. За допомогою бібліотек на зразок PyQt та PyQtGraph можна реалізувати інтерактивні графіки: спектрограму (waterfall), що показує еволюцію спектра в часі, панелі для точної настройки частоти та смуги, індикатори рівнів сигналу, маркери для вимірювань тощо. Наприклад, у роботі з SDR вже використовуються програмні інтерфейси, що поєднують одночасно графік у часі, спектр та водоспад, а також елементи керування параметрами приймача [5]. Це значно спрощує інтерпретацію: можна візуально виявити появу нового сигналу, простежити його тривалість, оцінити зайнятість каналу, розпізнати тип модуляції за характерним спектральним відбитком тощо. Окрім візуалізації, програмно можна додати й елементи автоматичного аналізу: наприклад, алгоритми детектування перевищення порогового рівня (для сигналізування про появу передавача) або навіть класифікацію сигналів методом машинного навчання. Таким чином, гнучкість ПЗ дозволяє не тільки відобразити дані, а й надати користувачу інструменти для їх трактування – аж до підказок на основі штучного інтелекту. Усе це сприяє підвищенню ефективності радіомоніторингу, особливо коли оператор повинен контролювати великий потік різнорідних сигналів.

Масштабованість системи. Завдання моніторингу можуть вимагати охоплення великої території або одночасного спостереження за багатьма діапазонами. Масштабування традиційними засобами (додавання більшої кількості апаратних аналізаторів або постів) швидко стає економічно невигідним і технічно складним. Нинішні методи спектрального нагляду не масштабуються для покриття розподілених мереж і широкого частотного спектра. Шляхи вирішення: програмно-орієнтований підхід надає більше гнучкості при масштабуванні. По-перше, можна розгорнути розподілену мережу недорогих сенсорів на базі SDR: замість кількох великих центрів спостереження – десятки

і сотні малих вузлів, дані від яких збираються у хмарну платформу. Такий краудсорсинговий підхід уже пропонується дослідниками: завдяки здешевленню SDR з’явилася ідея залучити масових користувачів (наприклад, смартфони з підключеними SDR-модулями) до автоматичного збору спектральних даних у режимі 24/7. Це дозволило б радикально розширити територіальне покриття моніторингу при помірній вартості. По-друге, навіть у межах одного потужного вузла, програмна система легше масштабується, ніж апаратна: додавання нових функцій або підтримки нового діапазону частот зводиться до встановлення відповідних модулів ПЗ, тоді як апаратний комплекс потребував би заміни блоків. Крім того, Python-орієнтоване рішення є портативним: його можна розгорнути на різних апаратних платформах (настільний ПК, ноутбук, одноплатний комп’ютер на зразок Raspberry Pi) залежно від потреб у масштабі і ресурсах. Таким чином, програмна реалізація спрощує як горизонтальне масштабування (розгортання багатьох вузлів моніторингу), так і вертикальне масштабування (розширення функціоналу та продуктивності окремої системи), чим частково або повністю знімає обмеження традиційних підходів.

1.3 Постановка задачі

З урахуванням проведеного аналізу, метою роботи є розробка програмної системи моніторингу радіосигналів в системах обізнаності – тобто застосунку, що надасть можливість відслідковувати радіосигнали в реальному часі та сприятиме підвищенню обізнаності операторів щодо радіообстановки. Щоб досягти цієї мети, розроблювана система повинна вирішувати ряд конкретних завдань і відповідати визначеним вимогам, а саме:

- моніторинг спектра в реальному часі;
- нагляд за широким діапазоном частот;
- візуалізація та інтерфейс користувача;
- збір та збереження даних;
- базовий аналіз та оповіщення.

Моніторинг спектра в реальному часі. Система має приймати радіосигнали з SDR-пристрою і виконувати їх спектральний аналіз на льоту, відображаючи поточний стан спектра у вибраному частотному діапазоні. Це забезпечить виявлення появи нових сигналів або змін у ефірі без помітних затримок.

Нагляд за широким діапазоном частот. Система повинна забезпечувати сканування (або одночасний перегляд) заданого діапазону частот, характерного для цільового застосування. Наприклад, для ситуативної обізнаності в межах певної служби може знадобитися моніторинг діапазону авіаційного зв’язку, діапазонів Wi-Fi, стільникових мереж тощо – залежно від сфери застосування.

Візуалізація та інтерфейс користувача. Програмний продукт має надавати зручний графічний інтерфейс, що включає відображення поточної спектральної характеристики (графік «частота-потужність») і спектрограми (водоспаду) для бачення динаміки сигналів. Інтерфейс повинен бути інтерактивним: користувач зможе змінювати налаштування (центральну частоту, смугу пропускання, посилення приймача тощо) і одразу спостерігати результат на екрані.

Збір та збереження даних. Система повинна мати функції логування або запису виявлених сигналів. Це може включати збереження спектральних знімків, запис сирих IQ-даних за певних умов (наприклад, при виявленні аномального сигналу) для подальшого аналізу, або ведення журналу зайнятості частотних каналів.

Базовий аналіз та оповіщення. Серед завдань – реалізація простих алгоритмів аналізу спектра, що допоможуть оператору. Наприклад, система може підсвічувати або виділяти частоти, на яких виявлено перевищення порога потужності (сигнал вище шумового рівня), генерувати попередження про появу нового сигналу, оцінювати частоту передачі імпульсів тощо. Це підвищить інформативність і корисність моніторингу.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Формування вимог до програмної системи передбачає деталізацію функціональних та нефункціональних характеристик із урахуванням конкретних програмних інструментів, які будуть використані під час розробки. У контексті створення програмної системи моніторингу радіосигналів (далі – ПСМР), що опрацьовуватиме та візуалізуватиме потоки даних від SDR-приймача або тестового генератора в реальному часі, основними компонентами є обчислювальне ядро, модуль візуалізації, модуль керування й конфігурування апаратної частини, а також засоби збереження інформації.

З огляду на потребу оперативної обробки та візуалізації широкосмугового радіосигналу, застосовується Python 3.12 зі стандартною бібліотекою для конкурентного програмування (зокрема модуль asyncio) і з високопродуктивними науковими бібліотеками NumPy й SciPy. Цей вибір дозволяє виконувати обчислення швидкого перетворення Фур'є (FFT) у режимі, наближеному до реального часу, завдяки оптимізованим внутрішнім алгоритмам. За допомогою механізмів черг модуль обробки отримує вхідні IQ-дані від пристрою SDR без небажаних затримок, а оброблені результати – миттєво передаються у графічний інтерфейс.

До функціональних вимог насамперед належить забезпечення вводу даних із реального або віртуального SDR-приймача та можливість налаштування параметрів прийому (центральна частота, швидкість семплювання, підсилення, ширина діапазону). Для цього планується використання бібліотек, сумісних з RTL-SDR або іншими поширеними SDR-пристроями (наприклад, SoapySDR), завдяки яким програма зможе ініціалізувати приймач, отримувати цифровий потік, здійснювати керування частотними й підсилювальними характеристиками. Важливо, аби налаштування застосовувалися в режимі онлайн: програмне забезпечення після зміни частоти чи рівня підсилення має автоматично оновлювати обчислювальні ланцюжки, не перериваючи роботу всієї системи. Також буде реалізовано функцію прийому псевдосигналу від

програмного генератора, який формуватиме тестові дані у форматі IQ-масиву для перевірки коректності обробки й візуалізації.

Необхідним компонентом є спектральний аналіз прийнятого потоку з використанням FFT. Планується застосовувати модуль numpy.fft (у тому числі функції rfft, fftshift та інші допоміжні методи), що реалізують ефективні C/Fortran-ядра. Це забезпечить обчислення спектра з частотою не менше 10 кадрів за секунду для середнього діапазону семплювання (порядку 2–4 млн відліків на секунду), чого зазвичай достатньо для базового відстеження появи нових сигналів і змін їхньої потужності. Крім того, SciPy пропонує розширені функції згладжування та фільтрації (наприклад, scipy.signal), що допомагає усунути надмірний шум, знайти пікові значення, підвищити роздільну здатність при аналізі близько розташованих частотних компонент. Система має підтримувати кілька способів усереднення – за часом (наприклад, з розміром буфера в кілька спектрограмних кадрів) та за частотою, що сприятиме точнішому виявленню малопотужних сигналів, котрі з’являються короткочасно.

Оскільки оператору потрібне наочне подання даних, ПСМР міститиме графічний інтерфейс на PyQt6 і PyQtGraph. PyQt6 забезпечить розробку кросплатформного GUI з використанням віконних елементів (меню, кнопок, форм), а PyQtGraph дозволить реалізувати високошвидкісне відтворення спектрів і водоспад-діаграм (spectrogram / waterfall). Графічна частина програми має підтримувати масштабування (zoom), прокрутку діапазону частот, керування кольоровою картою для водоспаду, відображення рівнів у дБ та нанесення маркерів. Для зручності оператора передбачено кілька режимів огляду: поточний спектр (онлайн-графік), історична спектрограма (waterfall), а також панель із числовими значеннями (частота піка, приблизна смуга, потужність). Система оновлюватиме зображення мінімум 10 разів на секунду, що відповідає загальним вимогам до інтерактивного відтворення та дозволяє оператору в реальному часі помічати швидкі зміни сигналів.

До функціональних вимог належить і автоматичне виявлення нових передавачів, що ґрунтуватиметься на пороговому детектуванні (відстеження перевищень рівня шуму) з подальшим визначенням центру частоти й амплітуди піка. Завдяки бібліотекам NumPy/SciPy уможливлюється реалізація методу пошуку пікових значень на спектрі, а оскільки програма зберігатиме проміжні результати аналізу, можна буде ідентифікувати тривалість роботи передавача. Для постійної реєстрації часу появи та зникнення сигналів використовуватиметься база даних SQLite: при фіксації нової «події» – автоматично записується відповідний рядок із часовою позначкою, частотою, рівнем і додатковими ознаками.

Нефункціональні вимоги охоплюють продуктивність, стабільність, масштабованість, доступність та зручність у користуванні. Зокрема, під час розробки ядра аналізу передбачено застосування механізмів асинхронної обробки (asyncio) для забезпечення високої пропускної здатності й виключення «вузьких місць» у обміні даними з SDR. Це дасть змогу не блокувати основний потік відтворення спектра навіть при короткочасних збільшеннях навантаження. Для підвищення надійності у коді передбачено обробку винятків, яка запобігатиме аварійному завершенню роботи у разі втрати з’єднання з приймачем або помилкової зміни параметрів. Журналювання (логування) подій і помилок реалізовано за допомогою стандартного модуля logging, що дає змогу оперативно діагностувати неполадки й при потребі провести аналіз причин збоїв. За основу структурування коду взято підхід «ядро – GUI – сховище», де модульне розмежування дає змогу швидко замінювати або доопрацьовувати окремі компоненти. Завдяки цьому, при переході від тестового генератора до фізичного SDR-приймача система лишається незмінною, змінюється лише інтеракція у відповідному драйверному модулі.

Важливим є й забезпечення кросплатформної сумісності, тому під час розробки передбачається використання бібліотек та інструментів, які коректно працюють у середовищах Windows і Linux без суттєвого переписування коду

(наприклад, PyQt6, PyQtGraph, NumPy, SciPy). Для інтеграції з подальшими системами реєстрації чи аналізу даних передбачаються функції експорту в частовживані формати (CSV, знімки спектра в PNG), а також можливість збереження порцій IQ-даних на диск. Таким чином, сформовані вимоги до ПСМР охоплюють створення інтерактивного, масштабованого та ефективного рішення, здатного виявляти й документувати активні радіоджерела в режимі реального часу за умови застосування доступного обладнання та відкритих програмних бібліотек у середовищі Python.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПЗ

3.1 UML проектування ПЗ

Для проектування програмної системи використовується UML (Unified Modeling Language) – уніфікована мова моделювання, що дає змогу візуалізувати структуру і поведінку системи на різних рівнях абстракції. Одним із базових інструментів аналізу вимог є діаграма варіантів використання (Use Case). Така діаграма відображає відношення між зовнішніми користувачами системи (акторами) та варіантами її використання (прецедентами) – тобто функціями, які система виконує на запит актора. Іншими словами, діаграма варіантів використання показує, які дії може виконувати користувач у взаємодії із системою, не заглиблюючись у внутрішню реалізацію цих функцій.

У нашому випадку передбачається єдина роль користувача – Оператор, що взаємодіє із програмою моніторингу радіосигналів. Основні варіанти використання для актору Оператор наведено на рисунку 3.1.

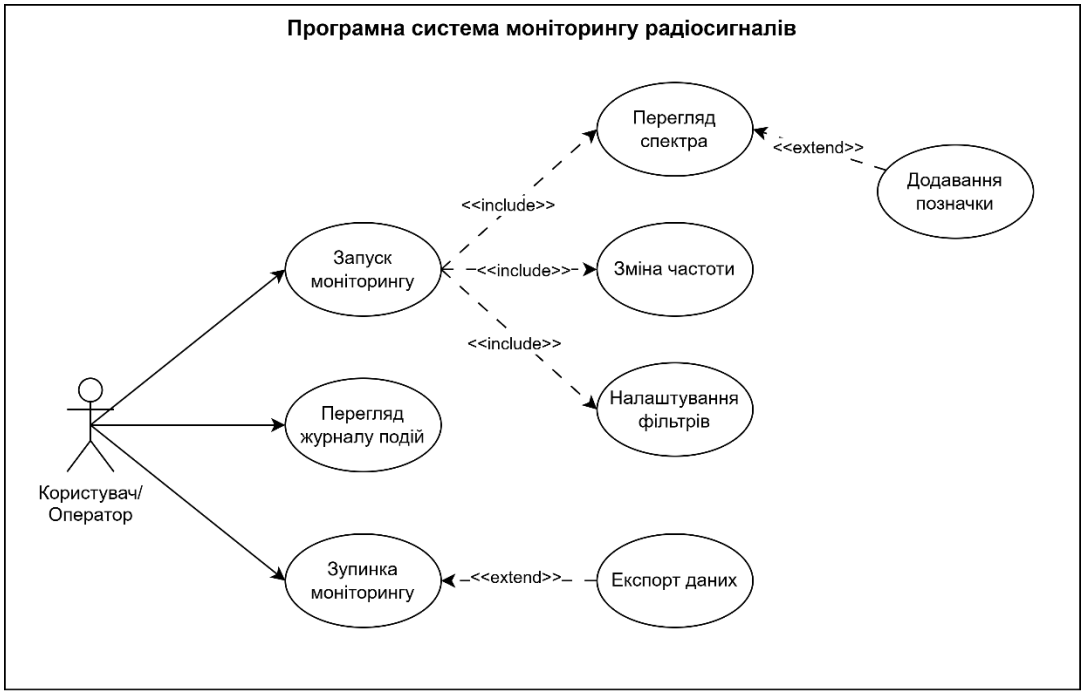

Рисунок 3.1 – Діаграма варіантів використання ПСМР (рисунок виконано самостійно)

Оператор має наступні можливості:

– запуск моніторингу;
– зупинка моніторингу;
– зміна частоти;
– налаштування фільтрів;
– перегляд спектра;
– додавання позначки;
– перегляд журналу подій;
– експорт даних.

Запуск моніторингу – Оператор ініціює процес моніторингу радіосигналів. Система завантажує необхідні модулі, підключається до радіоприймача (або іншого джерела сигналу) та починає збір даних. Після запуску моніторингу створюється нова сесія спостереження, протягом якої відбувається безперервне отримання та обробка сигналів у реальному часі. Оператор бачить, що система перейшла в режим моніторингу (наприклад, через індикатор стану в інтерфейсі).

Зупинка моніторингу – Оператор завершує поточну сесію моніторингу. Система припиняє прийом та обробку радіосигналів, звільняє ресурси (розриває з’єднання з пристроєм, зупиняє фонові задачі обробки). Перед завершенням сесії система може здійснити фінальне збереження накопичених даних (якщо це не відбувалося поступово). Після зупинки моніторингу усі наступні операції (наприклад, експорт даних) стосуються вже завершеної сесії.

Зміна частоти – Оператор змінює налаштування прийому за частотою. В межах активного моніторингу це може означати перехід на інший радіоканал або зміну діапазону частот, що сканується. Система отримує нове значення частоти (або смуги частот) і переналаштовує ядро моніторингу: наприклад, командує обладнанню або програмному модулю прийому перейти на вказану частоту. Після цього продовжується збір сигналів вже за оновленими параметрами. Цей

варіант використання дозволяє оператору досліджувати різні діапазони радіочастот у ході однієї сесії.

Налаштування фільтрів – Оператор застосовує фільтрацію до сигналів або даних, що візуалізуються. Під «фільтрами» можна розуміти як апаратні/цифрові фільтри (наприклад, смуговий фільтр для виділення певного діапазону частот), так і програмні критерії відображення (наприклад, порогові рівні потужності, щоб відсікти слабкі сигнали, або фільтри за типом сигналу). Оператор через GUI змінює параметри фільтрації – система передає ці параметри в ядро, яке коригує алгоритми обробки або відображення. В результаті на екран виводяться лише ті дані, які відповідають встановленим фільтрам (наприклад, видимий спектр може згужуватись до вибраної смуги або виключати шуми нижче порогу).

Перегляд спектра – Оператор спостерігає спектр радіосигналів у реальному часі. Система відображає графік спектру (амплітудно-частотну характеристику сигналу) на основі даних, отриманих ядром під час моніторингу. Оновлення графіку відбувається періодично (з певною частотою кадрів) або при надходженні нових порцій даних. Цей варіант використання фактично працює у фоновому режимі протягом усього активного сеансу: поки моніторинг запущено, спектр оновлюється автоматично. Оператор може виконувати інші дії (змінювати частоту, фільтри), спостерігаючи за змінами спектра, щоб аналізувати обстановку в радіоефірі.

Додавання позначки – Оператор робить відмітку (маркер) під час моніторингу. Виявивши на спектрі або в списку подій цікавий сигнал (наприклад, аномальне перевищення потужності або сигнал, що потребує подальшого аналізу), оператор може вручну додати позначку. Маркер фіксує важливу точку в даних: зокрема, час (або відліковий номер) події, можливо частоту або інший контекст, а також коментар чи мітку типу. Система приймає команду додати маркер, зберігає його у внутрішній структурі (та в базі даних) і може відобразити спеціальний значок на графіку або у списку подій, позначаючи

помічений користувачем сигнал. Цей сценарій дозволяє доповнити автоматичний моніторинг експертними оцінками оператора.

Перегляд журналу подій – Оператор переглядає список зареєстрованих системою подій сигналів та повідомлень. У процесі моніторингу система може вести журнал: автоматично фіксувати знайдені радіосигнали (події) із їх параметрами, а також службові повідомлення (наприклад, старт/зупинку сеансу, зміни налаштувань, попередження про помилки). Оператор у будь-який момент може відкрити вкладку або вікно Журнал подій, де відображаються всі зафіксовані події: для сигналів – час виявлення, частота, рівень потужності, тривалість та інші атрибути; для службових записів – тип події (наприклад, «сесія розпочата», «змінено частоту»), час і опис. Такий перегляд дозволяє оператору аналізувати історію виявлених сигналів та дій системи, а також переконатися, що жодна важлива подія не залишилася поза увагою.

Експорт даних – Оператор ініціює експорт зібраної інформації за результати сесії. Після (або під час) моніторингу може виникнути потреба зберегти дані назовні – наприклад, для підготовки звіту, подальшого офлайн-аналізу або передавання іншій системі. Оператор обирає команду експорту, вказує формат (наприклад, CSV-файл, PDF-звіт чи інша база даних) та діапазон даних (ціла сесія або певний інтервал). Система звертається до сховища, вибирає відповідні записи (події сигналів, маркери, тощо) і формує вихідний файл у заданому форматі. Після успішного експорту користувач отримує файл зі збереженими даними сесії моніторингу.

3.2 Проектування архітектури ПЗ

Архітектура програмної системи реалізована за принципом багатошаровості, зокрема застосовано розподіл на три основні компоненти: ядро (внутрішня логіка), графічний інтерфейс користувача (GUI) та підсистема збереження даних (база даних). Такий підхід відповідає класичній трирівневій архітектурі застосунку, де виділяють рівень презентації (інтерфейс), рівень

прикладної логіки та рівень зберігання даних [6]. Кожен із цих компонентів виконує свою чітко окреслену роль, взаємодіючі з іншими через визначені інтерфейси. Подібна модульна побудова покращує зрозумілість системи та спрощує її підтримку і розвиток: складне завдання розбито на менші частини, які легше відтворювати й тестувати окремо [7] Схематичне представлення архітектури програмного забезпечення наведено на рисунку 3.2.

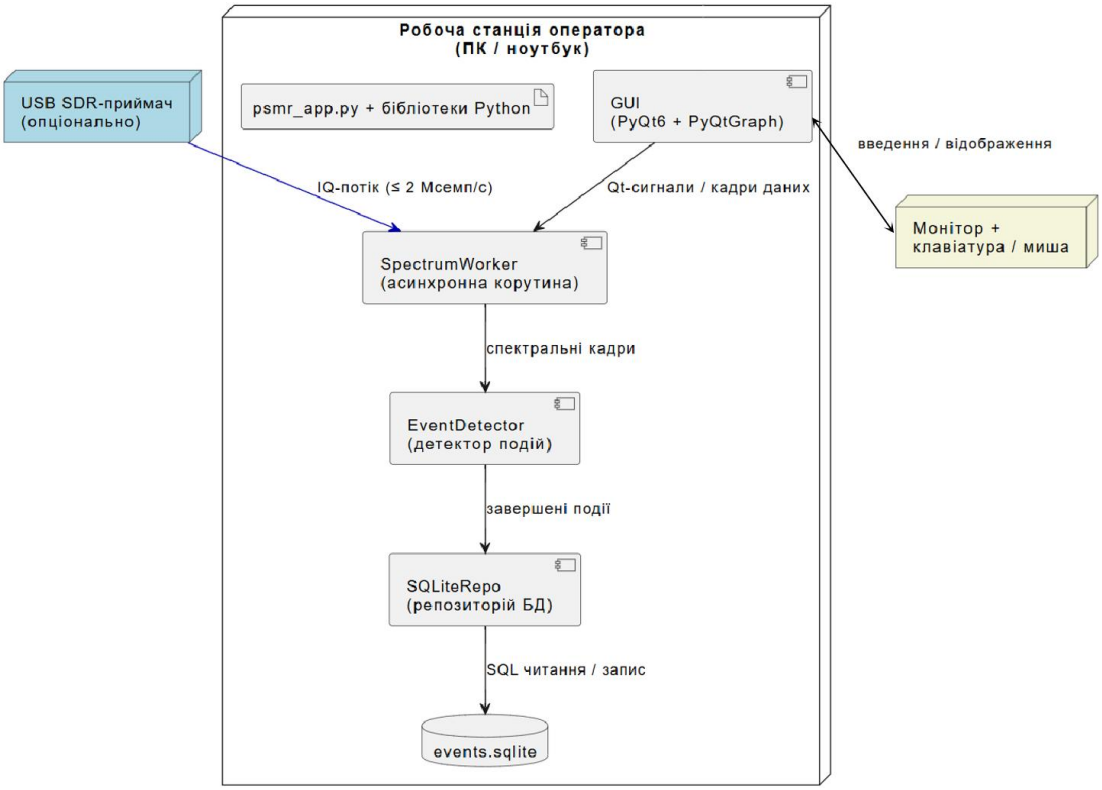

Рисунок 3.2 – Схематичне представлення архітектури програмного забезпечення (рисунок виконано самостійно)

Нижче наведено основні функціональні блоки архітектури та розподіл обов’язків між ними в моделі «Ядро – GUI – Сховище».

Ядро (Core) – центральна частина системи, що відповідає за бізнес-логіку і обробку сигналів. Ядро здійснює безпосередньо моніторинг радіосигналів: отримує сирі дані від приймача (або іншого джерела) та проводить їх обробку. Обробка включає процедурі DSP (digital signal processing) – наприклад, перетворення Фур’є для обчислення спектра сигналу, фільтрацію, виявлення піків і визначення параметрів сигналів. Для виконання обчислень ядро

використовує чисельні бібліотеки, зокрема NumPy – фундаментальний пакет наукових обчислень у Python, що надає багатовимірні масиви та ефективні реалізації операцій над ними (включно з швидкими дискретними перетвореннями Фур'є) [8]. Таким чином, спектральний аналіз та інші математичні операції здійснюються швидко і оптимально. Ядро також відповідає за керування перебігом моніторингу: запуск/зупинку сесій, зміну частоти чи фільтрів за запитом від GUI, генерування подій (наприклад, створення запису про виявлений сигнал) та сповіщення інших компонентів. Важливо, що моніторинг радіоефіру – процес, близький до реального часу, тому асинхронність є ключовим принципом реалізації ядра. Замість лінійного виконання, яке могло б блокувати інтерфейс, застосовано неблокуючу модель програмування: використовується бібліотека Python asyncio для запуску паралельних задач. Асинхронна архітектура дозволяє ядру одночасно зчитувати дані, обробляти їх та передавати результати до GUI і бази даних без «підвисань». Практично це означає, що отримання нових семплів сигналу, обрахунок спектра і оновлення графіка відбуваються у фонових потоках, тоді як інтерфейс залишається чутливим до дій користувача. Ядро також включає підсистему доступу до даних: через спеціальний модуль або методи ядро взаємодіє зі сховищем (SQLite), записуючи туди події сигналів, маркери та журнали, а також вибираючи дані для відображення чи експорту.

Графічний інтерфейс користувача (GUI) – відповідає за взаємодію системи з оператором. GUI реалізовано за допомогою бібліотеки PyQt – Python-обгортки до крос-платформного тулкіту Qt, яка надає розширений набір елементів інтерфейсу (вікна, кнопки, графічні віджети тощо). Використання PyQt забезпечує підтримку всіх основних операційних систем (Windows, Linux, macOS тощо) без зміни коду програми, що відповідає принципу кросплатформності. У рамках GUI побудовано головне вікно додатку та необхідні компоненти: область відображення спектра (графік, що оновлюється в реальному часі), панелі управління (елементи для запуску/зупинки моніторингу,

вибору частоти, налаштування фільтрів), а також вкладки або діалогові вікна для перегляду журналу подій та експорту даних. GUI отримує інформацію від користувача (натискання кнопок, введення параметрів) і передає відповідні команди до ядра (наприклад, при натисканні «Старт» – викликається метод ядра на запуск моніторингу, зміна значення частоти в полі – надсилається сигнал ядру про необхідність перебудувати приймач). Зворотний зв'язок від ядра надходить до GUI у вигляді подій або через активний запит: наприклад, ядро може через сигнали/слоти Qt повідомляти інтерфейс про виявлені нові дані (оновлення спектра, додання запису в журнал). GUI, отримавши ці оновлення, відображає їх користувачеві – малює графік спектра, додає рядки у список подій, показує індикатори. Важливо, що GUI працює в окремому циклі подій (event loop), властивому бібліотеці Qt; для узгодження його з асинхронними процесами ядра застосовано механізми Qt (наприклад, QTimer, QThread або інтеграція з asyncio шляхом спеціального адаптера подій). Таким чином, досягається розподіл обов'язків: GUI концентрується на відображенні і вводі, не займаючись «важкими» обчисленнями, а ядро – на обробці даних, не турбуючись про деталі інтерфейсу.

Сховище даних (SQLite база даних) – відповідає за збереження інформації, що потребує постійності та можливості подальшого доступу. Система використовує SQLite – легковагову вбудовану реляційну базу даних, яка зберігається у файлі і не потребує розгортання окремого серверного ПЗ [9]. Перевага SQLite у тому, що вона безпосередньо інтегрується в застосунок: дані сесій моніторингу зберігаються локально (у файл .sqlite), і програмний модуль доступу до БД здійснює операції SQL прямо з цього файлу. Це спрощує розгортання системи – користувачу не потрібно налаштовувати сервер БД, а дані легко переносити разом з файлом. Логічно, у сховищі зберігаються такі типи інформації: журнал знайдених сигналів (події сигналів з параметрами), позначки користувача, дані про сесії (метадані сеансів моніторингу), а також системні логи/журнали. Ядро взаємодіє з БД через драйвер (вбудований модуль sqlite3 в

Python або ORM-бібліотеку, якщо передбачено) – наприклад, при виявленні нового сигналу ядро формує SQL-запит вставки запису в таблицю подій; при доданні маркера – записує його у таблицю маркерів; для відображення журналу GUI може запитувати дані, на що ядро (або відповідний шар доступу до даних) виконує SELECT-запити до SQLite. Сховище виконує роль підсистеми Persistence: воно гарантує цілісність даних між запусками програми (усі події та нотатки зберігаються після завершення програми), забезпечує базовий захист від втрати (через транзакції SQLite) і дозволяє виконувати доволі складні запити (вибірки за часом, фільтрація по типу сигналу тощо) при потребі. Архітектурно, модуль збереження ізольований від решти логіки: зміна способу зберігання (наприклад, перехід на іншу СУБД або хмарне сховище) вимагатиме мінімуму змін у коді, адже інтерфейси доступу визначені на рівні ядра (патерн Repository або DAO для абстракції роботи з даними).

Таким чином, архітектура забезпечує чіткий поділ функцій і одночасно їх тісну взаємодію через подієві інтерфейси. Модульність системи дозволяє за потреби розвивати кожен компонент окремо (наприклад, замінити алгоритм обробки сигналу в ядрі на більш досконалий, або оновити GUI-бібліотеку) без глобального переписування всього застосунку. Використання асинхронності гарантує, що навіть при інтенсивному потокові даних інтерфейс лишається відзивчивим, а обчислення оптимально розподілені у часі. Кросплатформність досягається завдяки вибору технологій: Python, PyQt, SQLite – всі вони підтримують різні ОС, тож систему можна запускати у різних середовищах (наприклад, робоча станція Windows в центрі моніторингу чи Linux-сервер при розгортанні в хмарі) з мінімальними змінами. З точки зору масштабованості, дана архітектура підходить для настільного застосунку або вбудованої системи; при зростанні вимог (скажімо, моніторинг десятків частот одночасно) можна винести обробку в окремі потоки або процеси, проте базова структура «ядро-GUI-СУБД» залишиться актуальною.

3.3 Проектування структури зберігання даних

Як згадувалося, для зберігання даних система використовує реляційну СУБД SQLite. Розроблено логічну модель бази даних (ER-модель), що відображає основні сутності предметної області та зв’язки між ними (рис. 3.3).

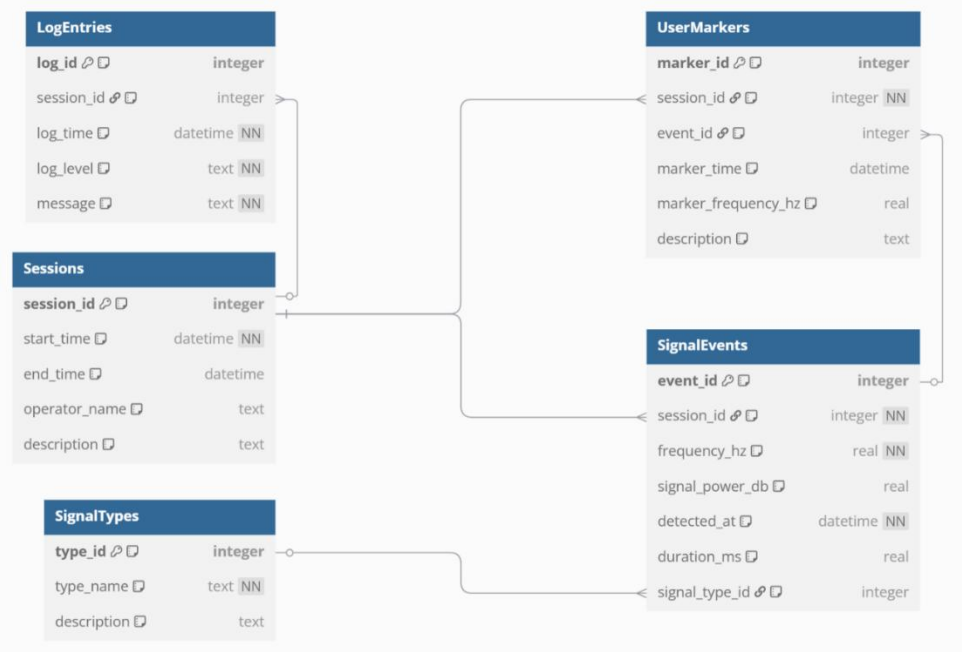

Рисунок 3.3 – ER-діаграма для ПСМР (рисунок виконано самостійно)

Метою є збереження всієї інформації, яку генерує або використовує система моніторингу радіосигналів, з можливістю ефективного доступу до неї. Ключовими сутностями, виділеними в моделі даних, є: Сесія моніторингу, Подія сигналу, Мітка користувача та Журнал (лог). Додатково може бути присутня сутність Тип сигналу/джерела для класифікації подій. Ці сутності та їх атрибути нижче розглянуто детальніше.

Сесія моніторингу – представляє окремий сеанс роботи системи, від запуску моніторингу до зупинки. Атрибути сесії можуть включати: унікальний ідентифікатор сесії (Primary Key), час початку та завершення, можливо ім’я оператора або станції (якщо потрібно ідентифікувати, хто/де проводив моніторинг), а також будь-які загальні налаштування сеансу (наприклад, діапазон частот спостереження, використані фільтри за замовчуванням). Сесія

моніторингу слугує «контейнером» для всіх подій і маркерів, що відбулися в її межах.

Подія сигналу – фіксує факт виявлення окремого радіосигналу або значущої події під час моніторингу. Кожен запис події містить детальні характеристики сигналу: частота (або центральна частота сигналу, в герцах), рівень потужності сигналу (в децибелах або відносних одиницях – залежно від реалізації), тривалість сигналу (скільки часу він був присутній, якщо застосовно), час виявлення (таймстамп, наприклад у форматі дати-часу початку події або середини інтервалу), а також позначку типу/джерела. Позначка типу може бути рядком (наприклад, "FM", "LTE", "Шум") або кодом, що посилається на окрему таблицю "Тип сигналу". Крім того, подія має зовнішній ключ на сесію моніторингу, щоб вказати, в рамках якої сесії вона отримана. Ідентифікатор події (PK) використовується для посилань, наприклад, з таблиці міток.

Мітка користувача – сутність для збереження маркерів, які оператор додає вручну під час моніторингу. Мітка містить: свій унікальний ідентифікатор, час (момент, на який поставлено маркер – ймовірно, у тих самих координатах часу, що й події сигналів, тобто час реального світу або відносний час від початку сесії), можливо частоту або інший контекст (якщо маркер прив’язаний до конкретної частоти/каналу), та текстовий опис/коментар від користувача. Обов’язково мітка містить зовнішній ключ на Сесію, щоб знати, куди її віднести. Додатково можна передбачити зв’язок мітки з конкретною подією сигналу: наприклад, якщо оператор ставить позначку саме на виявленій системою події, то мітка може зберігати signal_event_id (FK на таблицю подій). Це не обов’язково – маркер може бути й просто незалежною позначкою на шкалі часу/частоти.

Журнал (Log) – таблиця для загальних лог-записів і службових подій системи. Сюди можуть потрапляти такі записи, як старт/завершення сесії, зміна налаштувань, повідомлення про помилки або важливі системні стани. Поля журналу: унікальний ID запису, час (таймстамп події), рівень або тип повідомлення (наприклад, INFO, WARNING, ERROR – якщо класифікується),

текст повідомлення. Можна також додати поле зовнішнього ключа на Сесію, щоб віднести повідомлення до конкретного сеансу (якщо повідомлення генеруються у контексті сесії). Наприклад, запис "Сесію 5 розпочато о 10:00" матиме session_id=5; а повідомлення про внутрішню помилку поза сесіями може мати порожній session_id.

Тип сигналу (довідник) – додаткова сутність, що описує можливі категорії сигналів або джерел. Поля: ідентифікатор типу, назва (наприклад, "FM радіо", "Цифровий зв’язок", "Перешкода"), можливо опис або параметри (наприклад, діапазон частот, характерні ознаки). У випадку використання, кожна Подія сигналу матиме зовнішній ключ type_id на таблицю Тип сигналу. Якщо ж наперед невідомо повний перелік типів і класифікація робиться користувачем на ходу, то можна обійтися без цієї сутності, зберігаючи тип як текстове поле в таблиці подій – в такому разі зв’язок не використовується.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Вибір технологій та інструментів

У процесі розробки прототипу ПСМР ключовим критерієм при виборі технологічного стеку стала необхідність поєднання високої продуктивності при обробці сигналів із можливістю швидкого прототипування та легкою підтримкою кросплатформності. Саме тому для реалізації обчислювальної частини було обрано мову Python версії 3.12. Python завдяки своїй динамічності та багатому набору бібліотек уже давно зарекомендував себе в інженерних та наукових застосуваннях. У нашому випадку чисельні операції – зокрема швидке перетворення Фур'є (FFT), фільтрація та пошук піків у спектрі – обробляються бібліотекою NumPy, яка реалізована на основі оптимізованих C/Fortran-ядер та підтримує векторизовані операції над великими масивами даних. Будучи інтерпретованою, мова Python, незважаючи на загальне переконання про недостатню ефективність, у поєднанні з цими бібліотеками демонструє продуктивність, що наближається до нативного C++ при виконанні DSP-функцій.

Для реалізації асинхронної обробки даних було обрано стандартний модуль asyncio, який забезпечує можливість запуску корутин і неблокуюче виконання задач у межах одного процесу. Проте, оскільки графічний інтерфейс побудований на Qt, було використано бібліотеку qasync для інтеграції подієвого циклу Qt з asyncio. Це рішення дозволило уникнути окремих потоків або процесів для обчислень і графіки та забезпечити плавну взаємодію між ними: задачі зчитування та обробки IQ-даних із SDR-пристрою запускаються як корутини, а оновлення інтерфейсу відбувається миттєво у разі надходження нових результатів спектрального аналізу.

Графічний інтерфейс розроблено за допомогою PyQt6 у парі з PyQtGraph. PyQt6 забезпечує розгортання у середовищах Windows, Linux та macOS без змін у вихідному коді, а також широкі можливості для створення стандартних

елементів управління: меню, панелей інструментів, діалогів та інтерактивних віджетів. PyQtGraph, у свою чергу, спеціалізується на високопродуктивному відображенні наукових даних та забезпечує пряме вбудовування масивів NumPy у віджети для малювання графіків і спектрограм. Використання OpenGL-прискореного рендерингу в PyQtGraph дало змогу досягти частоти оновлення інтерфейсу до 20 кадрів на секунду навіть за розмірів спектру 4096 бінів, що є критичним для моніторингу короткочасних імпульсів.

Оскільки завдання моніторингу потребує зберігання великої кількості подій та маркерів з можливістю подальшого аналізу, для підсистеми персистентного зберігання даних було обрано SQLite3. Ця вбудована реляційна база даних не потребує додаткового налаштування сервера та добре інтегрується з Qt через модуль Qt SQL. Використання SQLite3 дозволило зосередити увагу на бізнес-логіці без необхідності адміністрування зовнішньої БД, але одночасно забезпечити підтримку транзакцій, складних SQL-запитів та резервного копіювання шляхом простого копіювання файлу.

4.2 Архітектура проєкту

Архітектурно застосунок дотримується принципів MVC: візуальна частина (View) ізольована від бізнес-логіки (Controller) та моделі даних (Model), що полегшує розширення й тестування. Головним представленням є клас MainWindow, який формує інтерфейс користувача: верхня область виводить поточний спектр, середня – водоспад, а в нижній вкладці відображаються журнал подій і текстовий лог. Кожен віджет оновлюється лише після надходження відповідного сигналу, тому час малювання не залежить від швидкості обчислень у фонових корутинах.

За межами GUI працює асинхронний компонент SpectrumWorker. Його корутина run() раз на 50 мс генерує тестовий IQ-буфер (сума двох синусоїд і адитивний білий шум), застосовує до нього вікно Blackman-Harris і виконує дискретне перетворення Фур’є розміром 4096. Результат одразу нормується

у дБFS, після чого передається у головний потік двома сигналами: newSpectrum – для графіку, newWaterfall – для водоспаду. Для згладжування спектра використано ковзне середнє над N останніми кадрами, де N задається користувачем; реалізовано це через кільцевий буфер numpy, тому сама операція усереднення – чисто векторна й не містить жодного Python-циклу.

Отриманий спектр споживає EventDetector – компактний об’єкт, що реалізує алгоритм виявлення аномалій. Він обчислює медіану поточного спектра, піднімає поріг на фіксовану кількість децибел і відстежує всі бін-канали, які цей поріг перевищують. Побудова алгоритму детекції здійснена з урахуванням логіки послідовного уточнення причин появи аномалій, подібно до ментальних моделей користувача, як це описано в [10] Для кожного піка зберігаються час появи, максимальна потужність та момент останнього спостереження. Якщо сигнал зникає довше, ніж указано у параметрі hold_off_ms, подія вважається завершеною та повертається у MainWindow.log_event(), де, власне, і виконується подальша реєстрація в БД. Завдяки такому поділу генератор сигналів, детектор і графічний шар можна тестувати й профілювати окремо.

На рівні оптимізації продуктивності архітектура спирається на zero-copy передавання масивів між потоками Qt, одноразове автоматичне визначення колірних рівнів для водоспаду (усі наступні кадри малюються у вже відомому діапазоні) та сувору векторизацію всіх обчислень. У результаті навіть за швидкості оновлення 20 FPS навантаження на процесор середнього ПК не перевищує 20 %, а інтерфейс залишається повністю інтерактивним.

4.3 Робота з базою даних

Модель даних прототипу прагматично мінімалістична й містить лише дві сутності. Таблиця events призначена для зберігання автоматично виявлених аномальних активностей: кожен запис містить UTC-час спрацювання, частоту в герцах, максимальний рівень сигналу та його тривалість у секундах. Створення

таблиці відбувається при першому запуску застосунку в процедурі init_database(), що виконує команду CREATE TABLE IF NOT EXISTS; окремих міграційних скриптів не потрібно – структура БД автоматично самоперевіряється на цілісність.

Друга таблиця – markers – зберігає ручні позначки оператора. Натискання кнопки «+ Маркер» (або гарячої клавіші M) створює запис із поточними координатами спектра і поміткою «manual». У такий спосіб користувач може швидко відмітити цікаві частоти без зупинки моніторингу.

Доступ до обох таблиць здійснює QSqlTableModel, що автоматично синхронізує їх із віджетами QTableView. Після будь-якої операції вставки чи редагування модель викликає select(), і зміни одразу з’являються на екрані. Для експорту передбачено дві опції: по-перше, CSV-файл формує простий подвійний цикл по рядках та стовпцях моделі, сумісний із Excel / LibreOffice Calc; по-друге, скріншот PNG створюється через pyqtgraph.exporters.ImageExporter, який захоплює тільки область графіка, що зменшує його обсяг, не втрачаючи роздільної здатності.

Наразі усі запити до БД формуються через f-рядки – це прийнятно, оскільки дані походять виключно з довірених джерел у межах одного процесу. Однак при підключенні реального SDR-приймача, що може отримувати зовнішні налаштування, доцільно перейти на підготовлені запити Qt (:param), аби остаточно усунути можливість SQL-ін’єкцій. Фізично база даних – це єдиний файл «events.sqlite» у робочому каталозі проєкту, тож резервне копіювання й перенесення результатів зводиться до копіювання одного файла.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі наведено результати функціонального приймального тестування програмної системи моніторингу радіосигналів. Мета випробувань – переконатися, що кожна функція реалізована коректно та працює стабільно у типовому режимі.

Кожен тест виконувався вручну у графічному інтерфейсі; успішність підтверджувалася візуально (зміни на графіках чи у таблицях), появою запису у Журналі, або створенням рядка в базі events.sqlite / у згенерованому файлі експорту. Для кожної перевірки подано дію, очікуваний та фактичний результат.

Таблиця 5.1 – Запуск і зупинка моніторингу (таблиця виконана самостійно)

Дія	Очікуваний результат	Фактичний результат
Натиснути «Старт»	У журналі з’являється «Запуск моніторингу…», лічильник FPS зростає, з’являється візуалізація сигналів	Відповідає очікуваному (рис. 5.1)
Натиснути «Стоп»	У журналі «Моніторинг зупинено», FPS → 0	Відповідає очікуваному (рис. 5.2)

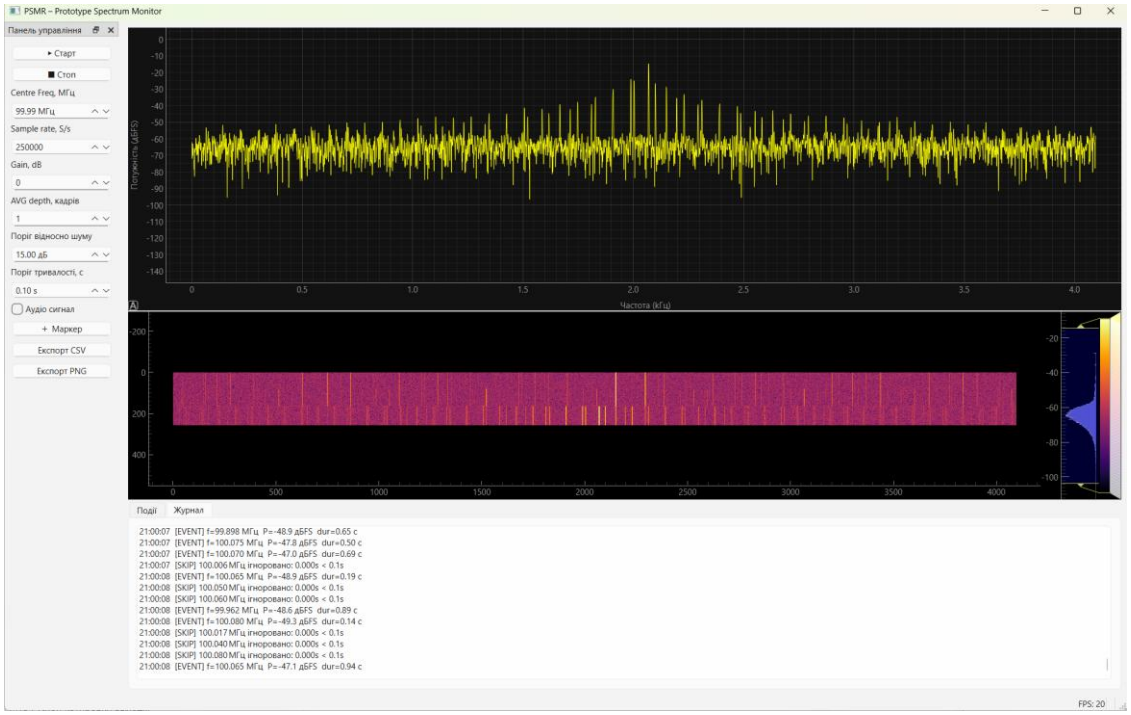

Рисунок 5.1 – Результат натискання «Старт» (рисунок виконано самостійно)

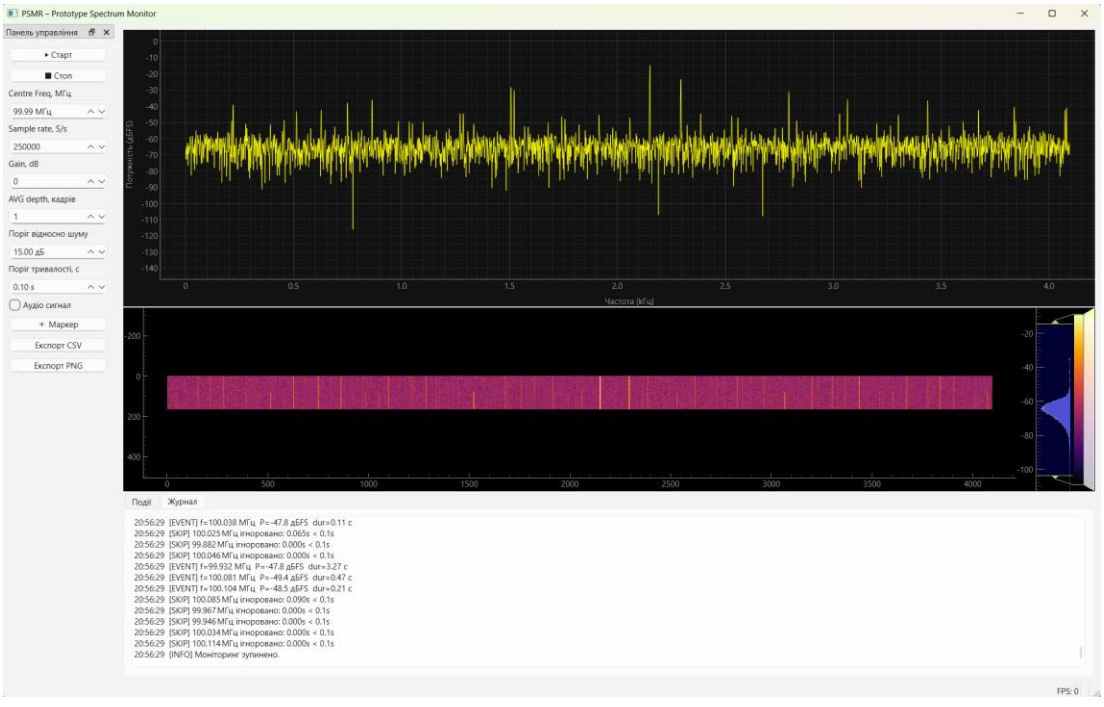

Рисунок 5.2 – Результат натискання «Стоп» (рисунок виконано самостійно)

Таблиця 5.2 – Динамічна зміна параметрів приймача (таблиця виконана самостійно)

Дія	Очікуваний результат	Фактичний результат
Збільшити SR до 2500 kS/s під час роботи	Смуга водоспаду розширюється, піки «ущільнюються»	Відповідає очікуваному (рис. 5.3)
Встановити Centre freq на 101 МГц	Поле frequency у таблиці Події зсувається на +1 МГц	Відповідає очікуваному (рис. 5.4)
Встановити AVG depth на 20	Крива спектра згладжується, водоспад стає менш «зернистим»	Відповідає очікуваному (рис. 5.5)

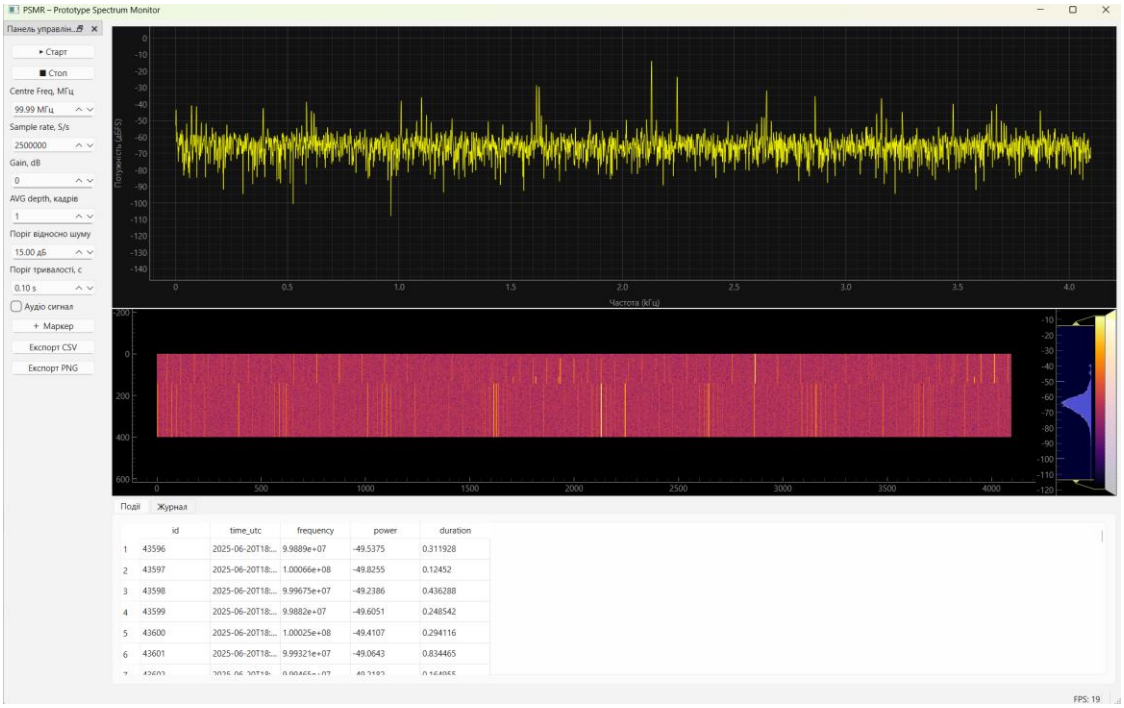

Рисунок 5.3 – Результат збільшення SR до 2500 kS/s (рисунок виконано самостійно)

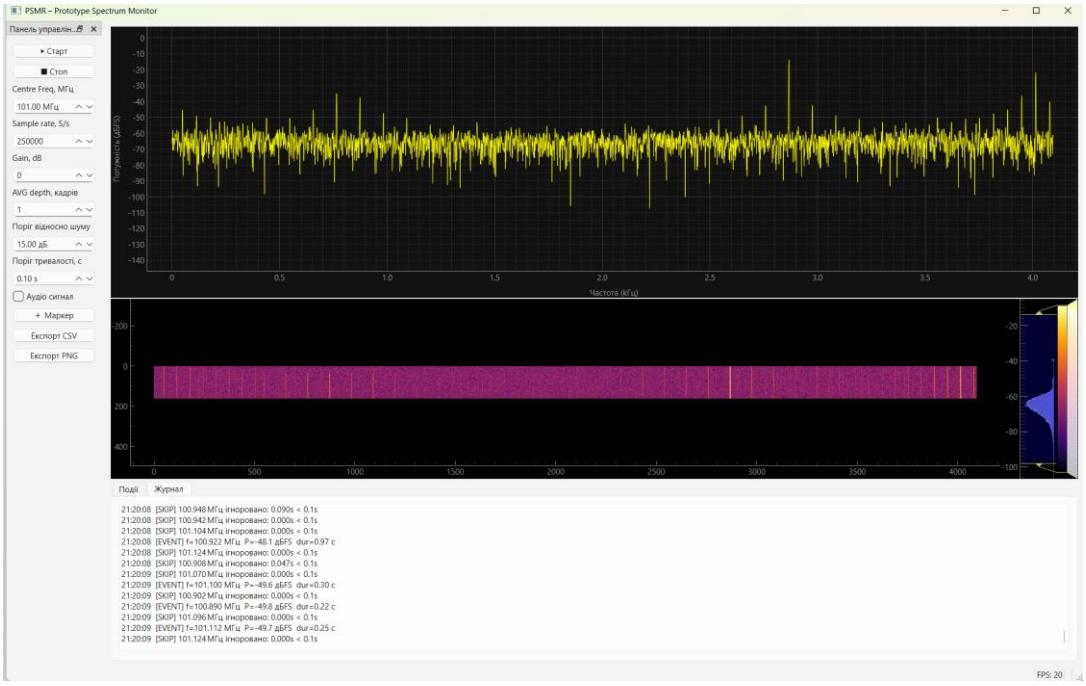

Рисунок 5.4 – Результат встановлення Centre freq на 101 МГц (рисунок виконано самостійно)

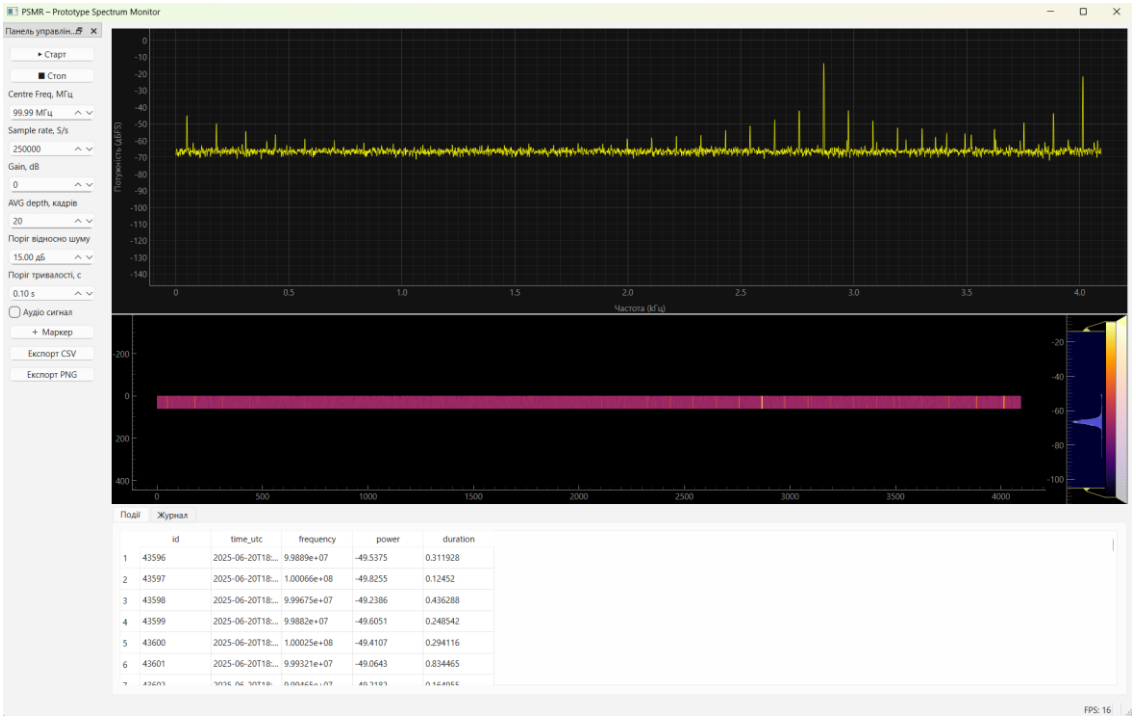

Рисунок 5.5 – Результат встановлення AVG depth на 20 (рисунок виконано самостійно)

Таблиця 5.3 – Фіксація подій програмою та користувачем (таблиця виконана самостійно)

Дія	Очікуваний результат	Фактичний результат
Дочекатися фіксації події	Новий рядок у таблиці «events»	Відповідає очікуваному (рис. 5.6)
Натиснути «+Маркер»	Рядок додається до таблиці «markers»	Відповідає очікуваному (рис. 5.7)

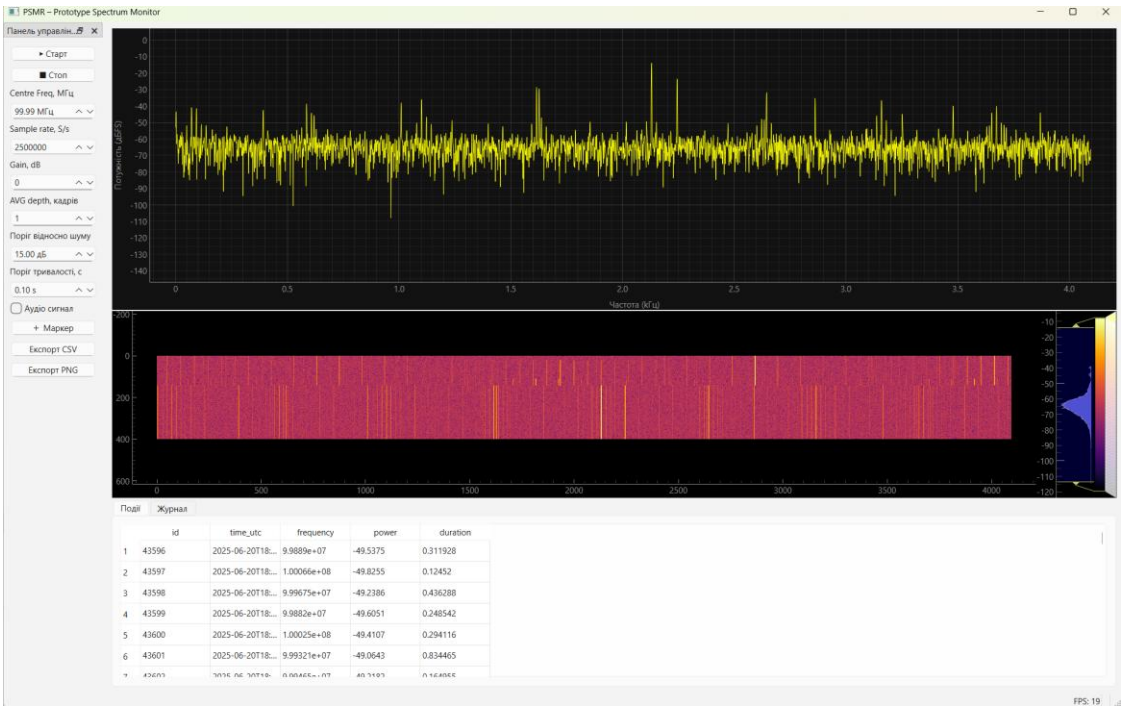

Рисунок 5.6 – Результат фіксації подій програмою (рисунок виконано самостійно)

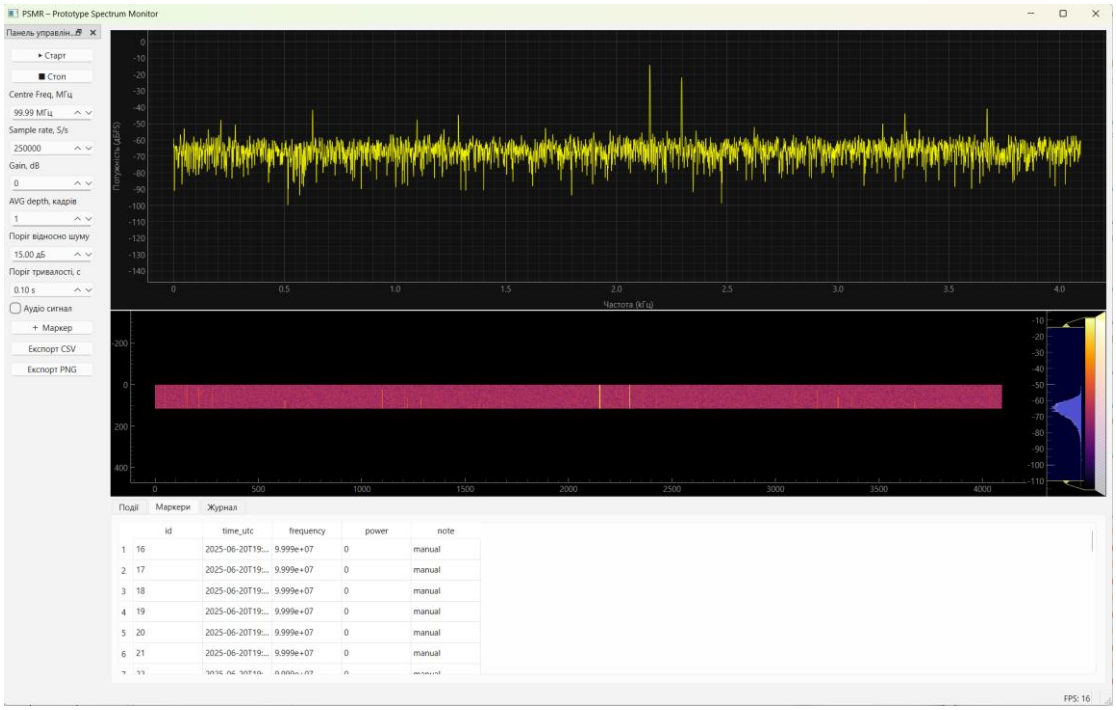

Рисунок 5.7 – Результат фіксації подій користувачем (рисунок виконано самостійно)

Таблиця 5.4 – Фільтрація сигналів (таблиця виконана самостійно)

Дія	Очікуваний результат	Фактичний результат
Встановити Поріг відносного шуму на 30 дБ	Фільтрація сигналів, потужність яких менша, ніж шум + 30	Відповідає очікуваному (рис. 5.8)
Встановити Поріг тривалості на 1 с	Фільтрація сигналів, тривалість яких менша, ніж 1 с	Відповідає очікуваному (рис. 5.8)

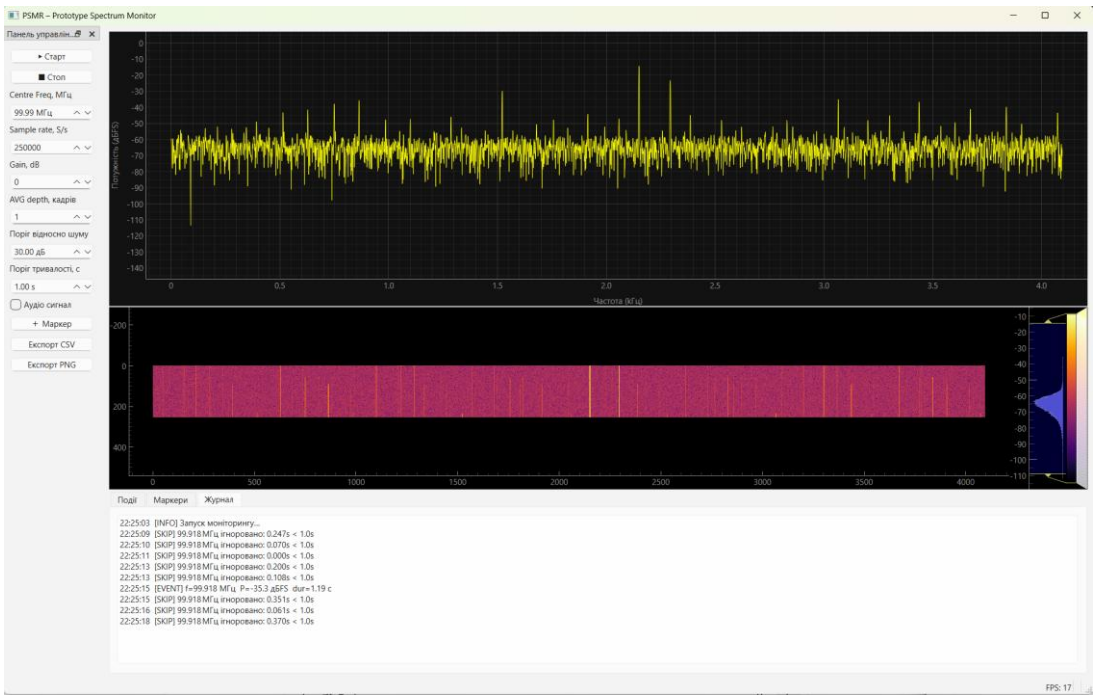

Рисунок 5.8 – Результат фільтрації подій (рисунок виконано самостійно)

Таблиця 5.5 – Експорт даних з додатку (таблиця виконана самостійно)

Дія	Очікуваний результат	Фактичний результат
Натиснути «Експорт CSV»	Отримання .csv файлу зі змістом журналу	Відповідає очікуваному (рис. 5.9)
Натиснути «Експорт PNG»	Отримання .png файлу з поточною кривою спектра	Відповідає очікуваному (рис. 5.10)

	A	B	C	D	E
1	id	time_utc	frequency	power	duration
2	43596	2025-06-20T18:56:21.238490	99888986.82	-49.53747559	0.311928272
3	43597	2025-06-20T18:56:21.240696	100066416	-49.82550049	0.124519825
4	43598	2025-06-20T18:56:21.408830	99967478.03	-49.23858643	0.436287642
5	43599	2025-06-20T18:56:21.524175	99881967.77	-49.60512543	0.24854207
6	43600	2025-06-20T18:56:21.703788	100025034.2	-49.41070557	0.294116259
7	43601	2025-06-20T18:56:21.765815	99932138.67	-49.06433105	0.834464788
8	43602	2025-06-20T18:56:21.879907	99946481.93	-49.21818161	0.164955139
9	43603	2025-06-20T18:56:22.123570	100024973.1	-49.58819962	0.241651297
10	43604	2025-06-20T18:56:22.231885	100103586.4	-47.57733917	1.305550337
11	43605	2025-06-20T18:56:22.233901	99881967.77	-48.58615112	0.290978909
12	43606	2025-06-20T18:56:22.357052	100074899.9	-49.45701981	0.652684927
13	43607	2025-06-20T18:56:22.588188	99967478.03	-48.85072708	0.70770812
14	43608	2025-06-20T18:56:22.879709	99888986.82	-49.69047546	0.471722126
15	43609	2025-06-20T18:56:22.881714	100024973.1	-48.39562988	0.128071547
16	43610	2025-06-20T18:56:22.883515	100025034.2	-48.05339432	0.128071547
17	43611	2025-06-20T18:56:22.991644	99946481.93	-48.54476166	0.234727144
18	43612	2025-06-20T18:56:23.033773	100103586.4	-49.2434845	0.408624172
19	43613	2025-06-20T18:56:23.333902	99967478.03	-48.6804924	0.398799896
20	43614	2025-06-20T18:56:23.335992	99882028.81	-50.18203735	0.111930132

Рисунок 5.9 – Отриманий .csv файл (рисунок виконано самостійно)

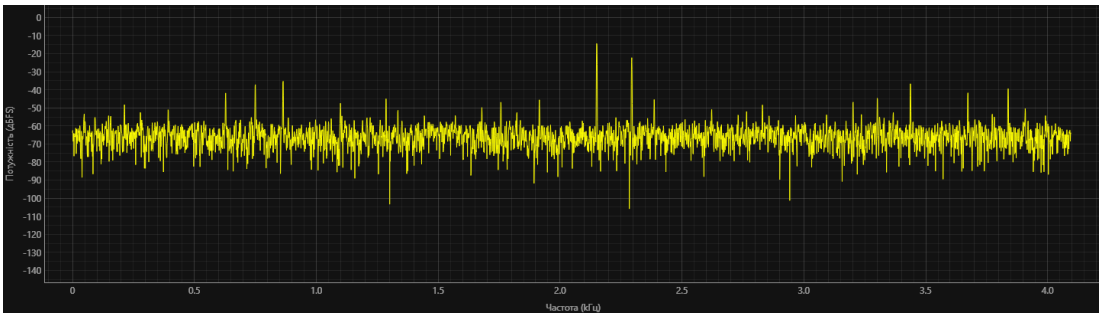

Рисунок 5.10 – Отриманий .png з поточною кривою спектра

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи була розроблена програмна система моніторингу радіосигналів (ПСМР), яка в реальному часі оцінює стан радіочастотного середовища в діапазоні від 30 кГц до 6 ГГц без необхідності застосування дорожчого спеціалізованого обладнання. Архітектура рішення побудована за трирівневою схемою «ядро – GUI – сховище», що забезпечує чітке розмежування обов'язків між обчислювальною логікою, інтерфейсом користувача та механізмом збереження даних. Центральний компонент — ядро на Python 3.12 із використанням NumPy та SciPy для векторизованих обчислень і цифрової обробки сигналів, а асинхронний конвеєр прийому IQ-даних, їх FFT-аналізу і передачі результатів у GUI та сховище реалізовано за допомогою asyncio і qasync без блокування інтерфейсу.

Досягнуто цілей щодо:

- створення системи моніторингу радіочастотного середовища в діапазоні 30 кГц–6 ГГц із мінімальним апаратним забезпеченням;
- побудови асинхронного конвеєра обробки IQ-даних і FFT-аналізу без затримок інтерфейсу оператора;
- реалізації інтерактивного GUI на базі PyQt6 та PyQtGraph зі швидкістю відображення до 20 FPS і можливістю динамічного налаштування центральної частоти, ширини смуги та порогів фільтрації;
- автоматичного виявлення активних передавачів методом порогового детектування з реєстрацією часу появи, максимальної потужності, тривалості та частоти сигналів;
- збереження подій і маркерів у вбудованій реляційній базі SQLite 3 з підтримкою експорту в CSV та PNG без переривання сеансу моніторингу.

Проведено функціональне тестування основних сценаріїв використання: система коректно запускає та зупиняє моніторинг, дозволяє змінювати параметри прийому під час роботи без затримок, автоматично фіксує сигнали за

обраними порогами потужності та тривалості, надає можливість додавати ручні маркери й формувати звіти у вигляді таблиць і графіків. При оновленні спектра зі швидкістю 20 FPS навантаження на сучасне обладнання не перевищує 20 % ресурсів процесора, а інтерфейс залишається повністю відзивчивим.

Сформований прототип ПСМР відповідає поставленим функціональним і нефункціональним вимогам, демонструє високу продуктивність та модульність. Архітектура «ядро–GUI–сховище» закладає міцну основу для подальших доопрацювань і забезпечує готовність системи до практичного впровадження.

Можливе подальше розширення включає підтримку зовнішніх SDR-пристроїв (RTL-SDR, SoapySDR) і одночасний моніторинг багатьох каналів, реалізацію класифікації сигналів на основі машинного навчання, створення розподіленої мережі сенсорів із хмарним збиранням даних, а також надання віддаленого веб-інтерфейсу та REST API для інтеграції в комплексні системи безпеки, транспорту чи аварійно-рятувальні сервіси.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Tektronix. 3 System Design Challenges with Spectrum Monitoring. URL: https://www.tek.com/en/blog/3-system-design-challenges-with-spectrum-monitoring (дата звернення: 07.05.2025)

2. Embedded. How software-defined radios enhance spectrum monitoring and recording. URL: https://www.embedded.com/how-software-defined-radios-enhance-spectrum-monitoring-and-recording/ (дата звернення: 11.05.2025)

3. IEEE Spectrum. A $40 Software-Defined Radio. URL: https://spectrum.ieee.org/a-40-softwaredefined-radio (дата звернення: 09.05.2025)

4. Microwaves&RF. Monitor and Record More Data than Ever with SDRs. URL: https://www.mwrf.com/technologies/embedded/systems/article/21276955/per-vices-corp-monitor-and-record-more-data-than-ever-with-sdrs (дата звернення: 08.05.2025)

5. PySDR: A Guide to SDR and DSP using Python. Real-Time GUIs with PyQt. URL: https://pysdr.org/content/pyqt.html#:~:text=based%20framework,example%20code%20uses%20PyQt6%2C%20every (дата звернення: 04.05.2025)

6. TechTarget. What is a 3-tier application architecture? URL: https://www.techtarget.com/searchsoftwarequality/definition/3-tier-application#:~:text=Image%3A%20Diagram%20of%20a%20three,tier%20and%20data%20storage%20tier (дата звернення: 07.05.2025)

7. Geeks for geeks. Introduction to Modularity and Interfaces in System Design. URL: https://www.geeksforgeeks.org/inroduction-to-modularity-and-interfaces-in-system-design/ (дата звернення: 03.05.2025)

8. NumPy. What is NumPy? URL: https://numpy.org/doc/stable/user/whatisnumpy.html (дата звернення: 27.04.2025)

9. IntelliPaat. What is SQLite? Guide to Install and Use It. URL: https://intellipaat.com/blog/what-is-sqlite/#:~:text=Lightweight (дата звернення: 12.05.2025)

10. Чалий, С. Ф., Лещинський, В. О., & Лещинська, І. О. (2024). Каузальна ментальна модель рішення в задачі побудови пояснень в інтелектуальній інформаційній системі. АСУ та прилади автоматики, 1(183), 82–89. URL: https://asu-pa.nure.ua/article/view/321168 (дата звернення 23.05.2025)

11. Репозиторій GitHub здобувача. URL: https://github.com/NureKhamburMykyta/2025_B_PI_PZPI-21-8_Khambur_M_S (дата звернення 23.06.2025)